EWIO$_2$
Remote Control API

Table of contents

1. Introduction

The communication between EWIO$_2$ and web interface is managed through HTTP over TCP/IP. The application layer that is transferred with HTTP can, however, also be used by other applications than a web browser.

Therefore, in principle any communication coming through a web interface can also be run with applications that use API.

In addition, general functions (as for SQL instructions) are available for additional functionality.

2. Principles of establishing communication

Each communication is protected with a TAN. It is therefore necessary to conduct a login procedure before the actual sending or receiving of user data. Within these, any number of communication steps (functions) are possible.

Communication steps:

Request ID

Login

Function 1

Function 2

…

Logout

3. Overview of functions

3.1. Administrative functions

3.1.1. Request session ID

The server generates a random ID that is sent to the client.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<pw-md5-hash>&type=session_id&module=login

Return: <Session ID>

3.1.2. Login

The client sends a link of session ID and password (= TAN) to the server. If the TAN is correct, this server returns a LOGIN OK, otherwise a LOGIN ERROR.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=login&module=login

Return: LOGIN OK / LOGIN ERROR

3.1.3. Logout

If communication is ended on the client side, the client will send a logout to the server. Otherwise, the logout is initiated automatically by the server after 30 minutes without communication.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=logout&module=login

Return: LOGOUT OK

3.1.4. Reboot

If communication is terminated on the client side with a EWIO$_2$ reboot, the client sends a reboot to the server. After the return message is sent, the EWIO$_2$ is rebooted. After the reboot, a new session ID must be requested and the login procecure must be repeated.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=reboot&module=login

Return: REBOOT OK

3.2. General functions

3.2.1. Load file (text file)

The client requests a text file that is currently on the EWIO$_2$.

Method: GET
URL: <server-ip>/cgi-bin/getTxtFileFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=fileload&module=<Source address of file>

Return: <data contents>

3.2.2. Save file (text file)

The client sends a text file to the server.

Method: POST
URL: <server-ip>/cgi-bin/setTxtFileToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=fileload&
module=<Target address of file>&data=<Data contents>

Return: -

3.2.3. Load file (binary file)

The client requests a binary file that is currently on the EWIO$_2$.

Method: GET
URL: <server-ip>/cgi-bin/getBinFileFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=fileload_bin&
module=<Source address of file>

Return: *<data contents>*

3.2.4. Save file (binary file)

The client sends a binary file to the server.

Method: POST
URL: <server-ip>/cgi-bin/setBinFileToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=fileload_bin&
module=<Target address of file>&data=<Data contents>

Return: -

3.2.5. Request directory contents

The client requests the contents of the directory declared under "module". The result is the directory contents in JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=dir_download&module=<path of dir>

Return: <JSON structure>

```
{
"files":
[
"counter.conf",
"ewio2server.ini",
"IoDriver.conf"
]
}
```

Example of JSON structure for the contents of a directory, here: Directory /var/opt/etc

3.3. Load / save device configuration

The device configuration is subdivided into multiple topics which are combined in a configuration data set. The configuration data set will either be in text format or in JSON format, depending on topic.

Configuration data sets on the following topics can be loaded and saved:

Topic	Module parameter	Data format	Load / save	Description
Gerät	device	JSON	load only	Device model and serial number
Version	version	JSON	load only	API version and software version
Date and time	datetime	JSON	Load and save	Configuration of date, time, time zone and timeserver
Device configuration	devicebase	JSON	load only	Device model, serial number, MAC address, software version and capacity of the flash drive
Memory	memory	JSON	load only	Capacity of flash drive and SD card
I/O connections	io_driver	Text	Load and save	Configuration of I/O connection of the EWIO$_2$

3.3.1. Load device configuration

The client requests configuration data on a particular topic. The topic is declared as a parameter "module".

(Ex.: Topic date and time ➔ module=datetime)

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=configuration&
module=<Topic>

Return: <Configuration data set>

3.3.2. Save device configuration

The client sends configuration data to the EWIO$_2$ on a topic that should be stored there. The topic is declared as a parameter "module".

