PowerDog Web API Description

Revision History

Name	Date	Reason For Changes	Version
Stefan Voit	11.01.2013	Create	0.a
Stefan Voit	17.01.2013	Changed KWH to WH; added uid to unexposed getPowerDogs call	0.b

Confidential Statement:

References

```
PowerDog®: www.power-dog.eu
PowerDog API: api.power-dog.eu
PowerDog API Generator: api.power-dog.eu/generator
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 API

The PowerDog API is a server side API that allows users to access their PowerDog Device Data directly from the server. The API can be access using the standardized XMLRCP Protocol (http://de.wikipedia.org/wiki/XML-RPC) and is located under http://api.power-dog.eu/ (more precise: http://api.power-dog.eu:80/index.php).

General

In order to request data from the API it is required to hold a valid "api key". This key is a unique identifier for a certain user. They key can be acquired by calling getApiKey with the users email address and the md5 hashed password. Once this key is acquired it will be valid until the user changes its email address. All further calls hold use the api key instead of email/password to authorize calls. This means that once the api key is acquired the login information are not required anymore in external applications, it is enough and recommended to store only the api key.

Calls

The PowerDog API currently implements the following serverside calls:

- General
    1. `getApiKey(email,md5_pass)`
    2. `getPowerDogs(apikey)`
- Photovoltaics
    1. `getInverters(apikey, PowerDogID)`
    2. `getStringData(apikey,SensorID,StringNum,UTC_TIMESTAMP_FROM,UTC_TIMESTAMP_TO)`
    3. `getPhotovoltaicBorders(apikey,PowerDogID,month,year)`
    4. `getStringDayData(apikey,SensorID,StringNum,day_from,day_to,month,year)`
    5. `getStringMonthData(apikey,SensorID,StringNum,month_from,month_to,year)`
    6. `getStringYearData(apikey,SensorID,StringNum,year_from,year_to)`
- Sensors
    1. `getSensors(apikey,PowerDogID)`
    2. `getSensorData(apikey,SensorID,UTC_TIMESTAMP_FROM,UTC_TIMESTAMP_TO)`
- Counters
    1. `getCounters(apikey,PowerDogID)`
    2. `getCounterData(apikey,SensorID,UTC_TIMESTAMP_FROM,UTC_TIMESTAMP_TO)`

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://api.power-dog.eu/');
$reply=$client->getPowerDogs($apikey);
if($reply["valid"]) {
      #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("getPowerDogs",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 getPowerDogs using `print_r($reply)` looks like:

```
Array
(
    [valid] => 1
    [powerdogs] => Array
        (
            [0] => Array
                (
                    [timezone] => Europe/Vienna
                    [id] => 11
                    [name] => Familie Hütter
                    [description] => Anlage Familie Hütter
                    [address_zip] => 5280
                    [address_city] => Braunau
                    [address_country] => AUT
                    [address_street] => Mozartstrasse 33
                    [address_no] => 1
                    [longitude] => 13.04096222
                    [latitude] => 48.24741364
                    [published] => 1
                    [owner] => 41
                )

            [1] => Array
                (
                    [timezone] => Europe/Vienna
                    [id] => 61
                    [name] => Bauhof Salzhalle
                    [description] => diese Anlage ist keinem User zugeordnet
                    [address_zip] => 94060
                    [address_city] => Pocking
                    [address_country] => GER
                    [address_street] => Schmiedweg
                    [address_no] => 12
                    [longitude] => 13.30531597
                    [latitude] => 48.40455246
                    [published] => 1
                    [owner] => 41
                )

        )
)
```

A valid reply will always hold “valid=1” in the first level of the array to indicate there was no error in this request. If valid id not 1 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.

Call Procedure

The Order of the calls usually follows a certain procedure since the reply data links to data in other calls.

If the api is called for the first time the external application is usually not in possession of an api key. Which means before they can call anything else they have to call `getApiKey(email,md5_pass)`to aquire the users key from the API. Once the external application is in possession of this key it can call any other server side function with this key.

Most serverside functions require information like PowerDogID, StringNum or SensorID, wich means the external application has to acquire this data before it can access this data.

For each final set of data there is a call procedure that has to be followed to receive the right data from the API. Examples are:

- Getting all PowerDog IDs with all data from those PowerDogs for the certain user:
    1. Call `getApiKey(email,md5_pass)`
    2. Call `getPowerDogs(apikey)`
- Getting a list of Inverters for a certain PowerDog with ID "**PDI**"
    1. Call `getApiKey(email,md5_pass)`
    2. Call `getPowerDogs(apikey)`
    3. Call `getInverters(apikey, PDI)`
- Getting day data for inverter string **2** with StringNum "**SNUM**" from PowerDog with ID "**PDI**"
    1. Call `getApiKey(email,md5_pass)`
    2. Call `getPowerDogs(apikey)`
    3. Call `getInverters(apikey, PDI)`
    4. Call `getStringData(apikey,SNUM,2,UTC_TIMESTAMP_FROM,UTC_TIMESTAMP_TO)`

Those examples only represent some of the procedures to acquire information. Basically it is all build up as:

