

Register Data Examples

This page contains a few basic examples of obtaining historical register data from an eGauge meter using the WebAPI.

These examples use the [eGauge Python library](#) but other languages and libraries may be used.

Get Instantaneous values

Request: `dev.get('/register?reg=3+5+7+9&rate')`

Parameters:

`reg=3+5+7+9` asks for those register IDs

`rate` asks for the instantaneous rate of change at the time

No `time` parameter was provided, so it gets the latest value

Response:

```
{
  "ts": "1679506470.000420096",
  "registers": [
    {"name": "Panel 3 Mains", "type": "P", "idx": 3, "did": 0, "rate": 2400},
    {"name": "Panel 3 L1", "type": "P", "idx": 5, "did": 2, "rate": 551},
    {"name": "Panel 3 L2", "type": "P", "idx": 7, "did": 3, "rate": 1080},
    {"name": "Panel 3 L3", "type": "P", "idx": 9, "did": 4, "rate": 769},
  ],
}
```

Get 6 hour period of historical data

Request: `dev.get('/register?reg=3+5+7+9&rate&time=1672556400:3600:1672578000')`

Parameters:

`reg=3+5+7+9` asks for those register IDs

`rate` asks for the instantaneous rate of change at the time

`time=1672556400:3600:1672578000` is:

- `1672556400`: Jan 1 12:00AM (start time)

- 3600: number of seconds to skip between rows (makes 1 hour granularity)
- 1672578000: Jan 2 6:00AM (end time)

Response:

```
{
  "ts": "1679505959.000420096",
  "registers": [
    {"name": "Panel 3 Mains", "type": "P", "idx": 3, "did": 0, "rate": 4185},
    {"name": "Panel 3 L1", "type": "P", "idx": 5, "did": 2, "rate": 506},
    {"name": "Panel 3 L2", "type": "P", "idx": 7, "did": 3, "rate": 2902},
    {"name": "Panel 3 L3", "type": "P", "idx": 9, "did": 4, "rate": 777},
  ],
  "ranges": [
    {
      "ts": "1672578000",
      "delta": 3600.0,
      "rows": [
        ["150555067851", "64158660544", "57705425071", "28690982234"],
        ["150552980706", "64157713471", "57704901028", "28690366205"],
        ["150550413006", "64156764160", "57703892567", "28689756277"],
        ["150548310776", "64155810457", "57703366875", "28689133443"],
        ["150546201980", "64154853051", "57702841741", "28688507186"],
        ["150543949276", "64153896433", "57702279596", "28687773245"],
        ["150541826993", "64152938219", "57701751003", "28687137769"],
      ],
    },
  ],
}
```

Notes

Parse output the same way that would be done for XML data.

Each row is *delta* seconds (3600) older than the previous row. The columns are in order of the registers, that is the first value in each row is for "Panel 3 Mains", the second value in each row is for "Panel 3 L1", and so on.

Row 1:

150555067851 is the cumulative value of "Panel 3 Mains" at time "ts" (1672578000 = Jan 1 6:00AM)

64158660544 is the cumulative value of "Panel 3 L1" at time "ts" (1672578000 = Jan 1 6:00AM)

57705425071 is the cumulative value of "Panel 3 L2" at time "ts" (1672578000 = Jan 1 6:00AM)

Row 2:

150552980706 is the cumulative value of "Panel 3 Mains" at time "ts"-"delta"*1 (1672578000-3600*1 = Jan 1 5:00AM)

Row 3:

150550413006 is the cumulative value of "Panel 3 Mains" at time "ts"-"delta"*2 (1672578000-3600*2 = Jan 1 4:00AM)

Row 4:

150548310776 is the cumulative value of "Panel 3 Mains" at time "ts"-"delta"*3 (1672578000-3600*3 = Jan 1 3:00AM)

Row 7:

28687137769 is the cumulative value of "Panel 3 L3" at time "ts"-"delta"*6 (1672578000-3600*6 = Jan 1 12:00AM)

For example, "Panel 3 Mains", between 12:00AM and 6:00AM on Jan 1 used (150555067851-150541826993)/3,600,000 == 3.679 kWh

Get 3 particular timestamps

Request: `dev.get('/register?reg=3:5&time=1672556400,1672642800,1672729200')`

Parameters:

`reg=3:5` asks for registers starting at ID 3 and ending at ID 5

`time=1672556400,1672642800,1672729200` requests the values at timestamps of:

- `1672556400`: Jan 1 12:00AM
- `1672642800`: Jan 2 12:00AM
- `1672729200`: Jan 3 12:00AM

Response:

```
{
  "ts": "1679507855.000420096",
  "registers": [
    {"name": "Panel 3 Mains", "type": "P", "idx": 3, "did": 0},
    {"name": "Panel 3 Mains*", "type": "S", "idx": 4, "did": 1},
    {"name": "Panel 3 L1", "type": "P", "idx": 5, "did": 2},
  ],
  "ranges": [
    {
```

```

    "ts": "1672556400",
    "delta": 60.0,
    "rows": [["150541826993", "182491473811", "64152938219"]],
  },
  {
    "ts": "1672642800",
    "delta": 60.0,
    "rows": [["150591166950", "182556552340", "64175776009"]],
  },
  {
    "ts": "1672729200",
    "delta": 60.0,
    "rows": [["150704245365", "182691087288", "64205578174"]],
  },
],
}

```

Notes

Parse the same as XML output. There are 3 ranges, one for each of the requested timestamps, and each register's cumulative value at that time.

For example:

The last range shows on January 1 ("ts" of 1672729200) that "Panel 3 L1*" (middle column) had a cumulative value of 64205578174 watt-seconds.

The second to last range shows on January 2 ("ts" of 1672642800) that "Panel 3 L1*" (middle column) had a cumulative value of 64175776009 watt-seconds.

This means between January 1 and January 2, the "Panel L1" register changed by (64175776009-64205578174) = -29802165 watt-seconds, which is $-29802165/3600000 = -8.27$ kWh

Depending on the orientation of the amperage sensor (CT), the power/energy will register as positive or negative. If there is unidirectional power (no back-feeding, etc.) it is usually safe to take the absolute value between dates. However, unexpected power polarity can also be caused by physical installation issues that are generating inaccurate data.

Please visit kb.egauge.net for the most up-to-date documentation.