(Ex.: Topic date and time ➔ module=datetime)

The configuration must be in the data format of the topic. With configuration data sets in JSON format, undeclared data remains unchanged. With configuration data sets in text format, the saved configuration is activated as a whole (undeclared data result in default values).

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=configuration&
module=<Topic>&data=<Configuration data>

Return: -

```
{
      "model":"EWIO2-MW-BM",
      "serial":"00000018"
}
```

Example of JSON structure of the device configuration device

```
{
      "api-version":1,
      "kernel-version":"4.9.88-ewio+g5ed2ee422",
      "package-version":"0.6",
      "build-number":"5f85065d"
}
```

Example of JSON structure of the device configuration version

```
{
      "datetime_utc":"2020-04-23 08:35:57",
      "datetime_local":"2020-04-23 10:35:57",
      "time_zone":"Europe\/Berlin",
      "time_server_1":"0.de.pool.ntp.org",
      "time_server_2":"1.de.pool.ntp.org"
}
```

Example of JSON structure of the device configuration datetime

Special aspects of saving the device configuration datetime:

- When time_server_...fields are declared, data on the current time is ignored.
- The field datetime_utc must be used to set the current time (the field datetime_local is ignored).
- After the device configuration datetime is saved, the API session should be ended with a reboot of the EWIO$_2$ (see **Fehler! Verweisquelle konnte nicht gefunden werden.**) to set all system components to the new configuration.


```
{
	"model":"EWIO2-MW-BM",
	"devicename":"EWIO2-a58649",
	"mac_address":"70:b3:d5:a5:86:49",
	"serialnumber":"00000018",
	"package_version":"0.6",
	"build_number":"5f85065d",
	"v_kernel":"4.9.88-ewio+g5ed2ee422",
	"flash_size":3625360,
	"flash_free":3536248
}
```


Example of JSON structure of the device configuration devicebase


```
{
	"sdcard_available":true,
	"sdcard_format":"vfat",
	"flash_size":3625360,
	"flash_free":3536240,
	"sd_size":3922944,
	"sd_free":3922912
}
```


Example of JSON structure of the device configuration memory

```
; IO Driver Configuration, automatically generated

FileWriteCount = 2

[device]
VerHardware IO = 1
VerHardware REL = 3

[relay outputs]
RelHandEnable = 15
RelHandActive = 0
RelHandValue = 0
RelDefault = 0

[transistor outputs]
DoHandEnable = 15
DoHandActive = 0
DoHandValue = 0
DoDefault = 0

[digital inputs]
DiDebounce 0 = 8
DiDebounce 1 = 8
DiDebounce 2 = 8
DiDebounce 3 = 8
DiDebounce 4 = 8
DiDebounce 5 = 8
DiDebounce 6 = 8
DiDebounce 7 = 8

[analogue outputs]
AoHandEnable = 7
AoHandActive = 0
AoHandValue 0 = 0
AoHandValue 1 = 0
AoHandValue 2 = 0
AoHandValue 3 = 0
AoDefault 0 = 0
AoDefault 1 = 0
AoDefault 2 = 0
AoDefault 3 = 0

[analogue inputs]
AiRange 0 = 1
AiRange 1 = 1
AiRange 2 = 1
AiRange 3 = 0
AiSensor 0 = 0
AiSensor 1 = 0
AiSensor 2 = 0
AiSensor 3 = 0

[sensor characteristics]
SensorTable 0
```

Example of text structure of the device configuration io_driver

3.4. Functions for applications

3.4.1. Load application list

The client requests the list of the current applications on the EWIO$_2$. The parameter "module" is not used and can be empty or can just be omitted. The result is the list of the current applications in JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=applications&
module=

Return: <JSON structure>

```
{
"apps":
[
"example1",
"example2"
]
}
```

Example of JSON structure of an application list

3.4.2. Load application

The client requests an application currently residing on the EWIO$_2$. The application is declared as a parameter "modul" without extension.