```
User(Api Key) → PowerDog (PID) → Inverter/Sensor/Counter (SID) → Data
```

Call: getApiKey

Function: `getApiKey(email,md5_pass)`
Purpose: get “api key” for a certain user
Example: `getApiKey('abc@gmx.at','900150983cd24fb0d6963f7d28e17f72')`

Arguments:

1. **email** - string - Email Address of the customer (lowercase)
2. **md5_pass** - string- md5 hashed password (lowercase)

Reply (Example):

```
Array
(
    [valid] => 1
    [apikey] => 90f47b75edc159ba8333a16ef37bd431
)
```

Call: getPowerDogs

Function: `getPowerDogs(apikey)`
Purpose: get PowerDogs and PowerDog information for the user of the api key
Example: `getPowerDogs('90f47b75edc159ba8333a16ef37bd431')`

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)

Reply (Example):

```
Array
(
    [valid] => 1
    [powerdogs] => Array
        (
            [0] => Array
                (
                    [timezone] => Europe/Vienna
                    [id] => 11
                    [name] => Familie Hütter
                    [description] => Anlage Familie Hütter
                    [address_zip] => 5280
                    [address_city] => Braunau
                    [address_country] => AUT
                    [address_street] => Mozartstrasse 33
                    [address_no] => 1
                    [longitude] => 13.04096222
                    [latitude] => 48.24741364
                    [published] => 1
                    [owner] => 41
                    [uid] => PD201-00002
                )

            [1] => Array
                (
                    [timezone] => Europe/Vienna
                    [id] => 61
                    [name] => Bauhof Salzhalle
                    [description] => diese Anlage ist keinem User zugeordnet
                    [address_zip] => 94060
                    [address_city] => Pocking
                    [address_country] => GER
                    [address_street] => Schmiedweg
                    [address_no] => 12
                    [longitude] => 13.30531597
                    [latitude] => 48.40455246
                    [published] => 1
                    [owner] => 41
                    [uid] => PD201-00003
                )
        )
)
```

Call: getInverters

Function: `getInverters(apikey, PowerDogID)`
Purpose: get inverters including inverter information for a certain PowerDog
Example: `getInverters ('90f47b75edc159ba8333a16ef37bd431',11)`

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **PowerDogID** – long – PowerDog ID from getPowerDogs (`[id] => x`)

Reply (Example):

```
Array
(
    [valid] => 1
    [inverters] => Array
        (
            [B1_A2] => Array
                (
                    [BUS] => 1
                    [ADDRESS] => 2
                    [Manufactor] => sma
                    [Modulfield] => 1
                    [Modulfield_Name] => Haus
                    [Monitoring] => on
                    [Capacity] => 2500
                    [SerialNo] => 2000049568
                    [Type] => WR25
                    [Strings] => 1
                    [desc] => WR 1
                    [StringList] => Array
                        (
                            [0] => Array
                                (
                                    [STRING] => 1
                                    [Capacity] => 2500
                                )

                        )

                    [id] => 26161
                )

        )
    [id] => 11
)
```

Call: getStringData

Function: `getStringData(apikey,SensorID,StringNum,UTC_TIMESTAMP_FROM,UTC_TIMESTAMP_TO)`
Purpose: get Day Data (5 min values) of a certain inverter string (certain mmp tracker)
Example: `getStringData('90f47b75edc159ba8333a16ef37bd431',26161,1, 1357020000, 1357106399)`
(get data from 1.1.2013, 0:0:0-23:59:59)

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **SensorID** - long – SensorID (Inverter ID), unique, from `[id] => 26161 in getInverters`
3. **StringNum** - int – String ID (usually 1-3) ,MMP Tracker ID, ids are from `StringList in getInverters`
4. **UTC_TIMESTAMP_FROM** - long – timestamp from in UTC
5. **UTC_TIMESTAMP_TO** - long – timestamp to in UTC

Reply (Example):

```
Array
(
    [sensor_id] => 26161
    [string_num] => 1
    [valid] => 1
    [datasets] => Array
        (
            [1357024640] => Array
                (
                    [PAC] => 6
                    [PDC] => 25
                    [UDC] => 227
                    [TEMPERATURE] => 0
                    [TIMESTAMP_LOCAL] => 1357028240
                    [TIMESTAMP_UTC] => 1357024640
                )

