eGauge XML API

(v1.51)

eGauge Systems LLC

September 15, 2023

1 Deprecation notice and JSON WebAPI

The eGauge XML API has been superseded by the newer and more powerful JSON-based WebAPI. The JSON WebAPI is faster, more flexible, and has configuration and status capabilities which the XML API does not.

The XML API will continue to function but not receive new features or enhancements. It is encouraged to use the JSON-based WebAPI rather than the XML interface.

More information about the JSON WebAPI can be found at `http://egauge.net/kb/webapi`.

2 Overview

This document describes how to read raw XML data from an eGauge device using CGI queries. This document applies to firmware versions v1.00 or newer. A firmware upgrade may be necessary to obtain support for newer query parameters or push options.

There are two types of queries: instantaneous and stored data queries. The former reads the most recent values of all measured data, whereas the latter reads (portions of) the historical data stored in a database built into the eGauge device.

Numeric data is returned either as integer strings or floating-point strings. The underlying format for integer strings is either unsigned 32-bit integers (U32) or signed 64-bit integers (S64). The range of U32 extends from 0 to 4,294,967,295. The range of S64 extends from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The underlying format for floating-point strings is the IEEE-754 64-bit floating point format.

S64 values are circular: after reaching the maximum positive value, they wrap around to the smallest negative value (and vice versa). Also note that JavaScript cannot natively handle 64-bit values and care must be taken to avoid overflows.

Unix time-stamps use U32 format, while integer register values use S64 format. Any decimal values use the IEEE-754 64-bit floating point format.

For additional information and examples, please visit `http://egauge.net/support/kb/xmlapi`.

3 Instantaneous Data

Instantaneous data is updated once a second. It is fetched via the URI reference:

    /cgi-bin/egauge?*params*

The possible values for *params* are described in the next section. Multiply query-parameters can be specified by separating them with an ampersand (e.g., **inst&tot** to specify both **inst** and **tot**).

3.1 Query Parameters (instantaneous data)

Parameter:	Type:	Description:
tot	n/a	Requests that the totals and virtual registers calculated from the physical registers be included in the output.
noteam	n/a	Requests that only locally measured values be reported (legacy format). The values of any registers acquired from remote devices and non-power registers are omitted. **DEPRECATED**.
teamstat	n/a	Requests that the teaming status be reported. This includes register "excess" values.
v1	n/a	Requests that the output be in v1.00 format as opposed to the legacy v0.01 format. Firmware v1.2 and above will default to v1.00-output.
inst	n/a	Requests that, along with the normal register values, the instantaneous rate-of-change of each register also be reported.