(Ex.: S0M_convert ➔ module= S0M_convert)

The application has a special structure and is in JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=application&
module=<Application>

Return: <JSON structure>

3.4.3. Save application

The client sends an application to the EWIO$_2$ that should be stored there. The file is declared as a parameter "modul" without extension.

(Ex.: S0M_convert ➔ module= S0M_convert)

The application has a special structure and must be in JSON format.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=application&
module=<Application>&data=<JSON structure>

Return: -

```
{
    "name": "S0M_convert",
    "kill": 0,
    "run": 1,
    "script": "#!/bin/sh\n# Description: Convert S0/M response file from pulse
to counter values\n\n# Cycle (in us):\nCYCLE=1000000\n\n# Declaration:\n\n#
Initialization - runs one time:\n\n# execute program only with 2 arguments
\nif [ $# -ne 2 ]; then    \n  exit    \nfi    \n\n# execute
conversion\n/config/bin/counter/counter_convertS0M $1 $2\n\n"
}
```

Example of JSON structure of an application (here: S0M_convert)

3.5. M-Bus functions

3.5.1. Load counter list

A list of the configured M-Bus counters is requested by the client.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=counter_list&
module= mbus_add_auto

Return: <JSON structure>

```
{
"counter":
[
{"ID":3,"BusAdr":"19235000-SYS-10-14","Name":"19235000-SYS-10-14"},
{"ID":13,"BusAdr":"01040904-GAV-90-2","Name":"01040904-GAV-90-2"},
{"ID":15,"BusAdr":"19235200-SYS-10-14","Name":"19235200-SYS-10-14"}
]
}
```

Example of JSON structure of a counter list

3.5.2. Delete counter configuration

The declared counter should be deleted from the configuration for M-Bus counters on the EWIO$_2$.

The counters to be deleted must be declared in a special JSON structure:

{"counter": [<name_1>, <name_2>, ...>]}

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=counter_remove&
module=mbus_remove&data=<JSON structure>

Return: -

```
data={
    "counter": [
        "19235200-SYS-10-14"
    ]
}
```

Example of JSON structure of a list for counters to be deleted

3.5.3. Save counter configuration

The configuration of an M-Bus counter is saved. The configuration must be in a special JSON format.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=counter&
module=<Counter Bus addr>&data=<JSON structure>

Return: -

3.5.4. Load counter configuration

The configuration of the declared M-Bus counter is requested by the client.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=counter&
module=<Counter Bus addr>

Return: *<Configuration>*

```
data={
    "ID":1,
    "BusAdr":"19235200-SYS-10-14",
    "PrimAdr":10
    "BR":9600,
    "Name":"19235200-SYS-10-14",
    "Sort":1,
    "ZNummer":"123456",
    "Mandant":"",
    "AbnNr":"",
    "Gewerk":0,
    "Hersteller":"",
    "Konto":"",
    "Ort":"",
    "Kommentar":"",
    "btype":"MBus",
    "ztype":"",
    "meteringcode":{
        "Land": "DE",
        "Betreiber": 123456,
        "Plz": 12345,
        "MePuId": "EXAMPLE0123456789012"
    }
}
```

Example of JSON structure of a counter configuration

3.5.5. Load data point list

The list of data points of a M-Bus counter is requested. The list is represented in a special JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=channels&
module=<Counter Bus addr>

Return: *<JSON structure>*


```
{
    "channel": [
        {
            "ID":1,
            "Zaehler_ID":1,
            "Freeze_ID":null,
            "Obis_ID":"",
            "Faktor_U":1,
            "Faktor_I":1,
            "EAblesung":"2020-06-05 11:20:00",
            "Intervall":5,
            "ZwTyp":"Volts",
            "MEinheit":"1.000000 V",
            "MessTyp":"",
            "Primaer":0,
            "App":"",
            "AppArgs":"",
            "Faktor":1
        },
        {
            "ID":2,
            "Zaehler_ID":1,
            "Freeze_ID":null,
            "Obis_ID":"",
            "Faktor_U":1,
            "Faktor_I":1,
            "EAblesung":"2020-06-05 11:20:00",
            "Intervall":5,
            "ZwTyp":"Energy",
            "MEinheit":"10 Wh",
            "MessTyp":"",
            "Primaer":0,
            "App":"",
            "AppArgs":"",
            "Faktor":1
        }
    ]
}
```