            [1357024939] => Array
                (
                    [PAC] => 16
                    [PDC] => 37
                    [UDC] => 227
                    [TEMPERATURE] => 0
                    [TIMESTAMP_LOCAL] => 1357028539
                    [TIMESTAMP_UTC] => 1357024939
                )

            [1357025236] => Array
                (
                    [PAC] => 28
                    [PDC] => 48
                    [UDC] => 227
                    [TEMPERATURE] => 0
                    [TIMESTAMP_LOCAL] => 1357028836
                    [TIMESTAMP_UTC] => 1357025236
                )
            -- omitted --
      )
)
```

Call: getPhotovoltaicBorders

Function: `getPhotovoltaicBorders(apikey,PowerDogID,month,year)`
Purpose: get start and end time in hours (24h) for a certain inverter string. This function is used to get the start and end time of the day chart. Make sure to interate over all used inverters and find the smallest and biggest number.
Example: `getPhotovoltaicBorders('90f47b75edc159ba8333a16ef37bd431',11,1,2013)`
(get start and end time for January 2013)

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **PowerDogID** – long – PowerDog ID from getPowerDogs (`[id] => x`)
3. **month** - int – Month in integer
4. **year** - int – Year in integer

Reply (Example):

```
Array
(
    [min] => 8
    [max] => 17
)
```

Call: getStringDayData

Function: `getStringDayData(apikey,SensorID,StringNum,day_from,day_to,month,year)`
Purpose: get Day Data (usage vales per day) of a certain inverter string
Example: `getStringDayData('90f47b75edc159ba8333a16ef37bd431',26161,1,1,31,1,2013)`
(get day datasets for January 2013)

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **SensorID** - long – SensorID (Inverter ID), unique, from `[id] => 26161 in getInverters`
3. **StringNum** - int – String ID (usually 1-3) ,MMP Tracker ID, ids are from `StringList in getInverters`
4. **day_from** - int – Start Day in integer (usually 1)
5. **day_to** - int – End Day in integer (usually 31)
6. **month** - int – Month in integer
7. **year** - int – Year in integer

Reply (Example):

```
Array
(
    [sensor_id] => 26161
    [string_num] => 1
    [valid] => 1
    [datasets] => Array
        (
            [2013-01-01] => Array
                (
                    [WH] => 2367
                    [PAC_MAX] => 914
                    [DAY] => 01
                    [MONTH] => 01
                    [YEAR] => 2013
                )

            [2013-01-02] => Array
                (
                    [WH] => 2727
                    [PAC_MAX] => 1389
                    [DAY] => 02
                    [MONTH] => 01
                    [YEAR] => 2013
                )

            [2013-01-03] => Array
                (
                    [WH] => 863
                    [PAC_MAX] => 377
                    [DAY] => 03
                    [MONTH] => 01
                    [YEAR] => 2013
                )

             -- omitted --
        )

)
```

Call: getStringMonthData

Function: `getStringMonthData(apikey,SensorID,StringNum,month_from,month_to,year)`
Purpose: get Month Data (usage vales per Month) of a certain inverter string.
Example: `getStringMonthData('90f47b75edc159ba8333a16ef37bd431',26161,1,1,12,2012)`
(get month datasets for 2012)

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **SensorID** - long – SensorID (Inverter ID), unique, from `[id] => 26161 in getInverters`
3. **StringNum** - int – String ID (usually 1-3) ,MMP Tracker ID, ids are from `StringList in getInverters`
4. **month_from** - int – Start Month in integer (usually 1)
5. **month_to** - int – End Month in integer (usually 12)
6. **year** - int – Year in integer

Reply (Example):

```
Array
(
    [sensor_id] => 26161
    [string_num] => 1
    [valid] => 1
    [datasets] => Array
        (
            [2012-02] => Array
                (
                    [WH] => 59107
                    [MONTH] => 02
                    [YEAR] => 2012
                )

            [2012-03] => Array
                (
                    [WH] => 274458
                    [MONTH] => 03
                    [YEAR] => 2012
                )

            [2012-04] => Array
                (
                    [WH] => 315231
                    [MONTH] => 04
                    [YEAR] => 2012
                )
             -- omitted --

        )

)
```

Call: getStringYearData

Function: `getStringYearData(apikey,SensorID,StringNum,year_from,year_to)`
Purpose: get Day Year (usage vales per year) of a certain inverter string
Example: `getStringYearData('90f47b75edc159ba8333a16ef37bd431',26161,1,2010,2014)`
(get year datasets for certain years)

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **SensorID** - long – SensorID (Inverter ID), unique, from `[id] => 26161 in getInverters`
3. **StringNum** - int – String ID (usually 1-3) ,MMP Tracker ID, ids are from `StringList in getInverters`
4. **year_from** - int – Start Year in integer (usually 2010)
5. **year_to** - int – End Year in integer (usually current year)

Reply (Example):

