USER MANUAL ezeio mkII

Go to **https://eze.io** to manage and configure

- ezeio MkII I/O Expander (TEMPORARILY UNAVAILABLE)
- IMPORTANT INFORMATION
- Introduction to the ezeio
- Registration
- Indicators
- Installing the ezeio
- Connecting Sensors and Peripherals
- User Interface
- Drivers
- Expressions
- Script reference
- API reference

Download the quick-start guide here

ezeio-mkii-quickstart_letter.pdf

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/start

Last update: **2024-08-05 17:31**

API reference

Version 1 API

API endpoint : https://api.eze.io/v1

Credentials

API credentials must be created for the account. The credentials consists of an ID number and a 24 character key. Please keep the API credentials secret.

All API calls require authentication using Digest Auth (RFC 2617). The API ID number shall be used as the Digest Auth Username, and the API key shall be used as the Password.

Alternatively, credentials can be supplied in “ezeAPIkey” and “Authorization” Bearer token headers respectively.

All API calls require HTTPS using TLS v1.2 or TLS v1.3. Calls via unencrypted HTTP are not allowed.

All API calls return data encoded in JSON. The response include the time of the request (reqtime) and a result code in (status). If the call was successful, the status will be “OK”. If there was an error, status will be a string detailing the error.

This section contains example code in PHP, however most development frameworks and languages can be used to interface to the ezeio API. There is no dependency on any special language or operating system.

Please note that using the API requires knowledge in programming and standard web technologies. eze System is happy to support with specifics related to the API, but we can't teach programming.

Available API functions

- control
- eventlog
- group
- samplelog
- scancode

- status
- subscribe
- syslog

API fair use

The API functionality is a shared resource, and as such requires 'good behavior' from those that use it. Access to the API features are logged and monitored by eze System, and if we detect inefficient or for any reason concerning usage we will contact the user and/or suspend the IP/access key until the problem is resolved.

The API also implements flood control counters to limit excessive calls as follows;

- Per 60 seconds : 100 API calls per key
- Per 24 hours : 50000 API calls per key

If these limits are exceeded, an error is returned. The counters are automatically reset at the end of the interval.

Example code

This example shows how to call the “syslog” API endpoint using PHP.

ezeioAPIexample.php

```php
<?php
    define("APIURI", "https://api.eze.io/v1/syslog");

    // API keyID and key needs to be set up in eze.io under Groups->API.
    define("APIKeyID", "00000");
    define("APIKey", "12345abcde12345abcde12345abcdeff");

    $params = array(
        "ABC000",               // ezeio serial
        "from=2021-11-01",      // start of data
        "to=2021-11-10",        // end of data
        "fields=1,12,cRSSI",    // list of fields we care about
        "1h",                   // 1h interval data..
        "mean"                  // ..as averages
    );

    // Set up cURL
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, APIURI."/".implode("/", $params));
```

```php
    // All API calls use Digest AUTH
    curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_DIGEST);
    curl_setopt($ch, CURLOPT_USERPWD, APIKeyID.":".APIKey);

    // Set a 5s timeout, return any received data, ignore ssl errors
    curl_setopt($ch, CURLOPT_TIMEOUT, 5);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

    // Execute the cURL request
    $response = curl_exec($ch);
    if(curl_errno($ch))         // if there's an error..
        die(curl_error($ch));   // quit and show what the problem is
    curl_close($ch);

    // Decode the json reply into an associative array
    $json = json_decode( $response, TRUE );

    // Show what we received
    print_r($json);
```

Below is a minimal example fetching current status using Python

ezeioAPIexample.py

```python
import requests
from requests.auth import HTTPDigestAuth
from pprint import pprint # Just used to format the output

# API credentials from eze.io -> Groups Settings -> Manage API Keys
apikeyid = '00000'
apikey = '12345abcde12345abcde12345abcdeff'

# API endpoint and request - see doc.eze.io
apiurl = 'https://api.eze.io/v1/status/ABC000'

# Send request
r = requests.get(apiurl, auth=HTTPDigestAuth(apikeyid, apikey))

# Convert response to a dictionary
data = r.json()

# Dump whole dictionary to screen
pprint(data)

# Print the value of field 1
```

```
print(data['fields']['1']['value'])
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/start

Last update: **2024-09-05 21:50**

control

Set a field value or output status

Description

```
https://api.eze.io/v1/control/{serial}/{command}/[additional commands]
```

Directly control a field value or output status of a controller. Note that this command will generate traffic to the controller, and is therefore rate limited to avoid excessive traffic. The field has to be set as “Writable”.

The immediate return data is a confirmation from the servers that the command was accepted. It is not an acknowledgement that the command(s) were actually received or executed on the controller.

If the API key is set up with a callback URI, a separate message will be sent to this URI when the command has been confirmed by the controller. See below for the format of the return messages.

Parameters

`serial`	ezeio serial number (XYZ123)
`command`	Command. See below

Possible commands:

To set the value of a field: `field[FIELDNO]=value`

```
https://api.eze.io/v1/control/XYZ987/field[12]=4711
```

To set the state of a register: `register[DEVICENO,REGISTERNO]=value`

```
https://api.eze.io/v1/control/XYZ987/register[2,34]=56
```

To set the state of an output, bypassing logic: `output[OUTPUTNO]=value`

```
https://api.eze.io/v1/control/XYZ987/output[2]=100
```

Multiple commands can be used in the same call.

Example usage

Set the value of two fields:

```
https://api.eze.io/v1/control/XYZ987/field[12]=888/field[5]=999
```

Log flush

Setting the value of a field will also cause any pending buffered log samples to be sent to the servers. This only applies to the control field command. Setting register or output value does not cause a log flush.

Tip: To force a log flush without making any changes to field values, write value 0 to field[0]. This has no effect on any valid field, but will still cause a log flush.

Return value

JSON formatted data

(below examples have whitespaces added for readability)

The immediate server reply (acknowledging receipt of the command on the servers):


```
{
  "reqtime":"2019-06-16T19:50:18Z",
  "actions":[
    "Set field 12 to 888",
    "Set field 5 to 999",
    "Callback URI: https://mydomain.com/mycallbackscript.php"
  ],
  "status":"OK",
  "exec_time":0.011375188827514648
}
```


The callback message from the controller, acknowledging processing of the command:


```
{
  "apikeyid":10009,
  "subject":"CONTROL_ACK",
  "time":"2019-06-16T19:45:52Z",
  "meta":{
    "serial":"XYS-987",
    "group":2,
    "accountgroupid":0,
    "name":"Test controller in lab",
    "note":"Here is a note",
    "tzofs":"0"
  },
  "adc":{
```

```
    "1":2,
    "2":2,
    "3":7,
    "4":2,
    "5":2,
    "6":2,
    "7":2,
    "8":29977,
    "Vin":12213,
    "Vbat":12090,
    "V5":5106
  },
  "out":{
    "1":0,
    "2":0,
    "3":0,
    "4":0
  },
  "pos":{
    "x":38677889,
    "y":-121174896,
    "z":0,
    "signal":-1
  },
  "fields":{
    "1":{
      "name":"Uptime",
      "unit":"s",
      "assettag":"uptime",
      "value":1102790
    },
    "2":{
      "name":"Temp (C)",
      "unit":"°C",
      "assettag":"temp",
      "value":26.6
    },
    "3":{
      "name":"Temp (F)",
      "unit":"°F",
      "assettag":"",
      "value":79.879997
    }
  }
}
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/control

Last update: **2023-08-08 23:26**

eventlog

Return data from the event log

Description

```
https://api.eze.io/v1/eventlog/{serial}/from={datetime}/[to={datetime}]
```

Parameters

`serial`	The serial number of the ezeio (XYZ987)
`from=datetime`	The beginning of the range. Format per RFC3339, or 1m/1h/1d
`to=datetime`	The end of the range. Format per RFC3339. If omitted, current time is assumed.

The `from` parameter can be given as number of minutes/hours/days relative to current time. For example `from=90m` means “90 minutes ago”.

Under no circumstances will the call return more than 300 rows of data.

Example usage

```
https://api.eze.io/v1/eventlog/XYZ987/from=2017-03-01/to=2017-03-15
```

Return value

JSON formatted data including the event specifics as well as a snapshot of all field values at the time of the event.

(below examples have whitespaces added for readability)

```
Example query:
https://api.eze.io/v1/eventlog/baa157/from=2019-02-01/to=2019-03-01

{
  "reqtime":"2019-10-28T20:58:06Z",
  "reqfrom":"2019-10-20T00:00:00Z",
  "reqto":"2020-01-18T00:00:00Z",
  "events":[
    {
    "time":"2019-10-28T20:48:17Z",
    "eventid":72857,
    "type":"ALARM",
```

```
    "source":"SCRIPT",
    "sourcename":"USER",
    "actionname":"My homemade script",
    "param1":"111.000000",
    "param2":"222.000000",
    "param3":"333.000000",
    "param4":"444.000000",
    "message":"The text sent from the script call",
    "meta":{
      "serial":"BAA-144",
      "group":102,
      "accountgroupid":2,
      "uptime":544,
      "name":"Test board 144",
      "tzofs":"-420"
    },
    "adc":{
      "1":4,
      "2":2,
      "3":2,
      "4":2,
      "5":2,
      "6":4,
      "7":1554,
      "8":29665,
      "Vin":12350,
      "Vbat":12172,
      "V5":5054
    },
    "out":{
      "1":0,
      "2":100,
      "3":100,
      "4":47
    },
    "pos":{
      "x":0,
      "y":0,
      "z":0,
      "signal":0
    },
    "fields":{
      "1":{
        "name":"Humidity",
        "unit":"%",
        "assettag":"",
        "value":"38.4"
      },
```

```
      "2":{
        "name":"Temperature",
        "unit":"°F",
        "assettag":"temp",
        "value":"82.45"
      }
    }
  ],
  "status":"OK",
  "exec_time":0.025
}
```


From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/eventlog

Last update: **2021-08-24 20:33**

group

Retrieve status of a group, group tree and optionally the assigned controllers.

Description

```
https://api.eze.io/v1/group/{mode}/["systems"]
```

The API key used will determine the starting group.

Parameters

`mode`	The mode of the request. See below.
`“systems”`	If given, the command will return all the ezeio controllers in each group.

Possible modes are:

`one` : The root group of the API key.
`tree` : The whole tree of groups, starting with the root group of the API key.
`flat` : Like `tree` but as a flat array.
`{groupno}` : If a group id number is given, return data on this group only.

Example usage

Fetch data for the API key's group, and the controllers therein:

```
https://api.eze.io/v1/group/one/systems
```

Fetch data for a specific group (id=56):

```
https://api.eze.io/v1/group/56
```

Fetch data for all groups and all controllers:

```
https://api.eze.io/v1/group/tree/systems
```

Additional options for more detail (may be combined):

`“fields”`	Return all field names and values for each controller
`“alarms”`	Return most recent alarm state for each controller
`“services”`	Return service settings for each controller

Example:

```
https://api.eze.io/v1/group/one/systems,fields,alarms
```

To return data for select fields, use the `tag` parameter:

```
https://api.eze.io/v1/group/one/systems/tag=TEMP
```

This will return field status only for those fields that have the assettag “TEMP”

Return value

JSON formatted data

(below examples have whitespaces added for readability)

This shows data from a call requesting hierarchical data including controllers:


```
{
  "reqtime":"2019-06-16T21:27:12Z",
  "groups":[
    {
      "id":"2",
      "name":"ACME Inc",
      "description":"ACME IoT System",
      "is_account":"1",
      "systems":[
        {
          "ezeid":"XYZ001",
          "name":"West monitor",
          "note":null
        },
        {
          "ezeid":"XYZ002",
          "name":"East monitor",
          "note":"No special note"
        }
      ],
      "children":[
        {
          "id":"55",
          "name":"Client A",
          "description":"",
          "is_account":"0",
          "systems":[
            {
              "ezeid":"XYZ101",
              "name":"Client A demo unit",
```

```
            "note":null
          }
        ]
      },
      {
        "id":"56",
        "name":"Client C",
        "description":"",
        "is_account":"0",
        "systems":[
          {
            "ezeid":"XYZ103",
            "name":"Home office",
            "note":null
          }
        ]
      }
    ]
  }
  ],
  "status":"OK",
  "exec_time":0.029423952102661133
}
```


From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/group

Last update: **2024-06-07 16:51**

samplelog

Return data from the sample log

Description

```
https://api.eze.io/v1/samplelog/{serial}/from={datetime}/[to={datetime}]/[fields]/[interval[,aggregation]]/["raw"]
```

Parameters

`serial`	The serial number of the ezeio (XYZ987)
`from=datetime`	The beginning of the range. Format per RFC3339, or 1m/1h/1d
`to=datetime`	The end of the range. Format per RFC3339. If omitted, current time is assumed.
`fields` (optional)	Comma separated list of fields. Possible fields are 1-90, GPSx, GPSy, GPSz, GPSsignal. If not given, all fields are returned.
`interval` (optional)	Sample interval. Given as a number followed by 'd', 'h' or 'm' for day, hour or minute.
`aggregation` (optional)	Method of aggregation. Only valid if interval is given.
“`raw`” (optional)	Only return values actually stored in database.

The `from` parameter can be given as number of minutes/hours/days relative to current time. For example `from=3d` means “3 days ago”.

If no interval is given, an automatic interval will be calculated based on the size of the range. This is to ensure the amount of data returned is reasonable.

If range >= 365 days, interval will be 1 day.
If range >= 60 days, interval will be 1 hour.
If range >= 7 days, interval will be 10 minutes.
If range >= 24 hours, interval will be 1 minute.
For smaller ranges than 24 hours, the interval will be 1 seconds.

Under no circumstances will the call return more than 15000 rows of data.

Aggregation modes:

`mean` : The mean (average) of samples in the interval
`mode` : The mode (most common value)
`median` : The middle value of sorted samples in the interval
`min` : The smallest value
`max` : The largest value
`sum` : The sum of all samples in the interval
`spread` : The difference between the highest and the lowest value in the interval

`count` : The number of samples in the interval

Only data from fields configured for higher logging rates than 10 minutes will be returned. See the syslog API for other fields.

Example usage

15 day interval using defaults:

```
https://api.eze.io/v1/samplelog/XYZ987/from=2017-03-01/to=2017-03-15
```

Specifying fields:

```
https://api.eze.io/v1/samplelog/XYS987/from=2018-10-05T06:30:00Z/to=2018-10-06T06:30:00Z/fields=2,4,12,GPSx,GPSy
```

Hourly max values:

```
https://api.eze.io/v1/samplelog/XYZ987/from=2019-02-01/to=2019-03-01/fields=1,2/interval=1h,max
```

Return value

JSON formatted data

(below examples have whitespaces added for readability)

```
Example query:
https://api.eze.io/v1/samplelog/baa157/from=2019-02-01/to=2019-03-01/fields=1,2

{
  "reqtime":"2019-06-16T22:51:49Z",
  "data":[
    {
      "time":"2019-02-01T00:00:00Z",
      "f1":null,
      "f2":null
    },
    {
      "time":"2019-02-01T00:10:00Z",
      "f1":432609,
```

```
      "f2":25.67
    },
    {
      "time":"2019-02-01T00:20:00Z",
      "f1":433209,
      "f2":25.709999
    },
    {
      "time":"2019-02-01T00:30:00Z",
      "f1":433809,
      "f2":25.709999
    },
    // ** Entries removed for brevity **
    {
      "time":"2019-02-28T22:00:00Z",
      "f1":786047.1666666666,
      "f2":25.016666666666666
    },
    {
      "time":"2019-02-28T23:00:00Z",
      "f1":789648,
      "f2":25.151667000000003
    }
  ],
  "status":"OK",
  "exec_time":0.027250051498413086
}
```


From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/samplelog

Last update: **2021-09-24 23:04**

scancode

Create, Read, Update or Delete records related to access control codes in the ezeio system.

Description

```
https://api.eze.io/v1/scancode/{serial}/{command}
```

The ezeio can store up to 324 scan code records. Each record has settings for when it is valid, limits on number of uses and what actions the code will perform when used.

The mechanics of how the code is entered/triggered in the ezeio is handled by a driver. The driver may interface with a keypad, card reader or other device, but this is not relevant to the scancode API. Please contact eze System for details.

The scancode API allows a third party system to manage the codes in the ezeio database. Currently the eze.io user portal do not allow access to the scancode data, but this is on our development roadmap.

The scan code record

Each scan code has the following properties:

`slot`	The record number in the ezeio (1-324)
`code`	The key/card scancode (max 10 characters, required when creating the record, optional on updates)
`name`	A plain-text name of the record (optional, default blank)
`userid`	An associated user id number (32 bit integer) (optional, default 0)
`notbefore`	An absolute time (UTC) defining when the scancode if first valid (default 'now')
`notafter`	An absolute time (UTC) defining when the scancode is no longer valid (default 'now'+30 days
`validweekdays`	An array of (7) flags (0/1) corresponding to Sunday-Saturday. Scancode is only valid if the corresponding flag is set. (default all '1')
`validfrom`	Time of day in HH:MM format. The scancode is only valid after this time. (optional, default 00:00, local time)
`validto`	Time of day in HH:MM format. The scancode is only valid before this time. (optional, default 24:00, local time)
`maxuse`	Max number of times the scancode can be used. (0-32767, optional, default 32767=infinite)
`maxlife`	Time in minutes from the first time the scancode is accepted until it will expire (0-32767, optional, default null)
`command`	The command code that will be reported to the driver when scancode is validated (0-255, optional, default 0)
`param1`	A parameter code that will be reported to the driver when scancode is validated (0-32767, optional, default 0)

param2	A parameter code that will be reported to the driver when scancode is validated (0-32767, optional, default 0)

URL Parameters

serial	ezeio serial number (XYZ123)
command	Command. See below

Possible commands:

To retrieve an existing record:

```
https://api.eze.io/v1/scancode/XYZ987/get[17]
```

The system will reply with a record similar to this (whitespace added for readability):


```
{
    "reqtime": "2024-10-21T21:01:54Z",
    "actions": [
        "Fetched 1 records"
    ],
    "scancodes": [
        {
            "slot": 17,
            "name": "Bob front gate access",
            "userid": 4711,
            "code": 131072,
            "notbefore": "2024-10-01 00:00:00",
            "notafter": "2025-12-31 23:59:59",
            "validweekdays": [
                0,
                1,
                1,
                1,
                1,
                1,
                0
            ],
            "validfrom": "07:30",
            "validto": "19:00",
            "maxuse": 400,
            "maxlife": null,
            "remainuse": 259,
            "usebefore": null,
            "command": 1,
            "param1": 0,
            "param2": 0
```

```
        }
    ],
    "status": "OK",
    "exec_time": 0.0208
}
```

In the example above, we see that Bob's scan code is 131072. When he enters/scans this code, and all time constraints are validated, the command code “1” will be sent to the driver, which we assume will trigger the gate to open.

If the scancode is used before the `notbefore`, or after the `notafter`, it will be ignored.

In the example above, the scancode will only be accepted on weekdays (not Saturdays or Sundays), and only between 07:30 and 19:00.

The `maxuse` setting will limit the number of times the code is accepted. The remaining count is reported in `remainuse`.

As an alternative to limiting the number of times a scancode can be used, the `maxlife` setting can be set to automatically expire the code a certain time after its first use. So for example, if we had set the maxlife in the example above to 1440 (number of minutes in 24h), Bob could start using the code any time between the `notbefore` and `notafter` times, but after first use he would only be able the use the code for 24 hours. Note that a scancode can either use the `maxuse` OR the `maxlife` feature (not both at the same time).

To retrieve an existing record:

```
https://api.eze.io/v1/scancode/XYZ987/getall
```

This will return all existing records. The format is identical to above, but with multiple records in the scancodes-array.

To create a new record or to update an existing record: `set`

```
https://api.eze.io/v1/scancode/XYZ987/set
```

This should be sent to the api as a 'POST' message, with a application/json body in the following format:

```
[
  {
    "slot": 17,
    "name": "Bob front gate access",
    "userid": 4711,
    "code": 131072,
```

```
    "notbefore": "2024-10-01 00:00:00",
    "notafter": "2025-12-31 23:59:59",
    "validweekdays": [
      0,
      1,
      1,
      1,
      1,
      1,
      0
    ],
    "validfrom": "07:30",
    "validto": "19:00",
    "maxuse": 400,
    "maxlife": null,
    "command": 1,
    "param1": 0,
    "param2": 0
  }
]
```


The `slot` property is mandatory, but all other properties are optional and will be populated with default values if the record is new, or left unchanged if the record exists. The system will reply with a status showing what action(s) were taken, like this:


```
{
  "reqtime": "2024-10-30T01:22:19Z",
  "actions": [
    "Updated #17"
  ],
  "status": "OK",
  "exec_time": 0.011
}
```

To delete a record: `del`

```
https://api.eze.io/v1/scancode/XYZ987/del[17]
```

Deletes record 17

To delete a record: `delall`

```
https://api.eze.io/v1/scancode/XYZ987/delall
```

Deletes all scancode records.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/scancode

Last update: **2026-03-17 18:38**

status

Retrieve the most recent known status of an ezeio.

Description

```
https://api.eze.io/v1/status/{serial}/[sections=X,Y]/[fields=X,Y]
```

This will retrieve the most recently reported status of the given controller. Note that the information is taken from the servers most recent status. This message does not generate traffic to the actual controller. Thus, it is normal that this status can be up to 10 minutes old.

Parameters

`serial`	ezeio serial number (XYZ123)
`sections` (optional)	Limit the reply to certain sections. Default is “meta, adc, out, pos, fields, fieldmeta”.
`fields` (optional)	Limit the reply to certain fields. Default is to receive all fields.

Available sections are:

`meta`	ezeio metadata
`adc`	status of on-board inputs
`out`	status of on-board outputs
`pos`	GPS/position data
`fields`	field values
`fieldmeta`	include field metadata with each field record
`device`	device metadata and status
`alarms`	alarm status (if available)

Example usage

Fetch default status record from serial XYZ123

```
https://api.eze.io/v1/status/XYZ123
```

Fetch only field values and position from serial XYZ123

```
https://api.eze.io/v1/status/XYZ123/sections=fields,pos
```

Return value

JSON formatted data

(below example has added whitespaces for readability)


```
{
  "reqtime":"2019-06-16T19:29:38Z",
  "time":"2019-06-16T19:20:00Z",
  "meta":{
    "serial":"XYZ-123",
    "groupid":"8",
    "accountgroupid":null,
    "name":"Test unit in lab",
    "note":"Here is a note",
    "tzofs":"0",
    "link":"Cell",
    "RSSI":-87,
    "sim_iccid":"89148000003622803539"
  },
  "adc":{
    "1":0,
    "2":2,
    "3":0,
    "4":4,
    "5":2,
    "6":2,
    "7":0,
    "8":29963,
    "Vin":12282,
    "Vbat":12144,
    "V5":5109
  },
  "out":{
    "1":0,
    "2":0,
    "3":0,
    "4":0
  },
  "pos":{
    "x":38677889,
    "y":-121174896,
    "z":0,
    "signal":-1
  },
  "fields":{
    "1":{
      "name":"Uptime",
      "unit":"s",
      "assettag":"uptime",
```

```
      "value":1101238
    },
    "2":{
      "name":"Temp (C)",
      "unit":"°C",
      "assettag":"temp",
      "value":26.48
    },
    "3":{
      "name":"Temp (F)",
      "unit":"°F",
      "assettag":"",
      "value":79.664001
    }
  },
  "status":"OK",
  "exec_time":0.048024892807006836
}
```


From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/status

Last update: **2024-02-21 19:10**

subscribe

Request a websocket data stream, and commands to manage the data stream.

Description

Request ticket and meta data:

```
https://api.eze.io/v1/subscribe/ticket
```

The `subscribe/ticket` call requests a websocket ticket and returns metadata for the group and ezeio units that are in context of the given API credentials.

Subsequent calls using the subscribe command are used to manage the data stream.

This functionality requires firmware 21012701 or later in the device.

Parameters

The `subscribe/ticket` call has no parameters

Example usage

Using the subscribe API requires the following steps:

Step 1 : Request ticket and metadata

Call the following API endpoint using valid API credentials:

```
https://api.eze.io/v1/subscribe/ticket
```

This call will return a JSON object (see example below) with metadata listing the systems that will be accessible through the websocket.

(whitespaces added for readability)

```
{
    "reqtime": "2021-01-31T14:15:16Z",
    "ws": "wss:\/\/api.eze.io\/ws\/wstktXXXXXXXXXXXXXXXXXXXXXX",
```

```
    "api":
"https:\/\/api.eze.io\/v1\/subscribe\/wstktXXXXXXXXXXXXXXXXXXXXXX\/",
    "account": {
        "id": "123",
        "name": "eze System",
        "description": "eze System - Testaccount"
    },
    "systems": [{
            "serial": "ABC-123",
            "name": "Demo unit",
            "note": "This is a demo unit in our lab",
            "lastseen": "2021-01-31T14:10:11Z",
            "fields": [{
                "fieldno": "1",
                "name": "Tank Level: Distance from top",
                "unit": "in",
                "decimals": "2",
                "assettag": "TANKLVL",
                "loginterval": "0"
            }, {
                "fieldno": "2",
                "name": "Output flow",
                "unit": "gal/min",
                "decimals": "1",
                "assettag": "",
                "loginterval": "60"
            }]
        },
        {
            "serial": "ABC-456",
            "name": "Other demo unit",
            "note": "This is also a demo unit in our lab",
            "lastseen": "2021-01-31T14:09:56Z",
            "fields": [{
                "fieldno": "2",
                "name": "Air temperature",
                "unit": "F",
                "decimals": "1",
                "assettag": "",
                "loginterval": "300"
            }, {
                "fieldno": "3",
                "name": "Relative humidity",
                "unit": "percent",
                "decimals": "0",
                "assettag": "",
                "loginterval": "0"
            }, {
```

```
                "fieldno": "4",
                "name": "Voltage",
                "unit": "V",
                "decimals": "1",
                "assettag": "V",
                "loginterval": "0"
            }]
        }
    ],
    "status": "OK",
    "exec_time": 0.055
}
```


The ws property is the complete websocket URI. This URI is valid only for 10 seconds following the call to subscribe/ticket.

The api property should be saved until the websocket is disconnected, as it is required for subsequent calls to update the data flow.

Step 2 : Open websocket and start receiving data

Use the ws URI from step 1 to open a websocket connection. The websocket will automatically receive all updates to any system (ezeio) covered by the initial API call, as the data becomes available to the cloud servers.

The data received will have a type, describing what kind of data this is:

LOGDATA	The field data is from the 'fast log'. The interval is determined by the log interval setting on each field.
STATUS	The field data is from the 'status log'. The interval is fixed to 10 minutes.

Note that the ezeio normally buffers data before the data is uploaded to the cloud servers, so the data may be delayed with up to 20 minutes

The STATUS updates will be sent every 10 minutes even if a faster subscription is active.

This is an example of what a STATUS update looks like (whitespace added for readability):


```
{
  "channel": "export:123",
  "data": {
    "type": "STATUS",
    "serial": "ABC-123",
    "time": "2021-01-31T14:20:00Z",
    "fields": [
      {
        "1": 73.4
      },
```

```
      {
        "2": 8.112
      }
    ]
  }
}
```

Note that the fields array will include all configured fields for this unit - regardless of their log setting. All fields are always logged every 10 minutes.

A LOGDATA update has the following format (whitespace added for readability):

```
{
  "channel": "export:123",
  "data": {
    "type": "LOGDATA",
    "serial": "ABC-123",
    "time": "2021-01-31T14:21:15Z",
    "timeout": 0,
    "fields": [
      {
        "2": 9.021
      }
    ]
  }
}
```

Note that this message only includes the fields that are configured for fast logging (interval less than 10 minutes). Please see below for the meaning of the `timeout` property.

Step 3 : Request subscription changes

With an open websocket and while receiving data from the ezeio system, you can call the subscribe API to request immediate updates or to cancel updates from a certain ezeio.

To request unbuffered log updates, the call is:

```
https://api.eze.io/v1/subscribe/wstktXXXXXXXXXXXXXXXXXXXXXXX/ABC123
```

The log updates will revert to normal (buffered) mode after 30 minutes.

To cancel updates, the call is:

```
https://api.eze.io/v1/subscribe/wstktXXXXXXXXXXXXXXXXXXXXXXX/-ABC123
```

Multiple requests can be sent in the same command:

```
https://api.eze.io/v1/subscribe/wstktXXXXXXXXXXXXXXXXXXXXXXX/-ABC123,ABC124,AB
```

```
C321,-ABC322
```

Up to 50 devices can be included in the same command

Example code to set up the websocket channel and receive data (PHP)

```php
<?php
    define("APIURI", "https://api.eze.io/v1/subscribe/ticket");

    // API keyID and key needs to be set up in eze.io under Groups->API.
    define("APIKeyID", "12345");
    define("APIKey", "XXXXXYYYYYXXXXXYYYYYXXXXXYYYYY");

    // Using the textalk/websocket client
    // https://github.com/Textalk/websocket-php
    require('vendor/autoload.php');
    use WebSocket\Client;

    // Request a websocket key and metadata using cURL
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, APIURI);

    // All API calls use Digest AUTH
    curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_DIGEST);
    curl_setopt($ch, CURLOPT_USERPWD, APIKeyID.":".APIKey);

    // Set a 5s timeout, and return any received data
    curl_setopt($ch, CURLOPT_TIMEOUT, 5);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);

    // Execute the cURL request
    $response = curl_exec($ch);
    curl_close($ch);

    // Decode the json reply into an associative array
    $json = json_decode( $response, TRUE );

    // Status should be 'OK'
    if($json["status"] != "OK")
        die("ERROR:\n" . print_r($json, TRUE));

    // Open the websocket
    $wsclient = new WebSocket\Client( $json["ws"] );

    // Loop forever (until websocket disconnects)
    while ( true ) {
```

```
        try {
            $message = $wsclient->receive();

            // Process the data - here we just print it for testing
            print_r( json_decode( $message, TRUE ) );
        }
        catch (\WebSocket\ConnectionException $e) {
            // If the websocket was disconnected, exit the loop
            if( !$wsclient->isConnected() )
                break;
        }
    }
    $wsclient->close();
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/subscribe

Last update: **2021-01-28 23:49**

syslog

Return data from the system log.

Description

```
https://api.eze.io/v1/syslog/{serial}/from={datetime}/[to={datetime}]/[fields]
/[interval[,aggregation]]/["raw"]
```

Parameters

`serial`	The serial number of the ezeio (XYZ987)
`from=datetime`	The beginning of the range. Format per RFC3339 or 1d/1h/1m
`to=datetime`	The end of the range. Format per RFC3339. If not given, current time is assumed.
`fields` (optional)	Comma separated list of fields. Possible fields are 1-90, GPSx, GPSy, GPSz, GPSsignal. If not given, all fields are returned.
`interval` (optional)	Sample interval. Given as a number followed by 'd', 'h' or 'm' for day, hour or minute.
`aggregation` (optional)	Method of aggregation. Only valid if interval is given.
“`raw`” (optional)	Only return values actually stored in database.

The `from` parameter can be given as number of minutes/hours/days relative to current time. For example `from=17h` means “17 hours ago”.

If no interval is given, an automatic interval will be calculated based on the size of the range. This is to ensure the amount of data returned is reasonable.

If range >= 365 days, interval will be 1 day.
If range >= 60 days, interval will be 6 hours.
If range >= 7 days, interval will be 1 hour.
For smaller ranges than 7 days, the interval will be 10 minutes.

Under no circumstances will the call return more than 5000 rows of data.

Aggregation modes:

`mean` : The mean (average) of samples in the interval
`mode` : The mode (most common value)
`median` : The middle value of sorted samples in the interval
`min` : The smallest value
`max` : The largest value
`sum` : The sum of all samples in the interval
`spread` : The difference between the highest and the lowest value in the interval
`count` : The number of samples in the interval

Example usage

15 day interval using defaults:

```
https://api.eze.io/v1/syslog/XYZ987/from=2017-03-01/to=2017-03-15
```

Specifying fields:

```
https://api.eze.io/v1/syslog/XYS987/from=2018-10-05T06:30:00Z/to=2018-10-06T06:30:00Z/fields=2,4,12,GPSx,GPSy
```

Available fields are:

#	Field # value, where # is an integer, 1-90
ADC#	Raw ADC reading, where # is 0-10. 8=DC in, 9=Terminal DC, 10=5V terminal
out#	Raw output status, where # is 0, 1, 2 or 3
fRunning	1=Logic running. 0=Logic halted
ethComm	1=Ethernet connection available. 0=No Ethernet connection
ethLink	1=Ethernet link availabe. 0=No Ethernet link
GPSx	Location x-coordinate (latitude)
GPSy	Location y-coordinate (longitude)
GPSz	Location z-coordinate (elevation)
GPSsignal	GPS received signal
gpsLock	1=GPS received has locked in signals from the satellites. 0=no lock
groupid	The group ID associated with this unit
uptime	Uptime in seconds
gsmComm	1=Cellular data connection established. 0=No cellular data
gsmLink	1=Cellular service found. 0=No cell service
gsmNet	1=Cellular IP acquired. 0=No connection
cID	Cell tower ID
cLAC	Cell location area code
cMCC	Cell network MCC
cMNC	Cell network MNC
cRSSI	Cell received signal level

Hourly max values:

```
https://api.eze.io/v1/syslog/XYZ987/from=2019-02-01/to=2019-03-01/fields=1,2/interval=1h,max
```

Return value

JSON formatted data

(below examples have whitespaces added for readability)

```
Example query:
https://api.eze.io/v1/syslog/baa157/from=2019-02-01/to=2019-03-01/fields=1,2

{
  "reqtime":"2019-06-16T22:51:49Z",
  "fieldmeta":[
    {
      "no":"1",
      "name":"Uptime",
      "tag":"",
      "decimals":"0",
      "unit":"s"
    },
    {
      "no":"2",
      "name":"Temperature",
      "tag":"temp",
      "decimals":"1",
      "unit":"°C"
    }
  ]
  "data":[
    {
      "time":"2019-02-01T00:00:00Z",
      "f1":null,
      "f2":null
    },
    {
      "time":"2019-02-01T00:10:00Z",
      "f1":432609,
      "f2":25.67
    },
    {
      "time":"2019-02-01T00:20:00Z",
      "f1":433209,
      "f2":25.709999
    },
    {
      "time":"2019-02-01T00:30:00Z",
      "f1":433809,
      "f2":25.709999
    },
    // ** Entries removed for brevity **
    {
      "time":"2019-02-28T22:00:00Z",
```

```
      "f1":786047.1666666666,
      "f2":25.016666666666666
    },
    {
      "time":"2019-02-28T23:00:00Z",
      "f1":789648,
      "f2":25.151667000000003
    }
  ],
  "status":"OK",
  "exec_time":0.027250051498413086
}
```


From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/apiref/syslog

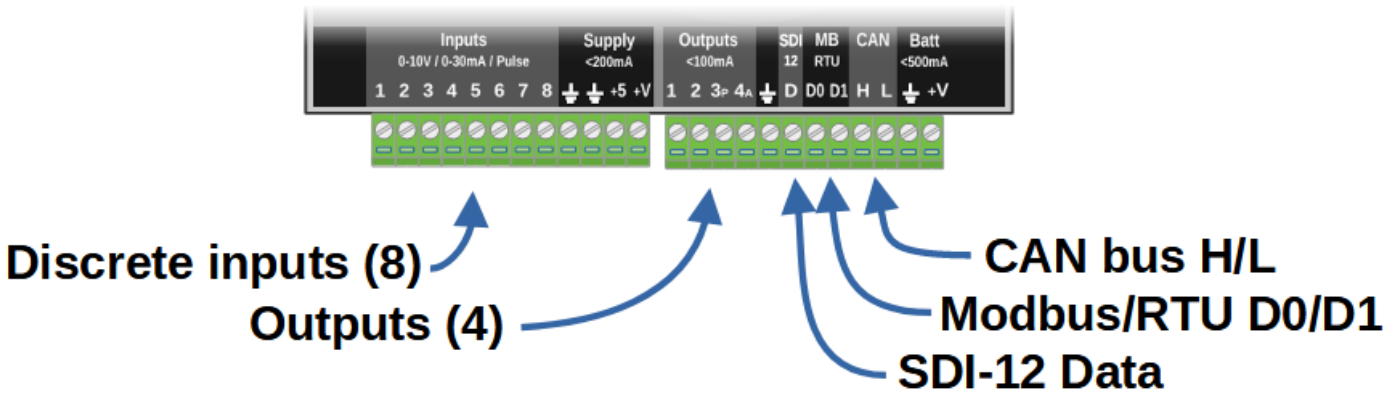
Last update: **2021-09-24 23:04**

Connecting Sensors and Peripherals

The ezeio can connect to a large variety of sensors and peripherals. eze System manufacture and sell some compatible sensors and devices, but many third-party devices also work with the ezeio thanks to the ezeio having a multitude of interfaces.

Simply connecting a sensor or device is usually not enough to make it work. You will also need to configure the ezeio to use the sensor or device. See Devices & Drivers for a place to start.

The two green screw terminals on the bottom of the ezeio has the following functionality:



The terminals are removable to simplify wiring.

Discrete inputs

The 8 discrete inputs are individually configurable to measure voltage (0-10VDC), current (0-30mADC), resistance (0-500000 Ohm) or pulses.

More details : Inputs

Outputs

Output 1 and 2 are active on/off outputs, and outputs the supply voltage when active. Output 3 can be used as a PWM output, or as a on/off output like 1&2. Output 4 outputs a DC voltage 0-10V (max 20mA).

More details : Outputs

SDI-12

The SDI-12 port conforms to the SDI-12 v1.3 standard, and allows for connection of various sensors and instruments. Sensors for environmental monitoring often support SDI-12.

More details : SDI-12

Modbus/RTU

The Modbus/RTU port is commonly used with third party devices, such as energy meters, temperature controllers, sensors, VFD's, I/O expansion and much more. Modbus/RTU is one of the most common interfaces in the industry.

More details : Modbus/RTU

CAN

The CAN port allows for interfacing with other CAN/J1939 devices. This is common in vehicles and engine control applications.

More details : CAN

Modbus/TCP

The Ethernet port also supports third party devices, compatible with Modbus/TCP.

More details : Modbus/TCP

ezeio custom expansion

The ezeio also has an expansion port on the side for eze System's own expansion devices.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections

Last update: **2024-05-08 19:29**

CAN

The ezeio supports the CAN hardware standard.

Currently, the CAN port is only used to interface with ezeio proprietary hardware.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections/can

Last update: **2021-09-24 22:06**

Inputs

There are eight (8) inputs on the ezeio hardware. More inputs can be added using the ezeio expansion modules https://ezesys.com/ezeio-mkii-i/o-expander. *Other types of inputs can be added using third party devices connected via Modbus, CAN or SDI-12*.

The ezeio and ezeio I/O Expander inputs are compatible with a wide variety of sensors, either directly or through the use of an amplifier or transmitter. These common, industry standard, sensor outputs are directly supported.

Input Mode	Sensor output type	Function	Sensor examples
Voltage	0-10VDC	Measuring an external voltage between 0 and 10V relative to ground (default)	pressure transducers, air quality
Current	0-30mA (4-20mA)	Measuring a current, 0-30mA, typically used with 4-20mA sensors	pressure transducers, flow meters
Resistive	0-500,000 Ohms	Measuring an external resistance (between the input and ground)	switches, buttons, potentiometers
Thermistor	Ohms to temp curve	Like Resistive, but outputting the temperature	2.25K @ 25C, 10K @ 25C, 100K @ 25C
Pulse Count		Counting the number of pulses	Gas, Water, and Electric meters
Pulse Rate	Hz	Measuring the frequency of pulses	Gas, Water, and Electric meters

Settings in the ezeio's device drivers electronically configure the input to the desired mode. The drivers can also convert/scale the raw value to engineering units. For more information see the Devices section of this manual.

Input Drivers

Discrete Input driver

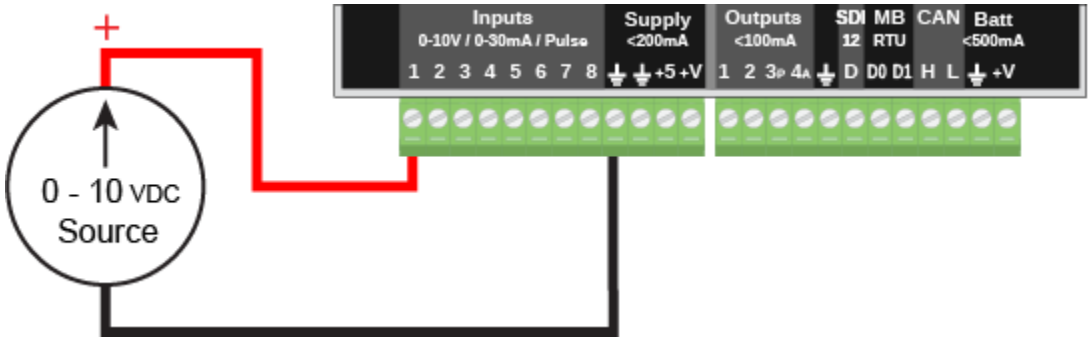
Pulse Input driver

Do not exceed the rated voltage (10 VDC) or current (30 mA) for the inputs.

Voltage setting (0-10 VDC)

Selecting a voltage setting switches the input to high impedance mode, and the RAW input value will reflect the voltage on the input in mV (0-10000 mV). The impedance when in 0-10V mode is 69200 Ohm.

Input setting : 0-10V	
Input range	0 - 10240mV
Resolution	2.5mV
Accuracy	+/- 5mV
Impedance	62900 Ohm
Excitation	no



Changing the range for higher voltages

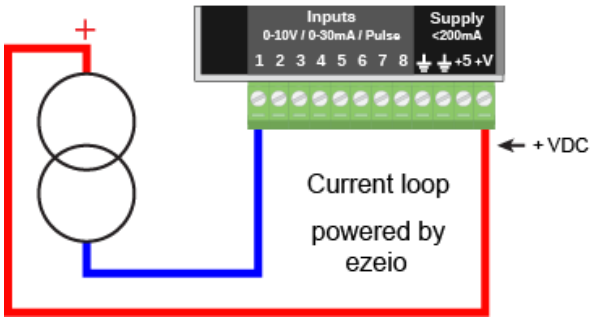
If you need to monitor a higher voltage than 10V, you can connect an in-line 270kOhm resistor. This will extend the input range to 0-54.2V with a resolution of 13.24mV. We do not recommend this method for voltages exceeding 50V. For higher voltages, use a suitable voltage transducer.

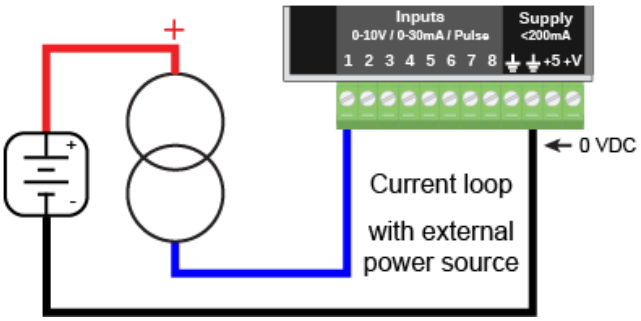
Current setting (0-30mA)

When set for 0-30mA, the input will be internally connected to ground via a 200 Ohm resistor. The input scaling switches to reflect micro-Ampere though the resistor.

This mode is commonly used for industry standard 4-20mA sensors.

Input setting : 0-30mA	
Input range	0 - 30000µA
Resolution	12.5µA
Accuracy	+/- 25µA
Impedance	200 Ohm
Excitation	no



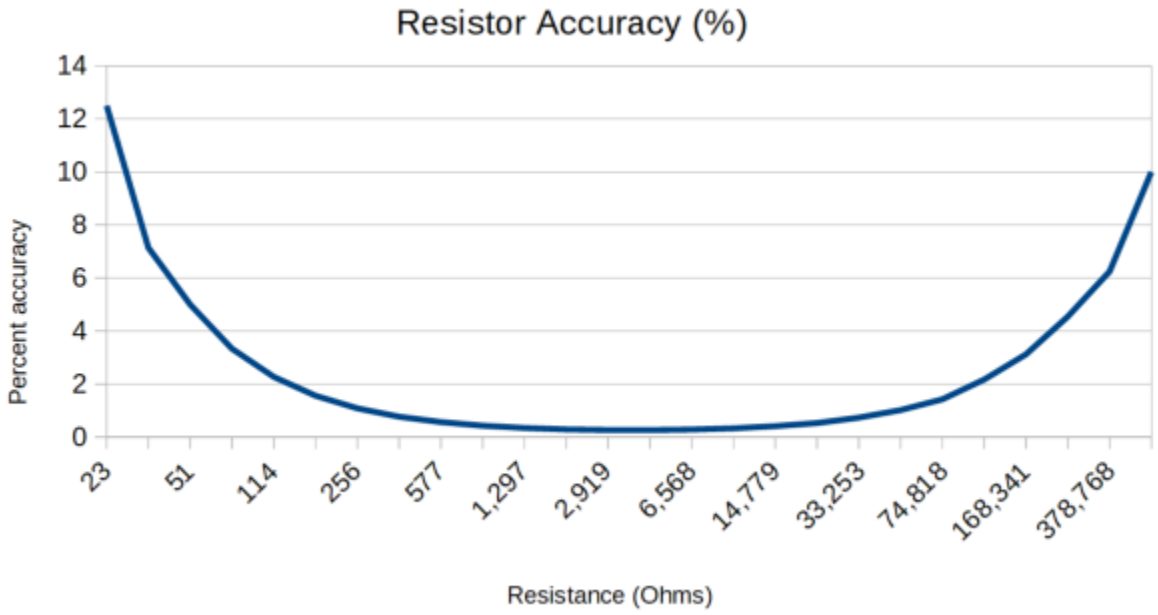


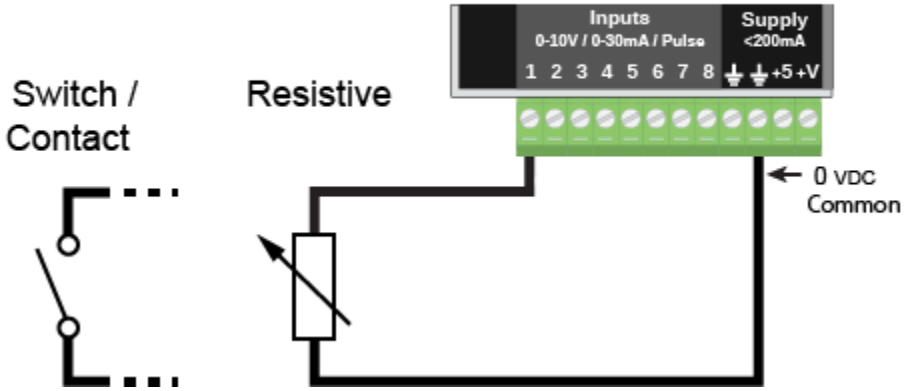
Resistive setting (Ohms)

When the input is set to monitor an external resistance, an internal 4780 Ohm resistor is enabled to connect the input to the internal +5V. This allows a small current to run through the external resistor, and this allows the ezeio to measure the voltage and convert it to resistance.

An internal reference is used to find the ratio and more accurately calculate the external resistance in Ohms.

Input setting : Resistance	
Input range	0 - 1MOhm
Resolution	Varies over the range
Accuracy	Better than 2% in the 200Ohm - 100kOhm range
Impedance	not applicable
Excitation	yes, 4870 Ohm to 5V (1mA short circuit)





Thermistor setting

A thermistor is a resistor that changes resistance with temperature. The input setting is very similar to when an external resistor is used, but the input value is recalculated using the Steinhart-Hart math to reflect a temperature.

There are five different types of thermistors supported:

Type	Typical application	Beta
10k, Type 2	Indoor temperatures, 0°C to +60°C (30-140°F)	3800K
10k, Type 3	Indoor temperatures, -5°C to +60°C (20-140°F)	3500K
10k	Outdoor temperatures, -25°C to +60°C (-10-140°F)	3380K
2k2	Refrigeration systems, -40°C to +20°C (-40-70°F)	3800K
100k	Heating systems, boilers, +50°C to +150°C (120-300°F)	4000K
PT1000 [1)]	Wide range, -70°C to +370°C (-90-700°F) (reduced resolution)	n/a

The thermistors will continue to work outside the above temperatures, but the accuracy will be lower at higher and lower temperatures.

The PT1000 type sensor will work over a very wide temperature range, but lower resolution than a thermistor (a few °K typically).

Input setting : Thermistor	
Input range	20000 to 65000 (°K x100)
Resolution	Varies over the range
Accuracy	0.5°K when around 10kOhm
Impedance	not applicable
Excitation	yes, 4870 Ohm to 5V (1mA short circuit)

The ezeio inputs does not directly support PT100 or Thermocouple type sensors. If you need to use these types of sensors, use a signal conditioner to convert the signal to 0-10V or 4-20mA, or use a digital expander with suitable inputs.

Pulse count/frequency

When set to monitor pulses, the ezeio input can be configured either in resistive mode or in 0-10V mode. If the pulse source is a passive switch (a.k.a dry contact or potential free contact), like a KYZ output or a opto-coupler transistor output, the resistive mode is suitable and will apply enough current through the circuit for the ezeio to detect the pulses. If the pulse source is active, outputting a switched voltage, the 0-10V input mode is more suitable.

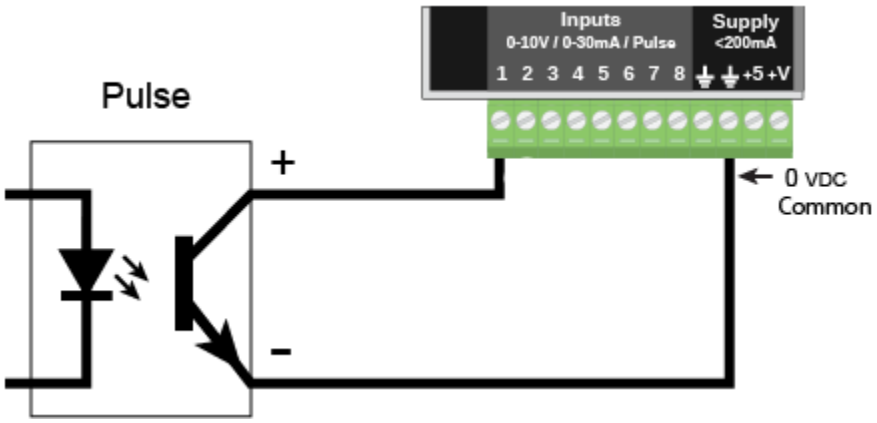
The maximum frequency that can be detected reliably by the ezeio is 400Hz.

If the pulse interval is larger than 20s (0.05Hz), the ezeio will consider this as no pulses (0 Hz).

Input setting : Pulse	
Input range	50-400000 mHz (mHz = Hz x 1000)
Resolution	
Accuracy	
Impedance	not applicable
Excitation	optional

The register value has the unit mHz (Hertz x1000). To convert to other units, use these examples:

Input	**Desired output**	**Conversion**
	Pulses per second (Hz)	`r(x,x)*0.001`
	Pulses per minute (ppm)	`r(x,x)*0.06`
	Pulses per hour (pph)	`r(x,x)*3.6`
2L/pulse	Liters per minute	`2*r(x,x)*0.06`
12 pulses per Liter	Liters per hour	`1/12*r(x,x)*3.6`
14 gal/pulse	Gallons per hour	`14*r(x,x)*3.6`
100 pulses per kWh	kW	`1/100*r(x,x)*3.6`
0.5kWh per pulse	W	`0.5*r(x,x)*3600`



1)

PT1000 is not technically a thermistor, but works similarly

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections/inputs

Last update: **2024-05-08 19:29**

Modbus/RTU

The Modbus Port Setup driver is required to activate the Modbus port and configure bus setting

Modbus/RTU is a very common protocol for control and monitoring applications. It is supported by a wide range of device types from almost all manufacturers of control electronics. Modbus/RTU is popular because it is very reliable and simple to use and implement.

For detailed information about Modbus, please refer to http://modbus.org/

The ezeio supports Modbus/RTU over RS-485 serial multidrop bus, and the ezeio will act as the master device, polling other devices on the bus. All devices on the bus shall be connected in parallel.

RS-485 is not the same thing as Modbus/RTU. If your device specification does not explicitly state support for Modbus/RTU, it most likely is not compatible with the ezeio.

The Modbus/RTU protocol requires exactly one master device (the ezeio), and anywhere from one to 32 slave devices sharing the same communication bus. Note that there can only be a single master on the bus. All slave devices must be set up with unique addresses (see below).

Communication wiring

Modbus/RTU is using standard RS-485 connections. RS-485 uses two conductors for data, often referred to as 'D0/D1', 'A/B', 'A-/B+' or sometimes just '-/+'.

Per Modbus/RTU standard, the correct naming for the data wires are 'D0/D1'. Please note that “A/B” or “+/-” designations are not standardized and may vary between device manufacturers.

Often the first step in troubleshooting is to simply swap the bus wires. When selecting a device driver the note field may contain device specific instructions like; Connect A to D1 or White wire to D0.

The data wires must be run as a single bus, with each device connecting to the next device in a chain. Avoid star configurations, and do not use long T-stubs.

End of line resistors

End of line resistors (also knows as termination resistors) are used to minimize the 'echo' of the data signal when it hits the end of the wire. Just like a audible echo, the reflected signal will bounce back-and forth on the wire and fade out with each bounce. The end of line resistor acts like a damper, absorbing the energy and thus reducing the effects of the echo. If the data wire is very long, the echo will take longer to fade out. On a short wire, the echo will disappear very quickly. Thus, end of line resistors are only required when the wire is long relative to the data rate.

Modbus/RTU typically runs at a relatively low data rate. 9600bps or 19200bps are typical default speeds for Modbus/RTU. At these rates, a 'long' wire is in the order of hundreds of meters. As a rule of thumb, when bus wires are shorter than 100m (300ft) no end of line resistors are required.

If your bus wiring is more than 100m (300ft), connect two 120 Ohm resistors between the D0 and D1 wires, one in each end of the bus wire. The ezeio can be connected anywhere on the bus. It does not need to be in one of the ends.

Some devices has built-in termination resistors. Make sure no more than two termination resistors are present on the bus.

Bus biasing

The ezeio has built-in biasing resistors. No additional biasing should be required.

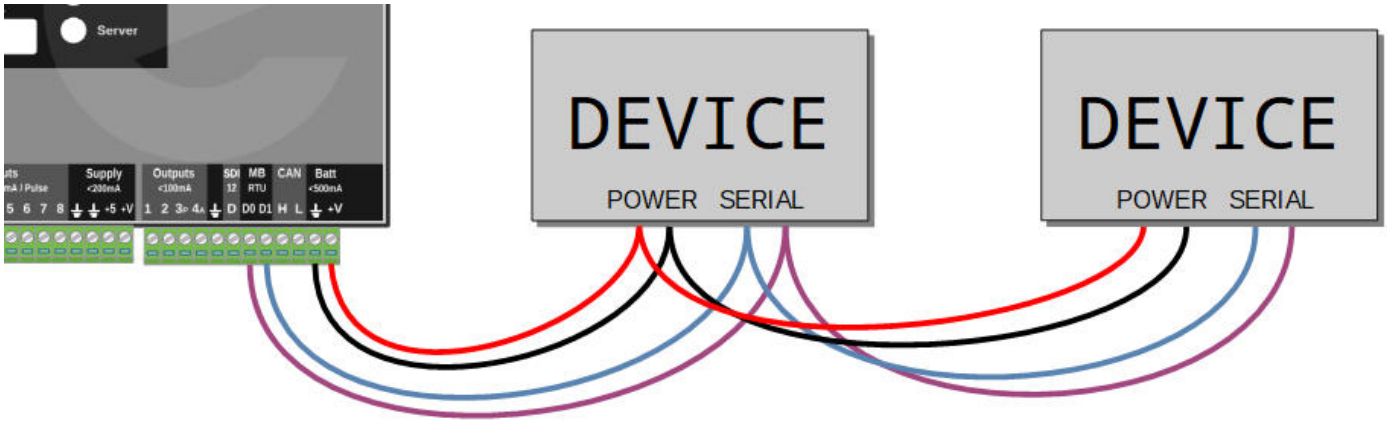
Ground

A common misconception is that RS-485 does not need a ground connection. This is wrong. Without a ground connection the bus may still work, but if there is a voltage potential between the devices there is a significant risk the communication will fail or even damage the hardware.

Always use a ground connection between RS-485 devices

Device power

Devices may be powered from the ezeio as shown in the wiring example below. Please observe the max limits on current that can be sourced from the ezeio. Devices may also be self-powered or use a separate power supply. In any case, the ground wire (black) must run from the ezeio to all connected devices.



Polling address

Each device must be set up with a unique polling address (1 through 254). The method of setting the address varies between devices. Some have a DIP switch or rotary encoder, while other devices have a screen/menu or require special software to be set up. Please refer to the manufacturer's installation manual for your particular device. The polling address must be unique on the bus. If two devices are set up with the same polling address, the ezeio will not be able to communicate with these devices.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections/modbusrtu

Last update: **2021-09-24 22:05**

Modbus/TCP

Modbus/TCP is a common protocol for connecting control and monitoring equipment. The fundamental protocol is the same as for Modbus/RTU, with the obvious difference that Modbus/TCP uses an Ethernet connection to link the devices together, while Modbus/RTU use a serial, RS-485 bus.

Another key difference between Modbus/TCP and Modbus/RTU is that in Modbus/RTU, only one device must be the 'master', and all the other devices must be 'slaves'. In Modbus/TCP, any device can be both a 'server' (providing information to a requestor) or a 'client' (requesting information from servers) - all at the same time.

Note that most Modbus/TCP servers allow a limited number of concurrent connections. Please refer to the documentation of the specific device you are using for these details. The ezeio allows up to 20 concurrent Modbus/TCP connections.

For detailed information about Modbus, please refer to http://modbus.org/

Modbus/TCP wiring

Since Modbus/TCP uses Ethernet, common networking hardware is used to connect devices together.

We recommend to always use an Ethernet switch between the ezeio and the device(s).

A switch is always required if more that one device is connected to the ezeio. If only a single device is connected, a direct connection may work. We have noticed some devices do not properly handshake when connected directly, so we strongly recommend to always use a switch even with a single device.

The ezeio supports standard Ethernet TP10/100, Auto MDX. The 'Auto MDX' part means that you can use either a regular Ethernet patch cable, or a 'crossover' Ethernet cable. The ezeio will detect the cable type and adapt accordingly.

All standard Ethernet wiring practices apply.

The ezeio uses the same connector for both cloud connectivity and Modbus/TCP. If the LAN provides public Internet access, you must use a correctly configured router to disallow any traffic from the public Internet to your control devices.

If the ezeio is using the built-in cellular modem for cloud connectivity, the Ethernet connection can be dedicated to the Modbus/TCP function, eliminating any security concerns.

Power considerations

The Ethernet connector on the ezeio is electrically isolated. It does not provide power, nor can does it accept power for the ezeio. Any devices connected to the ezeio via Ethernet must be separately powered. The ezeio does not support PoE.

Modbus/TCP addressing

By default the ezeio will expect a DHCP server to provide IP addressing for the LAN. If there is no DHCP server set up, you will need to manually set up fixed IP's in each device, including the ezeio. All devices and the ezeio must be configured for the same subnet (commonly a private network like 192.168.0.0/16, 172.16.0.0/12 or 10.0.0.0/8). Only the last octet is used for local addressing, so the subnet must be a /24 (255.255.255.0).

All devices and the ezeio must be configured for the same /24 subnet.

By default, the ezeio uses port 502 for both client and server Modbus/TCP traffic. The port number can be changed in the ezeio configuration. The TCP server can be turned off by setting the Server Port (under Configure/System) to 0.

Modbus server registers

The ezeio exposes the following registers:

Input registers Read (command 4)

Register	Address	Type	Description
30001	0	UINT_16	Input 1 raw value (mV, uA or Ohm)
30002	1	UINT_16	Input 2 raw value (mV, uA or Ohm)
30003	2	UINT_16	Input 3 raw value (mV, uA or Ohm)
30004	3	UINT_16	Input 4 raw value (mV, uA or Ohm)
30005	4	UINT_16	Input 5 raw value (mV, uA or Ohm)
30006	5	UINT_16	Input 6 raw value (mV, uA or Ohm)
30007	6	UINT_16	Input 7 raw value (mV, uA or Ohm)
30008	7	UINT_16	Input 8 raw value (mV, uA or Ohm)
30009	8	UINT_16	Supply voltage (mV)
30010	9	UINT_16	Batt terminal voltage (mV)
30011	10	UINT_16	5V terminal voltage (mV)
30012	11	UINT_16	Uptime (seconds), masked low 16 bits
30013	12	INT_32	Location Longitude (*10^6 °)
30015	14	INT_32	Location Latitude (*10^6 °)
30017	16	INT_32	Location Elevation (*10 m)
30019	18	UINT_16	GPS signal
30020	19	UINT_16	constant 23456 (5BA0 hex)
30021	20	UINT_32	constant 123456789 (075BCD15 hex)
30023	22	UINT_32	ezeio serial number

Register	Address	Type	Description
30025	24	UINT_32	RTC time (UNIX timestamp)

Holding registers Read (command 3)

Register	Address	Type	Description
40001	0	INT_16	Field 1 value, as integer masked to 16 bit
40002	1	INT_16	Field 2 value, as integer masked to 16 bit
/	/	/	/
40090	89	INT_16	Field 90 value, as integer masked to 16 bit
40101	100	INT_32	Field 1 value, as integer
40103	102	INT_32	Field 2 value, as integer
/	/	/	/
40280	279	INT_32	Field 90 value, as integer
40301	300	FLOAT	Field 1 value, as IEEE754 floating point
40303	302	FLOAT	Field 2 value, as IEEE754 floating point
/	/	/	/
40480	479	FLOAT	Field 90 value, as IEEE754 floating point
41001	1000	INT_32	User defined cell, address 0 *(see script command SetVal/GetVal)*
41003	1002	INT_32	User defined cell, address 1
/	/	/	/
42023	2022	INT_32	User defined cell, address 511

Holding register Write (command 6)

NOTE: The field will only accept the value written if the field is configured as Writeable.

Register	Address	Type	Description
40001	0	INT_16	Field 1 value, as integer masked to 16 bit
40002	1	INT_16	Field 2 value, as integer masked to 16 bit
/	/	/	/
40090	89	INT_16	Field 90 value, as integer masked to 16 bit
41001	1000	INT_16	Write to user defined cell 0, as 16 bit integer *(see script command SetVal/GetVal)*
41003	1002	INT_16	Write to user defined cell 1, as 16 bit integer
/	/	/	/
42023	2022	INT_16	Write to user defined cell 511, as 16 bit integer

Multiple holding registers Write (command 16 / 0x10)

NOTE: The field will only accept the value written if the field is configured as Writeable.

Register	Address	Type	Description
40001	0	INT_16	Field 1 value, as integer masked to 16 bit
40002	1	INT_16	Field 2 value, as integer masked to 16 bit
/	/	/	/
40090	89	INT_16	Field 90 value, as integer masked to 16 bit
40101	100	INT_32	Field 1 value, as integer
40103	102	INT_32	Field 2 value, as integer
/	/	/	/
40280	279	INT_32	Field 90 value, as integer
40301	300	FLOAT	Field 1 value, as IEEE754 floating point
40303	302	FLOAT	Field 2 value, as IEEE754 floating point
/	/	/	/
40480	479	FLOAT	Field 90 value, as IEEE754 floating point
41001	1000	INT_32	User defined cell, address 0 *(see script command SetVal/GetVal)*
41003	1002	INT_32	User defined cell, address 1
/	/	/	/
42023	2022	INT_32	User defined cell, address 511

Modbus/TCP security

Modbus/TCP is **not** a secure protocol. It relies on the security of the LAN. Typically a Modbus/TCP LAN should be dedicated to this functionality, and not have any Internet connectivity at all. Never allow incoming traffic on the same port as Modbus is using (TCP/502). Modbus/TCP is not designed for remote access and should never be routed on the Internet.

Modbus/TCP does not have any security or access control. Never use Modbus/TCP on public or untrusted networks.

If the Ethernet is connected to a public or untrusted network, we recommend turning off the TCP server by setting the Server Port (under Configure→System) to 0.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections/modbustcp

Last update: **2023-06-08 20:44**

Outputs

The ezeio has four (4) outputs built in. Additional outputs can be added using ezeio expansion devices, or third party devices connected via Modbus, CAN or SDI-12.

ezeio MkII I/O expander - https://ezesys.com/ezeio-mkii-i/o-expander

The function of each output is completely controlled by software. The function can be programmed/automatic, manual or a combination of manual and automatic. Common use for outputs are to control a heating or cooling system based on the input from a temperature sensor (like a thermostat), controlling indicator lights or lighting, sprinkler control and much more.

The four outputs on the ezeio are of three types:

Output number	Type	Range	Function
1 and 2	On/Off	0-100	Outputs 0V when off (<50) and supply voltage when on (>=50)
3	Pulse width	0-100	Outputs a 50Hz rectangular wave with duty cycle 0-100%
4	Analog	0-100	Outputs 0-10V based on its setting 0-100%

Note; Output 3 can be used as a standard on/off output if set to 0/100%.

From the software, all outputs are controlled the same way, using a 0-100% setting.

Connecting loads to the outputs

The ezeio outputs are active, meaning they source a voltage when on. The voltage from output 1, 2 and 3 is sourced directly from the supply voltage, so if the ezeio runs on 15V, there will be 15V coming out from the outputs when they are switched on (100%).

The analog output will output 0-10V regardless of the supply voltage. If the supply voltage is less than about 11V, the analog output will not be able to support the full voltage range.

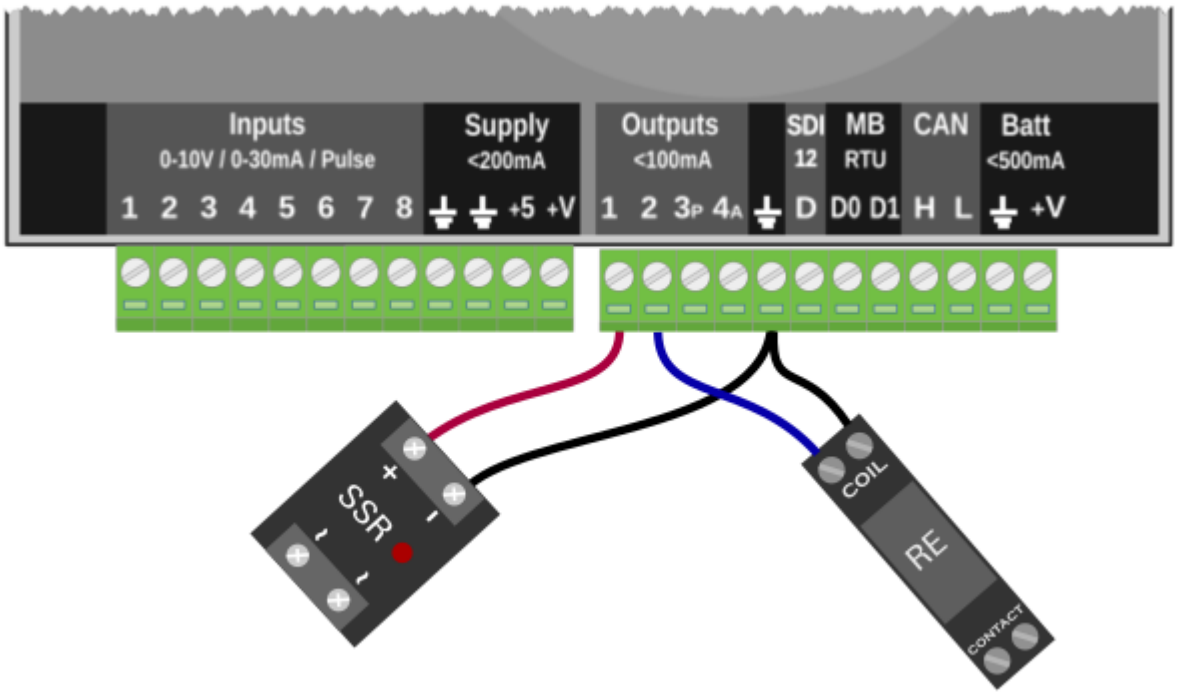
Do not connect external voltage to the outputs.

All outputs are fused with a self-resetting fuse. On output 1,2 and 3 the max continuous current output is 100mA. For the analog output, the max current output is 20mA.

Do not overload the outputs.

Any loads you connect to the outputs shall be connected between ground and the output only.

This shows a Solid State Relay connected to output #1 (note the polarity), and a standard relay connected to output #2:



Digital outputs 1 and 2

Output 1 and 2 are digital, meaning they are either ON or OFF. When the output is OFF, there is no voltage on the terminal. When the output is ON, the voltage from the ezeio supply is routed to the output terminal.

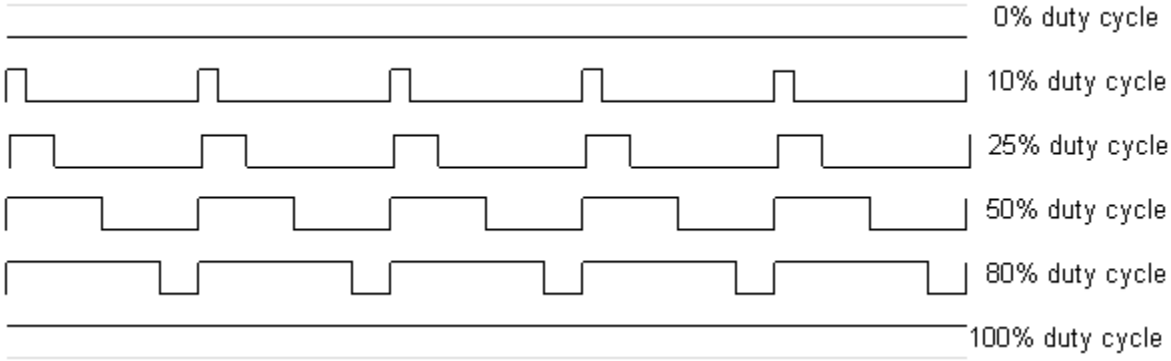
Up to 100mA current can be continuously drawn from the output when it is ON. Each output is equipped with a self-resetting fuse. If the current exceeds 200mA, the fuse will open and stay open until the load is removed.

From the software, all outputs are controlled by setting the value between 0 and 100 (%). Any value 0-49 will set the digital outputs to OFF. Any value from 50-100 will set the digital outputs on ON. When reading the state of a digital output, the value will either be 0 or 100.

Digital/PWM output 3

Output 3 is a PWM (Pulse Width Modulation) output. Cycle length is fixed at 50Hz. The percentage of time it stays “ON” (between 0% and 100%), during each cycle, is referred to as the “duty cycle” (see

illustrated below). This type of output is typically used to vary speed or position, as part of a feedback loop. We recommend using the ezeio's “User script” and/or “Fields” to create the programming logic controlling the PWM output.



Just like the two digital outputs (1&2), the output voltage when in the ON state is the same as the supply voltage to the ezeio. The PWM output also uses the same type of fuse as output 1&2.

This output can be used as a 3rd digital output. When set to 0% or 100%, the PWM output behaves exactly like a standard digital output.

Analog output 4

Output 4 is an analog output. It will output a voltage between 0V and 10V. This is an advanced control/automation feature that requires programming through the ezeio's user script (see script examples below).

From software, the analog output can be set to any value between 0 and 100, where 0 = 0V output, and 100 = 10V output. So to get 7.5V output, set the value to 75.

The analog output can supply max 20mA output.

The analog output is a low current control signal output. It is not designed to drive loads directly.

The analog output range is fixed at 0-10V. It does not depend on the supply voltage to the ezeio.

For the analog output to work correctly, the supply voltage to the ezeio must be at least 11V. If the supply voltage is lower than 11V, the analog output will not be able to output the full range.

Script Examples

```
   Setoutput(4, 75); // Set a static level

   SetOutput(4, GetField(19));  // Use a "Field" to set level

// Use logic to increase level

   if( (T1-T2) > FanTempDelta )  // Compare feedback value
      SetOutput(4, GetOutput(4)+5);  // Increase speed
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections/outputs

Last update: **2025-06-11 19:30**

SDI-12

SDI-12 is a communications protocol for intelligent sensors that typically are used to monitor environment data. Devices and sensors are available from multiple manufacturers.

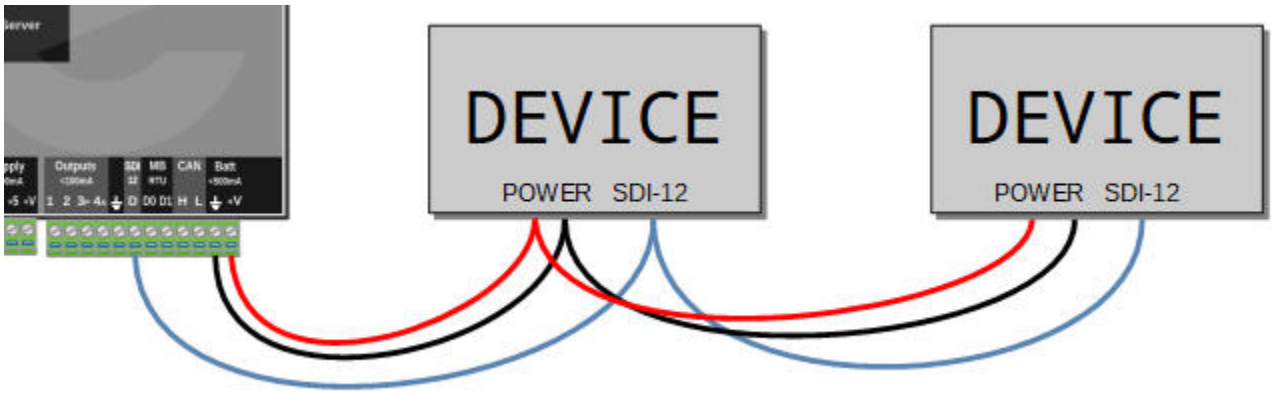
For detailed information about SDI-12, please refer to http://www.sdi-12.org/

The bus is multidrop, meaning multiple devices can be connected to the same data wire, and because each device is configured with a unique address, the ezeio can individually read data from each device.

The ezeio hardware port conforms to the SDI-12 v1.3 standard, but can also be configured for other purposes via software. The port can be used for example with a GPS receiver to monitor NMEA location messages, or to read certain proprietary protocols. When used in other modes, the standard SDI-12 functionality is not available.

Wiring SDI-12 devices

The SDI-12 communications uses a single wire for data going both to and from the device. The devices are usually designed to use very little power, and can be powered directly from the ezeio terminals. Up to nine (9) devices can be connected to the same bus.



We do not recommend using SDI-12 with more than about 60m (200ft) total bus length, although with few devices and good quality wire, longer buses are possible. Please refer to the SDI-12 specification for further details.

Powering SDI-12 devices

Most SDI-12 devices are designed to run on 12VDC. The ezeio can run on other voltages, so please make sure you use an appropriate power source if you are connecting SDI-12 devices.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/connections/sdi12

Last update: **2021-09-24 22:06**

Driver # not found

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/driverdetail

Last update: **2024-05-07 23:25**

Drivers

The “Device” page is where most links to hardware and data sources are created. This is done through the addition of drivers. Drivers are application programs that (in most cases) establish a communication link to hardware and provide an eze.io interface. This is similar to drivers, programs and apps that you would load on your phone, tablet or personal computer. For some such as “GPS receiver GN-803G” or the Expansion CAN port activation“ adding and saving changes is all that is required. Other driver's settings will range in complexity based on the requirements of the device or the level of customization provided by the driver.

Adding a device

To add a device, please follow these steps:

1. (From the Configure screen) select the “Devices” tab
2. At the bottom of the left panel click the Add Device button (This will bring up a list of available device drivers)
3. Select the appropriate driver and click Add Driver

Most drivers have some settings that needs to be properly set before the driver can be functional. For example, all Modbus RTU devices require a device address to be set.

Make your selections for the new driver and click Save Changes.

Finally, enable your new driver by checking the Active box, and then save your changes again.

The driver will now start working, and you should see values populate in the “Available register” list.

Select the register values that you want to show as fields by checking the corresponding checkboxes, and click Add selected to Fields. This will create new Fields, complete with “Data expression” and some default settings.

The following packages are available:

SKU	Package name	Purpose
	Basic services	Included with basic service subscription
SU2200	I/O expansion	I/O expansion devices
SU2201	Meters	Energy and flow meters
SU2202	Sensors	Sensors
SU2203	Temperature control	Temperature control applications, thermostats
SU2204	Light duty electrical	Light duty electrical control applications
SU2205	Heavy duty electrical	Heavy duty electrical control applications
SU2206	Engine control	Engine control applications

Detailed content of each package

Basic services

#	Name	Class	Manufacturer	Type	Product	Model
328	CATTRON CP750E/CP1000	MODBUS	Cattron	Engine control panel	CANplus	CP750-E/CP1000
97	CCS WattNode	MODBUS	CCS Continental Control System	Energy Meter	WattNode	Basic
3	CCS WattNode [OBSOLETE]	MODBUS	CCS Continental Control Systems	Energy Meter	WattNode	modbus
323	CDEBYTE MA01 Relay/IO	MODBUS	CDEBYTE	I/O Expander	2 RE, 2 AI, 2 DI	MA01-AACX2220
264	ebm-pabst fan 200 lite	MODBUS	ebm-pabst	Fan		84/112/150/200 lite
6	ezeio Generic Modbus Driver, 15 reg	MODBUS	eze System	generic	generic	test only
64	ezeio Modbus Port Setup	MODBUS	eze System	Modbus setting	ezeio mkII	
126	ezeio Fields via LAN	MODBUS	eze System	general purpose	ezeio	90 fields
168	ezeio 8I8O Expansion Module, Modbus	MODBUS	eze System	I/O Expander	ezeio I/O expander	8in, 8 dout, 2 aout
2	ezeio GPS receiver GN-803G	SDI12	eze System	GPS	GPS receiver	GN-803G
280	Wiegand keypad	SDI12	eze System	Wiegand keypad interface		
65	ezeio Expansion CAN port activation	CAN	eze System	ezeio I/O expander	ezeio mkII	

#	Name	Class	Manufacturer	Type	Product	Model
1	ezeio Standard I/O for ezeio mkII	CORE	eze System	Controller	ezeio mkII	mkII standard
4	ezeio Advanced I/O for ezeio mkII [OBSOLETE]	CORE	eze System	Controller	ezeio mkII	mkII standard
62	ezeio Discrete input	CORE	eze System	Discrete sensor	ezeio mkII	
68	ezeio Discrete output	CORE	eze System	Discrete output	ezeio mkII	
69	ezeio system	CORE	eze System	System info	ezeio	
105	ezeio Pulse input	CORE	eze System	Discrete input	Flow/energy meter	
122	ezeio Weather Forecast	CORE	eze System	Weather		
143	ezeio Runtime_Service	CORE	eze System	Runtime & Service	logic	
145	ezeio Thermistor	CORE	eze System	Thermistor	Resistive temperature probe	
156	ezeio MicroBMS logic	CORE	eze System	MicroBMS for small building		
177	ezeio PID control	CORE	eze System	PID control		
188	ezeio Lighting Control Logic	CORE	eze System	Lighting Control Logic	Logic	
205	General purpose register list	CORE	eze System	Register list		
220	DMX512 8 channel output	CORE	eze System	DMX Light control output		
238	Accumulator difference	CORE	eze System	Logic	Accumulator difference	
258	FormC alarm contact	CORE	eze System	Discrete alarm contact		
279	Gate/access door control	CORE	eze System	Gate control logic	Gate control	
296	ezeio Runtime tracker	CORE	eze System	Runtime tracking		
327	Thermostat logic	CORE	eze System	Thermostat Air Handler logic		
351	ezeio Pump Visualizer	CORE	eze System	Pump Visualizer		
24	ezeio Core drivers	CATEGORY	eze System	Core	ezeio	
71	Globalsat MR350S4 GPS	SDI12	Globalsat	GPS	MR350-S4	MR350-S4
246	Powercombo	MODBUS	Lithion	Battery Storage	Gridbox	10GB-100

#	Name	Class	Manufacturer	Type	Product	Model
226	Shelly relay output	MODBUS	Shelly	Relay output		
255	Shelly Pro EnergyMeter	MODBUS	Shelly	Energy meter	Pro 3EM	Pro 3EM
129	TopGNSS GPS RS232	SDI12	TopGNSS	GPS	GPS receiver	GN-8603G
269	TopGNSS Mushroom	SDI12	TOPGNSS	GPS		Mushroom
190	TZone THT-02/03 RH/Temp sensor	MODBUS	TZone	RH/Temp sensor	THT-02/03	
334	YJLink 4NTC expander	MODBUS	YJ-Link	I/O expander	Modbus NTC x 4	485-4NTC
11	RH/Temp sensor, Stick	MODBUS	ZGCJ	RH/Temp sensor	Stick sensor	ZGCJ

SU2200

Drivers I/O

#	Name	Class	Manufacturer	Type	Product	Model
342	NT57B08 Thermistor IO Expander	MODBUS	Eletechsup	IO Expander	NT57B08	NT57B08
31	ezeio Expander I/O	CORE	eze System	I/O Expander	ezeio I/O expander	8in, 8 dout, 2 aout
277	Generac Alarm Contact Board	CORE	Generac	Genset alarm contacts	NOC module	OK4929
295	ICP-DAS tm-TH8 Input expander	MODBUS	ICP-DAS	Input expander	tm-TH8	
94	Intech 2400-A16	MODBUS	IntechMicro	I/O expander	2400-A16	
95	Intech 2400-R2	MODBUS	IntechMicro	Relay module	2400-R2	
96	Intech 2400-M	MODBUS	IntechMicro	Input MUX expander	2400-M	
15	Procon Promux RO4	MODBUS	Procon	I/O Expander	Promux RO4	RO4
16	Procon Promux 16DO	MODBUS	Procon	I/O Expander	Promux 16DO	16DO
18	Procon Promux 8AI/I	MODBUS	Procon	I/O Expander	Promux 8AI/I	8AI/I
19	Procon Promux 8AI/V	MODBUS	Procon	I/O Expander	Promux 8AI/V	8AI/V
22	Procon Promux 8AO	MODBUS	Procon	I/O Expander	Promux PM8xO	PM8xO
20	Procon Promux 8TC	MODBUS	Procon	I/O Expander	Promux 8TC	8TC
21	Procon Promux 6RTD	MODBUS	Procon	I/O Expander	Promux 6RTD	PM6RTD
132	Procon Promux 16 DI	MODBUS	Procon	I/O Expander	Promux 16DI	16DI
133	Procon Promux 8DIO	MODBUS	Procon	I/O Expander	Promux 8DIO	8DIO
23	Procon Promux I/O expansion devices	CATEGORY	Procon	I/O Expander	Promux series	
26	Produal MIO 12-PT	MODBUS	Produal	I/O expander	MIO I/O module	MIP 12-PT
260	RVA LS9 alarm contact monitor	CORE	RVA	Tower light controller	GreenBOX	LS9

#	Name	Class	Manufacturer	Type	Product	Model
77	Temco T3-4AO module	MODBUS	Temco Controls	I/O expander	T3-4AO	T3-4AO
329	Temco I/O expander T3E-886	MODBUS	Temco Controls	I/O expander		T3E-886
267	Waveshare Analog Input 8CH expander	MODBUS	Waveshare	I/O expander	Analog Input 8CH	
268	Waveshare Analog Output 8CH expander	MODBUS	Waveshare	I/O expander	Analog Output 8CH	
272	Waveshare Digital I/O	MODBUS	Waveshare	I/O expander	8 digital in, 8 digital out	
299	WP9038 I/O expander	MODBUS	Wellpro	I/O expander	6AI 4DI 4DO	WP9038ADAM
326	WP8024 I/O expander	MODBUS	Wellpro	I/O expander	4RE 8DI	WP8024ADAM
300	SEEKU analog I/O expander	MODBUS	Winsun SEEKU	I/O expander	WSM01-1 8AIN/8AOUT	WSM01-1
301	SEEKU analog 16-Input expander	MODBUS	Winsun SEEKU	I/O expander	WSM01-2 16AIN	WSM01-2
302	SEEKU 8 PT100 expander	MODBUS	Winsun SEEKU	I/O expander	WSM02-2 8 PT100 inputs	WSM02-2

SU2201

Drivers Meters

#	Name	Class	Manufacturer	Type	Product	Model
14	ABB energy meter B23/B24	MODBUS	ABB	Energy Meter	B23/B24	B23/B24
185	ABB TotalFlow NGC	MODBUS	ABB	Gas Chromatograph	Totalflow NGC 8206	
173	AccuEnergy ACUDC 240	MODBUS	AccuEnergy	DC Energy and Power meter	AcuDC	AcuDC 240
207	AccuEnergy AcyRev1300	MODBUS	AccuEnergy	Energy meter	AcuRev1300	131x
179	Bender PEM353	MODBUS	Bender	Energy Meter	Linetraxx	PEM353
324	DC Power Energy meter	MODBUS	Bourns	DC Power meter	SSD-100A-R	SSD-100A-R
47	Carlo Gavazzi EM34x	MODBUS	Carlo Gavazzi	Energy Meter	Series 300	EM3xx - ET3xx
158	Carlo Gavazzi EM2x	MODBUS	Carlo Gavazzi	Energy Meter	Series 20	EM21/EM24
98	CCS WattNode-Advanced	MODBUS	CCS Continental Control System	Energy Meter	WattNode	Advanced
124	CCS WattNode-Advanced with Demand	MODBUS	CCS Continental Control System	Energy Meter	WattNode	Advanced with Demand

#	Name	Class	Manufacturer	Type	Product	Model
3	CCS WattNode [OBSOLETE]	MODBUS	CCS Continental Control Systems	Energy Meter	WattNode	modbus
41	CCS WattNode (advanced) [OBSOLETE]	MODBUS	CCS Continental Control Systems	Energy Meter	WattNode	WND-M1-MB
147	Circutor EDMk-MC	MODBUS	Circutor	Energy Meter	EDMk-MC	
42	DENT PowerScout RTU	MODBUS	Dent Instruments	Energy Meter	PowerScout	3037/12HD/48HD
43	DENT PowerScout TCP	MODBUS	Dent Instruments	Energy Meter	PowerScout	3037/12HD/48HD
201	Eastron single phase meter	MODBUS	Eastron	Energy meter	Single phase meter	SDM230
210	Eastron SDM120	MODBUS	Eastron	Energy meter	SDM120	
32	Elkor WattsOnII energy meter	MODBUS	Elkor Technologies	Energy Meter	WattsOn II	-
286	Elkor MCM-V-Mx Voltage meter	MODBUS	Elkor Technologies	Voltage meter	MCM-V-Mx	
127	ezeio Demand monitor	CORE	eze System	Static Demand window	Demand window montor	
131	ezeio CAISO Energy Pricing	CORE	eze System	CA ISO API	5-minute real time pricing	
178	Fineco EM115	MODBUS	Fineco Electric	Energy Meter	Single phase energy meter	EM115-Mod
82	Grundfos pump CIM200	MODBUS	Grundfos	Pump	CIM/CIU 200	CIM200
161	IME Nemo Electrical Meter	MODBUS	IME	Energy / Power Meter	Nemo D4	NEMO
123	Kamstrup Multical 62 flow meter	MODBUS	Kamstrup	Water flow meter	Multical 62 Comm module	HC-003-67
312	Kempower T500P40C5BS	MODBUS	Kempower	T500		
306	Keyence Air Efficiency Module	MODBUS	Keyence	Compressed Air Efficiency Module	MP-F series	
308	Keyence Energy Monitor	MODBUS	Keyence	Compressed Air Efficiency Module Energy Monitor	MP-F series	MP-FEA1
139	Lanry SC7 Water meter	MODBUS	Lanry Instruments	Ultrasonic Water meter	SC7	
46	LongrunCN LRF-2000M	MODBUS	LongrunCN	Flow meter, Ultrasonic	LRF-2000M	LRF-2000M

#	Name	Class	Manufacturer	Type	Product	Model
121	Measurlogic meter	MODBUS	Measurlogic	Energy Meter	DTS/SKT family	3xx,SKT
204	IT-DIN isolation monitor	MODBUS	Megacon	Isolation monitor	IT-DIN	
266	PZEM DC meter	MODBUS	Peacefair	Energy meter	PZEM DC	PZEM-017
273	PZEM AC meter	MODBUS	Peacefair	Energy meter	PZEM AC	PZEM-014/016
319	Schneider iEM3x50 energy meter	MODBUS	Schneider Electric	Energy meter	iEM3150 series	iEM3150/3250/3350
336	Schneider PM51xx energy meter	MODBUS	Schneider Electric	Energy meter	PM5110 / PM5111	PM511x
191	Siemens 7KT PAC1600	MODBUS	Siemens	Energy Meter	7KT PAC1600 energy meter	7KT1651, 1652, 1661, 1662, 1665, 1666
192	Siemens SENTRON POC1000	MODBUS	Siemens	SENTRON Power monitor receiver	SENTRON 7KN1110 POC1000	7KN1110 POC1000
193	Siemens SENTRON 3NA COM fuse	MODBUS	Siemens	SENTRON Fuse	SENTRON 3NA fuse	3NA COM
194	Siemens SENTRON 5SL6 Circuit breaker	MODBUS	Siemens	SENTRON Circuit breaker	SENTRON 5SL6 Circuit breaker	5SL6 COM
195	Siemens SENTRON 5SV6 Arc fault detector	MODBUS	Siemens	SENTRON Arc fault detector	SENTRON 5SLV6 Arc fault detector	5SLV6 COM
196	Siemens SENTRON 5ST3 aux switch	MODBUS	Siemens	SENTRON Aux switch	SENTRON 5ST3	5ST3 COM
200	Siemens PAC2200 energy meter	MODBUS	Siemens	Energy meter	PAC2200	PAC2200
249	Siemens SENTRON 5ST3 RCA	MODBUS	Siemens	SENTRON Remote Control Auxiliary	SENTRON 5ST3 RCA	5ST3 RCA
254	Siemens PAC4200 energy meter	MODBUS	Siemens	Energy meter	PAC4200	PAC4200
275	Siemens SENTRON ECPD 5TY1	MODBUS	Siemens	SENTRON ECPD	SENTRON 5TY1 ECPD/Circuit breaker	5TY1 COM
114	Socomec U-10	MODBUS	Socomec	Voltage meter	DIRIS Digiware	U-10
115	Socomec I-30	MODBUS	Socomec	Current meter	DIRIS Digiware	I-30
116	Socomec I-60	MODBUS	Socomec	Current meter	DIRIS Digiware	I-60
256	Socomec DIRIS MCM	MODBUS	Socomec	Energy meter	DIRIS MCM	12, 24, 48
282	Socomec Diris A40 / A-30 / A-20 / A-10	MODBUS	Socomec	Energy meter	Diris A40 / A-30 / A-20 / A-10	

#	Name	Class	Manufacturer	Type	Product	Model
284	Socomec DIRIS A-40	MODBUS	Socomec	Energy meter	DIRIS A-40	
229	TELE eCap Power Meter	MODBUS	TELE Haase	Energy meter	TELE eCap Power Sensor	eCap G4SR480V5A02CAA
197	WAGO Energymeter with push-in clamps	MODBUS	WAGO / KDK	Energy Meter	Meter with Push-in-CAGE CLAMP	879-30x0

SU2202

Drivers Sensors

#	Name	Class	Manufacturer	Type	Product	Model
152	Apogee SP-522	MODBUS	Apogee	Pyranometer	Thermopile Pyranometer	SP-522
10	Bewis Compass & Accelerometer	MODBUS	Bewis	Accelerometer	SEC345	SEC345
138	Campbell Scientific CS320	SDI12	Campbell Scientific	Pyranometer	Thermopile Pyranometer	CS320
154	CS Instruments VA520	MODBUS	CS Instruments	Flow sensor	VA520	
7	Davis Weather Station	SDI12	Davis Instruments	Weather Station	Vantage Pro	Pro2
289	Ultrasonic distance sensor 25-500cm	MODBUS	DYP	Ultrasonic distance sensor	DYP-A12	
39	E+E EE210 RH/Temp sensor	MODBUS	E+E Elektronik Messgeräte	RH/Temp sensor	EE210	EE210
40	E+E EE610 Differential Pressure Sensor	CORE	E+E Elektronik Messgeräte	Differential pressure sensor	EE610	EE610
330	PT100-Modbus PTA9B01	MODBUS	Eletechsup	PT100 temperature sensor transmitter		PTA9B01
150	ezeio Cabinet/door latch control	CORE	eze System	Latch control	Logic module	
257	Air science	CORE	eze System	Air analysis driver		
304	Halogen MP5-A Multiparameter sensor	MODBUS	Halogen Systems	pH / Chlorine sensor	MP5-A multiparameter	MP5-A
169	Hukseflux SR05-Dxxx	MODBUS	Hukseflux	Pyranometer	SR05	SR05-D1A3
265	ChemScan MPX4	MODBUS	In-Situ	Water Quality multi probe	ChemScan MPX4	MPX4

#	Name	Class	Manufacturer	Type	Product	Model
222	Concrete Sensor, Invisense	CORE	Invisense AB	Concrete sensor		
128	Linovision IOT Ambient noise meter	MODBUS	Linovision	Noise meter	IOT-S300NOIS	
146	Linovision IOT S300AQ Air Quality Sensor	MODBUS	Linovision	Air quality sensor	IOT-S300AQ	IOT-S300AQ
134	METER Atmos 22 Anemometer	SDI12	METER Group Inc.	Anemometer	Atmos 22 Ultrasonic	Atmos 22
292	METER Teros 11/12	SDI12	METER Group Inc.	Soil moisture and temperature sensor	TEROS 11/12	
180	Monnit Modbus Gateway	MODBUS	Monnit	Wireless sensor network	Serial Modbus Gateway	ALTA
181	Monnit Temp/RH	MODBUS	Monnit	Wireless RH/Temperature sensor	Relative Humidity sensor	ALTA
182	Monnit DoorSwitch	MODBUS	Monnit	Wireless Door switch	Door sensor	ALTA
242	Monnit Temperature sensor	MODBUS	Monnit	Wireless temperature sensor	Temperature sensor	ALTA
352	Monnit Pulse counter	MODBUS	Monnit	Wireless pulse counter	Pulse counter	ALTA
183	Monnit wireless system	CATEGORY	Monnit	Gateway (Serial Modbus)		
172	Ocean Vision IS-MOD-0100	MODBUS	Ocean Vision	Gas solenoid	i-Submerge	IS-MOD-0100
228	Modbus Injector	MODBUS	OceanVision	Modbus injector		IS-MOD-0200
227	pH Sensor	MODBUS	OceanVision / Nengshi	pH sensor	Digital pH sensor	IS-MOD-0150 / ASPS21xx
206	Omega SP-016 Heat Flux Probe	MODBUS	Omega	Heat Flux Probe	SP-016	
174	RH/Temp sensor	MODBUS	Open Source Hardware	RH/Temp sensor	HTB1/SHT20	
12	Produal PEL-M pressure sensor	MODBUS	Produal	Pressure sensor	PEL	PEL-M
285	Differential pressure sensor QDF70B	MODBUS	Qidian Automation	Differential pressure sensor	QDF70B	
36	Senix ToughSonic 14	MODBUS	Senix	Ultrasonic distance meter	ToughSonic 14	TSPC-N1S1-485A
203	Sensecap pH sensor	MODBUS	Sensecap	pH sensor	S-pH-01	version 2
8	Sentek Drill & Drop	SDI12	Sentek	Soil sensor	Drill & Drop	Series III, 48"

#	Name	Class	Manufacturer	Type	Product	Model
290	SignalFire Gateway v2	MODBUS	SignalFire	Wireless Gateway	DIN Gateway V2	
291	SignalFire Pressure Scout	MODBUS	SignalFire	Pressure sensor	Pressure Scout	
309	SignalFire Sentinel HART	MODBUS	SignalFire	HART interface	Sentinel HART	
310	SignalFire Sentinel Tilt	MODBUS	SignalFire	Inclinometer	Tilt Scout	Scout-TILT-3AABIS
311	SignalFire Sentinel Digital	MODBUS	SignalFire	Digital pulse input	Sentinel Node Digital	Sentinel-DI-XXXX
313	SignalFire Remote Start	MODBUS	SignalFire	Remote start controller	RSD stick	MBS-RSD-XX
287	Noise level sensor	MODBUS	Taida	Noise level sensor, Modbus	30-130dB	
92	Temco PM 2.5 sensor	MODBUS	Temco Controls	Air quality sensor	PM 2.5	
79	Terabee IND-TOF-1 distance sensor	MODBUS	Terabee	Distance sensor	Time of flight sensor	IND-TOF-1
142	TSI AEROTRAK	MODBUS	TSI	Air quality sensor	AEROTRAK	72xx,73xx,75xx
230	TurtleTough DSS Controller - ORP sensor	MODBUS	TurtleTough	ORP sensor	DSS Smart ORP	
231	TurtleTough DSS Controller - pH sensor	MODBUS	TurtleTough	pH sensor	DSS Smart pH	
232	TurtleTough DSS Controller - Wide ORP sensor	MODBUS	TurtleTough	Wide ORP sensor	DSS Smart Wide ORP	
233	TurtleTough DSS Controller - DO sensor	MODBUS	TurtleTough	Dissolved Oxygen sensor	DSS Smart DO	
234	TurtleTough DSS Controller - Ion sensor	MODBUS	TurtleTough	Ion sensor	DSS Smart Ion	
235	TurtleTough DSS Controller - Conductivity sensor	MODBUS	TurtleTough	Conductivity sensor	DSS Smart Conductivity	
45	Vaisala HMP60/110	MODBUS	Vaisala	RH/Temp sensor	Humidity and Temp probe	HMP60/110
274	WitMotion Vibration sensor	MODBUS	Wit-Motion	Vibration sensor	WTVB01-485 / WTVB02-485	
325	RH/Temp sensor	MODBUS	XY	Indoor RH/Temp sensor	XY-MD02	XY-MD02
38	YDHT RH/Temp sensor, Probe	MODBUS	YDHT	RH/Temp sensor	Waterproof sensor	YDHT-06-C

#	Name	Class	Manufacturer	Type	Product	Model
49	YSI EXO 599825 Sonde	MODBUS	YSI	Water Quality Sensor	EXO Modbus Adapter	599825
11	RH/Temp sensor, Stick	MODBUS	ZGCJ	RH/Temp sensor	Stick sensor	ZGCJ

SU2203

Drivers Temperature control

#	Name	Class	Manufacturer	Type	Product	Model
208	Bes-Tech DigiRTU	MODBUS	Bes-Tech	HVAC controller	Digi-RTU	
100	CAREL EVD Twin	MODBUS	CAREL	HVAC Controls	Expansion Valve Driver	EVD evolution twin
321	SelfCheck-Boiler	CORE	eze System	Burner control SelfCheck integration	SelfCheck	
322	Thermostat T32P	MODBUS	JacksonSystems	Thermostat	T32P/Chameleon	T32P
198	Marvair Commstat4	MODBUS	Marvair	Air conditioning unit	Commstat4	
119	Shimaden SR91	MODBUS	Shimaden	Temperature controller	Temperature controller	SR91
125	Shimaden SR23	MODBUS	Shimaden	Temperature controller	Temp/humidity controller	SR23
288	Siemens RWF55 Temp/Pressure controller	MODBUS	Siemens	Temperature and pressure controller	RWF55.5	
263	Siemens LMV5 BMS	MODBUS	Siemens Combustion Controls	Burner Management System	LMV5	

SU2204

Drivers Light duty electrical

#	Name	Class	Manufacturer	Type	Product	Model
89	ABB ACS550 drive	MODBUS	ABB	VFD	ACS550	ACS550
141	ABB XT5	MODBUS	ABB	Circuit breaker	SACE XT5	XT5
332	ABB ACS880 VFD	MODBUS	ABB	VFD	ACS880	ACS880
343	ABB ACS580 VFD	MODBUS	ABB	VFD	ACS580	ACS580
241	Airmaster S1	MODBUS	Airmaster	Compressor controller	Airmaster S1	S1
281	Airmaster SCCM	MODBUS	Airmaster	Compressor controller	Airmaster	SCCM / FORM

#	Name	Class	Manufacturer	Type	Product	Model
297	ASCO Series 300 Transfer Switch	MODBUS	ASCO / Schneider	Automatic Transfer Switch ATS	Series 300 Group G	
58	CAREL EVD evo twin (Bluon)	MODBUS	CAREL	HVAC Controls	Expansion Valve Driver	EVD evolution twin
60	CAREL EVD evo	MODBUS	CAREL	HVAC Controls	Expansion valve driver	EVD evolution
113	Danfoss FC202	MODBUS	Danfoss	VFD	VLT® AQUA Drive	FC202
341	Danfoss FC202 VFD	MODBUS	Danfoss	VFD	VLT® AQUA Drive	FC202
199	Eaton PowerXpert UPS	MODBUS	Eaton	UPS	PowerXpert	9PX 1000G interface
243	Smartpack S Charge controller	MODBUS	Eltek	Charge controller	Smartpack S	Smartpack S
93	Epever Tracer MPPT controller	MODBUS	EP Solar Technology Co.Ltd.	MPPT charge controller	EPEVER Tracer	Tracer-AN
17	Fuji Frenic ECO VFD	MODBUS	Fuji	VFD	Frenic	ECO
34	Fuji Frenic HVAC	MODBUS	Fuji	VFD	Frenic	HVAC
59	Fuji Frenic Mini VFD	MODBUS	Fuji	VFD	Frenic	Mini
202	Generac ATS X	MODBUS	Generac	Automatic Transfer Switch	TX series	TX611
262	Implenia Tunnel Fan Control	MODBUS	Holte	Fan control		
283	Fan Master/Slave Control	MODBUS	Holte	Multi Fan Control	Supervisor driver	
338	Holte Tunnel Fan Control logic	CORE	Holte	Fan VFD abstraction logic	Generic VFD	
186	KorePower KP-MC Battery Controller	MODBUS	KorePower	Battery management controller	KorePower MP-MC	KP-MC
117	Morningstar Tristar-MPPT	MODBUS	Morningstar	Battery charger	TristarMPPT	TSS-MPPT-30/45/60
261	Narada 48NPFC200	MODBUS	Narada	Backup battery	NPFC/MPLhE Series	48NPFC200
84	Outback Charge Controller	MODBUS	Outback Power	Charge Controller	FLEXmax	FLEXmax 60/80
85	Outback inverter	MODBUS	Outback Power	Inverter	Radian Series	
86	Outback FLEXnet-DC	MODBUS	Outback Power	System controller	FLEXnet-DC	
83	Outback AXS	MODBUS	OutbackPower	SunSpec Modbus Interface	AXS Port	

#	Name	Class	Manufacturer	Type	Product	Model
213	PCE LCWB Charger	MODBUS	PCE	Charger		EV11
149	WaterSam	MODBUS	WaterSam	Water sampler	WS Porti Mobile Sampler	
333	WWEC FC VFD	MODBUS	Wold Wide Electric	VFD	FlexControl	FC

SU2205

Drivers Heavy duty electrical

#	Name	Class	Manufacturer	Type	Product	Model

SU2206

Engine control

#	Name	Class	Manufacturer	Type	Product	Model
347	CATTRON CP750E/CP1000 REGEN	MODBUS	Cattron	Engine control panel REGEN	CANplus	CP750-E/CP1000
214	Controls Inc Panel (D)	MODBUS	Controls Inc	Engine control panel		
215	Controls Inc Panel (D) I/O	MODBUS	Controls Inc	Engine control panel Add-on		
216	Controls Inc Panel (D) Pump Config	MODBUS	Controls Inc	Engine control panel Add-on		
217	Controls Inc Panel (D) Safety	MODBUS	Controls Inc	Engine control panel Add-on		
218	Controls Inc Panel (D) Emission Cfg	MODBUS	Controls Inc	Engine control panel Add-on		
276	Controls Inc Panel Regen	MODBUS	Controls Inc	Engine control panel Add-on		
33	Controls Inc. Engine Control Panel	MODBUS	Controls Inc.	Engine Controller	XL-Series	
78	Controls Inc. Engine Control Panel, IO-ports	MODBUS	Controls Inc.	Engine Controller - I/O ports	C-Series/XL-Series	
303	Controls Inc. Engine Control Panel	MODBUS	Controls Inc.	Engine Controller	C-Series	

#	Name	Class	Manufacturer	Type	Product	Model
298	DeepSea Engine Controller General	MODBUS	Deep Sea Electronics	Engine Controller	DSE7410 MKII	GenComm
63	Enovation PowerCore MPC-10 Engine Controller	MODBUS	Enovation Controls / Murphy	Engine Controller	PowerCore MPC-10/TEC-10	MPC-10/TEC-10
67	Enovation PowerCore MPC-20 Engine Controller	MODBUS	Enovation Controls / Murphy	Engine Controller	PowerCore MPC-20	MPC-20
151	ezeio Generic Genset monitoring	CORE	eze System	Generic Genset	Logic driver, Discrete sensors	
211	Generac H-100 Genset Control Panel	MODBUS	Generac	Engine controller	H-100	
250	Generac PowerZone PRO	MODBUS	Generac	Engine controller	PowerZone PRO	PowerZone PRO
112	KOHLER-SDMO APM303	MODBUS	Kohler-SDMO	Engine Controller	Genset panel	APM303

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/start

Last update: **2025-01-29 16:52**

Accumulator Difference driver

Description

This driver monitors an accumulating field value, for example an energy (kWh) or volume (L) register that continuously counts up. The driver captures the field value at a given interval (hourly, daily etc..) and computes the difference between the start and end of the interval. Current value and the four previous are provided.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: the asset being monitored, the purpose of monitoring, etc.

eze System Notes - In this space we provide a description of the driver's function.

Accumulator Field - Use the drop-down menu to select the Field to be monitored.

Interval - Use the drop-down menu to select the differential's interval. Options are; Hourly, Daily, Weekly, Monthly or Yearly.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

Reg #	Reg Name	Description
1	**Accumulator value**	Current value in selected Field
2	**Current interval partial diff**	Differential between value at beginning of interval and current value
3	**Previous interval diff**	Most recent interval differential captured. **Daily example:** Yesterday's differential
4	**2nd interval diff**	Second most recent interval differential captured
5	**3rd interval diff**	Third most recent interval differential captured
6	**4th interval diff**	Fourth most recent interval differential captured
7	**Value at start of current interval**	Speed of travel is given as meters per second
8	**Interval identifier**	Current interval step, out of a complete cycle (expressed as a number). See table below

---

Interval option	Description	Cycle	Example of number reference
Hourly	Begins at top of hour	0 to 23	0 = Midnight to 1 am
Daily	Begins at Midnight	1 to 31	1 = 1st day of Month
Weekly (Mon-Sun)	Begins Monday	1 to 52	1 = Monday
Weekly (Sun-Sat)	Begins Sunday	1 to 52	1 = Sunday
Monthly	Begins first of the Month	1 to 12	1 = January
Yearly	Begins January 1st	N/A	

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/accumulator_diff

Last update: **2024-05-22 21:30**

Apogee SP-522 driver

Description

The SP-522-SS is an upward-looking thermopile pyranometer with a digital signal using Modbus RTU protocol over RS-232 or RS-485. The sensor incorporates a blackbody thermopile detector and acrylic diffuser with a rugged, self-cleaning sensor housing design, and high-quality cable terminating in pre-tinned pigtail leads for easy connection to dataloggers and controllers.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Modbus address - Enter the unique Modbus address (device ID) of the sensor.

Heater ON below (C) - Enter the temperature below which the sensors heater will turn on.

Sensor temp (C) field number - Use the drop-down menu to select the temperature Field used by the sensor.

Dewpoint (C) field number - Use the drop-down menu to select the Dewpoint Field used by the sensor.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. **Solar Radiation** - Radiation value in Watts per square meter.
2. **Raw millivolt** - Raw electrical output signal in millivolts.
3. **Sensor temperature (external)** - External sensor reading used by sensor
4. **Heater** - State of heater (on/off)
5. **Dewpoint (external)** - External sensor reading used by sensor (selected in driver config)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/apogee_sp522

Last update: **2024-05-22 21:45**

METER ATMOS 22 driver

Description

This driver is design to allow communication with the sensor via SDI-12. This sensor provides air speed, direction and temperature readings.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

SDI-12 address - Enter Device I.D. (SDI-12 address). This address must be unique on the bus.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

1. **Wind speed** - Average of six wind speed readings (read every 10 seconds). Reported in meters per seconds.
2. **Wind direction** - Average of six wind direction readings (read every 10 seconds). Reported in degrees 0-359.
3. **Gust wind speed** - Maximum wind speed reading of the last six measurements (read every 10 seconds). Reported in meters per seconds.
4. **North Wind speed** -
5. **East Wind direction** -
6. **Air Temperature** - Average of six temperature readings (read every 10 seconds). Reported in Celsius
7. **Orientation X** - Instantaneous measurement in degrees of tilt in X axis. Reported in degrees
8. **Orientation Y** - Instantaneous measurement in degrees of tilt in Y axis.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/atmos22

Last update: **2024-06-27 21:54**

Bewis Compass & Accelerometer driver

Description

This driver is design to allow communication with the Bewis two axis accelerometer and compass via Modbus RTU.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Modbus address - Use the drop-down menu to select the acceptable level of accuracy. Precise location only, Normal precision or Accept low signal. If horizontal accuracy falls below the selected level of precision the driver status will indicate an “App error”.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. **Pitch** - Fore and aft degrees of tilt (Range +40 to -40)
2. **Roll** - Side to side degrees of tilt (Range +40 to -40)
3. **Heading** - Compass heading in degrees (0-359)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/bewis_accelerometer

Last update: **2024-05-23 15:22**

Campbell Scientific CS320 driver

Description

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

SDI-12 address - Enter the unique SDI-12 address (device ID) of the sensor.

Heater ON below (C) - Enter the temperature below which the sensor's heater will turn on.

Dewpoint (C) field number - Use the drop-down menu to select the Dewpoint Field used by the sensor.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

1. **Solar Radiation** - Radiation value in Watts per square meter.
2. **Raw millivolt** - Raw electrical output signal in millivolts.
3. **Sensor temperature** - Temperature reading (in Celsius) provided by the sensor
4. **Heater** - State of heater (on/off)
5. **X-axis value** - Onboard level sensor measurement
6. **Y-axis value** - Onboard level sensor measurement
7. **Z-axis value** - Onboard level sensor measurement
8. **Dewpoint (external)** - External sensor reading used by sensor (selected in driver config)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/cs320

Last update: **2024-05-23 15:31**

CS Instruments VA520 driver

Description

This driver is design to allow communication with the VA520 via the ezeio's Modbus RTU serial port.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Modbus address -

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

Flow - Registers 1 through 15 offer current flow in various volumes and weights per second, minute and hour. These include cubic meters, normal cubic meters, liters, normal liters, cubic feet, normal cubic feet, and kilograms.

Volume/Total - Registers 16 through 22 offer totalized volumes in cubic meters, normal cubic meters, liters, normal liters, cubic feet, normal cubic feet and also in kilograms.

Velocity - Registers 23 through 26 offer current velocity measurements in meters per second (m/s), normal meters per second (Nm/s), cubic feet per minute (cfm) and Normal cubic feet per minute (Ncfm).

Temperature - Registers 27 and 28 provide median temperature readings in Celsius and Fahrenheit.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/cs_va520

Last update: **2024-05-23 15:25**

Demand monitor

Description

This driver produces metrics based on user defined energy consumption within a specified demand (time) window and instantaneous power reading from a connected power/energy meter. The resulting values can be used to drive demand shift automation in order to stay under demand window limits.

Configuration settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Window size - Select the number of minutes in your demand window (5, 10, 15, 20, 30, 60)

Delta-T for trend -

Power field - Enter the Field number for the power (kW) reading

---

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Target Energy (Writable) - User defined max energy consumption for demand window
2. Predicted Energy - Predicted energy total based on accumulated energy and current power usage
3. Window energy - Current energy total for current window
4. Momentary power - Current power (kW) reading of user define Field
5. Max power allowed - Max sustained average power consumption allowable to stay within demand window limit.
6. Power margin - Head room between current power reading and “Max power allowed”.
7. Time until target exceeded - Predicted elapsed time when user define target energy total will be exceeded, based on accumulated energy and current power usage
8. Window time remaining - Seconds remaining in demand window
9. Time over power budget - Number of seconds over user defined max energy, in current window.
10. Percent of previous window - Percentage comparison of current consumption to previous demand window.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/demand_monitor

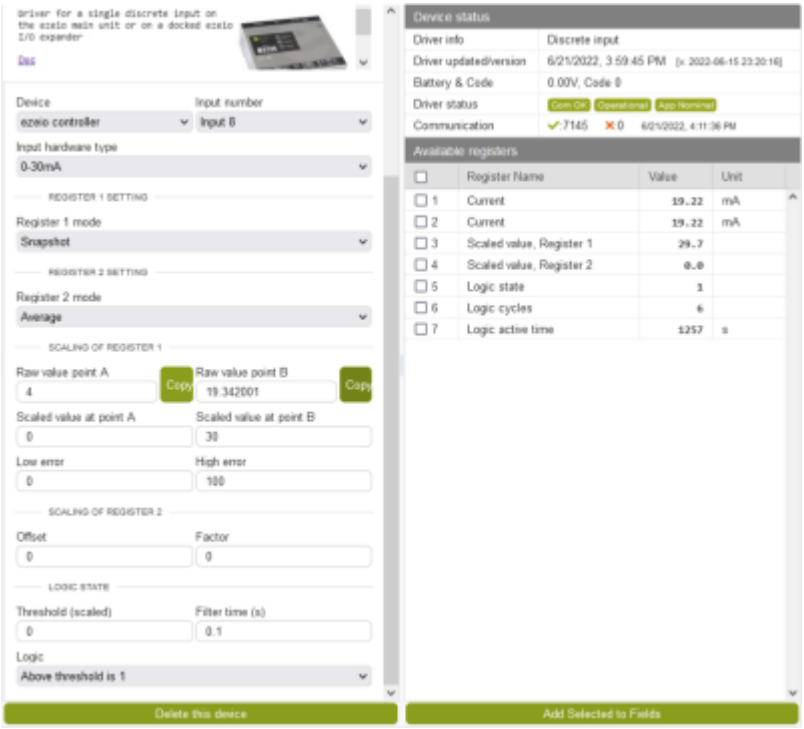
Last update: **2024-05-22 21:15**

Discrete Input driver

Configuration settings (center panel)

Description

This driver (located in the Core drivers folder) is designed for use with the ezeio MkII controller and the ezeio MkII I/O Expander. It is ideal for reading resistance (Ohms), and sensors with DC current (0-30 mA) or voltage (0-10 VDC) signals. *This driver does support pulse outputs and thermistors, specialized drivers are offered.* The settings on this driver map the source, configure circuitry on the ezeio, and produce signal readings, scaled values, a logic value and statuses. An instance of this driver can be added for each available input terminal on the controller or expanders. This method of individually configuring inputs allows the driver's name to reflect a unique sensor or device. Below are descriptions of the various settings.

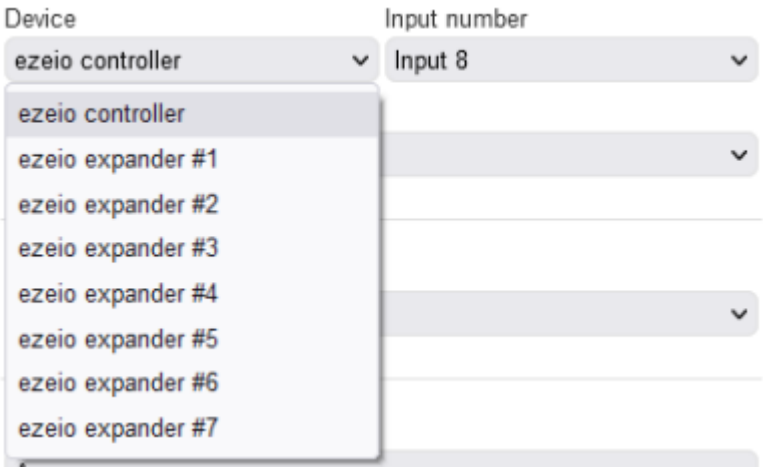


Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

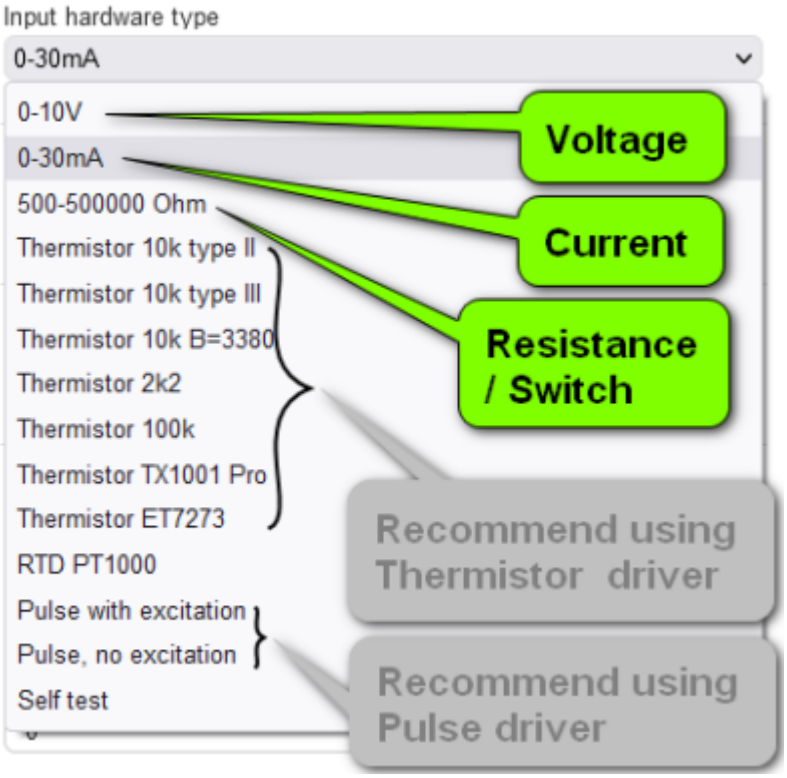
Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.



Device - Select the ezeio controller or the device number of a connected ezeio I/O Expander.

Input number - Select the input number (1-8) of the selected device.



Input hardware type - Select the type of sensor or signal type (0-10V, 0-30mA (4-20mA), Ohms, thermistor). Some of the thermistor choices are tailored to specific sensors and others are generic (see the table below for details). Options from pulse are shown, but we recommend using the “Pulse input” driver. *The Self test is diagnostic setting to test the circuitry of the input when nothing is connected*.

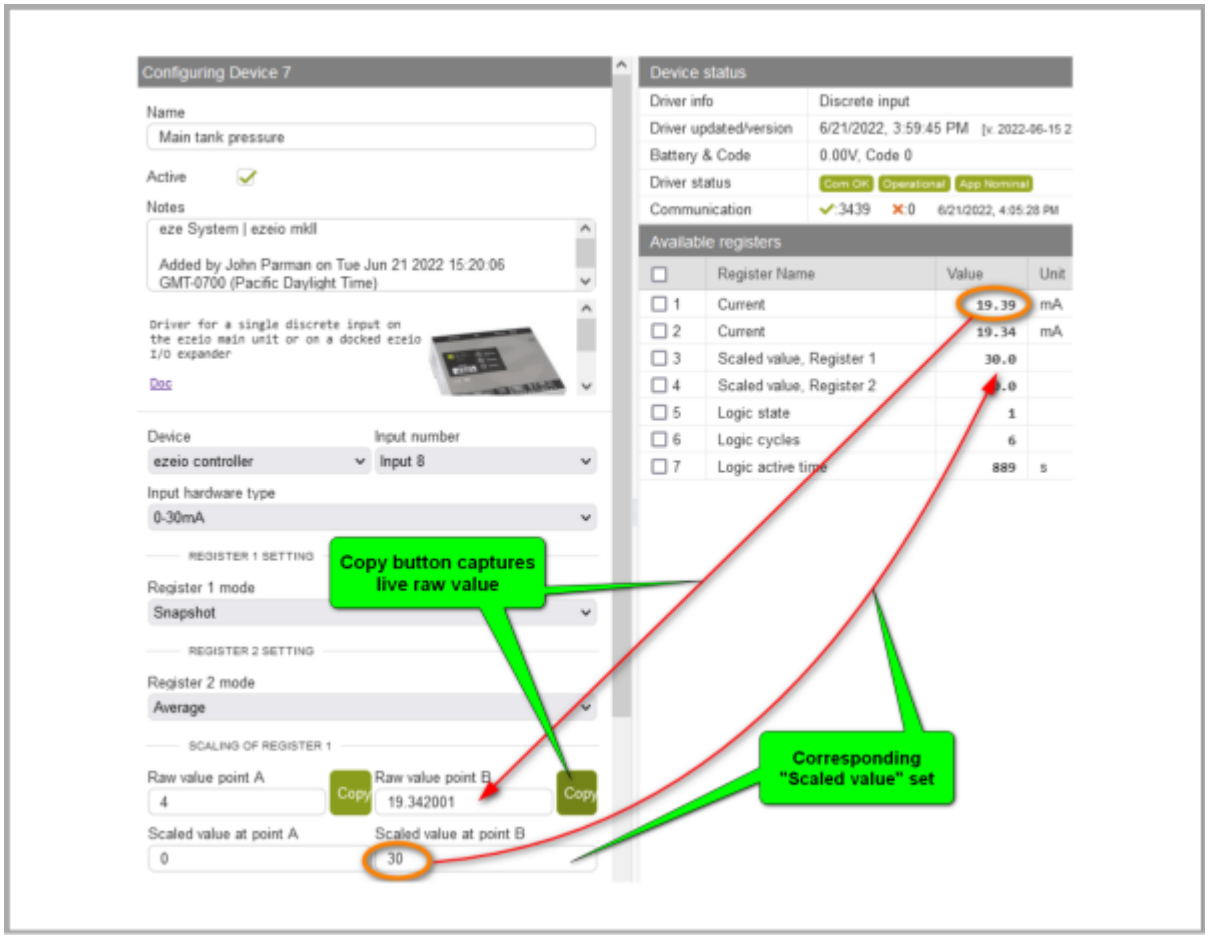
REGISTER 1 & 2 modes - Each discrete input is duplicated, providing the opportunity apply two modes of initial processing and two methods of scaling. The ezeio controller's microprocessor reads the onboard

inputs at a rate of 1000 times per second and exports the results every 1/10 of a second (*connected I/O Expander inputs are sampled slower as they are transmitted via CANbus*). Additional processing is applied to logged registers (see "Fields")

Register 1 & 2 modes - These modes control how the analog to digital converter calculates the live values. *Separate settings on the individual "Field" settings control how each channel is logged*. Choose one of the setting below for registers 1 & 2.

Mode	**Description**	**Effect**
Snapshot	Captured value is a single sample	99 samples are rejected
Minimum	Lowest of the 100 readings over sample interval	99 samples are rejected
Maximum	Highest of the 100 readings over sample interval	99 samples are rejected
Average	Average of the 100 readings over sample interval	All samples used to find avg.
Pulse rate	Rate is shown as frequency (Hz x 1000)	
Pulse count	Counting total number of pulse	
Pulse interval	milliseconds elapsed since last pulse	
Differential pulse count		

SCALING OF REGISTER 1 - Register 1 can be scaled to a unit value such as PSI or LPM with a 2 point linear conversion method. A rough conversion can be obtained by entering corresponding values from a table supplied by the sensor manufacture. For a calibrated conversion, use a trusted gauge or probe to provide readings at two points within the sensors range. Enter the value for one point and use the driver's copy function to capture the corresponding live raw reading. The result of this conversion is shown in register 3. In addition, a Low/High error range (in the scaled unit) can be entered, representing a normal or safe operating range for the application.



Raw value point A & B - By default, the range shown here is set based on the “Input hardware type” selected, such as 0-10V (0.0 - 10.0) or 0-30mA (4.0 - 20.0). This would correlate to the full scale of a typical sensor. These fields can be changed if required to match the output of your sensor or the scale range you wish to use in the next step.

Scaled value at point A & B - Enter values here that correlate to the Min/Max raw values. For example: 4.0mA = 0 psi and 20.0mA = 250 psi

Low error - High error - Enter values here that represent the normal range for the application. This could also represent the operating range of the sensor, so a negative value could indicate an open circuit. When the value is out of this range, the “Driver status” will indicate an “App error”. This error can be see on the Driver status, incorporated in expressions by use of ds(DVCSTAT_APPSTAT) and included in messages with the “Message Template Tag” [DVCSTAT#APPSTAT].

SCALING OF REGISTER 2 - This register's scaling utilizes a multiplying factor and an offset. Using the previous example of a 4-20mA output pressure transducer with a range of 0-250 psi, the offset would be 4 and the factor would be 15.625. The result of this scaling of register 2 is shown as register 4

Offset - Set a starting point or floor for the scaling such as 4 in the case of 4-20mA signals.

Factor - Multiply the raw value to create a linear conversion that matches your sensors scale. To find the factor, divide the sensor full scale range by the number of raw unit steps.

LOGIC STATE - This feature converts any ranging analog value into a digital value (or boolean) by

defining a threshold beyond which it is consider true (represented by 1, or 100).

Threshold (Scaled) - Enter a threshold value above or below which the state should be considered true (or ON).

Filter time - Filter out transient value changes by entering a number of seconds (and tenths) in this field. The value must then exceed the threshold for the proscribed time before it is considered true.

Logic - Set the true (or ON) state to be above or below the threshold value, with a value of 1 or 100. Choosing 100, make it easy to use the value to drive a digital output via “Fields”.

Thermistor options (use Thermistor driver for more options and features)

For thermistors, more options and features are available in the Thermistor driver

Thermistor Name	Source or Generic	Part #	Application	Range
10k type II	Generic	–		
10k type III	Generic	–		
10k type B=3380	eze System	BA0010		
2k2	eze System	BA0022		
100k	Generic	–		
TX1001 Pro	ThermoWorks	TX-1001X-OP		
ET7273		ET7273		

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

1	Raw electrical value	Unit type is set by “Input hardware type”, V, mA, Ohm
2	Raw electrical value	Unit type is set by “Input hardware type”, V, mA, Ohm
3	Scaled value, Register 1	Scaled by 2 point calibration
4	Scaled value, Register 2	Scaled by offset and factor
5	Logical state	Analog value converted to digital (on/off) based on threshold setting
6	Logical cycles	Number of times “Logical state” changes from off to on
7	Logical active time	Number of seconds logical state is “on” (runtime)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/discreteinput

Last update: **2024-05-09 15:30**

Discrete Output driver

Description

This driver (located in the Core drivers folder) is designed for use with the ezeio MkII controller and the ezeio MkII I/O Expander.

Configuration Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Doc (link)

Device - Select the ezeio controller or the address number of a connected ezeio I/O Expander.

Output number - Select the Output number of the selected device. Outputs 1 or 2 for the ezeio controller or outputs 1 - 8 for an I/O Expander.

On startup - When using the ezeio for control or automation you should consider the effects of a lost of power and the effects of the restart. This setting gives you three options for controlling an outputs state on startup of the ezeio, turn ON, turn OFF or return to the previous state.

Auto off - Enter the number of seconds an output should remain on, once triggered. Enter a zero if you would like the output to remain on until directed by automation logic or manually turned off.

CONDITIONAL SETTINGS (OPTIONAL)

This feature provides exclusive or additional control logic, by means of a Field value threshold. The “Control mode” options listed below determine the role (if any) the Field value plays in the output control logic.

Control mode	Driver Logic	Other Logic
No control condition	No effect on output	Complete control of output

Condition triggers the output to turn on	True condition will turn on output	Auto off and other logic may also control output
Condition triggers the output to turn on/off	True condition will turn on output & False condition will turn off	Auto off and other logic may also control output
Condition exclusively controls output state (No auto off)	True condition will turn on output & False condition will turn off	No effect on output
Condition must be true to turn output on	False condition will inhibit other logic from turning the output on	Primary control of output

Condition Field - Select Field (from list) to drive condition.

Logic - Select Greater than (above threshold) or Less than (below threshold).

(Logic) Value - Enter the threshold value for the condition.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Output state - On/Off state of relay
2. Auto off timer - Time remaining on Auto off timer

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/discreteoutput

Last update: **2024-05-09 16:34**

E+E Differential Pressure Sensor driver

Description

This driver is design to allow communication with E+E Elektronik's EE610 pressure differential sensor, via the ezeio MkII's Modbus serial port.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Input terminal - Select the ezeio input terminal connected to the EE610's output signal.

Sensor range setting - From the drop-down menu, select the pressure differential range that matches the DIP switch setting on the sensor.

-100 to +100 Pa
-50 to +50 Pa
-25 to +25 Pa
0 to 100 Pa

Zero adjust (Pa x10) - Enter the desired offset value (in Pascals times 10) to adjust to zero.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Differential pressure in **Pascal (Pa)**
2. Differential pressure in **millibar (mbar)**
3. Differential pressure in **inches of water column (in WC)**
4. Differential pressure in **millimeters of water column (mm H2O)**
5. Differential pressure in **pounds per square inch (psi)**

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/ee_press_diff

Last update: **2024-05-23 22:04**

E+E Temperature & Humidity driver

Description

This driver is design to allow communication with E+E Elektronik's EE210 humidity sensor, via the ezeio MkII's Modbus serial port. The driver may be compatible with other E+E humidity sensors with Modbus connectivity, such as EE072, EE150, and EE211.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Modbus Address - Enter the devices unique Modbus address (device ID)

Device model Units - Select “Metric” or “Imperial” as your preferred units of measure.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. **Temperature (Celsius) -** Ambient temperature in Celsius.
2. **Temperature (Fahrenheit) -** Ambient temperature in Fahrenheit.
3. **Relative Humidity (%) -** Percentage of ambient humidity (moisture).
4. **Water Vapor Partial Pressure (milibar) -** Pressure (in millibar) at which the water vapor in the air condenses to liquid.
5. **Water Vapor Partial Pressure (psi) -** Pressure (in psi) at which the water vapor in the air condenses to liquid.
6. **Dew point Temperature (Celsius) -** Temperature (in Celsius) at which the water vapor in the air condenses to liquid.
7. **Dew point Temperature (Fahrenheit) -** Temperature (in Fahrenheit) at which the water vapor in the air condenses to liquid.
8. **Wet bulb (Celsius) -** Corrected temperature (in Celsius) calculating the effect of evaporated cooling based on relative humidity.
9. **Wet bulb (Fahrenheit) -** Corrected temperature (in Fahrenheit) calculating the effect of evaporated cooling based on relative humidity.
10. **Absolute humidity (grams per cubic meter) -** Mass of water per volume (cubic meter).
11. **Absolute humidity (grams per cubic foot) -** Mass of water per volume (cubic foot).
12. **Mixing ratio (grams per kilogram) -** Ratio of grams of water to a kilogram of dry air
13. **Mixing ratio (grams per pound) -** Ratio of grams of water to a pound of dry air.
14. **Specific enthalpy (kilojoules per kilograms) -** The amount of energy (in Joules) per kilogram.
15. **Specific enthalpy (BTU per pound) -** The amount of energy (in BTU's) per pound.
16. **Frost point temperature (Celsius) -** Temperature (in Celsius) at which the water vapor in the air will freeze.
17. **Frost point temperature (Fahrenheit) -** Temperature (in Fahrenheit) at which the water vapor in the air will freeze.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/ee_temp_rh

Last update: **2024-05-23 21:57**

ezeio Fields via LAN driver

Description

This driver allows one ezeio access to the Fields of another ezeio/s over a local network. All 90 “Fields” are presented as “Available registers”. Both ezeio must be on the same IP subnet.

Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor, device or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

IP address - Enter the last octet of the remote units IP address

---

Features

- Fields from one ezeio can be added as “Fields”, and/or device registers, to another
- Writable “Fields” can be overwritten by the ezeio connected via the driver

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/fields_via_lan

Last update: **2024-07-01 22:50**

Form C alarm contact driver

This driver allows monitoring a Form-C (SPDT) contact, or two SPST (Form-A/B) contacts using a single ezeio input.

Using a few resistors, the driver can monitor each contact state individually, and also indicate if there is a wiring fault (short or open), all with just a single input. Note that the resistors shall be located physically as close to he monitored contact as possible to achieve the desired security function.

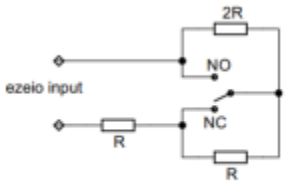
The driver settings allow for choosing between common resistor values.

If a **destination list** is selected, an alert will be sent. If no destination list is selected, the alert is only logged.

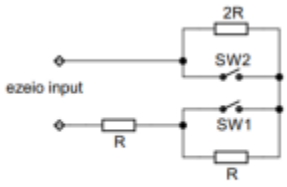
A **holdoff** setting allows for a delay before the alert is generated.

The **restore holdoff** prevents new alerts to be sent for a given time.

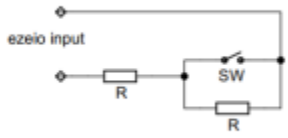
Note that wiring fault events are not affected by the holdoff timers, but will be immediately sent.



SPDT contact (Form-C)



Two SPST contacts



Single SPST contact

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/formc

Last update: **2024-06-03 20:40**

GPS receiver GN-803G driver

Description

This driver is design to allow communication with the GN-803G (TTL/UART) GPS receiver through the ezeio MkII's SDI-12 serial port or input #8 on ezeio with serial numbers BBx-xxx and higher.

Configuration settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Sensitivity - Use the drop-down menu to select the acceptable level of accuracy. Precise location only, Normal precision or Accept low signal. If horizontal accuracy falls below the selected level of precision the driver status will indicate an “App error”.

Timeout - Use the drop-down menu to select how long after the last time of fix (30s, 1min, 2min, or 10min) before indicting an “App error”.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Time of last fix - Gives the time of the last location calculation. Time of last fix should be within seconds of current UTC time.
2. Satellites in use - The number of satellites currently in view / utilized by the receiver.
3. Latitude - North-South coordinate
4. Longitude - East-West coordinate
5. Elevation - Approximate height relative to sea level, given in meters
6. Horizontal accuracy - Expressed as dilution of precision (DOP). In this metric readings of less than 1 are ideal and greater than 20 are poor.
7. Speed - Speed of travel is given as meters per second.
8. Course - Current direction of travel is given as degrees. Tip: Use “Display format in view” option “Degrees as direction” to display as compass direction. Example: 245 = Southwest
9. Time - Current UTC time is given a 6 digit number. Example: 181356 = 6:13:56 PM

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/gps_gn803g

Last update: **2024-05-10 17:56**

GPS receiver GN-8603G driver

Description

This driver is design to allow communication with the GN-8603G (RS232) GPS receiver through the ezeio MkII's SDI-12 serial port.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Sensitivity - Use the drop-down menu to select the acceptable level of accuracy. Precise location only, Normal precision or Accept low signal. If horizontal accuracy falls below the selected level of precision the driver status will indicate an “App error”.

Timeout - Use the drop-down menu to select how long after the last time of fix (30s, 1min, 2min, or 10min) before indicting an “App error”.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Time of last fix - Gives the time of the last location calculation. Time of last fix should be within seconds of current UTC time.
2. Satellites in use - The number of satellites currently in view / utilized by the receiver.
3. Latitude - North-South coordinate
4. Longitude - East-West coordinate
5. Elevation - Approximate height relative to sea level, given in meters
6. Horizontal accuracy - Expressed as dilution of precision (DOP). In this metric readings of less than 1 are ideal and greater than 20 are poor.
7. Speed - Speed of travel is given as meters per second.
8. Course - Current direction of travel is given as degrees. Tip: Use “Display format in view” option “Degrees as direction” to display as compass direction. Example: 245 = Southwest

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/gps_gn8603g

Last update: **2024-05-10 18:04**

GPS receiver GlobalSat MR350S4 driver

Description

This driver is design to allow communication with the MR350S4 (TTL/UART) GPS receiver through the ezeio MkII's SDI-12 serial port.

Configuration settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Sensitivity - Use the drop-down menu to select the acceptable level of accuracy. Precise location only, Normal precision or Accept low signal. If horizontal accuracy falls below the selected level of precision the driver status will indicate an “App error”.

Timeout - Use the drop-down menu to select how long after the last time of fix (30s, 1min, 2min, or 10min) before indicting an “App error”.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Time of last fix - Gives the time of the last location calculation. Time of last fix should be within seconds of current UTC time.
2. Satellites in use - The number of satellites currently in view / utilized by the receiver.
3. Latitude - North-South coordinate
4. Longitude - East-West coordinate
5. Elevation - Approximate height relative to sea level, given in meters
6. Horizontal accuracy - Expressed as dilution of precision (DOP). In this metric readings of less than 1 are ideal and greater than 20 are poor.
7. Speed - Speed of travel is given as meters per second.
8. Course - Current direction of travel is given as degrees. Tip: Use “Display format in view” option “Degrees as direction” to display as compass direction. Example: 245 = Southwest

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/gps_mr350s4

Last update: **2024-05-10 18:07**

Hukseflux Pyranometer SR05 driver

Description

This driver is design to allow communication with the

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Sensitivity - Use the drop-down menu to select the acceptable level of accuracy. Precise location only, Normal precision or Accept low signal. If horizontal accuracy falls below the selected level of precision the driver status will indicate an “App error”.

Timeout - Use the drop-down menu to select how long after the last time of fix (30s, 1min, 2min, or 10min) before indicting an “App error”.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication,

Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. **Solar Radiation** - Solar radiation in Watts per square meter
2. **Sensor voltage** - Solar radiation level in micro volts
3. **Sensor temperature** - Temperature of sensor body in Celsius
4. **Serial** - Serial number of sensor

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/hukseflux_sr05

Last update: **2024-05-23 22:08**

Linovison IOT-S300AQ Air Quality Sensor driver

Description

This driver is design to allow communication with the sensor via Modbus. This sensor provides instantaneous readings of the concentration of particulate matter. Readings are given for particles 2.5 microns or less and 10.0 microns or less. Using this information, the driver converts these reading into a US-EPA Air Quality Index value.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Modbus address - Enter Device I.D. (Modbus address). This address must be unique on the bus.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. **PM2.5** - Concentration of 2.5 micron particulate matter. Measurement in micrograms per meter squared.
2. **PM10** - Concentration of 10 micron particulate matter. Measurement in micrograms per meter squared.
3. **AQI US-EPA** - Air quality stated as the US EPA air quality index (AQI)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/iot-s300aq

Last update: **2024-06-17 19:25**

Linovison IOT-S300NOIS Noise Sensor driver

Description

This driver is design to allow communication with the sensor via Modbus. This sensor provides instantaneous as well a 60 second min, max and average values in dB.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Modbus address - Enter Device I.D. (Modbus address). This address must be unique on the bus.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

1. **Noise** - Instantaneous ambient noise reading in dB
2. **Noise 60s min** - Lowest noise level in the last 60 seconds
3. **Noise 60s max** - Highest noise level in the last 60 seconds
4. **Noise 60s mean** - Average noise level in the last 60 seconds

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/iot-s300nois

Last update: **2024-05-23 22:22**

Modbus port tool driver

Description

The Modbus port tool driver provides the means to set RTU serial bus parameters and directly read / write to devices connected via RTU (serial bus) or TCP (Ethernet).

Configuration settings

MODBUS RTU PORT SETTINGS

Modbus/RTU data rate (BAUD rate) - The speed of the bus can be set or changed to align with the devices connected to the Modbus RTU port. The drop-down menu list all seven speeds defined by the Modbus.org specifications (1200 to 115200 bps).

Modbus/RTU parity - Three “Parity” options are offered in the drop-down menu (No parity, Even parity , Odd parity).

DIAGNOSTIC REGISTERS

To apply the diagnostic settings it is not necessary the “Save changes”, simply click the associated “Set” button.

Monitor address - Enter the device I.D. (Modbus address) of the connected device you wish to read and or write to and click “Set”. If using TCP, the address is the last octet of the device's IP address.

Monitor port - The drop-down menu selections are; Monitor off, Modbus RTU or Modbus TCP. Make the selection and click “Set”. When not in use set to “Monitor off”.

If using another driver as the primary means of setting the RTU baud rate and parity (ezeio System info, Standard I/O, Advanced I/O), the settings must align. Conflicting settings may effect the RTU communication.

MONITOR REGISTERS

Two registers can be monitored live, on the right hand panel under “Available registers”, allowing users to see changes made via “Direct write”. The registers can also be logged by adding to to Fields.

Monitor register 1 & 2 - Enter the register addresses you wish to monitor and click “Set”

Register type - For each of the registers, set the type and click the “Set” button. The drop-down menu offerings are listed in the table below. Refer to the devices documentation and Modbus register map for the register type for each specific register

Type (abbreviated)	Description
INT16/HLD	Holding register, 16 bit integer
UINT16/HLD	Holding register, 16 bit unsigned integer
INT32/HLD	Holding register, 32 bit integer
INT32 Big Endian/HLD	Holding register, 32 bit integer w/ More significant byte sent first
FLOAT/HLD	Holding register, 32 bit Float
FLOAT Big Endian/HLD	Holding register, 32 bit Float, w/ Mort significant byte first
INT16/INP	Input register, 16 bit integer
UINT16/INP	Input register, 16 bit unsigned integer
INT32/INP	Input register, 32 bit integer
INT32 Big Endian/INP	Input register, 32 bit integer w/ More significant byte sent first
FLOAT/INP	Input register, 32 bit Float
FLOAT Big Endian/INP	Input register, 32 bit Float, w/ Mort significant byte first

DIRECT WRITE

Write register - Enter the address for the register of which you wish to modify the value and click “Set”.

Value - Enter the new value to be written and click the associated “Write” button

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/modbustool

Last update: **2024-05-09 23:13**

PID control driver

Configuration settings (center panel)

Description

Single sensor input, single process variable output Adjustable process speed, P/I/D and polarity

---

Configuration settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Setpoint min / max - Constrains the setpoint

Feedback sensor Field - Drop down menu lists all configured Fields. Select one to be used as the feedback for the PID loop.

Feedback min / max - Constrains the range of the value supplied by the feedback sensor's “Field”.

Output (PV) Field - Drop down menu lists all configured Fields. Select one to be used as the output for the PID loop.

Output min / max - Constrains the signal sent to the output

Proportional (PV) - Proportional factor

Integral (I) - Integral factor

Derivative (D) - Derivative factor

I-term min / max - Constrains the cumulative effect of the “Integral” factor.

Update speed (seconds) - Computation speed

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

1	Setpoint
2	Sensor (feedback)
3	Output (PV)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/pid

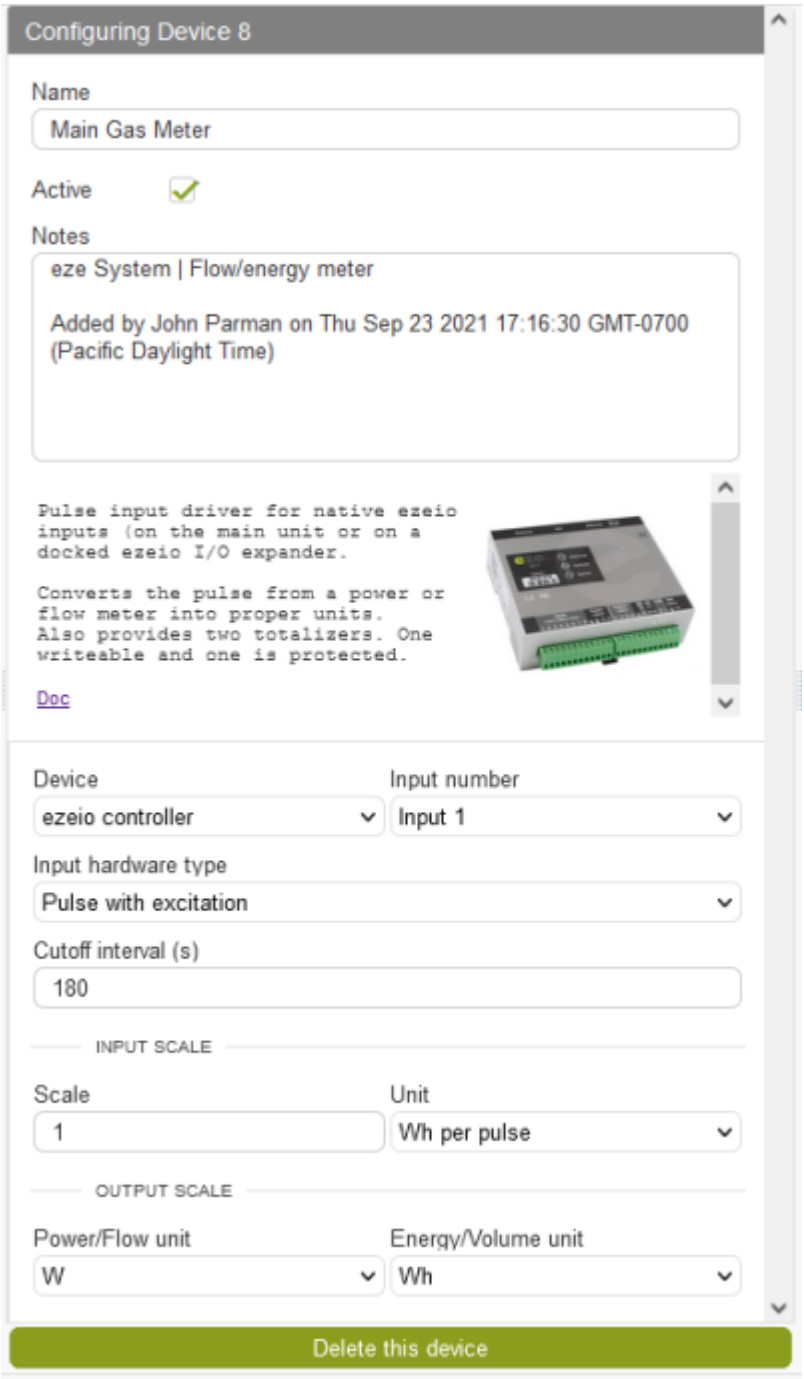
Last update: **2024-05-22 21:12**

Pulse Input driver

Description

This driver (located in the Core drivers folder) is designed for use with the ezeio MkII controller and the ezeio MkII I/O Expander. Pulse outputs on common on meters for gas, water, and electrical. Most are designed to provide a volume total, such as liters or Kilowatt hours. The ezeio can also determine a flow rate based on the interval between pulses or the frequency (both methods can be employed by the driver to produce the most accurate flow value). This feature works best when the meter is ideally sized for the application, generating dozens of pulses per minute. The closer the pulse interval is to the logging interval the less accurate the flow data will be. Configuration settings allow conversion from the meters native unit to any of the most commonly used energy and flow/power units. One of the count registers (reg. #5) is writable (by default) via “Fields”, allowing privileged users to zero or change the pulse count. Changes to the count on register #5 will effect the scaled unit value on register #2.

Configuration Settings



Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Deactivating will cause the pulse count registers to reset to zero when the driver is reactivated

Device - Select the ezeio controller or the address number of a connected ezeio I/O Expander.

Input number - Select the input number (1-8) of the selected device.

Input hardware type - Select Pulse with or without excitation. Most applications use the default “Pulse with excitation” (ezeio provides 5 VDC pull-up). Check the manufactures documentation to if your meter to determine the correct setting.

Cutoff interval (s) - When calculating rate from a totalizers pulses, the value is continually calculated as it waits for the next pulse. If the flow stops, the driver will continue to reduce the rate value as it waits for the next pulse. To avoid force the driver to zero the rate, enter the number of seconds after witch the flow should be determined to have stopped. This interval will differ based on the characteristics of the meter and flow.

(Input) Scale Enter the rating value of the meter (one pulse = x units).

(Input) Unit - Select the rating unit of the meter (kW, L/min, gal/h, etc.)

(Output) Power/Flow unit - Select the desired unit for the rate to be shown in register #1. This allows the rate to be converted to a different unit type (such as BTU/s to Watts) or shown as the same rating given by the manufacture.

Registers

Reg #1 holds the scaled rate value, based on frequency of pulses and/or interval between pulses. The name and unit of this register will vary based on the “Power/Flow unit” selected by the user, in the driver configuration. Selecting kW will set the name as Power and selecting L/min will set the name as Flow.

Reg #2 holds the scaled accumulated value of register #5. The name and unit of this register will vary based on the “Energy/Volume unit” selected by the user, in the driver configuration. Selecting kWh will set the name as Energy and selecting L will set the name as Volume. The value can be reset to zero or user defined value by writing a new value to register #5 through “Fields”.

Reg #3 is identical to register #2, but is intended to hold a value that will not be reset. This value is based on the scaled accumulated value of register #6. The name and unit of this register will vary based on the “Energy/Volume unit” selected by the user, in the driver configuration. Selecting kWh will set the name as Energy and selecting L will set the name as Volume.

Reg #4 holds the frequency of the pulses. Unit is Hertz.

Reg #5 holds the resettable pulse count. Can be reset to a any value via “Fields” or “Script”. When added to “Fields” via the Device tab, the “Writable” option is checked by default.

Reg #6 holds the total pulse count. The intention is for this register to hold the lifetime pulse count.

When added to “Fields” via the Device tab, the “Writable” option is unchecked by default.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/pulseinput

Last update: **2024-05-09 16:36**

Runtime / Service driver

Description

This driver creates a runtime clock from a Field value. The runtime is then used to send service reminder messages, based on a user defined service interval. When scheduled service is completed, a reference number is entered to reset the service interval counter to zero.

Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the asset or site. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device/asset, such as: location, wiring, scaling, etc.

Below the note field, you will see the eze System description, notes and instructions.

Run status field - A drop-down menu provides the means of selecting a specific Field on an ezeio to be used to indicate “Running”. The Field must be zero, if not running. Negative values and decimals are evaluated as running. If the source for the “Run status” does not produce a stable zero when the asset is not running, consider a means of filtering the input, such as using the logical state provided by the "Discrete input" driver.

Service interval - Enter the number of hours desired between servicing

Destination list for reminders - Use the drop-down menu to select one of the accounts "Destination lists". The predefined list of recipients will receive the service reminders. The reminders are sent at 60%, 80% and 100% of the service interval. When over 100% of service interval reminders are sent every 5%.

Reminder message - A default message is provided with minimal text and tags to provide the service interval statistics at each stage of the interval

Destination list for service reset - Use the drop-down menu to select one of the accounts “Destination lists”. The predefined list of recipients will receive the service reset notification.

Reset message - A default message is provided with minimal text and tags to provide the service ID and runtime data. The service ID is written to the Field mapped to register 8 of the driver. Any value can be written, intention is to provide a means of tracking or verification. Examples of possible service ID's

are; technicians ID # or work order #. Upon a submitting the “New value” the “Service reset” message will be sent.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/runtime

Last update: **2024-05-22 20:56**

Shelly relay driver

Description

This driver is designed for use with multiple Shelly relays via WiFi and Ethernet cable. Control logic features are identical to the those found in the eze Discrete output driver.

Configuration settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

IP Address (last octet) - Enter the last set of numbers in the IP address of the relay.

Output number - Enter the number of the relay (some Shelly models have multiple relays).

On startup - When using the ezeio for control or automation you should consider the effects of a lost of power and the effects of the restart. This setting gives you three options for controlling an outputs state on startup of the ezeio, turn ON, turn OFF or return to the previous state.

Auto off - Enter the number of seconds an output should remain on, once triggered. Enter a zero if you would like the output to remain on until directed by automation logic or manually turned off.

CONDITIONAL SETTINGS (OPTIONAL)

This feature provides exclusive or additional control logic, by means of a Field value threshold. The “Control mode” options listed below determine the role (if any) the Field value plays in the output control logic.

Control mode	Driver Logic	Other Logic
No control condition	No effect on output	Complete control of output

Condition triggers the output to turn on	True condition will turn on output	Auto off and other logic may also control output
Condition triggers the output to turn on/off	True condition will turn on output & False condition will turn off	Auto off and other logic may also control output
Condition exclusively controls output state (No auto off)	True condition will turn on output & False condition will turn off	No effect on output
Condition must be true to turn output on	False condition will inhibit other logic from turning the output on	Primary control of output

Condition Field - Select Field (from list) to drive condition.

Logic - Select Greater than (above threshold) or Less than (below threshold).

(Logic) Value - Enter the threshold value for the condition.

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Output state - On/Off state of relay
2. Auto off timer - Time remaining on Auto off timer
3. Voltage - Momentary voltage
4. Current - Momentary amperage reading
5. Power - Momentary power read
6. Energy - Total energy
7. WiFi RSSI - Signal strength of WiFi signal in dBm

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/shelly_relay

Last update: **2024-05-22 21:20**

ezeio System Info driver

Description

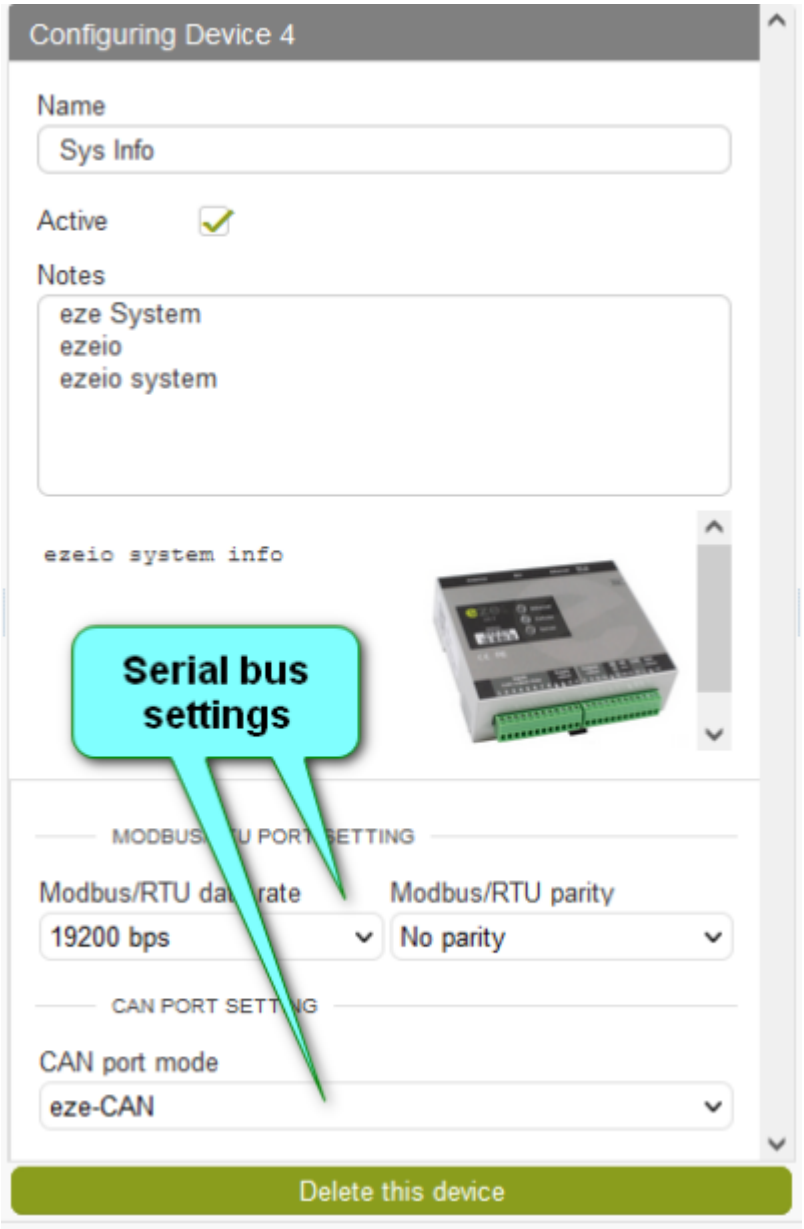
The ezeio MkII System Info driver provides system performance data as well as configuration settings for two of the ezeio's serial buses. In addition to the data and settings, registers 17 and 20 provide a means of locking in a location and measuring the units distance from that point, if it moves (requires a GPS receiver). This driver is loaded by default on units with serial numbers BBA and higher.

Configuration settings (center panel)

Modbus data rate (BAUD rate) - The speed of the bus can be set or changed to align with the devices connected to the Modbus RTU port. The drop-down menu list all seven speeds defined by the Modbus.org specifications (1200 to 115200 bps).

Modbus Parity - Parity can be set or changed by selecting one of three option in the drop-down menu.

CAN Port mode - The can port can be utilized for eze System MkII I/O Expanders (eze-CAN) or J1939 communication. Use the drop-down menu to make your selection.



Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication,

Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

Twenty registers offer a mix of performance date, statuses, GPS data, and a feature that locks in a GPS reference location. A description of each register is given below.

Available registers

	Register Name	View	Value	Unit
1	Uptime	87641	87641	s
2	Input Voltage	12.4	12.4	V
3	Battery Voltage	12.3	12.3	V
4	Logic voltage	5.1	5.1	V
5	Ethernet Link Status	1	1	
6	Ethernet Connection Statu	1	1	
7	Cellular Connection Status	0	0	
8	Cellular Signal	26	26	rssi
9	Cellular inhibit	0	0	s
10	GPS lock	0	0	
11	GPS signal	0	0	
12	GPS Latitude	38.67117	38.67117	°
13	GPS Longitude	-121.15787	-121.157	°
14	GPS Elevation	0.0	0.0	m
15	System Local Date	20240403	20240403	YMD
16	System Local Time	142427	142427	HMS
17	Ref Latitude	38.67117	38.67117	°
18	Ref Longitude	-121.15787	-121.157	°
19	Ref-GPS distance	0	0	m
20	Ref Location lock		0	
21	Group by location	55	55	

1. Uptime - Number of seconds the ezeio has been running. Power cycling or rebooting the ezeio will reset “Uptime” to zero.

2. Input voltage - Voltage reading at the barrel jack (“DC in” *top right*).

3. Battery voltage - Voltage reading at green screw terminals (“Batt” *lower right*).

4. Logic voltage - Voltage supplied for board components, inputs and “+5” supply on green screw terminal.

5. Ethernet Link Status - “1” indicates a connection to an Ethernet LAN.

6. Ethernet Connection Status - “1” indicates the ezeio is utilizing an Ethernet LAN to connect to the eze.io servers.

7. Cellular Connection Status - “1” indicated the ezeio is utilizing a Cellular network to connect to the eze.io servers.

8. Cellular Signal - Value shown is the (RSSI) received signal strength indicator.

9. Cellular inhibit - It is possible to temporarily turn off the cellular modem. This is achieved through script commands. If inhibited, the number of seconds remaining are shown in this register. If not connected via Ethernet, this information will not be transmitted as a live value. If you wish to track this data the register must be added to a “Field”.

10. GPS lock - “1” indicates the receiver has a lock on multiple satellites

11. GPS signal - The receivers signal strength is reported as DOP (Dilution Of Precision). Typically the value should be between 1 and 20. A lower number indicates higher precision (see the table at the bottom of the page for a complete breakdown). If the receiver does not have a GPS lock the value is reported as 255.

12. GPS Latitude - Last known Latitude

13. GPS Longitude - Last known Longitude

14 GPS Elevation - Last known Elevation

15 System Local Date - Current year, month, day, based on the time zone set under the “System” tab.

16 System Local Time - Current hour, minute, second, based on the time zone set under the “System” tab.

GPS Location Lock registers

Registers 17 through 20 are part of the GPS Location Lock feature. This feature works as a simple version of geo fencing for application such as rental equipment. When the lock is switched on, current GPS coordinates are saved as references (registers 17 & 18. If the ezeio (and GPS receiver) move from the reference coordinates the distance is calculated and shown in register 19, as meters. This value can be used in an alarm condition expression. *Example:* `r(1,19) > 300`

17 Ref Latitude - Latitude captured by “Ref Location lock” (register 20). This value will float, matching register 12, if “Ref Location Lock” is off.

18 Ref Longitude - Longitude captured by “Ref Location lock” (register 20). This value will float, matching register 13, if “Ref Location Lock” is off.

19 Ref-GPS distance - Distance (in meters) between current GPS coordinates and reference location.

20 Ref Location lock - An On/Off switch is provided when this register is added to “Fields”. Switching it on captures the current GPS coordinates and holds them in registers 17, 18. The difference between the current GPS coordinates (registers 12 & 13) and the locked reference Location (registers 17 & 18) is calculated in meters and shown in register 19.

Group area matching

Register 21 is convenient to use for irregular shaped geographical boundaries. The value is set to the number of the group with the smallest area overlapping the controller location. An example of how to use this in an alarm expression is `r(1,21) < 1`. This will trigger the alarm when no area matches the controller location.

21 Group by location - The location of the controller (as determined by GPS, by cell tower or manually set) is compared every 10 minutes with the group areas in all subgroups to the group where the controller is. This register will be set to the group number of the smallest area that overlaps the controller location. If no area overlaps with the controller location, the value will be 0 (zero). [New since April-24]

—-

GPS signal (DOP value)

DOP Value	Rating[5]	Description
<1	Ideal	Highest possible confidence level to be used for applications demanding the highest possible precision at all times.
1-2	Excellent	At this confidence level, positional measurements are considered accurate enough to meet all but the most sensitive applications.
2-5	Good	Represents a level that marks the minimum appropriate for making accurate decisions. Positional measurements could be used to make reliable in-route navigation suggestions to the user.
5-10	Moderate	Positional measurements could be used for calculations, but the fix quality could still be improved. A more open view of the sky is recommended.
10-20	Fair	Represents a low confidence level. Positional measurements should be discarded or used only to indicate a very rough estimate of the current location.
>20	Poor	At this level, measurements are inaccurate by as much as 300 meters with a 6-meter accurate device (50 DOP × 6 meters) and should be discarded.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/systeminfo

Last update: **2024-05-09 15:26**

Temp / RH Stick driver (TA1020)

Description

This driver is designed for use with the temperature and humidity sensor sold by eze System (P/N TA1020).

Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Settings

Address - Enter the RTU device number (must be unique on the bus)

Device status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

- Relative Humidity %
- Temperature C
- Dew point C
- Temperature F
- Dew point F

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/temp_rh1020

Last update: **2024-05-10 18:13**

Thermistor driver

Description

This driver is designed for use with the ezeio MkII controller and the ezeio MkII I/O Expander.

Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Input settings

Device - Select the ezeio controller or the address number of a connected ezeio I/O Expander.

Input number - Select the input number (1-8) of the selected device.

Sensor type - The drop-down menu contains a list of temperature probes both generic and mfg./model specific. This list will grow over time. If you don't find a suitable sensor type, feel free to contact eze System about adding your preferred sensor to the driver.

Unit - Choose you preferred unit (Celsius, Fahrenheit, or Kelvin)

Low & High limits - Set the thresholds for cooling and heating.

Offset - Enter a value here if a calibration offset is needed.

Control Output for Heating

Device - Select the ezeio controller or the address number of a connected ezeio I/O Expander.

Output number - Select the output number of the selected device.

Control Output for Cooling

Device - Select the ezeio controller or the address number of a connected ezeio I/O Expander.

Output number - Select the output number of the selected device.

Control Parameters

Control hysteresis -

Control holdoff - Temp readings must meet/exceed thresholds for the holdoff time before outputs will be triggered.

Min runtime - To avoid short run cycles, enter the minimum number of seconds the outputs will stay on.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/thermistor

Last update: **2024-05-09 23:02**

Temp / RH Sensor (TA1030)

Description

This driver is designed for use with the temperature and humidity sensor sold by eze System (P/N TA1030).

Settings (Center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Doc link - This links you to relevant documents such as the manufactures user manual.

Modbus Address - Enter the RTU device number (must be unique on the bus)

Device status & Registers (Right panel)

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

---

Available registers

1. Relative Humidity %

2. Temperature C
3. Dew point C
4. Temperature F
5. Dew point F

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/tz_temp_rh1030

Last update: **2024-05-10 19:06**

Davis Vantage Pro 2 driver

Description

This driver is compatible with Davis Instruments, Vantage Pro 2 series weather station. Weather station connects directly to ezeio's SDI-12 port, bypassing the Davis (cabled) Vantage Pro2 Console.

Settings (center panel)

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

User Notes - This space can be used to store information specific to the device and your application, such as: location, wiring, scaling, etc.

eze System Notes - In this space we provide: wiring instructions, product image, product details.

Rain collector size -

Status & Registers (right panel)

Device Status

Driver info - Typically this is the name from the program file

Driver updated/version - Date loaded or updated and the version of the driver

Battery & Signal - Signal indicates the progress through the script. If driver fails to operate, signal number indicates progress

Driver status - Color and text of status bubbles give a quick visual reference of Communication, Operation, and Application

Communication - Com count indicates number of successful and unsuccessful data packets received

Available registers

1. **Wind speed** - Current wind speed in meters per second (m/s)
2. **Wind direction** - Current wind direction in degrees (0-359, 0 = North)
3. **UV index** - Current UV measurement
4. **Rain rate** - Current rate of rain fall
5. **Solar radiation** - Current level of solar radiation in Watts per meter squared
6. **Temperature** - Current temperature in Fahrenheit
7. **Max wind** - Highest wind reading
8. **Humidity** - Current humidity reading
9. **Rain total** - Daily rain fall total
10. **Dewpoint** - Dewpoint calculation

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/vantage_pro2

Last update: **2024-05-23 21:51**

WattNode driver (Basic)

Description

This driver is designed for use with WattNode Modbus Module (WND) WattNode Modbus (WNC) and Multi-Circuit meter (MCM). This driver is included with the “Basic ezeio service” package and provides a limited set of power circuit measurements. The full set of measurements is available from the “WattNode Advanced” driver included in the “Meters” driver package.

Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default “Field” name (if added to “Fields”)

Active Check box - Check this box and “Save changes” to run the driver. The driver program can be suspended by unchecking the box and clicking on “Save changes”.

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Meter settings

Address - Enter the RTU device number (must be unique on the bus) or the last octet of the IP address, if connected via Ethernet/TCP

Module - Default setting is RTU for WattNode WND & WNC models. For Multi-Circuit Meter (MCM) select the corresponding Unit (circuit) number.

CT (A, B, & C) - Drop-down menus are provided for selecting CT ratings for each of the three phases metered.

Device status

Driver info -

Driver updated/version -

Battery & Signal -

Driver status -

Communication -

Available registers

- Total Energy (kWh)
- Total Power (kW)
- Power phase A (kW)
- Power phase B (kW)
- Power phase C (kW)
- Volt average ph-N (V)
- Volt phase A-N (V)
- Volt phase B-N (V)
- Volt phase C-N (V)
- Current phase A (A)
- Current phase B (A)
- Current phase C (A)
- Power factor average

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/wattnode_basic

Last update: **2024-05-10 19:24**

Weather forcast driver

Description

This driver provides 3 days of local weather forecast based on the ezeio's "system location"

Features

- 3 Days of forcasts
- Updates every 4 hours
- 18 Metrics

Settings

Name - The name is up to the user. Our suggestion is to choose a naming convention that makes the viewing the device list intuitive. Such as referring to the sensor and/or application. Keep the name short as it will be combined with the register name to create the default "Field" name (if added to "Fields")

Active Check box - Check this box and "Save changes" to run the driver. The driver program can be suspended by unchecking the box and clicking on "Save changes".

Notes - This space can be used to store information specific to the device, such as: location, wiring, scaling, etc.

Unit - Choose Metric or Imperial as your preferred unit type for the various weather measurements

Available daily metrics

- Average temp
- Min temp
- Max temp
- Humidity
- Wind speed
- Wind direction
- Pressure (Barometric)
- Precipitation
- Cloud cover
- UV Index
- Visibility
- Feels like

- Weather code
- Heat index
- Dewpoint
- Windchill
- Wind gust
- Sun hours

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/drivers/doc/weather_forcast

Last update: **2024-05-10 19:35**

Expressions

Within the configuration of an ezeio, mathematical expressions are used to produce a value/result for “Fields”, “Alarms” and “Conditions”. This powerful tool can produce mathematical and logical results using multiple constants and/or variables from the available resources on an ezeio. Expressions can be as simple as the constant “1” to indicate True/ON or as complex as necessary.

Simple example

Purpose - Show the value of register 8 on device 2:

```
r(2,8)
```

Complex example

Purpose - Low oil pressure siren (during work schedule)
Logic - If oil pressure is less than 25 and RPM is greater than 600 and hour is between 06:00 and 17:00

Enter the following in the “Field” relate to an output register

```
(r(3,4)<25) && (r(3,3)>600) && (hour()>=6) && (hour()<17)
```

Functions & Operators

Many of the valid functions and operators will be familiar to you, such as +, -, >, <, and =. Others come from advanced mathematics and logic, such as sin(), tan(), ||, and &&. Another category relates specifically to eze System terms and functions, such as Day() and K2F().

Syntax

The syntax structure of eze System expressions follows common mathematical and logic standards for precedence and functional format.

For example, 2+3*4 equals 2+(3*4) or 14, since multiplication takes precedence over addition.

Boolean logic

Any value greater than 0 is considered 'true'.

0 and all negative values are considered 'false'.

The result of a condition, such as (f(1)>45), results in the value 0 (if false) or 1 (true).

Field expressions

The Data Expression box under Field configuration determines the value of the field.

If left blank, the field value is not updated by the system, and will simply remain the same until changed by the user manually or by a script using the SetField function.

The most common field expression simply takes the value of a register and applies it to the field.

```
r(4, 2)   // get the value from device #4, register 2
```

A single expression can reference other fields and registers.

```
f(4) + f(5)          // the sum of the values in field #4 and field #5
r(1,1) * 100 - 50    // multiply the value of register 1 on device 1 with 100 and subtract 50
abs(f(10)-f(12))     // the absolute difference between field 10 and 12
Uptime()/60          // Set the field value to the number of minutes since last reboot
```

The Data Expression is evaluated at a fixed rate of 10 times per second (100ms interval). This can be used to create counters and accumulators.

Example: Run-time counter

If our Field #1 monitors the current draw of a device, and the device is considered running if the current exceeds 5A, we can use this expression to count the total run-time in seconds in field #2:

Field #2 Data expression:

```
f(2) + ( f(1)>5 )/10
```

Starting with the condition `(f(1)>5)`, this will return 0 (false) if the device is not running, and 1 (true) if it is running. So as long as the device is not running, we're adding zero to the f(2) counter. When the device is running we are adding 0.1 to the f(2) counter, and since this happens 10 times per second, we'll add a total of 1 each second.

This way we've implemented a run-time counter.

Example: Total volume accumulator

Assume our Field #1 holds the output of a flow sensor, in L/s (Liters per second).

We want out Field #2 to reflect the total amount in Liters.

Field #2 Data expression:

```
f(2) + f(1)/10
```

We simply add 1/10th of the value of the momentary flow to our accumulator f(2).

Example: kWh from kW

In this example, field 1 holds our momentary power in kW. We want to accumulate this into energy (kWh) over time.

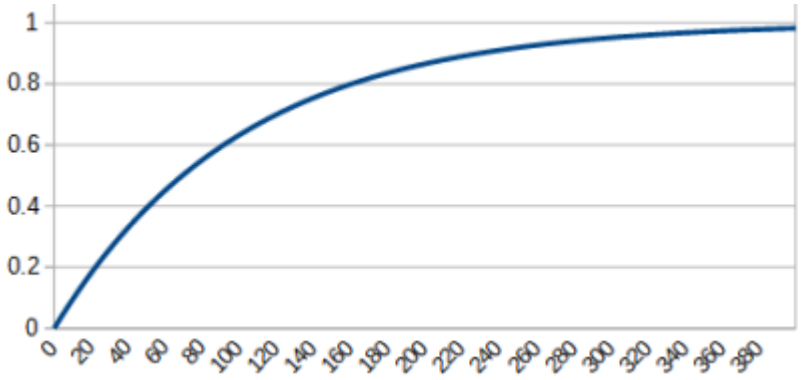
Field #2 Data expression:

```
f(2) + f(1)/36000
```

Since we are looking for kW-hours, we need to divide the momentary power with 36000 (3600 seconds in an hour, and we are re-evaluating 10 times per second).

Example: Signal filter

If data from a register (or other field) includes a lot of noise and needs to be filtered over time, a simple field expression can be used to implement a IIR filter function:




```
( f(THIS)*99 + r(1,1) )/100
```

Knowing that the expression evaluated 10 times per second, we are keeping 99% of the value, and adding 1% from the signal every 100ms. Thus the response time for this filter is about 7s for a 3dB (50%) response.

Reverse expressions

The reverse expression on the field is only relevant if the register referenced in the Data Expression is writable.

In most cases, the reverse expression can be left blank, and the ezeio will use the value from the field unchanged to write back to the register. Note also that the “Writable” checkbox needs to be checked in order to allow the field value to be changed by the user.

The reverse expression can be useful if there is math to scale the register value in the Data Expression. For example if we have a read & writable register with a temperature scaled x100 (so 4567 is 45.67 degrees), we would have a data expression like this:

```
r(1,1)/100
```

Now if the user want to set the temperature, we need to multiply the user-entered value with 100 before we write it back to the register. Thus our reverse expression need to be:

```
f(THIS)*100
```

The macro “THIS” always reference the current item.

Example: Simple control with Reverse expression

In combination with the “Continuous Write” checkbox, the reverse expression can be used for simple control logic.

When Continuous Write is checked, the reverse expression is evaluated every 100ms (10 times per second).

Assuming the Field is tied to a relay output in the Data Expression, we can use this to implement a simple control function by using the following expression:

```
( f(1)>40 ) * 100
```

This expression will look at field #1, and if it's larger than 40, it returns 100. If 40 or less, it returns 0. The result will directly control the relay, actuating it if the value of field 1 exceeds 40.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/start

Last update: **2022-11-16 23:50**

Alarm status

a(alarmno)

Return the current status of an alarm.

Description

```
a( alarmno )
```

This function returns the current status of a given alarm.

Parameters

alarmno	Alarm number (1 - 200)

Return value

Returns the current status of the alarm. The return value will be 0 if the alarm is not active. If the alarm is active, the return value is 1.

Note that the returned status depends on the holdoff and restore conditions of the alarm. The alarm condition may not be true, but if the restore condition is not met (yet) this function will still return 1 (true).

Example usage

```
a(12)
```

Returns 1 if the alarm is active. Otherwise returns 0.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/a

Last update: **2019-09-02 18:46**

Absolute value

abs(a)

Returns the absolute value of a.

Description

```
abs( a )
```

Returns |a|

Parameters

a	Value

Return value

This function returns the absolute value of a.

Example usage

```
abs( -123 )
```

Returns 123.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/abs

Last update: **2019-09-02 18:46**

Arc Cosine

acos(a)

Return the arc cosine of a.

Description

```
acos( a )
```

Return the arc cosine of a in radians.

Parameters

a	Value [-1 to +1]

Return value

arc cosine of a

Example usage

```
acos( 1 )
```

Returns 0.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/acos

Last update: **2019-09-02 18:46**

Analog input value

adc(inputnumber)

Returns the value of a physical input.

Description

```
adc( inputnumber )
```

Returns the value read on the ADC for the given input.

Parameters

`inputnumber`	Physical input number (1-8)

Return value

This function returns the value read on the given input. The unit depends on the configured mode.

Input configuration	Unit returned by this function
Voltage	mV
Current	uA
Resistance	Ohm
Thermistor	Degrees Kelvin
Pulse	Interval in ms

Example usage

```
adc( 4 )
```

Returns the value read from input #4

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/adc

Last update: **2024-08-01 20:14**

Alarm Information

ai(alarmno, mode)

Returns information of a given alarm.

Description

```
ai(3, AL_HOLDOFF)
```

Returns the remaining holdoff time for alarm #3.

Parameters

alarmno	Alarm number
mode	What information is requested

mode

AL_STATE	The alarm state (1=in alarm, 0=not in alarm)
AL_HOLDOFF	The remaining time on the holdoff timer (in seconds)
AL_ALARMHOLDOFF	The alarm holdoff setting (in seconds)
AL_RESTOREHOLDOFF	The restore holdoff setting (in seconds)
AL_ISALARM	Returns 1 if the alarm condition is true. Otherwise 0
AL_ISRESTORE	Return 1 if the restore condition is true. Otherwise 0
AL_RETRIGCOUNT	Number of times the alarm has be re-triggered
AL_RETRIGDELAY	Time remaining on the alarm retrigger delay (in seconds)
AL_RETRIGCOUNTSET	Retrigger count configured
AL_RETRIGDELAYSET	Retrigger delay configured (in seconds)

Return value

This function returns information about the alarm based on the mode parameter

Example usage

```
ai(3, AL_RETRIGCOUNT)
```

[command available in firmware 201008 and later]

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/ai

Last update: **2020-10-08 23:28**

Arc Sine

asin(a)

Return the arc sine of a.

Description

```
asin( a )
```

Return the arc sine of a in radians.

Parameters

a	Value [-1 to +1]

Return value

arc sine of a

Example usage

```
asin( 1 )
```

Returns 1.57079... (PI/2).

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/asin

Last update: **2019-09-02 18:46**

Arc Tangent

atan(a)

Return the arc tangent of a.

Description

```
atan( a )
```

Return the arc tangent of a.

Parameters

a	Angle in radians [-1 to +1]

Return value

arc tangent of a

Example usage

```
atan( 1 )
```

Returns 0.7854...

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/atan

Last update: **2019-09-02 18:47**

Bit

bit(value, bitno)

Returns the status of the given bit in an integer value

Description

```
bit(19, 1)
```

Returns the value of bit 1 (the second bit) in the value 19.

19 (decimal) written as binary is `10011`. Thus, bit 4, 1 and 0 are set. Bit 0 is the first from the right.

Parameters

`value`	The input value (64 bit integer)
`bitno`	The bit number to return

Return value

Returns 1 if the bit is set. 0 if the bit is not set.

Example usage

```
bit(211, 5) // returns 0
bit(211, 4) // returns 1
```

[Available in firmware 201008 and newer]

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/bit

Last update: **2020-10-09 01:07**

Boolean

bool(a)

Returns 1 if the value a is larger than 0. Returns 0 if a is zero or negative.

Description

```
bool( a )
```

Returns 1 if the value a is larger than 0. Returns 0 if a is zero or negative.

Parameters

a	Any value

Return value

If a>0, returns 1. Else returns 0.

Example usage

```
bool(3)
```

Returns 1.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/bool

Last update: **2019-09-02 18:48**

Celsius to Kelvin

C2K(celsius)

Returns the temperature converted from Celsius to Kelvin

Description

```
C2K( celsius )
```

Returns the given temperature in Kelvin.

Parameters

`celsius`	Temperature in Celsius

Return value

This function converts the given temperature from Celsius to Kelvin degrees.

Example usage

```
C2K(20)
```

Returns 293.15.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/c2k

Last update: **2019-09-02 18:48**

Cosine

cos(a)

Return the cosine of a.

Description

```
cos( a )
```

Return the cosine of a, where a is the angle in radians.

Parameters

a	Angle in radians

Return value

cosine of a

Example usage

```
cos( PI/2 )
```

Returns 0.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/cos

Last update: **2019-09-02 18:48**

Day

Day()

Returns the current day of the month.

Description

```
Day()
```

Returns the current day of the month, 1 to 31.

Parameters

none

Return value

This function returns the current day of the month in the time zone set for the ezeio.

Example usage

```
Day()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/day

Last update: **2019-09-02 18:48**

Device Status

ds(deviceno, subject)

Return the current status of a device.

Description

```
ds( deviceno, subject )
```

This function returns the current status of a device or the aggregate status of all devices.

Parameters

deviceno	Device number (1 - 40). If 0 (zero), the function returns the aggregate status of ALL devices.
subject	Subject to request (see below)

Subject	Description
DVCSTAT_BATTVOLT	Battery/supply voltage in mV
DVCSTAT_SIGNAL	Signal level, typically 0-100%
DVCSTAT_COMMCOUNT	Comm count, typically count of valid messages
DVCSTAT_COMMERR	Comm error count, typically number of failed communication attempts
DVCSTAT_LASTCOMM	Last comm, EPOCH timestamp of last successful communication
DVCSTAT_COMMSTAT	Device communication status (0-7, where 0=unknown, 1=error, 2,3,4=warning, 5,6,7=ideal)
DVCSTAT_OPSTAT	Device operational status (0-3, where 0=unknown, 1=error, 2=warning, 3=nominal)
DVCSTAT_APPSTAT	Device application status (0-7, where 0=unknown, 1=error, 2,3,4=warning, 5,6,7=nominal)
DVCSTAT_STAT	Aggregated status (0-3, 0=unknown, 1=error, 2=warning, 3=ok)

Return value

Returns the current status value of the subject requested.

Example usage

```
ds(2, DVCSTAT_COMMERR)
```

Returns the communication error count from device 2.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/ds

Last update: **2021-02-24 18:55**

Elevation

Elev()

Returns the elevation as reported by a GPS.

Description

```
Elev()
```

Returns the elevation in meters as reported by a GPS module. If no positioning data is available (or no GPS is attached), this will return 0.

Parameters

none

Return value

This function returns the most recent reported elevation.

Example usage

```
Elev()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/elev

Last update: **2019-09-02 18:49**

Field value from remote unit

=XYZ-987,12

Return recent value of a field from a remote unit

Description

```
={serial},{fieldno}
```

This function returns the most recent value of a field on a remote unit.

Note that the data is updated every 10 minutes (fixed), and that the two units must be assigned to the same account.

Parameters

`serial`	Serial number of remote ezeio
`fieldno`	Field number on remote ezeio (1 through 90)

Return value

Returns the most recent value of the field (updates every 10 minutes).

Example usage

```
=ABC-123,4   // returns the value of field #4 on unit ABC-123
```

Unlike other field expressions, this expression cannot be combined with other math. It must be the only thing in the field expression, and it only works as a field data expression. It will not work in alarms or reverse expressions.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/eqf

Last update: **2021-03-18 21:11**

Exponent

exp(a)

Returns the value of e raised to the a:th power.

Description

```
exp( a )
```

returns the value of e raised to the a:th power.

Parameters

a	Value

Return value

This function returns the exponential value of x.

Example usage

```
exp( 1 )
```

Returns 2.7182...

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/exp

Last update: **2019-09-02 18:49**

Field value

f(fieldno)

Return the current value of a field.

Description

```
f( fieldno )
```

This function returns the current value of a given field.

Parameters

`fieldno`	Field number (1 through 90)

Return value

Returns the current value of the field.

Example usage

```
f(15)
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/f

Last update: **2019-09-02 18:49**

Farenheit to Kelvin

F2K(farenheit)

Returns the temperature converted from Farenheit to Kelvin

Description

```
F2K( farenheit )
```

Returns the given temperature in Kelvin.

Parameters

`farenheit`	Temperature in Farenheit

Return value

This function converts the given temperature from Farenheit to Kelvin degrees.

Example usage

```
F2K(68)
```

Returns 293.15.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/f2k

Last update: **2019-09-02 18:50**

Field value from log data

fh(fieldno, seconds)

Return the field value from the log data, from `seconds` ago.

Description

```
fh( fieldno, seconds )
```

This function returns the logged value of a given field, from `seconds` ago. If the field is configured for faster logging than the default 10 minutes, the fh function will attempt to fetch the data from the 'fast log' buffer.

If no data can be found, the function will return 0.

This function is available in firmware 21060101 and newer

Parameters

`fieldno`	Field number (1 through 90)
`seconds`	Number of seconds back in time from where the data is fetched

Return value

Returns the value from the log buffer.

If no data can be found, the function returns 0.

Example usage

```
fh(15, 3600)
```

The above returns the data from field 15, 1h ago

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/fh

Last update: **2021-06-03 16:05**

Floor

floor(a)

Returns a rounded down to the closest integer.

Description

```
floor( a )
```

Returns the value of a rounded down to the closest integer.

Parameters

a	Value

Return value

This function returns the largest integral value not greater than a.

Example usage

```
floor(7.89)
```

Returns 7.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/floor

Last update: **2019-09-02 18:50**

From/In range

from(value, range)

Check if the `value` is between 0 and the `range`.

Description

```
from( value, range )
```

This function is commonly used with the time() function to evaluate a time range.

Parameters

`value`	Value to evaluate
`range`	Range to compare to

Return value

If the `value` is negative or larger than `range` the from() function returns 0. Otherwise the function returns the difference between `range` and `value`.

Example usage

```
from(35, 40)  // returns 5 (=40-35)

from(42, 40)  // returns 0

from(-4, 40)  // returns 0

from(time(17,00), 30)   // Will return a positive number between 17:00 and
17:29 (inclusive). Otherwise returns 0.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/from

Last update: **2019-09-02 18:50**

Hour, Minute, Second

HMS()

Return encoded hour-minute-second

Description

```
HMS()
```

This function returns the current local time of day as HHMMSS.

Parameters

none

Return value

Returns current time of day as HHMMSS.

Example usage

```
HMS()>123400   // returns 1 if time is past 12:34
HMS()>93000 && HMS()<180500   // returns 1 if time is between 9:30 and 18:05
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/hms

Last update: **2019-09-25 00:25**

Hour

Hour()

Returns the current hour

Description

```
Hour()
```

Returns the current hour, 0-23.

Parameters

none

Return value

This function returns the current hour in the time zone set for the ezeio.

Example usage

```
Hour()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/hour

Last update: **2019-09-02 18:50**

Kelvin to Celsius

K2C(kelvin)

Returns the temperature converted from Kelvin to Celcius

Description

```
K2C( kelvin )
```

Returns the given temperature in Celcius.

Parameters

`kelvin`	Temperature in Kelvin

Return value

This function converts the given temperature from Kelvin degrees to Celcius.

Example usage

```
K2C(293.15)
```

Returns 20.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/k2c

Last update: **2019-09-02 18:50**

Kelvin to Farenheit

K2F(kelvin)

Returns the temperature converted from Kelvin to Farenheit

Description

```
K2F( kelvin )
```

Returns the given temperature in Farenheit.

Parameters

`kelvin`	Temperature in Kelvin

Return value

This function converts the given temperature from Kelvin degrees to Farenheit.

Example usage

```
K2F(293.15)
```

Returns 68.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/k2f

Last update: **2019-09-02 18:51**

Latitude

Lat()

Returns the Latitude as reported by a GPS.

Description

```
Lat()
```

Returns the Latitude in degrees as reported by a GPS module. If no positioning data is available (or no GPS is attached), this will return 0.

Parameters

none

Return value

This function returns the most recent reported latitude

Example usage

```
Lat()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/lat

Last update: **2019-09-02 18:51**

Natural Logarithm

ln(a)

Return the natural logarithm of a.

Description

```
ln( a )
```

Returns the natural logarithm (base-e logarithm) of a.

Parameters

a	Value

Return value

This function returns natural logarithm of a.

Example usage

```
ln( 20 )
```

Returns 2.9957...

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/ln

Last update: **2019-09-02 18:51**

Common Logarithm

log(a)

Returns the common logarithm (base-10 logarithm) of a.

Description

```
log( a )
```

Returns the common logarithm (base-10 logarithm) of a.

Parameters

a	Value

Return value

This function returns the common logarithm of a, for values of a greater than zero.

Example usage

```
log( 100 )
```

Returns 2.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/log

Last update: **2019-09-02 18:51**

Longitude

Long()

Returns the Longitude as reported by a GPS.

Description

```
Long()
```

Returns the Longitude in degrees as reported by a GPS module. If no positioning data is available (or no GPS is attached), this will return 0.

Parameters

none

Return value

This function returns the most recent reported longitude

Example usage

```
Long()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/long

Last update: **2019-09-02 18:52**

Max

max(a, b)

Return the larger value of a or b.

Description

```
max( a, b )
```

This function returns the larger of the two values, a or b.

Parameters

a	Any value
b	Any value

Return value

If a>=b, returns the value a If a<b, returns the value b

Example usage

```
max(15, 17)
```

Returns 17.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/max

Last update: **2019-09-02 18:52**

Min

min(a, b)

Return the smaller value of a or b.

Description

```
min( a, b )
```

This function returns the smaller of the two values, a or b.

Parameters

a	Any value
b	Any value

Return value

If a>=b, returns the value b If a<b, returns the value a

Example usage

```
max(3, -4)
```

Returns -4.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/min

Last update: **2019-09-02 18:52**

Minute

Minute()

Returns the current minute

Description

```
Minute()
```

Returns the current minute, 0-59.

Parameters

none

Return value

This function returns the current minute in the time zone set for the ezeio.

Example usage

```
Minute()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/minute

Last update: **2019-09-02 18:52**

Month

Month()

Returns the current month.

Description

```
Month()
```

Returns the current month number (1-12)

Parameters

none

Return value

This function returns the current month in the time zone set for the ezeio.

Example usage

```
Month()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/month

Last update: **2019-09-02 18:52**

Operators and Constants

Operators and constants

The following operators are supported in expressions:

Operator	Name	Description	Example usage
+	Addition	Adds two numbers	`5+PI` (8.14159..)
-	Subtraction	Subtracts a number from another	`Year()-2000` (19)
*	Multiplication	Multiplies two numbers	`4*7` (28)
/	Division	Divides a number with another	`67/10` (6.7)
^	Exponent	Raises a number to a power	`10^3` (1000)
%	Modulus	Remainder of division	`2345%100` (45)
&	Binary AND	Binary AND	`14&7` (6)
|	Binary OR	Binary OR	`14|7` (15)
#	Binary XOR	Binary Exclusive OR	`14#7` (9)
~	Binary complement	Binary Complement	`~160` (95)
>	Greater than	Logical compare	`10>9` (1)
<	Less than	Logical compare	`10<9` (0)
>=	Greater than or equal	Logical compare	`10>=9` (1)
<=	Less than or equal	Logical compare	`10<=9` (0)
==	Equal to	Logical compare	`10==9` (0)
&&	Logical AND	Logical AND	`10&&9` (1)
||	Logical OR	Logical OR	`10||9` (1)
!	Logical NOT	Logical invert	`!(10==9)` (1)
!!	Make boolean	Forces result to be boolean	`!!8` (1)
E	10th power	Raise to 10th power	`3.14E3` (3140)
PI	π	Constant Pi	`PI` (3.14159...)
E	e	Natural Logarithm	`E` (2.71828...)
? :	Conditional	(C)?(X):(Y) If C is true then X, else Y	`(f(3)>100)?(f(4)):(f(5))`

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/operators

Last update: **2021-06-08 16:35**

Register value

r(deviceno, registerno)

Return the current value of a register.

Description

```
r( deviceno, registerno )
```

This function returns the current value of a given register.

Parameters

deviceno	Device number (1 - 40)
registerno	Register number on the device (1 - 150)

Return value

Returns the current value of the register.

Example usage

```
r(3,14)
```

Returns the value of register 14 on device 3.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/r

Last update: **2019-09-02 18:53**

Register value

ra(deviceno, registerno)

Return the age of a register.

Description

```
ra( deviceno, registerno )
```

This function returns the number of seconds passed since the register was updated.

Each time SetRegister is called, the age is reset to 0 - even if the value written to the register is the same.

Parameters

deviceno	Device number (1 - 40)
registerno	Register number on the device (1 - 150)

Return value

Returns the age of the register in seconds (up to 4095, 68 minutes and 15 seconds).

Example usage

```
ra(3,14)
```

Returns the age of register 14 on device 3.

FIRST AVAILABLE IN FW VERSION 25020101

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/ra

Last update: **2025-03-14 18:19**

Random number

rand(a)

Returns a random number between 0 and a.

Description

```
rand( a )
```

Returns a random number between 0 (inclusive) and a (exclusive).

Parameters

a	Upper limit for random number

Return value

This function returns a random number.

Example usage

```
rand(100)
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/rand

Last update: **2019-09-02 18:53**

Round

round(a)

Returns a rounded to the nearest integer.

Description

```
round( a )
```

Returns the value of a rounded to the nearest integer, or away from zero if the fraction is 0.5.

Parameters

a	Value

Return value

This function returns a rounded to the nearest integer.

Example usage

```
round(7.89)
```

Returns 8.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/round

Last update: **2019-09-02 18:53**

Real time / EPOCH

RTC()

Returns the epoch / unixtime.

Description

```
RTC()
```

Returns the number of seconds since 1970-01-01 00:00:00.

Parameters

none

Return value

This function returns the epoch/unix timestamp as seconds.

Example usage

```
RTC()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/rtc

Last update: **2019-09-02 18:54**

Schedule status

sch( )

Returns the status of the schedule number referenced Returns 1 if currently in schedule. Returns 0 if currently out of schedule.

Description

```
sch( # )
```

Returns 1 if “in schedule. Returns 0 if “out of schedule”.

Parameters

#	Schedule #

Return value

If “in schedule, returns 1. Else returns 0.

Example usage

```
sch(3)
```

Returns 1.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/sch

Last update: **2020-07-23 22:06**

Second

Second()

Returns the current second

Description

```
Second()
```

Returns the current second, 0-59.

Parameters

none

Return value

This function returns the current second.

Example usage

```
Second()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/second

Last update: **2019-09-02 18:54**

Sine

sin(a)

Return the sine of a.

Description

```
sin( a )
```

Return the sine of a, where a is the angle in radians.

Parameters

a	Angle in radians

Return value

sine of a

Example usage

```
sin( PI/2 )
```

Returns 1.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/sin

Last update: **2019-09-02 18:54**

Square root

sqrt(a)

Returns the square root of a.

Description

```
sqrt( a )
```

Returns the square root of the value a.

Parameters

a	Value

Return value

This function returns the square root of the value a.

Example usage

```
sqrt( 256 )
```

Returns 16.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/sqrt

Last update: **2019-09-02 18:54**

System Item

sys(subject)

Return a system status value.

Description

```
sys( subject )
```

This function returns the current value of a system item.

Parameters

subject	Subject to request (see below)

Subject	Description
SYSITEM_GPSX	Current GPS longitude in degrees x 10^6
SYSITEM_GPSY	Current GPS latitude in degrees x 10^6
SYSITEM_GPSZ	Current GPS elevation in meter x 10
SYSITEM_GPSSIGNAL	Current GPS reception signal
SYSITEM_GPSLOCK	0 if GPS is not locked. 1 if locked.
SYSITEM_UPTIME	Number of seconds since last reset
SYSITEM_CELLINHIBIT	0 if normal operation. >0 if cell modem is turned off
SYSITEM_RSSI	Cell modem RSSI (received signal), 1 (bad) to 31 (good)
SYSITEM_ETHLINK	0 if no hardware link detected. 1 if ok
SYSITEM_ETHCONN	0 if not connected via Ethernet. 1 if connected
SYSITEM_CELLCONN	0 if not connected to cell network. 1 if connected
SYSITEM_YEAR	Current year (eg 2020)
SYSITEM_MONTH	Current month (1-12)
SYSITEM_DAY	Current day of the month (1-31)
SYSITEM_WDAY	Current weekday (0=Sunday, 6=Saturday)
SYSITEM_YDAY	Current day of the year
SYSITEM_HOUR	Current hour (0-23)
SYSITEM_MINUTE	Current minute (0-59)
SYSITEM_SECOND	Current second (0-59)
SYSITEM_EPOCH	Epoch time stamp (seconds since 1970-01-01 00:00:00 UTC
SYSITEM_RND	Random number 0-2^{32}

Returns the current status value of the subject requested.

Example usage

```
sys(SYSITEM_MONTH)
```

Returns the current month number.

```
sys(SYSITEM_RND) % 47
```

Returns random value between 0 and 46

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/sys

Last update: **2020-01-24 18:15**

Tangent

tan(a)

Return the tangent of a.

Description

```
tan( a )
```

Return the tangent of a in radians.

Parameters

a	Value

Return value

Tangent of a in radians

Example usage

```
tan( PI/4 )
```

Returns 1.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/tan

Last update: **2019-09-02 18:54**

This (field/alarm number)

THIS

Holds the index number of the current field or alarm

Description

```
THIS
```

THIS is useful to self-reference a field or alarm. For example to let a field self-increment, the following expression can be used:

```
f(THIS)+1
```

THIS is also useful in reverse math expressions to scale the field value to a suitable register value. This example scales a 0-10 mA value in the field to uA and offsets it to industry standard 4-20mA.

```
f(THIS)*1600+4000
```

Parameters

Not applicable

Return value

Returns the index value of the current field or alarm.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/this

Last update: **2023-08-13 07:24**

Time to

time(hour, minute)

Return the difference between current local time and the given time.

Description

```
time( hour, minute )
```

This function returns the difference in minutes between the current local time of the controller and the time given in the command.

Parameters

`hour`	Hour (0-23)
`minute`	Minute (0-59)

Return value

Returns the difference in minutes between the current local time of the controller and the given time. The return value is the difference in minutes, in the range -720 to 720. The return value will be negative if the given time is in the future, and positive if the given time is in the past.

Example usage

Assuming local time is 22:00;

```
time( 21, 45 )  // returns 15
```

```
time( 2, 30 )  // returns -270
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/time

Last update: **2019-09-02 18:55**

Uptime

Uptime()

Return number of seconds since last reset.

Description

```
Uptime()
```

This function returns the number of seconds since last reset.

The cause of a reset may be:

- Power being removed/restored
- Software reset due to firmware update
- Software reset requested by the user
- Software or hardware error (Watchdog reset)
- Waking up from sleep mode

Parameters

none

Return value

Returns the number of seconds since the last reset.

Example usage

```
Uptime()
```

Will return 86400 after 24 hours running.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/uptime

Last update: **2019-09-02 18:55**

Week

Week()

Returns the current week number.

Description

```
Week()
```

Returns the current week number of the year.

Parameters

none

Return value

This function returns the current week number of the current year.

Example usage

```
Week()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/week

Last update: **2019-09-02 18:55**

Weekday

Weekday()

Returns the current weekday.

Description

```
Weekday()
```

Returns the current weekday, 0 (Sunday) to 6 (Saturday).

Parameters

none

Return value

This function returns the current weekday in the time zone set for the ezeio.

Example usage

```
Weekday()
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/weekday

Last update: **2019-09-02 18:55**

User defined memory

v(address) / vf(address)

Returns the value of a user-defined cell.

Description

```
v( ADDRESS )
```

```
vf( ADDRESS)
```

Returns the value of the cell.

v() assumes the cell contains an integer value.

vf() assumed the cell contains a floating point value.

Parameters

ADDRESS	the address of the cell (0-511)

Return value

This function returns the value of one of the user-defined cells.

Example usage

```
v(17)
```

NOTE: First appeared in firmware 21071901

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/year

Last update: **2021-07-20 01:04**

Year, Month, Day

YMD()

Return encoded year-month-day

Description

```
YMD()
```

This function returns the current local date as YYYYMMDD

Parameters

none

Return value

Returns current date as YYYYMMDD.

Example usage

```
YMD()>20190515   // returns 1 if date is past May 15, 2019.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/expref/ymd

Last update: **2019-09-25 00:27**

IMPORTANT INFORMATION

Important considerations

The ezeio is designed for use in a dry and clean location, such as indoors or in a protected wiring/electronics cabinet. Do not expose the ezeio to rain, direct sunshine, chemicals or excessive dust. Avoid extreme temperatures. Please see technical specifications for acceptable ranges.

The ezeio is a low voltage device. Never connect high voltages or currents to the ezeio, and only use the supplied or specified power supply to power the ezeio.

Do not run wires that connects to the ezeio (Ethernet, Modbus, CAN, SDI-12, inputs/outputs, antenna), together with high voltage/current wiring. Use separate conduits whenever possible, and avoid environments with excessive RF or magnetic interference, as this may cause malfunction or even damage the ezeio.

Take necessary precautions to avoid large static discharges onto the ezeio or its connections.

WARNINGS

To reduce risk of fire or electric shock, do not expose this product to rain or moisture. This product is designed for use indoors and only with the supplied AC adapter.

Unplug the AC adapter before working with any connections.

The ezeio® is a low voltage device.

Never connect high voltage to the inputs or outputs.

No user serviceable parts inside.

Do not open the ezeio enclosure. Breaking the warranty seal will void the warranty.

The radio is designed to have an antenna connected at all times, and may take harm from running without the antenna. **Always make sure the antenna is**

connected before connecting the ezeio® to power.

Do not use the ezeio in safety critical systems.

The ezeio depends on external services and systems to function. Do not use in applications where a functional failure can cause harm to personnel or property.

Suitable applications

The ezeio is not a fool-proof device. If set up incorrectly, it may not provide the intended functionality.

The ezeio is not trivial. While we have put a lot of effort into making the ezeio as easy as possible to use, it still requires experience with electrical circuits and logic flow. The ezeio is designed for professionals, not as a consumer product.

Do not use the ezeio for safety critical applications. Always consider what effects a malfunction of the ezeio and/or any external device/sensor/system may have, and ensure no damage to equipment or personnel can result from a failure in ezeio hardware, programming, logic, misuse of the system or any related systems or connections.

The ezeio is not designed with hardware redundancy. The ezeio is not designed for safety-critical applications.

Third party systems

The ezeio communicates using the Internet. Communication interruptions should be expected, and may be more or less frequent depending on region, infrastructure, weather, network service, political issues etc. Network issues like these are outside the control of eze System.

The ezeio system is designed to be highly reliable and robust. While we continuously monitor and maintain the hardware, firmware and software that is under our control, some features and functions relies on third party services. These include, but are not limited to;

- Network connectivity (cellular and/or hardwired)
- Email, SMS and other messaging services
- Internet hosted services

We carefully select reliable service partners, but eze System has no direct control of these services, and can not make any claims as to availability, suitability or accept any responsibility for any third party service.

Terms of use

See https://eze.io/login/tos.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/important_information/start

Last update: **2025-01-02 20:11**

Indicators

There are three LED indicators on the front of the ezeio. These can provide valuable information about the status of the unit and how the communication is working.

Indicator	Purpose
Ethernet	Shows the status if the Ethernet connection
Cellular	Shows the status of the cellular wireless connection
Server	Shows the status of the server communication and if the internal logic is running

If neither the Ethernet or the Cellular connection is successful, the connection to the server is not possible, so the Server LED will only be green if at least one of the Ethernet or Cellular connection are also green.

Startup sequence

Immediately on startup or reset, the LED's will show a green/red 'running' pattern for a few seconds.

If during startup the Ethernet and Cellular LED's flash RED rapidly while the Server LED is off, this means the input voltage is too high (>30 VDC).

If during startup the Cellular and Server LED's flash RED rapidly while the Ethernet LED is off, this means the input voltage is too low (<10.5 VDC).

Normal operation

During normal operation, each of the LED's will indicate the status if it's resource independent of the other LED's. Please check each LED's pattern against the below tables.

The ETHERNET indicator

	OFF	No Ethernet (physical) connection
	GREEN	Ethernet connection established
	slow RED blink	Ethernet connection disabled (in software)
	RED - RED - RED - pause	Waiting for DHCP server
	RED - RED - GREEN - pause	Fixed IP set. Looking for DNS.
	RED - GREEN - GREEN - pause	DHCP address acquired. Looking for DNS.
	slow GREEN blink	Fixed IP set. No gateway programmed. LAN comm only.

The CELLULAR indicator

	OFF	Cell modem is off / not available

🟢	GREEN, short off 1-5 times	Connected to network. 1-5 signal quality (5 is best)
🔴	fast RED blink	Cell modem powered off by software
🔴	short RED pulse 1s apart	Trying to start up cell modem
	short RED pulse 20s apart	SIM not detected / faulty / locked
🔴	RED - RED - RED - pause	Searching for cellular carrier network
🔴	RED - RED - GREEN - pause	Network found. Trying to register.
🔴	RED - GREEN - GREEN - pause	Registered with network. Trying to get data connection.
🔴	RED - GREEN alternating	Cellular connection lost. Trying to re-establish.
🔴	RED	Cellular connection lost. Waiting to reconnect.
🔴	RED slow blink	Cellular connection lost. Required to back off before retrying.

The SERVER indicator

🔴	fast RED blink	No connection to server. Logic is not active.
🔴	RED	No connection to server. Logic is running.
🟢	fast GREEN blink	Connected to server. Logic is not active.
🟢	GREEN	Connected to server. Logic is running.

It is normal that the server LED will flicker while green. This indicates normal communication activity.

Button function during startup

The button on the ezeio can be used to override normal startup.

DEFAULT COMMUNICATION SETTINGS

To force default communication settings (Ethernet DHCP and default cell modem settings), simply hold the button while power is applied, and keep holding the button until the LED startup roll is finished. Then release the button. This will also prevent any scripts and drivers to run.

The ezeio will revert to the programmed settings after the next reset.

KNOWN/FIXED ETHERNET IP

The Ethernet IP setting can be forced to 192.168.1.77 by holding the button when power is applied, but release the button during the LED startup roll. This only affects the Ethernet IP address. All other functions operate as normal.

The ezeio will revert to the programmed settings after the next reset.

HALT SCRIPTS AND DRIVERS

By pushing down the button during the startup roll, and holding it until the roll completes, the ezeio will start without running any scripts or drivers. The scripts can be started again by pushing the button once, or by restarting the ezeio.

STOPPING SCRIPTS DURING RUNTIME

The scripts and drivers can be stopped any time during runtime by clicking the button twice. To restart the scripts and drivers, push the button once. If the ezeio is restarted the scripts will still start normally.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/indicators/start

Last update: **2023-10-23 23:27**

Installing the ezeio

The ezeio itself is very easy to install.

Start by mounting the unit where you need it. Make sure it's in a dry location and not exposed to excessive dust or extreme temperatures.

More details on how to mount the hardware

Before connecting sensors and devices, make sure the antenna is mounted on the antenna bracket, and placed in a location free from large metal items that could shield the signal.

Proper antenna placement is critical for reliable operation.

- ALWAYS keep the antenna connected
- Keep the contacts dry and clean
- Do not damage the antenna cable
- Never mount the antenna inside a metal enclosure
- Mount the antenna away from metal objects and other cables

Then apply power using the power supply that came with the ezeio, and make sure the ezeio SERVER light goes green. This may take a few minutes that first time, but should not take longer than 5 minutes.

More details on the power requirements

After confirming the SERVER light go green (meaning the ezeio communicated with the cloud servers), disconnect power and continue adding your sensors and devices.

Connecting to the ezeio inputs

Connecting to the ezeio outputs

Modbus/RTU connections

Modbus/TCP connections

SDI-12 connections

CANbus / J1939

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/installation/start

Last update: **2024-07-22 18:15**

Installing the hardware

Before mounting the unit, please remember to register it online. The registration code is on the side label, and may be difficult to access after the unit has been installed.

The ezeio can be installed inside a larger enclosure, or used by itself mounted on a flat surface or DIN track.

The ezeio is designed for mounting on a standard 35mm DIN track. Simply snap the unit in place. There is a release tab that can be easily accessed by pulling our the green screw terminals. Pull the tab down to release the unit from the DIN track.

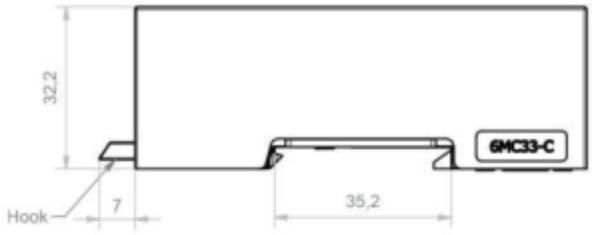
There are no restrictions on orientation, but for the purposes of this document we will refer to 'up' as the side where the antenna and Ethernet connectors are, and 'down' as the side where the green screw terminals are. The ezeio has no mechanical moving parts, and does not generate any significant heat.

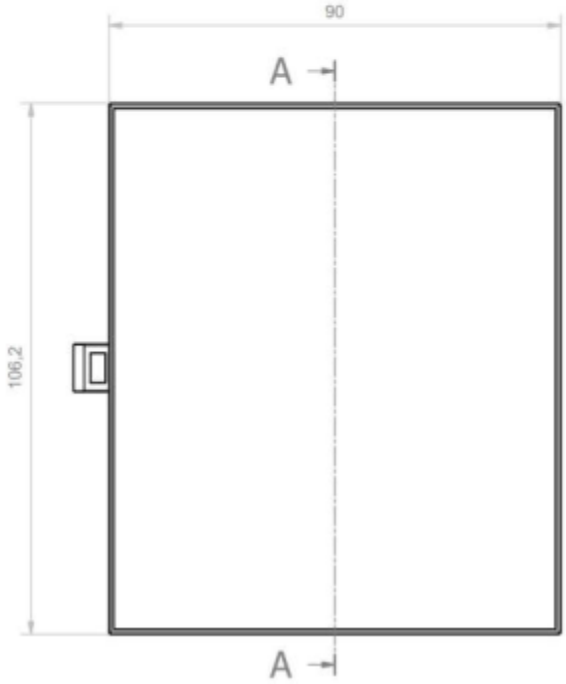
The three LED's on the front communicate valuable information about the communication status. Please make sure the LED's are easily visible for the user. *For information on the LED indication see* Indicators

Required space

The ezeio unit measures 107mm x 90mm x 33mm without terminals and connectors. We recommend allowing at least 50mm for connectors and wires on the top and bottom.

Expansion modules, if used, will connect on the right side of the unit, so plan for leaving open space as needed.





Download STEP drawing.

Screw mounting

If you are not using DIN rail, the ezeio can be mounted using two screws and the keyholes on the back. The screws need to be 74.35mm (2930 thou) center-to-center.

When using the ezeio with side-dock expansion modules, always use a DIN rail. Do not use screw mount with side-dock modules.

Antenna

Proper antenna placement is critical for reliable operation.

- ALWAYS keep the antenna connected
- Keep the contacts dry and clean

- Do not sharply bend or pinch the antenna cable
- Never mount the antenna inside a metal enclosure
- Mount the antenna away from metal objects and other cables

Use only the antenna supplied with the ezeio. The antenna and antenna mounting bracket has excellent performance when mounted correctly. Please follow these instructions carefully.

The antenna bracket can be mounted on a horizontal or vertical surface using two regular screws, or on a pole using a U-bracket. Make sure the bracket is mounted horizontally and the antenna pointing straight up.

Keep the antenna away from metal objects and cables.

Never mount the antenna inside a metal box.

Antenna on bracket

After mounting the antenna bracket, position the antenna on the bracket, and tighten the nut.

If it's absolutely necessary to use an extension cable for the antenna, keep it as short as possible and use high quality coax cable. Note that the losses in extension cables are significant. For example, 6m (20ft) of RG58 cable attenuates roughly 50% of the signal.

Wiring

The green screw terminals can be pulled out from the body of the ezeio to make access and wiring easier. Make sure you open the screw terminals properly before inserting the wire. Avoid running multiple wires into the same terminal. If you need multiple wires to connect to the same terminal, for example if you have a lot of ground connections, use an external terminal solution. Do not tighten the screws too hard, and check each wire by pulling it gently to make sure it has a good connection.

The antenna connector shall be finger tight. Do not use a wrench. Make sure the antenna connector is

free of dirt or water before connecting.

The Ethernet connector is a standard RJ45 network connector. There should be a solid 'click' when the plug is inserted. To release, press the tab and pull gently.

Do not use force on any of the connectors. With the exception of a small flat-head screw driver for the screw terminals, no tools are required to connect/disconnect the wires.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/installation/mounting

Last update: **2025-08-18 16:09**

Power supply and power connections

The ezeio shall be powered from a DC (Direct Current) source, such as a AC/DC adapter, power supply or battery. Use only supplied or recommended power supplies with the ezeio.

Self draw

	Min	Max	Comment
Input voltage	11V [2)]	30V	DC
Current	30mA [3)]	400mA [4)]	70mA average at 12V
Power	50mW	5W [5)]	0.8W average when using CAT-M1

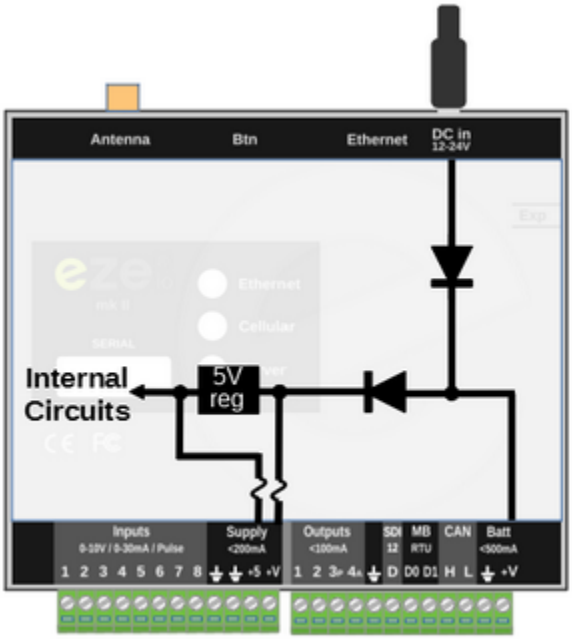
Note; the above power consumption does not include any external connections.

The ezeio will start operating as soon as power is applied. There is no power switch. The ezeio is designed to stay powered at all times. There are no moving mechanical parts or anything that causes wear on the hardware. It generates very little heat. There is no need for ventilation.

Feeding power to the ezeio - The power can be connected either via the DC Barrel jack on the top of the unit, or on the screw terminal. Do not supply power from more than one source at the same time.

Supplying power to sensors and peripherals - The terminals in the middle, marked 'Supply ,200 mA, +V' and +5 are designed to supply power to connected sensors and peripheral devices. If the ezeio is powered from the DC barrel jack, the screw terminals marked Batt, < 500 mA, '+V' can be used to route power to other devices as well. Please be careful to not overload the power supply. Do not load the 'Batt' output with more than 500mA. Do not load the 'Supply' terminal with more than 200mA.

Only the Supply +5 & +V terminals are protected with resettable fuses that will limit the output to about 200mA. There is no fuse on the Batt +V terminal. Overloading this terminal could damage the ezeio. If your sensors require more than 200 mA or you are unsure of their current draw, bypass the ezeio's circuitry and directly connect them to an adequate power supply.



All the terminals marked with a ground symbol are electrically connected together internally. This is also the same ground as the sleeve of the barrel jack, and the antenna connector.

Power connections on the green terminal

Terminal	Max current	Purpose
Supply +5	200mA	Regulated 5V DC output for powering sensors/GPS that require a regulated 5V supply.
Supply +V	200mA	Same voltage as the supply voltage. Fused with PTC. Intended to be used to power sensors that connects to the discrete inputs.
Batt +V	500mA	Same voltage as the supply voltage. Not fused. May be used as either a power input [6] or output to power sensors and devices. Please make sure not to exceed 500mA. If higher current is required, use a separate power supply.

Excitation current for passive sensors

When an input is configured for Resistive or Pulse mode, an internal switch will apply +5V through a 4870 Ohm resistor to the input terminal. There is usually no need for external excitation.

Outputs

The outputs emit the supplied DC voltage when turned on. Do not exceed 100mA on output 1-3. Do not exceed 20mA on output 4 (analog).

2)

The ezeio will operate on as little as 7V, but the analog output will not be able to supply 10V output when the supply voltage is less than 11V

3)

In normal operation and at 30V [4] [5],

Short bursts, while transmitting [6]

Only use the Batt +V terminal for input if the barrel jack is not used.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/installation/power

Last update: **2021-09-27 22:54**

Backup power options

Lithium Ferrite Phosphate batteries has much better life span than lead-acid/AGM batteries. They also include protection circuits that prevent them from discharging too much, while lead-acid batteries are usually damaged permanently if they are allowed to completely discharge.

The ezeio is shipped with the charger set for LiFePO4 charging. **DO NOT CONNECT a lead-acid battery** to the charger unless you adjust the voltage appropriately

Below are some sources for suitable LiFePO4 batteries:

Compatible batteries

Brand	Distributor	Link
Nermak	Amazon	https://www.amazon.com/dp/B0B78HTRDL
BOTKU	Amazon	https://www.amazon.com/dp/B0CLB1Y8BR
ERYY	Amazon	https://www.amazon.com/dp/B0CT8H2F8L
GoldenMate	Amazon	https://www.amazon.com/dp/B0CN38TNDR
Donghot	Amazon	https://www.amazon.com/dp/B0D55Z8HRQ
Icanmeto	Amazon	https://www.amazon.com/dp/B0D2QDNFLC
Howell	Amazon	https://www.amazon.com/dp/B095K38M6C
Haya	Amazon	https://www.amazon.com/dp/B0BXGSPF7R

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/installation/power/backup

Last update: **2025-07-31 23:15**

Introduction to the ezeio

Thank you for purchasing or considering the ezeio® !

What is the ezeio®?

The ezeio is a complete solution for monitoring, alarming, control and automation of commercial and industrial equipment.

The ezeio hardware connects to sensors, meters, thermostats, VFD’s and other control devices locally via a number of industry standard interfaces.

It connects securely and seamlessly via the Internet (Ethernet or Cellular) to the eze.io cloud application, where the user can access all data in real time as well as historical.

All configuration settings and programming is done via the cloud interface, allowing multiple concurrent users, automatic synchronization and secure access from anywhere without any special software or setup.

Common applications include:

- Monitoring energy meters (electrical, water, gas)
- M&V applications (energy saving, improvements)
- Monitoring refrigeration systems (temperature, pressure)
- Controlling & monitoring HVAC systems (thermostats, room sensors)
- Construction site monitoring (cement curing, heating/cooling, alarms)
- Automating thermal energy storage systems (*TES)
- Technical alarm systems (fan monitors, temperature, tank levels)
- Lighting control, monitoring and scheduling/automation
- Irrigation control, monitoring and scheduling/automation
- Battery / EV charging, monitoring and control
- Vehicle tracking, monitoring (GPS)

The ezeio system is designed for easy deployment in geographically spread out, multi-dicipline applications where traditionally several single-purpose systems were needed.

The ability to support different kinds of sensors, meters, actuators and applications within a single, low cost yet complete and secure system makes the ezeio system unique.

Model information

Part number	Model	Cloud connectivity
EM2010	ezeio Cat-M1	Ethernet + IoT (4G/5G) cellular

Please contact eze System for other cloud-connectivity options.

Local connectivity includes;

	Quantity	Comment
Discrete inputs	8	Individually configurable for 0-10V, 0-30mA, Resistor, Thermistor, Pulse
Discrete outputs	4	2 digital (voltage output), 1 Pulse Width, 1 Analog 0-10V
SDI-12 serial	1	Serial I/O for SDI-12 sensors or GPS receiver
Modbus/RTU	1	Serial (RS485) port supporting the Modbus/RTU protocol
Modbus/TCP	1	Ethernet (TP10/100)[7] supporting the Modbus/TCP protocol
CAN	1	CANbus compatible port

Additional inputs and outputs can be added using ezeio expansion devices and/or third party devices.

Standard & Optional Features; [8]

- Data logging with over 50 days of local buffer storage [9]
- Flexible alarming functions
- Reporting to email, SMS, Voicecalls, Pushover, API and more
- Direct control from web
- Complete remote access to configuration and programming
- Easy to configure local logic
- Scheduling with 10 year calendar
- Powerful Scripting running on hardware
- REST API
- Built-in Real Time Clock
- Low power consumption (<1Watt)
- Smart drivers for common Modbus, SDI12 and CAN devices
- Multiple dashboards
- Easy management and hierarchical groups
- Unlimited user access with multiple permission levels from view only to full admin

Functional overview

Communication

The ezeio connects to the Internet via Cellular or Ethernet. A cellular modem is built in to the unit, but an external antenna is required. By default, if there is an Ethernet connection that allows Internet access, *Ethernet is prioritized*. If for any reason the Ethernet connection is not available, the ezeio will automatically switch to use the cellular connection. If the Ethernet becomes available after the cellular connection has been established, the ezeio will switch to the Ethernet after a few seconds.

All communication via Ethernet and Cellular is secure, encrypted and controlled. The ezeio is always a 'client', which means that it will actively seek a connection to the servers. Typically this means there is no special network configuration required, and the ezeio should be expected to work on any network setup - even behind routers and firewalls. However if the network blocks certain traffic or uses proxy

technologies, there is a risk that the ezeio will not work on that network, and fall back to the cellular connection if coverage is available.

Ethernet specifications

Hardware	Ethernet TP10/100BASE-T, Auto MDI-X
Max cable length	100m (320ft)
Addressing	DHCP (default) or static IP
Protocol	Proprietary, encrypted, Port TCP/443
Security	PKI handshake, 128 bit encryption

Cellular specifications

Antenna	SMA, standard rugged knob-style antenna with mounting bracket. 1m cable.
Technology	LTE CAT-M1
Carrier support	Global, multi-carrier with localization
SIM card	Built-in [10)]

Inputs and Outputs

The ezeio connects to local sensors and devices through the green screw terminal. There are several connection options.

Port type	Qty	Description
Inputs	8	0-10V, 0-30mA, Resistor, Thermistor, Switch, Pulse count/rate
Outputs	4	On/Off (2), 50Hz PWM (1), Analog 0-10V (1)
SDI-12	1	For environmental sensors or GPS
Modbus/RTU	1	Modbus/RTU master, RS485, 2400-115200bps (19200 bps standard)
Modbus/TCP	1	Server or Client, standard Ethernet
CAN	1	For I/O expansion, configurable for other protocols
Antenna	1	Cellular LTE CAT-M1 (optional GSM 2G/3G) SMA standard
Power supply		Barrel DC jack input or screw terminal
Power output		Terminals for fused or passthrough DC as well as regulated +5VDC
Ground		Inputs, outputs and supply use a common ground. Multiple ground terminals

Discrete inputs

There are eight (8) discrete inputs on the ezeio. Each input can be individually configured to monitor voltage, current, resistance or pulses. The configuration is done via the web interface at https://eze.io.

More information about connecting to the inputs is available here.

Discrete outputs

There are four (4) discrete outputs on the ezeio. The function of each output is completely controlled by software.

Output 1 and 2 are purely on/off. The will output the supply voltage when on, and no voltage when off.

Output 3 is a Pulse Width output. It will output a 50Hz rectangular wave, with duty cycle from 0-100%. The high state is the supply voltage. This output can also be used as a regular on/off output by only setting it to 0 and 100% in software.

Output 4 outputs an analog voltage between 0 and 10V. The software controls this in 100mV steps (0-100%).

More information about connecting to the outputs is available here.

SDI-12

SDI-12 is a bidirectional communications protocol for relatively simple sensors, and commonly used for monitoring environment data. See http://www.sdi-12.org for more details.

The ezeio supports the standard SDI-12 protocol and multiple drivers are available for different types of sensors. The SDI-12 port can also be used as a generic serial input to read GPS data and other nonstandard devices.

More information about SDI-12 is available here.

Modbus/RTU

Modbus/RTU is also a bidirectional communication protocol, but faster and more powerful than SDI-12. Developed 1979, it is still by far the most common protocol in industrial automation systems, and supported by multiple manufacturers in many industries.

The ezeio supports standard Modbus/RTU as a 'master', meaning the ezeio controls the communication between multiple devices. eze System provides drivers for many types of devices from various manufacturers.

Several devices can be connected at the same time on the same Modbus/RTU cable. The protocol is very robust, uses low cost bulk wires and can be used in noisy environments and over long distances.

More information about Modbus/RTU is available here.

Modbus/TCP

Modbus/TCP is similar to Modbus/RTU, but uses Ethernet network signalling to connect to the devices. This allows for faster data transfers than over Modbus/RTU and has the benefit of using standard plug-and-play Ethernet hardware (switches, cabling). Modbus/TCP can be routed using standard WiFi hardware, allowing one ezeio to communicate directly to another ezeio or other device within range. The downside is somewhat more complex configuration, shorter range without using additional hardware and higher device cost.

The ezeio supports standard Modbus/TCP as both a 'server' and a 'client'. There are multiple drivers available for various devices.

More information about Modbus/TCP is available here.

CAN

The ezeio has a CAN port, which by default is used for expanding the system with additional I/O hardware and terminal made by eze System. When used for this purpose, the protocol is proprietary to eze System, and other devices cannot be connected to the CAN port.

The CAN port is controlled by software and may be configured for other protocols, such as NMEA2000, J1939, DeviceNet or CANopen. Please contact eze System for options.

Operation

Startup

The ezeio will automatically connect to the cloud services when power is applied. It is designed to be always-on, and always stay connected to the servers. If the communication is interrupted, the ezeio will automatically attempt to reconnect until the connection is restored.

Configuration

The user can configure each ezeio with multiple sensors and devices, to collect data (log), set alarm conditions, construct automation logic and directly remote control the units.

All configuration is managed through the cloud portal at https://eze.io. Any changes to the configuration of an ezeio unit is stored in the cloud databases, and automatically synchronized to the unit as soon as possible - typically within a few seconds assuming the unit is powered up and connected.

Data acquisition

Captured data is first buffered in the built-in memory, which holds over 50 days of 10 minute interval data, or 300000 samples at other intervals. Buffered data is uploaded automatically to the cloud

databases as soon as the connection allows, and can be easily viewed in real time or historically, and downloaded as CSV or via API. The data is kept in the servers databases for a minimum of three years.

Events

If the ezeio is programmed to trigger alarm events, messages can be sent via email, SMS, Pushover or voice calls to any number of recipients. The conditions that trigger such events is completely user programmable, and can be changed at any time from the portal.

Logic

The user may construct any logic using a powerful scripting language, and download this logic into the ezeio unit(s). Such logic will run independent of the servers.

Management system

On the server/portal side, the user may manage multiple units, provide access to multiple users with varying privileges, design dashboards, reports and have a quick overview of the real-time state of all units. The system scales easily from a single ezeio to thousands.

ezeio hardware and firmware

The ezeio hardware is highly optimized for its purpose. An embedded microcontroller (ARM Cortex-M4) manages all the communications, logic, I/O's, ports and memories. There are several different types of memory chips for programs, configuration, recorded data and status, as well as power management, I/O interfaces, real-time clock/calendar etc. The inputs and outputs are protected against excessive voltages/currents and the hardware is designed for wide temperature and rugged for interference expected in harsh environments.

CPU	ARM Cortex-M4 @ 120MHz
Memory	On-chip flash and RAM for system and configuration
	32MB flash buffer for log data
Interfaces	Ethernet, CAN, Modbus RS485, SDI-12, Discrete I/O
Cellular	LTE CAT-M1 global
RTC	Automatically synchronized with servers. Supercap backup (~24h)
Power	12-24VDC supply (switching [11]), <1W self-draw average

There are no provisions for directly connecting to the hardware for configuration. All configuration is managed through the servers https://eze.io.

Embedded software

The ezeio runs a proprietary software stack, designed from scratch by eze System. The software is embedded in the microcontroller and runs immediately when power is applied. The configuration, drivers and user designed logic is synchronized automatically from the cloud servers, and runs locally on the ezeio.

Each ezeio unit has unique keys for validating with the ezeio cloud system and will refuse any other attempts to communicate than with the ezeio servers. The system is inherently secure, designed for direct use on the public Internet with no need for additional VPN, firewalls or other third-party security measures.

In case the communication is interrupted, all logic continues to run, and any messages/log data is buffered and automatically sent to the servers when the communication is restored.

Capacity and performance

Field width	64 bit, double precision
Max fields	90 (logged data points)
Log buffer, fixed	56 days, 10 minute interval, all fields and system status
Log buffer, configurable	262000 blocks of 11 fields each
Log interval	10 minute standard, configurable down to 5s, configurable to record mean/max/min/snapshot/trend
Event buffer	8000 events
Alarms	300, individual condition, holdoff and restore settings. Each alarm may trigger up to four separate actions.
Schedules	30 daily/weekday or by 10-year calendar
Register width	32 bit, as signed integer or floating point
Max registers	2000 total, max 150 per device
Devices	Up to 40 total
Hardware inputs	8 on main hardware. Expandable to 64 with ezeio expansion. 12 bit resolution, 1000Hz sampling speed (fixed)
Hardware outputs	4 on main hardware. Expandable to 74 with ezeio expansion. Types: On/Off max 200mA active output, PWM 50Hz/RC servo, Analog 0-10V 0.1V step
Expression processing	10Hz, fields and alarms
Script execution	Approx 200k instructions per second
Script language	See Script reference

[7)]

Same physical port may be used for cloud connectivity

[8)]

Some features require additional paid subscription

[9)]

Logging intervals 10 min, 5 min, 2 min, 1 min, 15 sec, 5 sec

[10)]

SIM is provided by eze System, and not user replaceable

[11)]

Power consumption mostly independent of input voltage

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/introduction/start

Last update: **2022-07-07 16:59**

ezeio MkII I/O Expander (TEMPORARILY UNAVAILABLE)

Description

This device is designed by eze System to provide additional input and output capacity to the MkII ezeio. Each I/O Expander adds 8 discrete inputs, 8 digital outputs and 2 analog outputs. Multiple I/O Expanders can be connected to one ezeio MkII controller. Two versions of the MkII I/O are currently available EM2088 (eze-CAN only) and EM2088-MB which provides the option of connecting via Modbus RTU. Docking to the ezeio's EXP port (eze-CAN) allows the expanders I/O's to function as if they were onboard resources of the ezeio, including the use of ezeio drivers (discrete input, pulse input, thermistor, and digital output) for configuration. Utilizing the Modbus RTU port allows the I/O Expander/s to be located remotely (up to 1000 meter of hard wired serial bus length). For the Modbus RTU option a dedicated driver is used to configure the 8 inputs.

Do not connect both serial buses. Only one form of serial bus communication can be used to connect to the ezeio controller.

	eze-CAN	**Modbus RTU**
Bus length	Docks directly	1000 meters
Sample speed	100 ms	Varies base on # & performance of MB slaves
Driver options	Same as ezeio controller	8I8O Expansion Module
Power	Supplied thru ezeio controller EXP port	Barrel jack or Modbus port

Model information

Part number	**Model**	**System connectivity**	**For use with**
EM2088	ezeio I/O Expander	eze CAN	ezeio controller only
EM2088-MB	ezeio I/O Expander + MB	eze CAN / Modbus RTU	ezeio controller or 3rd party Modbus RTU master

Local connectivity includes;

	Quantity	**Description**	**Comment**
Discrete inputs	8	Individually configurable for 0-10V, 0-30mA, Resistor, Thermistor, Pulse	
Discrete outputs	8	2 digital (voltage output)	Outputs supply voltage
Analog outputs	2	0-10VDC (voltage output)	Recommend greater than 12V supply to insure full 0-10 range
Modbus/RTU	1	Serial (RS485) port supporting the Modbus/RTU protocol	Only operates as slave device
EXP port	2	eze CAN Expansion ports for docking to ezeio and Expanders	Only operates as client of ezeio

Discrete inputs

There are eight (8) discrete inputs on the ezeio. Each input can be individually configured to monitor voltage, current, resistance or pulses. The configuration is done via the web interface at https://eze.io.

More information about connecting to the inputs is available here.

Discrete outputs

There are 10 outputs on the I/O Expander. The function of each output is completely controlled by software.

Output 1 thru 8 are purely on/off. They will output the supply voltage when on, and no voltage when off.

Analog (Output) 1 & 2, output an analog voltage variable between 0 and 10V. The software controls this in 100mV steps (0-100%).

More information about connecting to the outputs is available here.

Functional overview

As described above, the I/O Expander provides the ezeio with additional input and output capacity. The two options for communicating with the ezeio require different installation and set-up processes. There are also performance difference between the two modes of operation. Below are complete instructions and operation details for connecting via the eze CAN EXP port followed by those for connection via Modbus RTU.

Connect via EXP port

Features & Benefits

- Dock side by side on DIN rail using 6 pin header
- Dock up to 7 I/O Expanders to one ezeio MkII
- Inputs are sampled at the same rate as native inputs on the ezeio MkII controller

---

Physical installation

- Disconnect power from the ezeio
- Install the 6 pin header into the EXP port on the left side of the I/O Expander. Take care to properly align the pins, as the DC supply voltage can damage the CAN buffer if connected to com pins.
- With both devices released from the DIN rail, align the 6 pin header with the ezeio's EXP and press

the devices together.
- Hang connected devices on the top rail of the DIN rail. Then press on the bottom/center of the label on each device to snap onto the DIN rail.

Changing device ID/address

Default address is 1, unless otherwise noted

- Connect or power cycle the ezeio
- In the first 30 seconds of start-up, press the button (at the top of the I/O Expander) 3 times
- The I/O Expander's LEDs will begin to roll red and green. Enter the desired address by clicking the button (5 clicks = address 5)
- The I/O Expander's LEDs will confirm the entry by flashing the address number (5 flashes = address 5)

Adding Inputs and Outputs to eze.io configuration

Use any of the drivers listed below to add inputs and outputs to the ezeio controllers configuration

- Discrete input driver
- Pulse input driver
- Thermistor driver
- Discrete Digital Output driver

- eze Expander I/O driver (legacy) *Allows basic configuring of all 8 discrete input in one driver*

Connect via Modbus RTU

In this communication mode the I/O Expander functions like a 3rd party device. Since Modbus RTU is an open protocol, it could be sold separately and use with almost any Modbus master.

- When connecting to an ezeio via Modbus RTU a device driver is required. Search for available I/O Exp drivers on the eze.io configuration's “Device” tab or the Drivers section of this manual.
- If utilizing the I/O Expander in conjunction with a 3rd party Modbus master, refer to the I/O's register map.

One of the effects of using this communication mode is a slower register update rate. Modbus RTU operates at a much slower speed than CANbus/eze-CAN. Update rates will vary based on the number of devices and registers polled. Update rate could range from several times per second to a slow as once per minute, with high traffic and/or comm issues.

More information about Modbus/RTU is available here.

Features & Benefits

- Distribute I/O Expanders along serial bus (up to 1000 meter for wired bus)
- Connect up to 40 I/O Expanders to one ezeio MkII (total of all devices per ezeio = 40)
- Power over bus wire or from local power supply

Physical installation

- 4 wire (3 if powered locally) serial bus (daisy chain) wiring. Twisted pair data cable is recommended (such as Cat3, 5 or 6).
- Connect D0, D1, Ground and +V (if powered over bus) from green terminal on ezeio controller to green terminal (top) of I/O Expander.
- It may be necessary to provide power directly to the I/O Expander based on several factors; distance from ezeio, wire gauge, current draw and supply voltage.

Changing device ID/address

Default address is 1, unless otherwise noted

- Connect or power cycle the ezeio
- In the first 30 seconds of start-up, press the button (at the top of the I/O Expander) 3 times
- The I/O Expander's LEDs will begin to roll red and green. Enter the desired address by clicking the button (5 clicks = address 5)
- The I/O Expander's LEDs will confirm the entry by flashing the address number (5 flashes = address 5)

Changing data bit rate (baud rate)

Default address is 19,200, unless otherwise noted

- Connect or power cycle the ezeio
- In the first 30 seconds of start-up, press the button (at the top of the I/O Expander) 4 times
- The I/O Expander's LEDs will begin to roll red and green. Enter the desired baud rate by clicking the button the corresponding number of times (3 clicks = 9,600)
- The I/O Expander's Modbus LED will flash green when ever it receives a correctly formatted and addressed message.

Register Value = Baud rate

1. = 2,400
2. = 4,800

3. = 9,600
4. = 19,200
5. = 38,400
6. = 57,600
7. = 115,200

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/ioexpander

Last update: **2025-10-20 16:53**

ezeio I/O Exp - Register map

Modbus/RTU slave registers

Read-only input (command 0x03) registers:

Register	Address	Type	Description
30001	0	UINT_16	Input 1 raw value (mV, uA or Ohm)
30002	1	UINT_16	Input 2 raw value (mV, uA or Ohm)
30003	2	UINT_16	Input 3 raw value (mV, uA or Ohm)
30004	3	UINT_16	Input 4 raw value (mV, uA or Ohm)
30005	4	UINT_16	Input 5 raw value (mV, uA or Ohm)
30006	5	UINT_16	Input 6 raw value (mV, uA or Ohm)
30007	6	UINT_16	Input 7 raw value (mV, uA or Ohm)
30008	7	UINT_16	Input 8 raw value (mV, uA or Ohm)
30009	8	UINT_16	Supply Voltage (mV)
30010	9	UINT_16	5V output (mV)
30011	10	UINT_32	Input 1 raw value (mV, uA or Ohm)
30013	12	UINT_32	Input 2 raw value (mV, uA or Ohm)
30015	14	UINT_32	Input 3 raw value (mV, uA or Ohm)
30017	16	UINT_32	Input 4 raw value (mV, uA or Ohm)
30019	18	UINT_32	Input 5 raw value (mV, uA or Ohm)
30021	20	UINT_32	Input 6 raw value (mV, uA or Ohm)
30023	22	UINT_32	Input 7 raw value (mV, uA or Ohm)
30025	24	UINT_32	Input 8 raw value (mV, uA or Ohm)
30027	26	UINT_32	Supply Voltage (mV)
30029	28	UINT_32	5V output (mV)
30031	30	UINT_32	Input 1 pulse count
30033	32	UINT_32	Input 2 pulse count
30035	34	UINT_32	Input 3 pulse count
30037	36	UINT_32	Input 4 pulse count
30039	38	UINT_32	Input 5 pulse count
30041	40	UINT_32	Input 6 pulse count
30043	42	UINT_32	Input 7 pulse count
30045	44	UINT_32	Input 8 pulse count
30047	46	UINT_32	not used
30049	48	UINT_32	not used
30051	50	UINT_32	Input 1 pulse interval (ms)
30053	52	UINT_32	Input 2 pulse interval (ms)
30055	54	UINT_32	Input 3 pulse interval (ms)
30057	56	UINT_32	Input 4 pulse interval (ms)
30059	58	UINT_32	Input 5 pulse interval (ms)

Register	Address	Type	Description
30061	60	UINT_32	Input 6 pulse interval (ms)
30063	62	UINT_32	Input 7 pulse interval (ms)
30065	64	UINT_32	Input 8 pulse interval (ms)
30067	66	UINT_32	not used
30069	68	UINT_32	not used
30071	70	UINT_32	Input 1 pulse frequency (Hz*1000)
30073	72	UINT_32	Input 2 pulse frequency (Hz*1000)
30075	74	UINT_32	Input 3 pulse frequency (Hz*1000)
30077	76	UINT_32	Input 4 pulse frequency (Hz*1000)
30079	78	UINT_32	Input 5 pulse frequency (Hz*1000)
30081	80	UINT_32	Input 6 pulse frequency (Hz*1000)
30083	82	UINT_32	Input 7 pulse frequency (Hz*1000)
30085	84	UINT_32	Input 8 pulse frequency (Hz*1000)
30087	86	UINT_32	not used
30089	88	UINT_32	not used
30091	90	UINT_16	Output 1 status (0/100)
30092	91	UINT_16	Output 2 status (0/100)
30093	92	UINT_16	Output 3 status (0/100)
30094	93	UINT_16	Output 4 status (0/100)
30095	94	UINT_16	Output 5 status (0/100)
30096	95	UINT_16	Output 6 status (0/100)
30097	96	UINT_16	Output 7 status (0/100)
30098	97	UINT_16	Output 8 status (0/100)
30099	98	UINT_16	Analog Out 1 status (0-100)
30100	99	UINT_16	Analog Out 2 status (0-100)
30101	100	UINT_32	Uptime (s)
30103	102	UINT_16	constant value 23456
30104	103	UINT_32	constant value 123456789
30106	105	UINT_16	constant value 57872 (=hE21O)

Read/write holding (command 0x04) registers:

Register	Address	Type	Description
40001	0	UINT_16	Input 1 mode
40002	1	UINT_16	Input 2 mode
40003	2	UINT_16	Input 3 mode
40004	3	UINT_16	Input 4 mode
40005	4	UINT_16	Input 5 mode
40006	5	UINT_16	Input 6 mode
40007	6	UINT_16	Input 7 mode
40008	7	UINT_16	Input 8 mode
40091	90	UINT_16	Output 1 status (0/100)

Register	Address	Type	Description
40092	91	UINT_16	Output 2 status (0/100)
40093	92	UINT_16	Output 3 status (0/100)
40094	93	UINT_16	Output 4 status (0/100)
40095	94	UINT_16	Output 5 status (0/100)
40096	95	UINT_16	Output 6 status (0/100)
40097	96	UINT_16	Output 7 status (0/100)
40098	97	UINT_16	Output 8 status (0/100)
40099	98	UINT_16	Analog Out 1 status (0-100)
40100	99	UINT_16	Analog Out 2 status (0-100)

Input modes:

Mode	Description	Unit
0	0-10V (high impedance)	mV
1	0-30mA (200 Ohm internal shunt to common)	uA
2	Pulse/Resistance (4.87k pull-up to +5V)	Ohm
3	Self-test (4.87k pull-up & 200 Ohm shunt both connected)	Ohm
6	Thermistor, 10k type II	Kelvin * 100
10	Thermistor, 10k type 3	Kelvin * 100
14	Thermistor, 2k2	Kelvin * 100
18	Thermistor, 100k	Kelvin * 100

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/ioexpander/regmap

Last update: **2022-05-27 21:56**

Registration

The registration process will create an account on the servers. This account will allow you to manage multiple ezeio units from one place, invite others to access the units and create a hierarchy of groups under your account for easy structuring of units and co-workers.

Registering for access to the ezeio system is very easy. There are three typical scenarios.
Please consider carefully what applies to you:

1) You want to set up new access for your organization, and you have not previously used the ezeio system.

⇒ Follow the process under First time access to eze.io.

2) You have access to ezeio controllers already, and want to add a new unit to your acccount.

⇒ Follow the process under Adding a controller to your account.

3) You want to give an additional user access to your systems.

⇒ Follow the process under Inviting someone else to your account

If you want to gain access to an ezeio but the system says it's already registered, please contact the owner of the ezeio and ask that they send you an invite.

First time access to eze.io / Creating your account

This action will create a new account. If your organization already has an account (see Adding a controller to your account). The system is designed to allow a single point of access for multiple controllers distributed throughout multiple groups.

To register your first ezeio, create a log-in to the eze.io portal to a new account, you will need the following:

- A valid email address
- The serial number of a new, not previously registered ezeio controller
- The registration code of the same ezeio controller

The serial number and registration code can be found on the side sticker of the ezeio.

If the ezeio controller has previously been registered, please contact the owner of it to get invited to the account (see Inviting someone else to your account).

1. Navigate to https://eze.io and click “Sign up here”.
2. Enter the serial number and registration code of your first controller (Registration code is case sensitive and must be entered with dashes as shown on the sticker).
3. Enter your email address and click the Register button.

Please make sure you enter this information correctly. The system will send a verification email to the address you enter, so you must have access to this email to complete the registration process.

If the email address has already been registered in the system, you will receive an error message, and you should go back to the log-in screen to log in using this email address.

Enter the information requested in the prompts following.

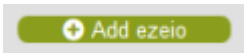
When done, check your email for the verification link. Please note that your login will not work until you clicked the link in the email. The link is only valid for a limited time, so do not delay checking your email for it.

Adding a controller to your account

If you have used the ezeio system in the past, and already have a log-in to the eze.io portal, please navigate to https://eze.io and enter your email and passcode to log in.

If you forgot your passcode, simply click the “Forgot Password” link to reset it.

When logged in, please follow these steps:

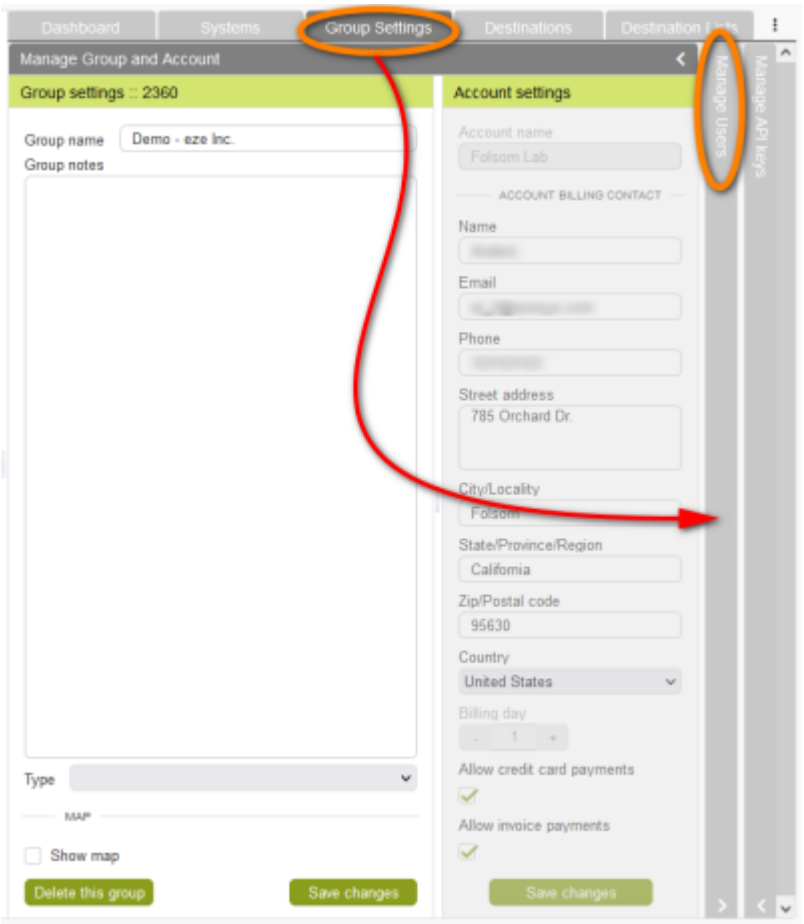
1. Select the group (folder) where you want to add the new ezeio
2. Select the Systems tab
3. Click the Add ezeio button 

4. You will be asked to enter the serial number and registration code, located on the side of the ezeio
5. Click add to complete

The controller will be added to your selected group immediately. Controls are listed in alpha numeric order on the “Systems” table in each group. If the controller was already registered, you will receive an error. In this case, please contact the owner of the hardware.

Inviting someone else to your account

To invite a new user to access controllers on your acc



1. Log in at https://eze.io with your email and passcode
2. Select the group folder where you want the new user to have access
3. Click the Group Settings tab
4. Click the Manage Users Banner (at the bottom of the panel)
5. Click Invite User

The system will ask you for the email address of the new user.

You may enter a short message that will be included in the invitation email. We recommend using this to make it clear to the new user that this is coming from you. The message template will include this text *"You have been sent an invitation to the ezeio management system at https://eze.io. Please click the link to gain access:* The invitation is valid for 72 hours.

You may also select the privileges the new user shall receive. If you do not select any privileges, the new user will only be able to log in and view some very basic information. You may change the privileges for the new user later, after the user has logged in for the first time.

Registration and use of the ezeio system is subject to our Terms Of Use.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/registration/start

Last update: **2023-11-21 16:58**

Script reference

ezeio script programming - the PAWN language

The scripting feature in the ezeio allows the user to implement advanced custom logic and control functionality on the ezeio. For most common applications, scripting is not necessary.

The ezeio uses a powerful script language called PAWN. The syntax is very similar to JavaScript, and should be easily understood by anyone working with programming in similar languages.

The scripting feature of the ezeio is intended for those with previous experience in programming. This manual does not attempt to teach you to write programs. If you have a specific feature that you need help with, please contact eze System.

We strongly recommend reading the following documents to get introduced to the PAWN language:

Introduction to PAWN

PAWN language guide

ezeio script administration

When working with ezeio scripts, you will edit your code on the userscript screen. When clicking “Commit”, the script is compiled on the servers, and automatically downloaded into the ezeio, where it will start to run as soon as the download completes.

Only one user defined script can run at the same time, but the editor allows you to have multiple scripts saved on the servers, and easily switch between them. The currently running script is marked with a checkmark.

If a script fails to compile due to a syntax error, the script will not be sent to the ezeio. If a previous version of the script, or a different script, was running, this will continue to run.

Script resources

The user script can occupy up to 128kB code (compiled bytecode), and up to 16kB RAM.

The estimated requirements for a script is displayed when compiling.

Programming pattern

Although the user script runs in a sandboxed runtime engine, the recommended programming pattern is similar to cooperative multitasking.

This means that you should avoid long-running loops, and instead make use of system callbacks provided in the function library.

The following system callback functions are defined:

@Tick	Called at a regular interval, set by SetTickInterval()
@Timer	Called when one of the millisecond timers expire, see SetTimer()
@Action	Called when an action is triggered due to an alarm event
@Key	Called when a button is pressed on a connected terminal device
@Scan	Called when a code is received from a scanner device

Some examples

The program below will print out “Hello” on the debug console.