3.2 Instantaneous Data (v1.00 format)

A sample output for the v1.00 format is shown in Figure 1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<data serial="0x78666e4d">
 <ts>1284607004</ts>
 <r t="P" n="Grid"><v>5196771697</v></r>
 <r t="P" n="Solar"><v>21308130148</v></r>
 <r t="P" n="Grg&amp;Bth (PHEV)"><v>17601054087</v></r>
</data>
```

Figure 1: Example of instantaneous data with **v1** query parameter.

The instantaneous data query returns a single element enclosed by **data** start and end tags. The **data** element may have a **serial** attribute which specifies the configuration serial number as a hexa-decimal string. This serial number is guaranteed to change whenever a change is made to the device configuration. Thus, the serial-number can be used to detect configuration-changes.

This configuration serial number has nothing to do with the device hardware serial number, and is not a unique identifier of the unit.

Within the **data** element, the following elements may appear:

Element Name:	Type:	Description:
ts	Integer (U32)	The device-local time at which the reported measurements were obtained. This is a UNIX timestamp (seconds since start of January 1st, 1970 UTC).
r	Struct	There is one **r** element per register configured in the device. Attribute **t** specifies the code identifying the register's type (see Section 3.2.1). Attribute **n** specifies the register name, which may contain HTML entities to encode special characters. Attribute **rt** may be set to the string **total** to indicate that the register is a total or virtual register whose value has been calculated from the physical registers. Two sub-elements may appear for each **r** element: **v** and **i**.
v	Integer (S64)	A cumulative register value expressed in a type-specific unit. Subtracting two consecutive readings and dividing by the number of seconds elapsed between the samples gives the average rate of change for the register.
i	Float	The average rate of change of the register value as measured for the most recent one-second interval.

3.2.1 Register Types

The table below specifies the code used to identify the register type, the physical quantity represented by the code, and the unit of measurement for the rate of change of the register values.

Code:	Physical Quantity:	Rate-of-change Unit:	Recording Unit:
Ee	Irradiance	W/m^2 (Watts per square meter)	W/m^2
F	Frequency	Hz (Hertz)	mHz
I	Current	A (Ampère)	mA
PQ	Reactive Power	var (Volt-Ampère reactive)	var
Pa	Pressure	Pa (Pascal)	Pa
P	Power	W (Watt)	W
Qv	Volumetric flow	mm^3/s	mm^3/s
Q	Mass-flow	g/s (gram per second)	g/s
R	Resistance	Ω (Ohm)	Ω
S	Apparent Power	VA (Volt-Ampère)	VA
THD	Total Harmonic Distortion	%	$m\%$
T	Temperature	C (Centigrade Celsius)	mC
V	Voltage	V (Volt)	mV
#	Numeric	(unit-less)	(unit-less)
$	Monetary	*currency unit*$/s$	$2^{-29} \cdot$ *currency unit*$/s$
a	Angle	° (degrees)	$m°$
h	Relative humidity	%	0.1%
v	Speed	m/s (meter per second)	mm/s
Qe	Charge	Ah (Ampère-hours)	mAh

New register-type codes may be added over time. Software processing the eGauge XML data should be written such that it degrades gracefully when encountering an unknown register-type code.

Note that the above units apply to the rate-of-change of a register. The *value* of a register is the time-integral over the rate-of-change, so the register's unit is the above unit multiplied by time in seconds. For example, for power, the rate-of-change unit is Watts, and therefore the register value is Watt-seconds (which is equivalent to Joules). Watt-seconds can be converted to kilo-Watt-hours (kWh) by dividing by 3,600,000.

The database records all data as signed 64-bit quantities with the granularity shown in column "**Recording Unit**". Correspondingly, the stored data (see Section 4) is reporting these integer values and appropriate scaling may have to be applied to obtain the more user-friendly **Rate-of-change Unit**.

3.3   Team Status

An eGauge device configured to read out remote devices is part of a team whose members include the device itself and all remote devices.

The status of such a team can be obtained by passing the **teamstat** query parameter. The returned status indicates the availability and status of the configured registers, some of which may be acquired from one or more remote devices. A sample output of the team status format is shown in Figure 2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<status>
 <lag unit="ms">227</lag>
 <reg>
  <name>Grid</name>
  <available>1</available>
  <last_update>1312472842</last_update>
  <excess>0</excess>
  <last_val>0</last_val>
 </reg>
   :
 <reg>
  <name>Solar</name>
  <available>0</available>
  <last_update>1312472842</last_update>
  <excess>0</excess>
  <last_val>0</last_val>
 </reg>
</status>
```

Figure 2: Example of team status data (**teamstat** query parameter).

The team status is returned in a single element enclosed by **status** start and end tags. Within the **status** element, the following elements may appear:

Element Name:	Type:	Description:
lag	Integer (U32)	The amount of time by which the reported instantaneous data is behind real time. This lag is normally reported in in units of milli-seconds, as indicated by a value of **ms** for the attribute **unit**. Ideally, this lag should be close to zero, but may be larger when fetching data from remote devices that are slow to reach or that are temporarily unavailable.
reg	Struct	There is one such element for each register defined for the device. These elements appear in the same order as the register (**r**) tags in the v1.00 instantaneous data format. The sub-elements **name**, **available**, **last_update**, **excess**, and **last_val** may appear inside this element.
available	Boolean	Indicates whether the (remote-)device supplying the data for this register is currently reachable. A value of **1** indicates the device is reachable, a value of **0** indicates that the device is unreachable.
last_update	Integer (U32)	UNIX timestamp of when the value for this register was updated last.
excess	Integer (S64)	A non-zero value for this element indicates that the device supplying the data for this register was unreachable some time ago and the amount by which the current register-value is off from the true value.
last_val	Integer (S64)	The last value recorded for this register.