```
Array
(
    [sensor_id] => 26161
    [string_num] => 1
    [valid] => 1
    [datasets] => Array
        (
            [0] => Array
                (
                    [WH] => 2.75126e+06
                    [YEAR] => 2012
                )

            [1] => Array
                (
                    [WH] => 9982
                    [YEAR] => 2013
                )

        )

)
```

Call: getSensors

Function: `getSensors(apikey,PowerDogID)`
Purpose: get all sensors and sensor information of a certain PowerDog
Example: `getSensors('90f47b75edc159ba8333a16ef37bd431',11)`

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **PowerDogID** – long – PowerDog ID from getPowerDogs (`[id] => x`)

Reply (Example):

```
Array
(
    [valid] => 1
    [sensors] => Array
        (
            [onewire_1327954814] => Array
                (
                    [KEY] => onewire_1327954814
                    [Type] => Temperature
                    [Max] => 70
                    [Name] => Hzg.Vorlauf
                    [id] => 26221
                )

            -- omitted –

            [temeraturesensor_1332286336] => Array
                (
                    [KEY] => temeraturesensor_1332286336
                    [Type] => Global_Radiation
                    [Max] => 1200
                    [Name] => Solarstrahlung
                    [id] => 26291
                )

            [temeraturesensor_1354373373] => Array
                (
                    [KEY] => temeraturesensor_1354373373
                    [Type] => Temperature
                    [Max] => 30
                    [Name] => temp keller
                    [id] => 29531
                )

        )

    [id] => 11
)
```

Call: getCounters

Function: `getCounters(apikey,PowerDogID)`
Purpose: get all counters and counter information of a certain PowerDog
Example: `getCounters('90f47b75edc159ba8333a16ef37bd431',11)`

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **PowerDogID** – long – PowerDog ID from getPowerDogs (`[id] => x`)

Reply (Example):

```
Array
(
    [valid] => 1
    [counters] => Array
        (
            [impulsecounter_1327954495] => Array
                (
                    [KEY] => impulsecounter_1327954495
                    [Type] => Energy
                    [Max] => 3000
                    [Name] => Verbrauch
                    [Hardware] => Impulse
                    [id] => 26171
                )

            [pv_global] => Array
                (
                    [KEY] => pv_global
                    [Type] => Energy
                    [Max] => 2500
                    [Name] => PV
                    [Hardware] => pv_global
                    [id] => 26181
                )

             -- omitted --

            [iec1107_1355005323] => Array
                (
                    [KEY] => iec1107_1355005323
                    [Type] => Energy
                    [Max] => 2500
                    [Name] => Liefern
                    [Hardware] => iec1107
                    [id] => 30521
                )
        )

    [id] => 11
)
```

Call: getSensorData/getCounterData

Function: `getSensorData(apikey,SensorID,UTC_TIMESTAMP_FROM,UTC_TIMESTAMP_TO)`
Purpose: get Sensor/Counter day data (5 min data) within a certain time
Example: `getSensorData('90f47b75edc159ba8333a16ef37bd431', 26181, 1357020000, 1357106399)`
(get data from 1.1.2013, 00:00:00-23:59:59)

Arguments:

1. **apikey** - string - api key from getApiKey (lowercase)
2. **SensorID** - long – SensorID (Sensor/Counter ID), unique, from `[id] => 26181` in `getSensor/getCounter`
3. **UTC_TIMESTAMP_FROM** - long – timestamp from in UTC
4. **UTC_TIMESTAMP_TO** - long – timestamp to in UTC

Reply (Example):

```
Array
(
    [sensor_id] => 26181
    [valid] => 1
    [datasets] => Array
        (
            [1357024640] => Array
                (
                    [DATA] => 6
                    [TIMESTAMP_UTC] => 1357024640
                    [TIMESTAMP_LOCAL] => 1357028240
                )

            [1357024939] => Array
                (
                    [DATA] => 16
                    [TIMESTAMP_UTC] => 1357024939
                    [TIMESTAMP_LOCAL] => 1357028539
                )

            [1357025236] => Array
                (
                    [DATA] => 28
                    [TIMESTAMP_UTC] => 1357025236
                    [TIMESTAMP_LOCAL] => 1357028836
                )

             -- omitted --

        )
)
```

Open Source Clients

There are two open source clients provided by ecodata GmbH that implement a big part of the API. Both of the clients are provided under the LGPL which means they can be used commercial. Please see the open source client documentation for further details.

API Generator

ecodata GmbH provides an api url and html snipped generator that can be used to build up specific HTML Web Page integrations. The generator can be access under: http://api.power-dog.eu/generator/. Log in with the portal email address and the portal password to get access.

The Purpose of the generator is to provide and easy to use interface for generating HTML Code that can be integrated as widgets in foreign webpages: