PowerAPI – Local Device API Description

Revision History

Name	Date	Reason For Changes	Version
Stefan Voit	13.02.2013	Create	0.a
Stefan Voit	15.02.2013	Added set Calls	0.b

Confidential Statement:

The information in this document is confidential to the person to whom it is addressed and should not be disclosed to any other person. It may not be reproduced in whole, or in part, nor may any of the information contained therein be disclosed without the prior consent of the directors of ecodata GmbH (‘the Company’). A recipient may not solicit, directly or indirectly (whether through an agent or otherwise) the participation of another institution or person without the prior approval of the directors of the Company.

The contents of this document have not been independently verified and they do not purport to be comprehensive, or to contain all the information that a prospective investor may need. No representation, warranty or undertaking, expressed or implied is or will be made or given and no responsibility or liability is or will be accepted by the Company or by any of its directors, employees or advisors in relation to the accuracy or completeness of this document or any other written or oral information made available in connection with the Company.

Any form of reproduction, dissemination, copying, disclosure, modification, distribution and or publication of this material is strictly prohibited.

References

```
PowerDog®: www.power-dog.eu
ecodata GmbH: www.eco-data.de
XMLRPC: http://de.wikipedia.org/wiki/XML-RPC
libmaia: https://github.com/wiedi/libmaia
```

Notes

This document can be found under the following location:

```
http://api.power-dog.eu/documentation
```

The PowerAPI Local Device API

The PowerAPI is a device side API that allows users to access their PowerDog Device Data live from the PowerDog device over the network. The API can be access using the standardized XMLRCP Protocol (http://de.wikipedia.org/wiki/XML-RPC) and is located under http://POWERDOG_DEVICE_IP:20000 (it is running on port 20000 on the PowerDog device).

This API allows integration of the PowerDog Device into 3rd Party Applications.

The API is available since Version 1.60, is turned on by default and the default password is the Unlock Key.

General

The API itself is password protected. The default password is the “Unlock Key” of the PowerDog. The password can be changed in the PowerDog Configuration. The API can be disabled in the PowerDog Configuration.

Calls

The PowerAPI currently implements the following calls:

```
VariantMap getPowerDogInfo(String password);
VariantMap getSensors(String password);
VariantMap getCounters(String password);
VariantMap getRegulations(String password);
VariantMap getLinearDevices(String password);
VariantMap getAllCurrentLinearValues(String password);
VariantMap getCurrentLinearValues(String password,String comma_seperated_list_of_keys);
VariantMap setLinearSensorDevice(String password,String key,String current_value);
VariantMap setLinearCounterDevice(String password,String key,String current_value,String
countup_meter_reading);
```

The “get” Calls allow you to read values and configuration for the devices. The “set” calls allow you to write information to instanced “PowerAPI Sensors” or “PowerAPI Counters”.

Calling a Serverside Function

In order to call a serverside function an xmlrcp client is required. In **php** a function call would look like:

```
$client=new XMLRPCClient('http://10.2.1.122:20000/');
$reply=$client->getPowerDogInfo($password);
if($reply["ErrorCode"]==0) {
      #do valid stuff
}
else
{
      #handle error
}
```

For **Qt/C++** an XMLRPC Client can be "MaiaXmlRpcClient" from libmaia and a call could look like:

```
MaiaXmlRpcClient *client=new MaiaXmlRpcClient(QUrl("http://api.power-dog.eu/index.php"),
this);
QVariantList args;
args << QVariant(apikey);
client->call("getPowerDogInfo",args,this,SLOT(getPowerDogsReply(QVariant&)), \
this,SLOT(getPowerDogsError(int,QString)));
```

Libmaia does the error handling by itself which means we do not have to check for valid in the reply.