Example of JSON format of data point list

3.5.6. Save data point configuration

The configuration of a data point of a M-Bus counter is saved. The configuration must be in a special JSON format.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=channel&
module=<Data point ID>&data=<Data point Konfiguration>

Return: -

3.5.7. Load data point configuration

The configuration of a data point of a M-Bus counter is requested. The configuration is represented in a special JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=channel&
module=<Data point ID>

Return: *<JSON structure>*

```
{
    "channel": [
        {
            "ID":1,
            "Zaehler_ID":1,
            "Obis_ID":"",
            "Faktor_U":1,
            "Faktor_I":1,
            "EAblesung":"2020-06-05 11:20:00",
            "Intervall":5,
            "ZwTyp":"Volts",
            "MEinheit":"1.000000 V",
            "MessTyp":"",
            "Primaer":0,
            "App":"",
            "AppArgs":"",
            "Faktor":1
        }
    ]
}
```

Example of JSON format of data point configuration

3.5.8. Load M-Bus parameters for the documentation

The configuration of a counter with all associated data points is output as a text file.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=output&
module=mbus_<Counter Bus addr>

Return: *<Text>*

```
Counter ID: 1
Query Number: 1
Counter Name: 00243216-ABB-32-2
MBUS-ID: 00243216-ABB-32-2
Baudrate: 2400
Counter Number:
Client:
Customer:
Craft: 0
Manufacturer:
Account:
Location:
Counter Type:
Bus Type: MBus
Meteringcode: DE12345612345EXAMPLE0123456789012

Data Points
       Channel ID: 1
       Freeze ID:
       OBIS ID:
       Readout From: 2020-04-16 11:15:00
       Interval: 5
       Description: Cumulation Counter
       Factor Voltage: 1
       Factor Current: 1
       Factor: 1
       Unit: 1.000000 V
       Prim. Counter: 0
       Application:
       Application Arguments:
```

Example of JSON structure of counter configuration

3.6. Modbus functions

3.6.1. Load counter type list

The templates of all configurable counter types are loaded. The counter types are contained in a special JSON structure.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=deviceType_list&
module=modbus_config

Return: <JSON structure>

```
{
"deviceTypes":
[
"modbus_schneider_Compact_NSX.json",
"modbus_kbr_multimess_4F144-1-LED.json",
"modbus_janitza_umg96s.json",
"modbus_mc_mr-do4.json",
"modbus_mc_mr-ai8_temperature.json",
"modbus_mc_mr-ci4_current-0-20.json",
"modbus_mc_mr-ai8_resistance.json",
"modbus_schneider_PM3255.json",
"modbus_schneider_iEM3250.json",
"modbus_mc_mr-di10.json",
"modbus_janitza_umg604.json",
"modbus_janitza_umg503.json",
"modbus_econ_sens+.json",
"modbus_mc_mr-tp.json",
"modbus_mc_mr-ci4_current-4-20.json",
"modbus_mc_mr-ai8_voltage.json",
"modbus_mc_mr-ci4_voltage.json",
"modbus_mc_mr-si4.json",
"modbus_mc_mr-di4.json",
"modbus_schneider_iEM3255.json",
"modbus_mc_mr-ao4.json"
]
}
```

Example of JSON format of counter type list

3.6.2. Load counter list

A list of the configured Modbus counters is requested by the client.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=counter_list&
module= modbus_output

Return: <JSON structure>

```
{
"counter":
[
{"ID":14,"BusAdr":"11","Name":"mr do4"}
]
}
```

Example of JSON structure of a counter list

3.6.3. Delete counter configuration

The declared counter should be deleted from the configuration for Modbus counters on the EWIO$_2$.
The counters to be deleted must be declared in a special JSON structure:
{"counter": [<name_1>, <name_2>, ...>]}

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=counter_remove&
module=modbus_remove&data=<JSON structure>

Return: -

```
data={
    "counter": [
        "14"
    ]
}
```

Example of JSON structure of a list for counters to be deleted

3.6.4. Save counter configuration

The configuration of a Modbus counter is saved. The configuration must be in a special JSON format.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=modbus_counter&
module=<Counter Bus addr>&data=<JSON structure>

Return: -

3.6.5. Load counter configuration

The configuration of the declared Modbus counter is requested by the client.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=modbus_counter&
module=<Counter Bus addr>

Return: <JSON structure>

```
{
    "ID":2,
    "BusAdr":6,
    "BR":0,
    "Name":"mr-ai8",
    "Sort":2,
    "ZNummer":"",
    "Mandant":"",
    "AbnNr":"",
    "Gewerk":0,
    "Hersteller":"",
    "Konto":"",
    "Ort":"",
    "Kommentar":"",
    "btype":"Modbus",
    "ztype":"mc_mr-ai8",
    "meteringcode": {
        "Land": "DE",
        "Plz": 12345,
        "Betreiber": 123456,
        "MePuId": "EXAMPLE0123456789012"
    }
}
```

Example of JSON structure of a counter configuration

3.6.6. Load data point list

The list of data points of a Modbus counter is requested. The list is represented in a special JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=modbus_channels&module=<Counter Bus addr>

Return: *<JSON structure>*

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=modbus_channels&module=<Counter Bus addr>

Return: *<JSON structure>*


```
{
    "channel": [
        {
            "ID":11,
            "Zaehler_ID":2,
            "Freeze_ID":null,
            "Rec":0,
            "Obis_ID":"",
            "Faktor_U":1,
            "Faktor_I":1,
            "EAblesung":"2020-06-05 11:20:00",
            "Intervall":1,
            "ZwTyp":"Volts",
            "MEinheit":"1.000000 V",
            "App":"",
            "AppArgs":"",
            "Faktor":1
        },
        {
            "ID":12,
            "Zaehler_ID":2,
            "Freeze_ID":null,
            "Rec":0,
            "Obis_ID":"",
            "Faktor_U":1,
            "Faktor_I":1,
            "EAblesung":"2020-06-05 11:20:00",
            "Intervall":1,
            "ZwTyp":"Input Resistance 1-C2",
            "MEinheit":"",
            "App":"",
            "AppArgs":"",
            "Faktor":1
        }
    ]
}
```


Example of JSON format of data point list

3.6.7. Save data point configuration

The configuration of a data point of a Modbus counter is saved. The configuration must be in a special JSON format.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=modbus_channel&
module=<Data point ID>&data=<JSON structure>

Return: -

3.6.8. Load data point configuration

The configuration of a data point of a Modbus counter is requested. The configuration must be in a special JSON format.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=modbus_channel&
module=<Data point ID>

Return: <JSON structure>

```
{
    "channel": [
        {
            "ID":12,
            "Zaehler_ID":2,
            "Obis_ID":"",
            "Rec":0,
            "Faktor_U":1,
            "Faktor_I":1,
            "EAblesung":"2020-06-05 14:15:00",
            "Intervall":5,
            "ZwTyp":"Input Resistance 1-C2",
            "MEinheit":"",
            "App":"",
            "AppArgs":""
        }
    ]
}
```

Example of JSON format of data point configuration

3.6.9. Load Modbus parameters for the documentation

The configuration of a counter with all associated data points is output as a text file.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=output&
module=modbus_<Counter Bus addr>