4   Stored Data

Stored data is updated once a minute. It is fetched via the URI reference:

```
/cgi-bin/egauge-show?params
```

This query returns energy data as rows of columns. Each row reports data for a specific point in time. The row consists of a fixed number of columns, with one column per configured register. Various query parameters *params* can be specified to select which data to retrieve and what format to return it in.

4.1 Query Parameters (stored data)

Parameter:	Type:	Description:
a	n/a	Requests that the totals and virtual registers calculated from the physical register values be included as the first columns in each row. These values are calculated according to the Totaling and Virtual Register rules configured for the device.
E	n/a	Requests that values are output relative to epoch. I.e., the value at "Date & time when recording started" will be zero.
b	n/a	Requests the output be returned in the data backup format.
c	n/a	Requests the output be returned in CSV (comma-separated value) format.
e	n/a	Requests the output of one extra data point beyond the requested range. This is similar, but not identical, to passing a value of $N+1$ for parameter **n**. This reason the two are not identical is because the data-base granularity may be coarser than requested. For example, you may be requesting data at one minute granularity, but if the data-base has only 1-hour granularity available, passing $N+1$ for parameter **n** has no effect, whereas **e** will ensure that the hourly data-point just beyond the last requested data-point is also included in the output. This parameter has no effect when the **T** or **w** parameters are specified. This parameter was introduced with firmware v1.2.
m	n/a	Specifies that **n** and **s** parameters are specified in units of minutes.
h	n/a	Specifies that **n** and **s** parameters are specified in units of hours.
d	n/a	Specifies that **n** and **s** parameters are specified in units of days.
S	n/a	Specifies that **n** and **s** parameters are specified in units of seconds
C	n/a	Specifies that the returned data be delta-compressed. That is, after the first row of data, each subsequent row's columns are expressed as a difference relative to the previous row's column-values. When combined with the CSV (**c**) parameter, the first row is always omitted.
n	Integer (U32)	Specifies the maximum number of rows to be returned.
s	Integer (U32)	Specifies the number of rows to skip after outputting a row. For example, `h&s=23` would skip 23 hours worth of data after a row is output, and would be equivalent to `d`.
f	Integer (U32)	Specifies the timestamp of the first row to be returned.
t	Integer (U32)	Specifies the timestamp of the last row to be returned.
w	Integer (U32)	Requests that only data newer than the specified timestamp returned. If the timestamp lies in the future, the query will complete immediately returning an empty **data** element whose **wait_time** attribute indicates how many seconds have to elapse before data younger than the specified timestamp will be available.
T	Integer-list (U32)	Specifies a list of timestamps, ordered by decreasing value (younger to older) for which to return data rows.
Z	string	Specifies the time-zone to use when exporting CSV data. For the description of the format of the Z parameter and building a timezone string, please visit `http://egauge.net/support/kb/xmlapi` As of firmware v1.12, it is possible to omit the value for this parameter. In this case, the device converts time-stamps using the device-local time-zone (specified through setting "Time Zone" in the "Date & Time" dialog).

4.2 Returned XML Data

A sample output for this query using parameters `m&n=3` is shown in Figure 3.

The stored data query returns a single element enclosed by **group** start and end tags. Just like the **data** element of the instantaneous response, the **group** element may have a **serial** attribute indicating the configuration serial number.

This configuration serial number has nothing to do with the device hardware serial number, and is not a unique identifier of the unit.

```
<?xml version="1.0" encoding="UTF-8" ?>
<group serial="0x37cdd096">
<data columns="3" time_stamp="0x4c9197e4" time_delta="60" epoch="0x47395980">
 <cname t="P">Grid</cname>
 <cname t="P">Solar</cname>
 <cname t="P">Grg&amp;Bth (PHEV)</cname>
 <r><c>5203642184</c><c>21308125431</c><c>17598056700</c></r>
 <r><c>5203503484</c><c>21308125526</c><c>17598116405</c></r>
 <r><c>5203368999</c><c>21308125626</c><c>17598176060</c></r>
</data>
</group>
```

Figure 3: Example of stored data.