The Reply

As a reply xmlrcp return an xml encoded and serialized message. This messages is unserialized by the client and converted back in its original structure (String, List, Array, int,…)

Raw

```
An example raw reply for an error is:
<?xml version="1.0" encoding="iso-8859-1"?>
<methodResponse>
<fault>
 <value>
  <struct>
   <member>
    <name>faultString</name>
    <value>
     <string>parse error. not well formed.&#10;&#10;error occurred at line 1, column 1,
             byte index 0
     </string>
    </value>
   </member>
   <member>
    <name>faultCode</name>
    <value>
     <int>-32700</int>
    </value>
   </member>
  </struct>
 </value>
</fault>
</methodResponse>
```

Please see the xmlrpc specs for more details on message serialization and unserialization. Also note that this is handled by the client automatically.

Logic

An actual reply, as for the php example above, for the call getPowerDogInfo using `print_r($reply)` looks like:

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [FullVersion] => 1.55 Codename: Honeypot
            [HardwareRevision] => PowerDog Rev. B
            [ProductionDate] => 2013-01-28 17:30:15
            [SerialNumber] => PD301-00054
            [Version] => 1.55
        )

)
```

A valid reply will always hold "ErrorCode=0" in the first level of the array to indicate there was no error in this request. If ErrorCode si not 0 the reply must be handled as an error. Depending on the call the content of the reply will be different. The exact structure is described in the call details.

Example Client

Please note that there is a PHP example for the PowerAPI calls under http://api.power-dog.eu/documentation.

Call: getPowerDogInfo

Function: `getPowerDogInfo(password)`
Purpose: get information on the PowerDog itself
Example: `getPowerDogInfo('abc123')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [FullVersion] => 1.55 Codename: Honeypot
            [HardwareRevision] => PowerDog Rev. B
            [ProductionDate] => 2013-01-28 17:30:15
            [SerialNumber] => PD301-00054
            [Version] => 1.55
        )

)
```

Call: getSensors

Function: `getSensors(password)`
Purpose: get all currently active Sensors of the PowerDog
Example: `getSensors('abc123')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [impulsesensor_1359727508] => Array
                (
                    [Current_Value] => 0
                    [Hardware] => Impulse
                    [Key] => impulsesensor_1359727508
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [LinearType] => sensor
                    [Max] => 50
                    [Name] => wind
                    [Setable] =>
                    [Type] => Speed_kmh
                    [Unit] => km/h
                    [Unit_1000] => Tkm/h
                    [Unit_1000000] => Mkm/h
                    [Unit_Time_Add] =>
                    [Valid] => 1
                )

            [onewire_1359724930] => Array
                (
                    [Current_Value] => 23.8125
                    [Hardware] => onewire
                    [Key] => onewire_1359724930
                    [Last_Read_Average] => 23.75
                    [Last_Usage] => 0
                    [LinearType] => sensor
                    [Max] => 50
                    [Name] => multitemp
                    [Setable] =>
                    [Type] => Temperature
                    [Unit] => °C
                    [Unit_1000] => T°C
                    [Unit_1000000] => M°C
                    [Unit_Time_Add] =>
                    [Valid] => 1
                )
)
```

Call: getCounters

Function: `getCounters(password)`
Purpose: get all currently active Counters of the PowerDog
Example: `getCounters('abc123')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [impulsecounter_1359724600] => Array
                (
                    [Current_Value] => 0
                    [Hardware] => Impulse
                    [Key] => impulsecounter_1359724600
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [LinearType] => counter
                    [Max] => 1000
                    [Name] => impuls2
                    [Setable] =>
                    [Type] => Energy
                    [Unit] => W
                    [Unit_1000] => kW
                    [Unit_1000000] => MW
                    [Unit_Time_Add] => h
                    [Valid] => 1
                )

            [powerapi_1360890336] => Array
                (
                    [Current_Value] => 0
                    [Hardware] => PowerAPICounter
                    [Key] => powerapi_1360890336
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [LinearType] => counter
                    [Max] => 500
                    [Name] => test
                    [Setable] => setLinearCounterDevice
                    [Type] => Energy
                    [Unit] => W
                    [Unit_1000] => kW
                    [Unit_1000000] => MW
                    [Unit_Time_Add] => h
                    [Valid] => 1
                )
)
```

Call: getRegulations

Function: `getRegulations(password)`
Purpose: get all currently active Counters of the PowerDog
Example: `getRegulations('abc123')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
             [regulation_1359854217] => Array
                (
                    [Current_Value] => 100
                    [Hardware] =>
                    [Key] => regulation_1359854217
                    [Last_Read_Average] => 100
                    [Last_Usage] => 0
                    [LinearType] => regulation
                    [Max] => 100
                    [Name] => relais
                    [Setable] =>
                    [Type] => Percent
                    [Unit] => %
                    [Unit_1000] => T%
                    [Unit_1000000] => M%
                    [Unit_Time_Add] =>
                    [Valid] => 1
                )

            [regulation_1359891714] => Array
                (
                    [Current_Value] => 0
                    [Hardware] =>
                    [Key] => regulation_1359891714
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [LinearType] => regulation
                    [Max] => 100
                    [Name] => switch
                    [Setable] =>
                    [Type] => Percent
                    [Unit] => %
                    [Unit_1000] => T%
                    [Unit_1000000] => M%
                    [Unit_Time_Add] =>
                    [Valid] => 1
                )
)
```

Call: getLinearDevices

Function: `getLinearDevices(password)`
Purpose: get all currently active Sensors, Counters and Regulations of the PowerDog. It is a combination of the getSensors, getCounters and getRegulations call.
Example: `getLinearDevices('abc123')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (

             [onewire_1360752649] => Array
                (
                    [Current_Value] => 956.94
                    [Hardware] => onewire
                    [Key] => onewire_1360752649
                    [Last_Read_Average] => 956.89
                    [Last_Usage] => 0
                    [LinearType] => sensor
                    [Max] => 1000
                    [Name] => owtest
                    [Setable] =>
                    [Type] => Pressure
                    [Unit] => hPa
                    [Unit_1000] => Bar
                    [Unit_1000000] => kBar
                    [Unit_Time_Add] =>
                    [Valid] => 1
             )

            [powerapi_1360890336] => Array
                (
                    [Current_Value] => 300.5
                    [Hardware] => PowerAPICounter
                    [Key] => powerapi_1360890336
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [LinearType] => counter
                    [Max] => 500
                    [Name] => test
                    [Setable] => setLinearCounterDevice
                    [Type] => Energy
                    [Unit] => W
                    [Unit_1000] => kW
                    [Unit_1000000] => MW
                    [Unit_Time_Add] => h
                    [Valid] => 1
                )
            ---- omitted ----
)
```

Call: getAllCurrentLinearValues

Function: `getAllCurrentLinearValues(password)`
Purpose: returns all current Sensor, Counter and Regulation Values of all attached Sensors, Counters and Regulations. This call can be used to receive an update of all values.
Example: `getAllCurrentLinearValues('abc123')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (


            [impulsesensor_1359727508] => Array
                (
                    [Current_Value] => 0
                    [Key] => impulsesensor_1359727508
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [Name] => wind
                    [Unit] => km/h
                    [Valid] => 1
                )

            [onewire_1359724930] => Array
                (
                    [Current_Value] => 24.25
                    [Key] => onewire_1359724930
                    [Last_Read_Average] => 24.25
                    [Last_Usage] => 0
                    [Name] => multitemp
                    [Unit] => °C
                    [Valid] => 1
                )



            [pv_global] => Array
                (
                    [Current_Value] => 0
                    [Key] => pv_global
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [Name] => prod
                    [Unit] => W
                    [Valid] => 1
                )
```

```
             [regulation_1359842025] => Array
                 (
                     [Current_Value] => 100
                     [Key] => regulation_1359842025
                     [Last_Read_Average] => 100
                     [Last_Usage] => 0
                     [Name] => gggg
                     [Unit] => %
                     [Valid] => 1
                 )