Return: *<Text>*

```
Counter ID: 2
Query Number: 2
Counter Name: mr-ai8
Modbus Address: 6
Counter Number:
Client:
Customer:
Craft: 0
Manufacturer:
Account:
Location:
Counter Type: mc_mr-ao4
Comment:
Bus Type: Modbus
Meteringcode: DE12345612345EXAMPLE0123456789013

Data Points
       Channel ID: 11
       Freeze ID:
       OBIS ID:
       Record Number: 0
       Readout From: 2020-04-16 11:15:00
       Interval: 1
       Description: Volts
       Factor Voltage: 1
       Factor Current: 1
       Factor: 1
       Unit: 1.000000 V
       Application:
       Application Arguments:

       Channel ID: 12
       Freeze ID:
       OBIS ID:
       Record Number: 0
       Readout From: 2020-04-16 11:15:00
       Interval: 1
       Description: Input Resistance 1-C2
       Factor Voltage: 1
       Factor Current: 1
       Factor: 1
       Unit:
       Application:
       Application Arguments:
```

Example of JSON structure of counter configuration

3.7. SQL functions, general

3.7.1. Send a SQL instruction

A SQL instruction with any content is sent from the client to the server.
The type of response is identified by the parameter "type":

db_str = response immediately as text string
db_json = response immediately in JSON format

db_str_<number> = response in following function call as text string
db_json_<nummer> = response in following functional call in JSON format

<number> = any random number under which the result of the instruction is available

The parameter "module" is not used and can be empty or can just be omitted.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=db_str_1234&
module=&data=<SQL statement>

Return: <Data> (only if response immediate)

3.7.2. Obtain the result of a SQL instruction

The result of a SQL instruction sent with response in the following function call is available in a temporary file. After the data is retrieved, the file is automatically deleted.

If the data is not yet available at the time of retrieval, "WEBGATE – NO FILE" is returned. The call must then be repeated after a specific period of time.

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=db_str_1234&module=

Return: <Data>

3.8. SQL functions, special

3.8.1. Load measured values from the data base

The number of 24 measured values of a particular data point of a declared time range is requested.

The data point – underscore – time range is declared as a module. The data point is declared with its data point ID.

Time range: -1 = last dates

0 = first dates

1 = dates beginning at (1|<date>)

Method: GET
URL: <server-ip>/cgi-bin/getParamFromServer.cgi

Parameter:
username=Administrator&password=<tan-md5-hash>&type=db_data&
module=<Data point ID_Zeitbereich>

Return: <JSON structure>

```
{
"data":
{"channelData":
[
{"Kanal_ID":2,"Zeit":"2020-05-08
11:15:00","Werte":0.000000,"Flags":"S;A;P;G;N;I;T","Grund":""},
{"Kanal_ID":2,"Zeit":"2020-05-08
11:20:00","Werte":0.000000,"Flags":"S;A;P;G;N;I;T","Grund":""},
{"Kanal_ID":2,"Zeit":"2020-05-08
11:25:00","Werte":0.000000,"Flags":"S;A;P;G;N;I;T","Grund":""},
{"Kanal_ID":2,"Zeit":"2020-05-08
11:30:00","Werte":0.000000,"Flags":"S;A;P;G;N;I;T","Grund":""},


....


{"Kanal_ID":2,"Zeit":"2020-05-08
13:05:00","Werte":0.000000,"Flags":"S;A;P;G;N;I;T","Grund":""},
{"Kanal_ID":2,"Zeit":"2020-05-08
13:10:00","Werte":0.000000,"Flags":"S;A;P;G;N;I;T","Grund":""}
]
}
}
```

Example of JSON format of requested measured values

3.8.2. Save measured values to the data base

One or more measured values are saved in the data base on the EWIO$_2$. These values must be in a special JSON format.

Method: POST
URL: <server-ip>/cgi-bin/setParamToServer.cgi

POST parameter:
Host:<server-ip>

Content-Type:application/x-www-form-urlencoded
Content-Length:<length>

Parameter:
username=Administrator&password=<tan-md5-hash>&type=db_send&
module=mbus&data=<JSON structure>

Return: -

```
{
"channelData":
[
{
"Kanal_ID":93,
"Zeit":"2020-06-05 13:45:59",
"Werte":1682,
"Flags":"S;A;P;G;N;I;T",
"Grund":""}
]
}
```

Example of JSON format of measured values to be saved

4. Data base structure

The data base structure of the EWIO$_2$ is generated with the following SQL statements (correspondences represent equivalent data base fields of the predecessor device EWIO-M):

```
CREATE TABLE `Counter` ( -- Basic properties of counter devices
        -- Unique ID of counter, corresponds to t_zaehler.ID
        `znr` INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Secondary address of M-bus counter, unused for other devices, corresponds to t_zaehler.BusAdr
        `address`    TEXT,
        -- Primary address of M-bus counter or Modbus address of Modbus counter, corresponds to t_zaehler.BusAdr
        `prim`       INTEGER,
        -- Name of counter (assigned by user), corresponds to t_zaehler.Name
        `name`       TEXT,
        -- Bus type of counter (can be MBUS, MODBUS or SYSTEM), corresponds to t_zaehler.BTyp_ID and t_btype.Kinds
        `bus` TEXT,
        -- Type of counter (selected by user, links to ID of row of MBusCounterTypes table for M-bus counter, name of Modbus-template
           for Modbus counter), corresponds to t_zaehler.ZTyp_ID and t_ztype.Kinds
        `devicetype` TEXT,
        -- Baud rate of M-bus counter, unused for other devices, corresponds to t_zaehler.BR
        `baud`       INTEGER,
        -- Rank of counter for measurement value retrieval, measurement values from counter with lowest rank are retrieved first,
           corresponds to t_zaehler.Sort
        `rank`       INTEGER
);
CREATE TABLE `Counter_extension` ( -- Extended informational properties of counter devices
        -- Unique ID of counter, links to related row of Counter table
        `znr` INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Craft property of counter (assigned by user)
        `craft`      INTEGER,
        -- Counter number property of counter (assigned by user), corresponds to t_zaehler.ZNummer
        `counternumber`    TEXT,
        -- Manufacturer property of counter (assigned by user)
        `manufacturer`     TEXT,
        -- Location property of counter (assigned by user)
        `location`    TEXT,
        -- Account property of counter (assigned by user)
        `account`    TEXT,
        -- Client property of counter (assigned by user), corresponds to t_zaehler.Mandant
        `client`     TEXT,
        -- Customer property of counter (assigned by user), corresponds to t_zaehler.AbnNr
        `customer`   TEXT,
        -- Metering-code country property of counter (assigned by user), corresponds to t_meteringcode.Land
        `mc_country`       TEXT,
        -- Metering-code operator property of counter (assigned by user), corresponds to t_meteringcode.Betreiber
        `mc_operator`      TEXT,
        -- Metering-code postcode property of counter (assigned by user), corresponds to t_meteringcode.Plz
        `mc_postcode`      TEXT,
        -- Metering-code counter number property of counter (assigned by user), corresponds to t_meteringcode.MePuId
        `mc_counternumber`       TEXT,
        -- Comment property of counter (assigned by user)
```

```
        `comment`  TEXT
);
CREATE TABLE `Datapoints` ( -- Basic properties of data points
        -- Unique ID of data point, corresponds to t_kanal.ID
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Unique ID of counter containing data point, links to related row of Counter table, corresponds to t_kanal.Zaehler_ID
        `znr`  INTEGER,
        -- Index of data point within its counter, enumerates data points of a counter starting at 1
        `dpnr`          INTEGER,
        -- Index of corresponding freeze data point within its counter, or 0 if data point does not have a freeze data point
        `freeze_id`    INTEGER,
        -- Modbus register address of Modbus counter data point, I/O line for system counter data point, unused for data points not
            belonging to a Modbus or system counter, corresponds to t_kanal.Rec only for Modbus data points
        `register`      INTEGER,
        -- Name of data point, corresponds to t_kanal.ZwTyp
        `name`          TEXT,
        -- Factor value of data point, applied before database insertion of measurement values, corresponds to t_kanal.Faktor_edit
        `factor`        REAL,
        -- Unit of data point, corresponds to t_kanal.Einheit_edit
        `unit` TEXT,
        -- Measurement value query frequency of data point, corresponds to t_kanal.Intervall
        `range`         INTEGER,
        -- Time of last data point readout or configuration change
        `last` INTEGER
);
CREATE TABLE `Datapoints_extension` ( -- Extended properties of data points
        -- Unique ID of data point, links to related row of Datapoints table
        `id`    INTEGER PRIMARY KEY UNIQUE,
        -- Time of first data point readout, corresponds to t_kanal.EAblesung
        `readout`    TEXT,
        -- Type of measurement processing for data point (0: normal, 1: average), corresponds to t_kanal.MessTyp
        `measuring` INTEGER,
        -- Primarity property of data point, corresponds to t_kanal.Primaer
        `prim`          INTEGER,
        -- Voltage factor of data point, applied before database insertion of measurements for data point related to a voltage,
            corresponds to t_kanal.Faktor_U
        `u`     REAL,
        -- Current factor of data point, applied before database insertion of measurements for data point related to a current,
            corresponds to t_kanal.Faktor_I
        `i`     REAL,
        -- Data type of Modbus data point, unused for data points not belonging to a Modbus counter
        `datatype`    TEXT,
        -- Temperature sensor type of data point, links to ID of row of TempSensorTypes table, corresponds to t_temp.SensTyp
        `sens_type`  INTEGER,
        -- Offset to be applied after temperature sensor data transformation of measurement value, corresponds to t_temp.Offset
        `sens_offset`           REAL,
        -- OBIS-ID of data point (assigned by user), corresponds to t_kanal.Obis_ID
        `obis_id`       TEXT,
        -- Application script to be executed during measurement value retrieval, corresponds to t_kanal.App
        `app` TEXT,
        -- Arguments to be supplied to application script executed during measurement value retrieval
        `app_args`    TEXT
);
```

```
CREATE TABLE `MBusCounterTypes` ( -- Types of M-bus counters
        -- Unique ID of M-bus counter type
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Name of M-bus counter type
        `name`      TEXT
);
CREATE TABLE `TempSensorTypes` ( -- Types of temperature sensors
        -- Unique ID of temperature sensor, corresponds to t_sensor.ID
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Name of temperature sensor type (shown to user in type selection list), corresponds to t_sensor.Name
        `name`      TEXT
);
CREATE TABLE `ModbusCounterModels` ( -- Types (models) of Modbus counters
        -- Unique ID of Modbus counter type
        `model_id` INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Default Modbus address of Modbus counter type
        `address` INTEGER,
        -- Name of Modbus counter type
        `modelname` TEXT UNIQUE,
        -- Template of Modbus counter data points
        `template` TEXT);
CREATE TABLE `ModbusCounterModelsDatapoints` ( -- Default data points of Modbus counter types devices
        -- Unique ID of default Modbus counter data point
        `db_id`     INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Unique ID of Modbus counter type, links to related row of ModbusCounterModels table
        `model_id`  INTEGER,
        -- Index of default datapoint within a Modbus counter type, enumerates data points of a counter type starting at 1
        `dpnr`      INTEGER,
        -- Modbus register address of default data point
        `register`  INTEGER,
        -- Name of default data point
        `name`      TEXT,
        -- Unit of default data point
        `unit` TEXT,
        -- Data type of default data point
        `format`    TEXT
);
CREATE TABLE `Measurements` ( -- Measurement values
        -- Unique ID of measurement value, corresponds to t_daten.ID
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        -- Unique ID of data point that provided the measurement value, links to related row of Datapoints table, corresponds to
           t_daten.Kanal_ID
        `dpid`      INTEGER,
        -- Sample time of measurement value, corresponds to t_daten.Zeit
        `sampletime`        TEXT DEFAULT CURRENT_TIMESTAMP,
        -- Value (result of all transformations applied during the readout process) of measurement value, corresponds to t_daten.Werte
        `value`     REAL DEFAULT 0,
        -- Flags of measurement value, corresponds to t_flags.Flags
        `flags`     TEXT,
        -- Additional description of measurement value
        `reason`    TEXT
);
```