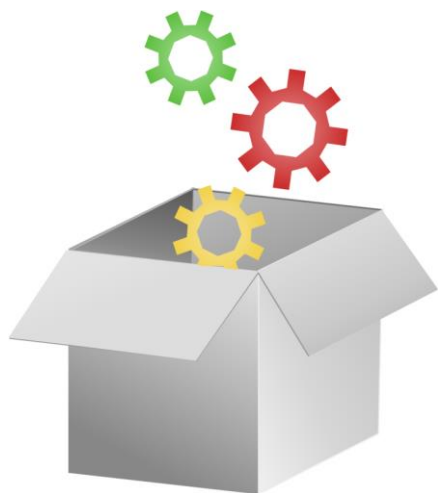


# MethodSCRIPT SDK Example – Linux



Last revision: April 20, 2020

© 2020 PalmSens BV

[www.palmsens.com](http://www.palmsens.com)

## 1 Contents

The example *MethodSCRIPTExample.c* found in the */MethodSCRIPTExample\_C\_Linux/src* folder demonstrates basic communication with the EmStat Pico. The example allows the user to start measurements on the EmStat Pico from a Linux PC using a simple C program which makes use of the MethodSCRIPT SDK (C libraries).

## 2 Basic Console Example (MethodSCRIPTExample.c)

This example demonstrates how to implement serial communication with the EmStat Pico to

- Establish a connection with the device
- Write a MethodSCRIPT to the device
- Read and parse the measurement data packages from the device
- Print the parsed data to the console
- Write the parsed data to a CSV file

This does not include error handling, method validation etc.

This example is build using Eclipse CDK (snap distribution) on Ubuntu 19.04.

The console output of one of the examples is shown below.

```
<terminated> (exit value: 0) C example original Debug [C/C++ Application]
Serial port successfully connected to EmStat Pico.

MethodSCRIPT sent to EmStat Pico.

Response begin

Measuring...

Receiving measurement response:
```

1	E(V): -0.500	i(A) : -4.967E-05	Status: OK	CR: 1mA (High speed)
2	E(V): -0.450	i(A) : -4.480E-05	Status: OK	CR: 1mA (High speed)
3	E(V): -0.400	i(A) : -3.982E-05	Status: OK	CR: 1mA (High speed)
4	E(V): -0.351	i(A) : -3.489E-05	Status: OK	CR: 1mA (High speed)
5	E(V): -0.301	i(A) : -2.990E-05	Status: OK	CR: 1mA (High speed)
6	E(V): -0.251	i(A) : -2.492E-05	Status: OK	CR: 1mA (High speed)
7	E(V): -0.201	i(A) : -1.999E-05	Status: OK	CR: 1mA (High speed)
8	E(V): -0.151	i(A) : -1.506E-05	Status: OK	CR: 1mA (High speed)
9	E(V): -0.102	i(A) : -1.014E-05	Status: Underload	CR: 1mA (High speed)
10	E(V): -0.052	i(A) : -5.152E-06	Status: Underload	CR: 1mA (High speed)
11	E(V): -0.002	i(A) : -2.240E-07	Status: Underload	CR: 1mA (High speed)
12	E(V): 0.048	i(A) : 4.704E-06	Status: Underload	CR: 1mA (High speed)
13	E(V): 0.098	i(A) : 9.632E-06	Status: Underload	CR: 1mA (High speed)
14	E(V): 0.147	i(A) : 1.462E-05	Status: OK	CR: 1mA (High speed)
15	E(V): 0.197	i(A) : 1.954E-05	Status: OK	CR: 1mA (High speed)
16	E(V): 0.247	i(A) : 2.453E-05	Status: OK	CR: 1mA (High speed)
17	E(V): 0.297	i(A) : 2.946E-05	Status: OK	CR: 1mA (High speed)
18	E(V): 0.347	i(A) : 3.444E-05	Status: OK	CR: 1mA (High speed)
19	E(V): 0.396	i(A) : 3.942E-05	Status: OK	CR: 1mA (High speed)
20	E(V): 0.446	i(A) : 4.435E-05	Status: OK	CR: 1mA (High speed)
21	E(V): 0.496	i(A) : 4.934E-05	Status: OK	CR: 1mA (High speed)

```
Measurement completed. 21 data point(s) received.
```