             [regulation_1359854217] => Array
                 (
                     [Current_Value] => 100
                     [Key] => regulation_1359854217
                     [Last_Read_Average] => 100
                     [Last_Usage] => 0
                     [Name] => relais
                     [Unit] => %
                     [Valid] => 1
                 )

             [regulation_1359891714] => Array
                 (
                     [Current_Value] => 0
                     [Key] => regulation_1359891714
                     [Last_Read_Average] => 0
                     [Last_Usage] => 0
                     [Name] => switch
                     [Unit] => %
                     [Valid] => 1
                 )

             [temeraturesensor_1360441491] => Array
                 (
                     [Current_Value] => 1315.26
                     [Key] => temeraturesensor_1360441491
                     [Last_Read_Average] => 1315.26
                     [Last_Usage] => 0
                     [Name] => radii
                     [Unit] => W/m2
                     [Valid] => 1
                 )

         )

)
```

Call: getCurrentLinearValues

Function: `getCurrentLinearValues(password, comma_seperated_list_of_keys)`
Purpose: Returns the values of the sensors/Counters/regulations, passed as a comma separated list of the keys. This function can be used to update values of certain sensors.
Example: `getCurrentLinearValues('abc123','onewire_1360752649,regulation_1360752480')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)
2. **comma_seperated_list_of_keys** - string – A List of keys in the format: “key1,key2,keyX”

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [onewire_1360752649] => Array
                (
                    [Current_Value] => 962.66
                    [Key] => onewire_1360752649
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [Name] => owtest
                    [Unit] => hPa
                    [Valid] => 1
                )

            [regulation_1360752480] => Array
                (
                    [Current_Value] => 100
                    [Key] => regulation_1360752480
                    [Last_Read_Average] => 0
                    [Last_Usage] => 0
                    [Name] => cycle
                    [Unit] => %
                    [Valid] => 1
                )

        )

)
```

Call: setLinearSensorDevice

Function: `setLinearSensorDevice(password, key, current_value)`
Purpose: Sets a Value to a “PowerAPI Sensor”. You can only call this function on Sensors that are [“Setable”]=setLinearSensorDevice
Example: `setLinearSensorDevice(‘abc123’,’powerapi_1360890336’,’22,33’)`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)
2. **key** - string – the key of the PowerAPI Sensor to set
3. **current_value** - string – the value to set as double string: “11.44”, “1,3398”, “1187,00”, “515”

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [Current_Value] => 300.5
            [Key] => powerapi_1360890336
            [Last_Read_Average] => 0
            [Last_Usage] => 0
            [Name] => test
            [Unit] => W
            [Valid] => 1
        )

)
```

Call: setLinearCounterDevice

Function: `setLinearCounterDevice(password, key, current_value, countup_meter_reading)`
Purpose: Sets a Value to a "PowerAPI Counter". You can only call this function on Counters that are ["Setable"]=setLinearCounterDevice. Please note that the meter reading value should be counting up
Example: `setLinearCounterDevice('abc123','powerapi_1360890337','22,33', '10002')`

Arguments:

1. **password** - string - the Password set for PowerAPI (Unlock key by default)
2. **key** - string – the key of the PowerAPI Sensor to set
3. **current_value** - string – the value to set as double string: "11.44", "1,3398", "1187,00", "515"
4. **countup_meter_reading** - string – the meter reading value as numeric string: "10000", "10001"

Reply (Example):

```
Array
(
    [ErrorCode] => 0
    [ErrorString] =>
    [Reply] => Array
        (
            [Current_Value] => 300.5
            [Key] => powerapi_1360890337
            [Last_Read_Average] => 0
            [Last_Usage] => 0
            [Name] => test
            [Unit] => W
            [Valid] => 1
        )

)
```