The most recent values are reported first in the data element, with following rows being **time_delta** seconds earlier than the previous row.

Within the **group** element, the following elements may appear:

Element Name:	Type:	Description:
data	Struct	One such element appears for each consecutive sequence of data rows. Attribute **columns** specifies the number of columns in each row. Attribute **time_stamp** specifies the UNIX timestamp (in hex) for the first row. Attribute **time_delta** specifies the number of seconds to be subtracted to get the next row's timestamp. Attribute **epoch** specifies the UNIX timestamp (in hex) of the time at which recording started. Attribute **delta** is equal to **true** if the data rows are delta-encoded (see below). Attribute **wait_time** specifies how many seconds have to elapse before the timestamp specified by the **w** parameter is available for reading.
cname	String	Specifies the register-name of a column in order of increasing column. This element may only appear in the first **data** element. In subsequent **data** elements, the register names must remain the same as for the first one. This element may have a **t** attribute which identifies the type of the register (see 3.2.1). If the **t** attribute is not present, a type-code of **P** (power) should be assumed.
r	Struct	Contains one row of data.
c	Integer (S64)	An individual cumulative register value. This value must be interpreted according to the register-type specified (or implied) by the corresponding **cname** declaration.

5 Push Data Setup

In addition to polling data from an eGauge device as described in the previous sections, it is also possible to setup eGauge to push data to an arbitrary URI. Both http (unencrypted) and https (encrypted) schemes are supported. The data is pushed in the same XML format as described in Section 4 and is sent as the body of an HTTP POST (not as a form). The Content-Type of this POST is text/xml.

The URI specifies the web-server to which the data should be pushed. Regardless of the scheme specified, this transmission uses HTTP/1.1 **chunked** transfer-encoding (i.e., there is *no* Content-Length header). Optionally, the data may be compressed with content-encoding **deflate** or **gzip** (see push-options below). Note that content-encoding (i.e., compression) is applied before the transfer-encoding. Normally, a web-server decodes the transfer-encoding, so the final recipient only has to handle content-encoding (if any).

There are two ways to setup an eGauge to push data to a web server: manual (custom) or via a push-service definition. The manual setup involves specifying all the communication parameters, including the URI, the time-interval between push updates, and the options with which the data should be pushed (e.g., specifying whether or not virtual registers should be pushed). The manual setup is cumbersome and error-prone for end-customers. Using a push-service definition is much more user-friendly and also allows a third-party service provider to automatically walk a customer through the signup or login procedure required on the server-side to accept data from the eGauge.

The push service is configured on the device under `Data Sharing` in `Settings→General Settings`. If an automatic push-service definition is used, the URI the eGauge pushes to and any options will be hidden.

The push-interval (Interval between push updates) may be shorter than the granularity of the data being pushed. A connection to the Push URI is established only if new data is available at the end of a push interval. This can be used to improve data transfer reliability across unreliable links. For example, if the push interval is 1 minute and the data-granularity 1 hour, the device will attempt to push new data once a minute until the data has been uploaded and then wait until the next hourly data-point becomes available before connecting to the Push URI again.

Unless specified otherwise, data is pushed with minute-by-minute granularity (to the degree available).

5.1 Automatic Push Data Setup via Push Service Definition

To setup a push-service definition, a third-party service provider needs to provide the following pieces of information to eGauge Systems LLC:

- The name of the service (arbitrary text string but should be no longer than 24 characters).

- The *control URI* (*cURI*) which will be used to signup the device with the third-party service.

Once this information has been received and processed by eGauge Systems LLC, a user will be able to sign up a device with the third-party service by opening the device's web-interface in a browser and clicking on `Settings→General Settings`. The third-party service can then be selected from the drop-down list under `Data Sharing`. When the user clicks on the `Save` button, the following actions take place if the selected push service has been changed:

1. The eGauge saves the name of the selected push service. The user may have to authenticate him- or her-self with proper credentials before this step can be completed (for example, username "owner" and password "default", assuming a direct/LAN connection to the device exists). The eGauge will mark the push service as being **inactive** until the following steps have been completed successfully.

2. The eGauge then redirects the user's browser to the service provider's cURI, passing various GET parameters in the request. Specifically, the browser is directed to URI:

   cURI`?activate&mfg=eGauge&did=`***name***`&sn=`***SN***`&regs=`***rlist***`&virt=`***vlist***`&ruri=`***rURI***`&opts=`***opts***

   where:

   cURI: The control URI specified by the third-party service provider.

   name: The hostname (device-id) of the device. Note that the hostname can be changed by the end-customer. The hostname is guaranteed to be unique only if the device is connected to a proxy-server and it is guaranteed to be unique only for that proxy-server.

   SN: The serial-number of the device, which is guaranteed to be unique and cannot be changed by the end-customer. However, the serial number could change, for example, if a device were to fail and is subsequently swapped out with a replacement device. This parameter is included only on newer devices (eGauge3 series or newer) and only when using firmware v2.1 or newer.

   rlist: Comma-separated list of the names of the registers that the device is recording. Each name is URI-encoded where blanks are replaced by the plus-sign (+) and any other non-alpha-numeric byte by a percentage-sign (%) followed by the hexa-decimal string encoding the byte's value.

   vlist: Comma-separated list of the names of virtual registers that have been defined for the device. Each name is URI-encoded.

   rURI: The return URI: this is the URI the browser is to be directed to upon successful completion of the signup-process with the third-party service. This value is also URI-encoded. Currently, this value is `http://DEV-URI/cgi-bin/protected/egauge-cfg?setpush` where `DEV-URI` is the URI used by the client configuring the push, which may be a local IP address, or a full proxy-server URI. This is subject to change and should not be assumed to be the same across devices.

   opts: Comma-separated list of all push options the device firmware supports.

3. Once the web browser opens the cURI, the third-party service can perform all the steps needed to prepare for receiving data from the device (e.g., create a new account for the user/device or associate an existing account with the device). Upon successful completion, the third-party service should then redirect the browser back to the rURI via an HTTP `POST` using encoding-type `multipart/form-data`. The POSTed form may pass values for any of the following part names:

   - **uri**: The URI that the eGauge should push data to. For example:

     `https://datacenter.example.com/push-data?01345a535a`

   - **interval**: The interval in seconds between updates.

- **options**: The options with which the data should be pushed. Multiple options can be specified by separating them with commas. Available options are:

  - **totals**: Also push the values of virtual (calculated) registers.
  - **sec**: Push second-by-second data.
  - **hour**: Push data with hour granularity (to the degree available).
  - **day**: Push data with day granularity (to the degree available).
  - **max=N**: Push at most *N* rows of data. *N* cannot be more than 900.
  - **skip=N**: After pushing a row of data, skip the the next *N* rows before pushing another one. For example, with minute granularity, "skip=14" would end up pushing one data row for every 15 minutes.
  - **secure**: When using the https scheme, certificate verification errors are usually ignored. When specifying this option, the web-server's certificate must verify correctly, otherwise, the data will not be pushed.
  - **deflate**: Compress pushed data using the deflate algorithm.
  - **gzip**: Compress pushed data using the gzip algorithm.
  - **old_first**: Push pending old data before newer data. Normally, the most recent N rows of data are pushed (as specified by the **max=N** option). However, if the **old_first** option is specified and there are more than *N* rows of data that have not been pushed yet, the old data will be sent first. If data is being generated faster than can be pushed, this option can have the effect that the pushed data falls further and further behind the current time.
  - **epoch**: Push values as they are relative to epoch (see egauge-show parameter "E").

    Note: The **deflate** and **gzip** options are supported starting with firmware version 2.03. Older firmware will ignore these options and send the data uncompressed.

- **user**: The user-name to be supplied during a push via HTTP basic authentication. If no user-name is specified, no basic authentication information will be supplied.

- **pw**: The password to be supplied during a push via HTTP basic authentication.

4. The eGauge extracts the parameters from the POSTed form and saves them. Assuming all parameters have valid values, the eGauge then starts pushing data to the third-party service.

5.1.1 Example of Return-URI Form Posting

The example below illustrates how a form can be POSTed to the return URI (rURI) using a web-page that employs JavaScript to extract the rURI and immediately post the form (using `form.submit()`).

```
<script type="text/javascript">
function init () {
    var ruri = "";

    /* Loop through GET parameters sent
    to the control URI (this page). */

    var m = location.search.substr (1).split ("&");
    for (var i = 0; i < m.length; ++i) {
        var nv = m[i].split ("=", 2);
        var name = nv[0];
        var value = nv.length > 1 ? nv[1] : null;

        /* Choose what we should do with
        parameters we acknowledge */
        switch (name) {
            case "ruri": ruri = decodeURIComponent (value); break;
            case "did": devname = decodeURIComponent (value); break;
            case "sn": serialnum = decodeURIComponent (value); break;
            // there's also "rlist", "vlist" and "opts"
        }
    }
    /* We've read the GET parameters, now do something with
    the values. Prepare your database for incoming data from
    the particular device, generate a unique push URI for the
    device, generate credentials, etc. */

    // Generate unique push URI based on device name and
    // set the form value of "uri", which will be the URI the eGauge to
    pushURI = "https://www.example.com/push-data?devname=" + devname;
    document.getElementById("uri").value = pushURI;

    // Finally, submit the form to the return URI
    var form = document.getElementById ("form");
    form.action = ruri;
    form.submit ();
}

window.onload = init;
</script>

<form id="form" method="POST" enctype="multipart/form-data">
  <!-- Note URI is set in JavaScript function -->
  <input type="hidden" name="uri" id="uri"
    value="">
  <!-- You can also set options, user, pw in JavaScript -->
  <input type="hidden" name="options" id="options" value="totals">
  <input type="hidden" name="user" id="user" value="push_username">
  <input type="hidden" name="pw" id="pw" value="push_password">
</form>
```

In a more realistic environment, the above web-page would be generated by server-side scripting (e.g., PHP). In such a case, the return URI could be hard-coded in the form and JavaScript would have to be used only to submit the form.

This example can be used to observe the sign-up process: save as a text file, and open in a browser, passing at least an **rURI** and **name** as GET parameters to it. For example:
```
http://cURI.html?name=eGauge10010&rURI=http://egauge10010.d.egauge.net/
cgi-bin/protected/egauge-cfg?setpush
```

5.2 Push Service Status Verification

Once a push service is enabled, the status can be checked at URI reference:

```
/push-status.html
```

This web page displays basic status information, such as the date and time of the last push attempt, the HTTP response code received during that attempt, the date and time of the last successful push, and the number of data rows delivered during that push.

The same information is also available in the form of an XML document at URI reference:

```
/cgi-bin/get?pushStatus
```

The following XML elements are returned within an **uploadStatus** element:

Element Name:	Type:	Description:
nextAttempt	Integer (U32)	Next time an attempt will be made to push data. This is a UNIX timestamp.
lastAttempt	Integer (U32)	Last time an attempt was made to push data. This is a UNIX timestamp.
lastSuccess	Integer (U32)	Last time data was pushed successfully. This is a UNIX timestamp.
lastUploadCount	Integer (U32)	Number of data rows sent during the last successful push.
lastResponseCode	Integer (U32)	HTTP response code received during the last push attempt.
lastTimestamp	Integer (U32)	Timestamp of most recent row of data that was pushed. This is a UNIX timestamp.

6 Miscellaneous tips and information

6.1 Identifying devices push data

To identify the device that is pushing data, a unique token may be encoded on the end of the push string. For example, if you have a data collector at `http://pushdata.example.com/collector/egauge`, you may set your eGauge to push to `https://pushdata.example.com/collector/egauge?devid=f6ys3sdrw56754c`. The data collector should read this GET parameter and determine the pushing device from the value.

NOTE: The **serial** attribute in stored and instantaneous data is guaranteed to change when configuration changes are made to the device. Thus, it cannot be used as a unique indicator of the device.