### 3 Communications

The *MSComm.c* from the MethodSCRIPT SDK (C libraries) acts as the communication object to read/write from/to the EmStat Pico. The port it will open is configured using the *BAUD\_RATE* define located in “MethodSCRIPTExample.h”. The serial port of the Pico can be obtained by issuing the following command in the terminal:

```
dmesg | grep FTDI
```

It is usually called “ttyUSBx” where x is a number.

The example implements the serial communication interface in the file “SerialPort\_Linux.c”. This file implement the same C-interface to provide a layer of abstraction. On Linux the serial port can be accessed using the TERMIOS library which abstracts the interface as if it is a file and provides open, read, write and close functions. Besides that the port has to be configured using cflags. Note: only baudrates of the type *speed\_t* are supported. The exact configuration for this example is as follows:

```
// Set baudrate for both input and output
speed_t baud_config = baud_to_termios(BAUD_RATE);
cfsetispeed(&config, baud_config);
cfsetospeed(&config, baud_config);

// Input flags - Turn off input processing and flow control
config.c_iflag &= ~(IXON | IXOFF | IXANY);

// Local mode flags - disable echo and put the interface in non-canonical
mode
config.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

// Output flags - Turn off output processing
config.c_oflag &= ~OPOST;

// Control mode flags - Turn off output processing and act as null-modem
config.c_cflag &= ~(PARENB | CSTOPB | CSIZE | CRTSCTS);
config.c_cflag |= CS8 | CREAD | CLOCAL;
```

This configuration is set in the *OpenSerialPort* function of the example. The complete abstracted interface for this example is shown below.

```
/// Opens the serial port to which the EmStat Pico is connected.
/// Returns: 1 on successful connection, 0 in case of failure.
int OpenSerialPort();

/// Writes the input character to the device
/// Returns: 1 if data is written successfully, 0 in case of failure.
int WriteToDevice(char c);

/// Reads a character read from the EmStat Pico
/// Returns: -1 on failure or the value of the received byte on success
int ReadFromDevice();

/// Closes the serial port
/// Returns: 1 if closed successfully, 0 in case of failure.
int CloseSerialPort();
```

The *ReadFromDevice* / *WriteToDevice* functions are required to initiate the *MSComm* communication library and have to be passed to the *init* function.

```
MSComm msComm;
RetCode code = MSCommInit(&msComm, &WriteToDevice, &ReadFromDevice);
```

### 3.1 Sending the MethodSCRIPT

The MethodSCRIPT can be read from a txt file stored in the PC. In this example, the MethodSCRIPT files are stored in the ScriptFiles directory. The code snippet below is used in the example to read the MethodSCRIPT from the file and in turn send it to the device.

```
int SendScriptFile(char* fileName)
{
    FILE *fp;
    //Including the string termination character (0)
    char str[MS_MAX_LINECHARS+1];

    fp = fopen(fileName, "r");
    if (fp == NULL) {
        printf("Could not open file %s", fileName);
        return FAILURE;
    }
    // Reads a single line from the script file and sends it to the device.
    while (fgets(str, MS_MAX_LINECHARS, fp) != NULL)
    {
        WriteStr(&msComm, str);
    }
    fclose(fp);
    return SUCCESS;
}
```

### 3.2 Receiving measurement data packages

This example uses the *MSComm* library to receive and parse the data packages from a measurement. In order to read and parse the measurement data packages from the device, the Receive Package function from the *MSComm* library can be used. This function requires a reference to an initiated *MSComm* struct (*msComm*) and it returns the parsed data in the referenced MeasureData struct (*data*)

```
code = ReceivePackage(&msComm, &data);
```

### 3.3 Parsing the measurement data packages

Each measurement data package returned by the function *ReadBuf()* in *MSComm* library, can be parsed further to obtain the actual data values. For example, here is a set of data packages received from a Linear Sweep Voltammetry (LSV) measurement on a dummy cell with 10kOhm resistance.

```
e\n
M0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n
```

While parsing a measurement package, various identifiers are used to identify the type of package. For example, In the above sample,

1. 'e' is the confirmation of the "execute MethodSCRIPT" command.
2. 'M' marks the beginning of a measurement loop.
3. 'P' marks the beginning of a measurement data package.
4. "\*" marks the end of a measurement loop.
5. "\n" marks the end of the MethodSCRIPT.

Most techniques return the data values Potential (set cell potential in V) and Current (measured current in A). The data values to be received from a measurement can be sent through 'pck' commands in the MethodSCRIPT.

In case of Electrochemical Impedance Spectroscopy (EIS) measurements, the following *variable types* can be sent with the MethodSCRIPT and received as measurement data values.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following metadata values can also be obtained from the data packages, if present.

- CurrentStatus (OK, Underload, Overload, Overload warning)
- CurrentRange (the current range in use)
- Noise

#### 3.3.1 Parsing the measurement data packages

Each measurement data package begins with the header 'P' and is terminated by a '\n'. The measurement data package can be split into data value packages based on the delimiter ';'. Each of these data value packages can then be parsed separately to get the actual data values.

The type of data in a data package is identified by its variable type:

- The potential readings are identified by the string "da"
- The current readings are identified by the string "ba"
- The frequency readings are identified by the string "dc"
- The real impedance readings are identified by the string "cc"
- The imaginary impedance readings are identified by the string "cd"

For example, in the sample package seen above, the *variable types* are  
*da7F85F3Fu* - "da" for potential reading and  
*ba48D503Dp,10,288* - "ba" for current reading.

The following 7 characters hold the 28-bit signed integer data value followed by one SI unit prefix character. The data value for the current reading (7 characters) from the above sample package is “48D503D” followed by the SI unit prefix ‘p’ (pico, which is 1e-12 A).

After obtaining variable type and the data values from the package, the metadata values can be parsed, if present.

### 3.3.2 Parsing the metadata values

The metadata values are separated based on the delimiter ‘,’ and each of the values is further parsed to get the actual value.

The first character of each metadata value metaData[0] identifies the type of metadata.

‘1’ - status  
 ‘2’ - Current range index  
 ‘4’ - Noise

The metadata status is a 1 character hexadecimal bit mask.

For example, in the above sample, the available metadata values for current data are, 10,288. The first metadata value is 10.

1 – metadata status – 0 indicates OK.

The metadata type current range is represented by a 2-digit hexadecimal value. If the first bit is high (0x80), it indicates a high-speed mode current range. The hexadecimal value can be converted to int to get the current range.

For example, in the above sample, the second metadata available is 288.

2 – indicates the type – current range

88 – indicates the hexadecimal value for current range index – 1mA. The first bit 8 implies that it is high speed mode current range.

## 3.3.3 Sample output

### 3.3.3.1 LSV

Here's a sample measurement data package from a LSV measurement on a dummy cell with 10kOhm resistance and its corresponding output.

```
Pda7F85F3Fu;ba4BA99F0p,10,288
```

Output:            E (V) = -4.999E-01  
                  i (A) = -4.999E-01  
                  Status : OK  
                  CR : 1mA (High speed)

### 3.3.3.2 EIS

Here's a sample measurement data package from an EIS measurement on a dummy cell with 10 kOhm resistance and its corresponding output.

```
PdcDF5DFF4u;cc896D904m,10,287;cd82DB1A8u,10,287
```

Output:            Frequency(Hz): 100.0  
                  Zreal(Ohm): 9885.956  
                  Zimag(Ohm): 2.995  
                  Status: OK  
                  CR: 200uA (High speed)