```
main()
{
    PDebug("Hello");
}
```

The following example adds the values of field 1 and field 2 together, and writes the result to field 3.

```
// This example adds the value of field 1 and 2 and writes the sum to field 3
// The value of field 3 is updated every 500ms (twice per second)

main()
{
    SetTickInterval(500);       // Make sure the @Tick function is called
twice per second
}

@Tick(uptime)
{
    new f1, f2;             // declare local variables

    f1 = GetField(1);       // set f1 to the value of field 1
    f2 = GetField(2);       // set f2 to the value of field 2

    SetField(3, f1+f2);     // set the value of field 3 with the sum of f1 and
f2
}
```

The example below illustrates the use of a global variable, the use of the main() function to initialize the values, and the @Tick(uptime) call. The program will use field number 1, and count from 100 down to 50, and then starting over.

```
// A simple demo program, counting down from 100 to 50, and then starts over

new myCounter;                  // Declare a 'global' variable

main()                          // The main() function will run on startup,
once
{
    SetTickInterval(1000);      // Make sure the @Tick function is called once
every 1000ms (1s)
    myCounter = 100;            // Initialize the counter with value 100
}

@Tick(uptime)                   // The @Tick function will be called once per
second
{
    myCounter = myCounter - 1;  // Count down the counter with 1

    if(myCounter <= 50) {       // If we reached 50 (or lower)
        myCounter = 100         // ..then set it back to 100
    }

    SetField(1, myCounter);     // Update field 1 with the counter value
}
```

The example below will run a 3-speed fan based on the input from a temperature sensor on field 1. Output 1 is used for low speed, output 2 is medium speed, and output 3 is high speed. The speed is selected based on how much the temperature exceeds a given setpoint (here set to 20.5 degrees C)

```
new @Speed;

main()
{
    SetTickInterval(5000);  // Update every 5s (5000ms)
}

@Tick(uptime)
{
    new Float:temp, Float:diff; // Declare "float" variables to handle
fractions
    new sp = 0;                 // Use sp to find the resulting speed

    temp = GetFieldFloat(1);    // Fetch the current temperature from Field 1
    diff = temp - 20.5;         // 20.5 is our setpoint. diff is the differece

    if(diff > 0.0) {            // Over setpoint?
        SetOutput(1, 100);      // ..yes, so enable low speed
        sp++;                   // count up our speed result
```

```
    }
    else
        SetOutput(1, 0);        // ..no, so shut off

    if(diff > 2.0) {            // 2 degrees or more over setpoint?
        SetOutput(2, 100);      //..yes, so enable mid-speed
        sp++;                   // count up our speed result
    }
    else
        SetOutput(2, 0);

    if(diff > 6.0) {            // 6 degrees or more over setpoint?
        SetOutput(3, 100);      //..yes, so run full speed
        sp++;                   // count up our speed result
    }
    else
        SetOutput(3, 0);

    @Speed = sp;                // Make the speed available to expressions as
"@Speed".
}
```

State machines

A common programming pattern in control applications is to use state machines. PAWN and the ezeio implements strong support for state machines. The following is a typical pattern showing the startup sequence of an engine. Note that there are three @Tick handlers; one for each state. Also note the “entry” and “exit” functions. For more detail, refer to the PAWN language guide.

```
  new count = 0;

  main()
  {
    SetTickInterval(100);   // set tick interval to 100ms (0.1s)
    state WAITING;          // start in the waiting mode
  }

  @Tick(uptime) <WAITING>
  {
     if( GetField(1) < 100 )  // condition to start up the process
        state IGNITION;
  }

// ***** the IGNITION state

  entry() <IGNITION>
```

```
  {
     SetOutput(1, 100);       // Turn on the ignition switch
     count = 0;
  }

  @Tick(uptime) <IGNITION>
  {
     if( GetField(2) > 100 )  // Did the engine start?
        state RUNNING;        // yes - we're running

     if(count++ > 50)
        state WAITING;        // didn't start in 5s? Give up and go back to
waiting.
  }

  exit() <IGNITION>
  {
     SetOutput(1, 0);         // Turn the ignition switch off
  }

// ***** the RUNNING state

  @Tick(uptime) <RUNNING>
  {
     if( GetField(2) < 100 )  // Check if the engine stopped
        state WAITING;        // ..go back to waiting
  }
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/start

Last update: **2026-01-23 21:51**

ABS

Macro that will return the absolute value

Description

```
ABS( number )
```

Returns the absolute value of the given number.

Works with both float and integer parameters.

Parameters

number	input value

Return value

Returns the absolute value of number.

Example usage

```
new Float:x;
new i;

x = ABS( 123.45 );      // 123.45
i = ABS( 456 );         // 456

x = ABS( -123.45 );     // 123.45
i = ABS( -456 );        // 456
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/abs

Last update: **2021-09-27 22:44**

AdjustField

Adjust the value of a field.

Description

```
AdjustField( fieldno, Float:value )
```

Adds the given value to the field value.

This command performs the addition with double (64-bit) math. It may be useful for accumulating values that can grow very large, in which case this command can be used to maintain precision.

Parameters

`fieldno`	Which field to adjust
`value`	Value to add

Return value

The float value of the field after being adjusted.

Example usage

```
AdjustField(4, 0.1);

// adds 0.1 to the value of field 4
```

NOTE: First appeared in firmware 20091601

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/adjustfield

Last update: **2020-09-16 23:23**

AllocRegisters

Reserve registers for a device driver

Description

```
AllocRegisters( deviceno, registercount )
```

This command allocates registers for a device driver. Repeated calls will not reset the registers unless the register count changes. There is no need to de-allocate registers as the system will automatically free registers that are not in use.

Up to 150 registers can be allocated for each device, up to 2000 registers total for the ezeio.

Parameters

deviceno	Device number (1-40)
registercount	Number of registers to allocate (1-150)

Return value

none

Example usage

```
    AllocRegisters(3, 8);        // Allocate eight registers for device 3

    SetRegister(3, 1, 1234, INT);   // Set register 1 on device 3 to the value 1234
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/allocregisters

Last update: **2023-07-27 17:17**

BufDec

Decode a part of a buffer into a scalar

Description

```
BufDec( const buf[], offset, method )
```

This command will decode a portion of a buffer into a scalar variable.

Parameters

`buf`	Data buffer
`offset`	Byte offset into the buffer where to start the decoding
`method`	What type of value is encoded in the buffer

Available methods:

Method	Length	Type	Description
BT_INT8	8 bit	signed integer	Single octet value
BT_UINT8	8 bit	unsigned integer	Single octet value
BT_INT16	16 bit	signed integer	Big endian integer
BT_UINT16	16 bit	unsigned integer	Big endian integer
BT_INT32	32 bit	signed integer	Big endian integer
BT_UINT32	32 bit	unsigned integer	Big endian integer
BT_INT32LE	32 bit	signed integer	Little endian integer
BT_UINT32LE	32 bit	unsigned integer	Little endian integer
BT_INT32BLE	32 bit	signed integer	Big-Little endian integer
BT_UINT32BLE	32 bit	unsigned integer	Big-Little endian integer
BT_INT32LBE	32 bit	signed integer	Little-Big endian integer
BT_UINT32LBE	32 bit	unsigned integer	Little-Bit endian integer
BT_FLOAT	32 bit	float	IEEE 732 floating point
BT_FLOATLE	32 bit	float	IEEE 732 floating point little endian
BT_FLOAT64	64 bit	float64	IEEE 732 64-bit floating point (converts to 32 bit float)
BT_FLOAT64LE	64 bit	float64	IEEE 732 64-bit floating point little endian (converts to 32 bit float)

Also available for decoding bitmaps:

Bit offset : **BT_BITOFS0** through **BT_BITOFS15**
Bit mask : **BT_BITMASK1** through **BT_BITMASK8**

The BT_BITOFSx and BT_BITMASKx methods can be OR:ed together for bitmask decoding.

Example: To extract a 3 bit value from bit position 5,6 and 7, use `BT_BITOFS5 | BT_BITMASK3`

Return value

Returns the decoded value

Note that the returned value is always a 32 bit cell (signed 32 bit integer or 32 bit float).

Example usage

```
    new x;
    new s{10};

    if(ModbusRead(5, MB_HOLDING, 100, 2, s) == 4) {  // Request and receive two registers (32 bits total)
        x = BufDec(s, 0, BT_INT32);                  // Decode the received data as a 32 bit integer
        // ..further processing
    }
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/bufdec

Last update: **2023-08-18 23:50**

C2F

Helper function to convert from Celsius degrees to Farenheit

Description

```
Float:C2F( Float:C )
```

Return Farenheit from degrees Celsius.

Parameters

C	Degrees Celsius

Return value

Degrees Farenheit

Example usage

```
new Float:F = C2F( 10.0 );
// F = 50.0;
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/c2f

Last update: **2020-08-14 23:36**

CLAMP

Macro that will return a value limited to a given range

Description

```
CLAMP( value, minimum, maximum )
```

Returns the `value` if it is within the range `minimum` to `maximum`.

If the `value` is smaller than `minimum`, returns `minimum`.

If the `value` is larger than `maximum`, returns `maximum`.

Works with both float and integer parameters.

Parameters

`value`	input value
`minimum`	smallest allowed value
`maximum`	largest allowed value

Return value

Returns the `value` limited to within the given range.

Example usage

```
new n, x;

n = 60;
x = CLAMP(n, 30, 90);   // 60
x = CLAMP(n, 10, 20);   // 20
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/clamp

Last update: **2019-11-19 00:19**

CSVtoInt

Extract a single parameter from a CSV formatted string.

Description

```
CSVtoInt( const data[], item, decimals )
```

Extract single numeric parameter from a CSV formatted string.

Parameters

`data`	CSV formatted string
`item`	Which item to extract. The first item is 1.
`decimals`	Number of decimals to extract

Assuming the input data is properly formatted as a CSV string (comma separated), this command finds the item by count and evaluates the item as a number. If decimals is larger than 0, that number of decimals will be added, and the resulting integer multiplied by 10 for each decimal.

Return value

The integer value of the item.

Example usage

```
new val;
new csvdata{} = "123, 1234.56, \"some string\", \"more data\"";

val = CSVtoInt( csvdata, 2, 3 );

// val will be 1234560
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/csvtoint

Last update: **2019-09-02 21:37**

CSVtoStr

Extract a single parameter from a CSV formatted string.

Description

```
CSVtoStr( const data[], destination[], maxlen, item )
```

Extract single parameter from a CSV formatted string.

Parameters

`data`	CSV formatted string
`destination`	Buffer for extracted parameter
`maxlen`	Max size of the destination buffer
`item`	Which item to extract. The first item is 1.

Assuming the input data is properly formatted as a CSV string (comma separated), this command finds the item by count and copies the string data into destination.

Return value

Number of characters copied into destination. If the item is not found, return 0.

Example usage

```
new param{128};
new csvdata{} = "123, 1234, \"some string\", \"more data\"";

CSVtoStr( csvdata, param, 128, 3 );

// param will now hold the text "some string" without quotes.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/csvtostr

Last update: **2019-09-02 21:37**

Dewpoint

Helper function to calculate dewpoint from temperature and relative humidity

Description

```
Float:Dewpoint( Float:temp, Float:RH )
```

Calculates dewpoint in degrees Celsius.

This functions uses the Magnus-Tetens formula.

Parameters

temp	Temperature in degrees Celsius (-45 to +60 °C)
RH	Relative humidity in percent (0-100)

Return value

Dew point in degrees Celsius.

Example usage

```
new Float:temperature = 18.5
new Float:relhum = 22.9;
new Float:dp;

dp = Dewpoint( temperature, relhum );

// dp is now 8.87
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/dewpoint

Last update: **2020-08-14 20:54**

DoAlarm

Trigger activation of an alarm.

Description

```
DoAlarm( alarmno, force, dparam, iparam1, iparam2, text, ... )
```

This command will change the state of an alarm into active and trigger any alarm actions accordingly. If the force parameter is 0/false and the alarm is already in active state, the actions are not re-triggered.

If the alarm is not active, the restore logic and holdoff will determine if/when the restore actions will execute.

Parameters

`alarmno`	The alarm number
`force`	If set to TRUE/1, the actions will be run even if the alarm was already in alarm state
`dparam`	A user-defined float value (accessed by [P2] in templates
`iparam1`	A user-defined integer value [P3]
`iparam2`	A user-defined integer value [P4]
`text`	User defined formatted text to include in the alarm message [TEXT]

Return value

Returns the previous state of the alarm. 1=alarm was already active. 0=alarm was not active.

Example usage

```
new t = 99;
DoAlarm(4, 0, 1.1, 200, 333, "Too hot! Temp=%d degrees!", t);
```

NOTE: First appeared in firmware 24060601

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/doalarm

Last update: **2024-06-07 19:34**

DoRestore

Trigger restore of an alarm.

Description

```
DoRestore( alarmno, force, dparam, iparam1, iparam2, text, ... )
```

This command will change the state of an alarm into restored and trigger any alarm actions accordingly. If the force parameter is 0/false and the alarm is not in active state, the actions are not re-triggered.

If the alarm is in alarm state, the restore logic and holdoff will determine if/when the restore actions will execute.

Parameters

`alarmno`	The alarm number
`force`	If set to TRUE/1, the actions will be run even if the alarm was already in restored state
`dparam`	A user-defined float value (accessed by [P2] in templates
`iparam1`	A user-defined integer value [P3]
`iparam2`	A user-defined integer value [P4]
`text`	User defined formatted text to include in the message [TEXT]

Return value

Returns the previous state of the alarm. 1=alarm was active. 0=alarm was not active.

Example usage

```
new m = 14;
DoRestore(4, 0, 1.1, 200, 333, "Door closed. Was open %d minutes.", m);
```

NOTE: First appeared in firmware 24060601

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/dorestore

Last update: **2024-06-07 19:35**

Eval

Evaluates the supplied string using the expression parser

Description

```
Float:Eval( const string[] )
```

Return the resulting value. If the expression is invalid, the return value is zero (0.000)

NOTE: The Eval script function does not support tags.

Parameters

`string`	Expression string to be evaluated

Return value

Floating point value of the expression after evaluation.

Example usage

```
   new Float:f;

   f = Eval( "12.34-2.34" );
   // The value of f is now 10.00

   f = Eval( "f(1)>f(2)" );
   // If the value of field 1 is larger than the value of field 2, f=1.000.
Otherwise f=0.000.
```

NOTE: First appeared in firmware 23112801

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/eval

Last update: **2023-11-30 18:00**

F2C

Helper function to convert from Farenheit degrees to Celsius degrees

Description

```
Float:F2C( Float:F )
```

Return degrees Celsius from Farenheit.

Parameters

F	Degrees Farenheit

Return value

Degreees Celsius

Example usage

```
new Float:C = F2C( 50.0 );
// C = 10.0
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/f2c

Last update: **2020-08-14 23:34**

facos

Find the arc-cosine of a value

Description

```
Float:facos( Float:value )
```

Find the arc-cosine of the given value.

The return value is in radians.

Parameters

`value`	input value

Return value

Returns the arc-cosine in radians of `value`.

Example usage

```
new Float:x;

x = facos( 1 );    // 0
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/facos

Last update: **2021-09-27 22:44**

fasin

Find the arc-sine of a value

Description

```
Float:fasin( Float:value )
```

Find the arc-sine of the given value.

The return value is in radians.

Parameters

`value`	input value

Return value

Returns the arc-sine in radians of `value`.

Example usage

```
new Float:x;

x = fasin( 1 );    // 1.570796..
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fasin

Last update: **2021-09-27 22:44**

fatan

Find the arc-tangent of a value

Description

```
Float:fatan( Float:value )
```

Find the arc-tangent of the given value in radians.

Parameters

`value`	input value in radians

Return value

Returns the arc-tangent `value`.

Example usage

```
new Float:x;

x = fatan( 1 );    // 0.785398..
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fatan

Last update: **2019-11-18 23:40**

fatan2

Find the 2-argument arc-tangent

Description

```
Float:fatan2( Float:x, Float:y )
```

Find the Euclidean plane for the point (x,y)

The return value is in radians.

Parameters

x	x-axis offset
y	y-axis offset

Return value

Returns the arc-tangent in radians for the point (x,y)

Example usage

```
new Float:x;

x = fatan2( 1, 1 );    // 0.785398...
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fatan2

Last update: **2019-11-18 23:43**

fcos

Find the cosine of a value

Description

```
Float:fcos( Float:value )
```

Find the cosine of the given value.

The value is assumed to be in radians.

Parameters

`value`	input value (angle in radians)

Return value

Returns the cosine of `value`.

Example usage

```
new Float:x;

x = fcos( PI/4 );    // 0.70710...
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fcos

Last update: **2019-11-18 23:27**

flog

Find the logarithm of a value

Description

```
Float:flog( Float:value, Float:base=10.0 )
```

Find the logarithm of the given value, using the given base. By default flog returns log10.

To return the natural logarithm, use `flog( value, E )`.

Parameters

`value`	input value
`base`	base (optional, default is 10)

Return value

Returns the logarithm of `value` using the given base.

Example usage

```
new Float:x;

x = flog( 100 );     // 2
x = flog( 100, E ); // 4.60517
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/flog

Last update: **2019-11-18 23:21**

ForceAlarm

Force activation of an alarm.

Description

```
ForceAlarm( alarmno )
```

This command will force the state of an alarm into active and trigger any alarm actions accordingly.

The alarm will be triggered even if the alarm is already active. If the alarm is not active, the restore logic and holdoff will determine if/when the restore actions will execute.

Parameters

`alarmno`	The alarm number

Return value

Returns the previous state of the alarm. 1=alarm was already active. 0=alarm was not active.

Example usage

```
   ForceAlarm(10);   // trigger alarm number 10
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/forcealarm

Last update: **2020-08-14 20:17**

ForceRestore

Force restore of an alarm.

Description

```
ForceRestore( alarmno )
```

This command will force the state of an alarm into restore and trigger any restore actions accordingly.

The restore actions will be tripped even if the alarm state is already restored. If the alarm condition is fulfilled, the alarm condition/holdoff will determine if/when the alarm activates again.

Parameters

`alarmno`	The alarm number

Return value

Returns the previous state of the alarm. 1=alarm was already active. 0=alarm was not active.

Example usage

```
   ForceRestore(10);   // trigger restore on alarm number 10
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/forcerestore

Last update: **2020-08-14 20:18**

fpow

Return a value raised to an exponent

Description

```
Float:fpow( Float:value, Float:exponent )
```

Raise `value` to `exponent`

Parameters

`value`	input value
`exponent`	exponent

Return value

Returns (value)$^{(exponent)}$

Example usage

```
new Float:x;

x = fpow( 3, 4 );   // 81

x = fpow( 7.39, E ); // 2.0001.. (=natural logarithm)
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fpow

Last update: **2023-10-26 18:12**

fround

Return a float value rounded off to an integer

Description

```
fround(Float:value, method=floatround_round)
```

Round the `value` to an integer, using the method given by `method`.

Parameters

`value`	Floating point value to be rounded
`method`	Rounding method, default to nearest integer

method is one of:

`floatround_round`	Round to nearest integer. 0.5 rounds up.
`floatround_floor`	Round down to next lower integer
`floatround_ceil`	Round up to next higher integer
`floatround_tozero`	Round towards zero

Return value

Returns the rounded result as an integer

Example usage

```
new x;

x = fround(3.14);                   // 3
x = fround(3.14, floatround_ceil);  // 4
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fround

Last update: **2019-11-18 23:01**

fsin

Find the sine of a value

Description

```
Float:fsin( Float:value )
```

Find the sine of the given value.

The value is assumed to be in radians.

Parameters

`value`	input value (angle in radians)

Return value

Returns the sine of `value`.

Example usage

```
new Float:x;

x = fsin( PI/4 );    // 0.70710...
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fsin

Last update: **2019-11-18 23:26**

fsqrt

Find the square root

Description

```
Float:fsqrt( Float:value )
```

Find the square root of the given value

Parameters

`value`	input value (must be non-negative)

Return value

Returns the square root of `value`

Example usage

```
new Float:x;

x = fsqrt( 12.25 );   // 3.5
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fsqrt

Last update: **2019-11-18 23:05**

fstrval

Evaluate a string to a float value

Description

```
Float:fstrval( const str[], index=0, limit=15 )
```

Evaluate the string, starting at the `index` position for a float value Max `limit` characters will be evaluated, starting at the `index` position.

Parameters

`str`	source string
`index`	start position in `str`

Return value

Returns the value as a float

Example usage

```
    new s{20};
    new Float:x;

    strcopy(s, "Hello3.14");

    x = fstrval(s, 5);

    // x now equals 3.14
```

NOTE: requires firmware 24020101 or newer.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/fstrval

Last update: **2024-02-05 23:17**

ftan

Find the tangent of a value

Description

```
Float:ftan( Float:value )
```

Find the tangent of the given value.

The return value is in radians.

Parameters

`value`	input value

Return value

Returns the tangent in radians of `value`.

Example usage

```
new Float:x;

x = ftan( PI/4 );    // 1
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/ftan

Last update: **2019-11-18 23:35**

GetAlarmInfo

Return the current information about the alarm

Description

```
GetAlarmInfo( alarmno, mode )
```

Fetch the given information about the current alarm state.

Parameters

alarmno	The alarm number
mode	One of the modes below

mode parameter, one of:

AL_STATE	The alarm state (1=in alarm, 0=not in alarm)
AL_HOLDOFF	The remaining time on the holdoff timer (seconds x10, so 100 = 10s)
AL_ALARMHOLDOFF	The alarm holdoff setting (seconds x10, so 100 = 10s)
AL_RESTOREHOLDOFF	The restore holdoff setting (seconds x10, so 100 = 10s)
AL_ISALARM	Returns 1 if the alarm condition is true. Otherwise 0
AL_ISRESTORE	Return 1 if the restore condition is true. Otherwise 0
AL_RETRIGCOUNT	Number of times the alarm has be re-triggered
AL_RETRIGDELAY	Time remaining on the alarm retrigger delay (seconds x10, so 100 = 10s)
AL_RETRIGCOUNTSET	Retrigger count configured
AL_RETRIGDELAYSET	Retrigger delay configured (seconds x10, so 100 = 10s)

Return value

Returns the requested information based on the mode parameter.

Example usage

```
new ds = GetAlarmInfo(10, AL_HOLDOFF);
// ds is the number of 10th of seconds remaining on the holdoff
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getalarminfo

Last update: **2020-10-08 23:21**

GetAlarmState

Return the current state of the alarm

Description

```
GetAlarmState( alarmno )
```

Fetch the state of a given alarm.

Parameters

`alarmno`	The alarm number

Return value

Returns the current state of the alarm. 1=alarm is active. 0=alarm is not active.

Example usage

```
if(GetAlarmState(10)) {
  // logic to run if alarm is active
}
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getalarmstate

Last update: **2019-09-02 21:38**

GetBit

Helper function to read the state of a single bit in a byte-buffer.

Description

```
GetBit( buffer, bitno )
```

Get the state of a single bit in a buffer

Parameters

buf	Reference to the byte buffer
bitno	The bit number (0-indexed)

Return value

Returns the state of the bit (1 or 0)

Example usage

```
new buffer{80};
new b;

// assuming the buffer is filled with data

b = GetBit(buffer, 18);

// b is the status of the 19th bit in the buffer
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getbit

Last update: **2020-08-14 20:40**

GetCellBit

Helper function to read the state of a single bit in a cell.

Description

```
GetCellBit( value, bitno )
```

Get the state of a single bit in the cell value

Parameters

`value`	32-bit cell value
`bitno`	The bit number (0-indexed)

Return value

Returns the state of the bit (1 or 0)

Example usage

```
new value = 9;  // (9 = binary 1001)
new b;

b = GetBit( value, 3 );

// b is now 1
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getcellbit

Last update: **2024-04-10 18:04**

GetDeviceStatus

Fetch the Comm / Op / App status of a given device (or aggregate status)

Description

```
GetDeviceStatus( deviceno, status item )
```

Returns the given status item for.

Parameters

deviceno	Device number to request status from (or 0 for all devices)
status item	The status item to return (see below)

Parameter	Description	Aggregate
DVCSTAT_BATTVOLT	The device battery/supply voltage in mV (if supported)	YES
DVCSTAT_SIGNAL	The reported device signal strength	YES
DVCSTAT_COMMCOUNT	Number of successful comm attempts	NO
DVCSTAT_COMMERR	Number of failed comm attempts	NO
DVCSTAT_LASTCOMM	Time of last status update	NO
DVCSTAT_COMMSTAT	Communication status, one of: DS_COM_UNKNOWN DS_COM_ERROR DS_COM_ISSUE DS_COM_OK DS_COM_BEST	YES
DVCSTAT_OPSTAT	Operational status, one of: DS_OP_UNKNOWN DS_OP_ERROR DS_OP_PART DS_OP_FULL	YES
DVCSTAT_APPSTAT	Application status, one of: DS_APP_UNKNOWN DS_APP_ERROR DS_APP_WARNING DS_APP_OK DS_APP_OK1 DS_APP_OK2 DS_APP_OK3 DS_APP_OK4	YES
DVCSTAT_STAT	Aggregated status, one of: (read only) DS_UNKNOWN DS_ERROR DS_WARNING DS_OK	YES

Parameter	Description	Aggregate
DVCSTAT_ACTIVE	Returns 1 if device is loaded/active (read only)	NO
DVCSTAT_REGCOUNT	Number of registers allocated (read only)	NO
DVCSTAT_TYPE	Driver id number	NO
DVCSTAT_VERSION	Driver version	NO
DVCSTAT_MODE	User defined mode	NO
DVCSTAT_PARAM	User defined parameter	NO

DVCSTAT_ACTIVE, _REGCOUNT, _TYPE, _VERSION, _MODE and _PARAM are available in firmware 21061101 and newer

Return value

Depends on parameter requested. See above.

Example usage

```
   if( GetDeviceStatus(0, DVCSTAT_STAT) < DS_OK ) {  // Get 'worst' status from all devices
      // DEVICE ERROR DETECTED -- SHUT OFF PROCESS
      SetOutput(1, 0);
   }
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getdevicestatus

Last update: **2021-06-14 22:45**

GetField/GetFieldFloat

Get the value of a field

Description

```
GetField( fieldno )
```

```
Float:GetFieldFloat( fieldno )
```

Fetches the value of a field

Parameters

`fieldno`	Which field to fetch

Return value

The value of the field. GetField will return the value rounded to the nearest integer.

Example usage

```
new x;
new Float:f;

x = GetField(4);
f = GetFieldFloat(4);

// x now holds the value of field 4, rounded to the nearest integer.
// f holds the value as a float.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getfield

Last update: **2020-09-16 23:17**

GetFieldLog

Get the value of a field from the log data

Description

```
GetFieldLog( fieldno, seconds )
```

```
Float:GetFieldLog( fieldno, seconds, FLOAT )
```

Fetches the value of a field from the log data buffer

Available in firmware 21060101 or newer

Parameters

`fieldno`	Which field to fetch
`seconds`	How far back in time the data should come from
FLOAT	By default, this function returns an integer value. Add 'FLOAT' as the last parameter to return a floating point value

Return value

The value of the field from the log data. The log buffers will be searched for the closest timestamp saved. If the field is set to log faster than the default 10 minutes, this fuction automatically selects the 'fastlog' data.

Example usage

```
   new x;
   new Float:f;

   x = GetFieldLog(4, 300);
   f = Float:GetFieldLog(4, 3600, FLOAT);

   // x now holds the value of field 4 from 5 minutes ago, rounded to the
nearest integer.
   // f holds the value from field 4 from 1h ago as a float.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getfieldlog

Last update: **2022-05-17 14:41**

GetInputMode

Get the input mode on a native ezeio input.

Description

```
GetInputMode( inputno )
```

This will get the hardware input mode on a ezeio input. The command only works on native ezeio inputs - on the ezeio controller or ezeio expansion I/O devices. Third party modules use different methods to change input configuration.

Parameters

`inputno`	Input number to query. Inputs 1-8 are on the ezeio. Use “DVCn+inputno” for expansion devices

Return value

Value	Mode	Hardware config	Input unit	Range
0	INMODE_10V	High impedance, >60kΩ	mV (Volt)	0-10000mV
1	INMODE_30mA	200Ω shunt to ground	µA (current)	0-30000µA
2	INMODE_PULLUP	Weak pull-up to +5V, <1mA	Ohm (resistance)	0-1000000Ω
3	INMODE_TEST		internal test mode	

Note that only the hardware mode is returned. While the SetInputMode command accepts various other modes, the only four possible hardware modes are the above. The various thermistor modes will all return as 2:INMODE_PULLUP.

Example usage

```
   SetInputMode(3, INMODE_30mA);          // Set input 3 on ezeio to measure current
   x = GetInputMode(3);                   // x will be assigned the value 1 (INMODE_30mA)
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getinputmode

Last update: **2025-07-10 18:28**

GetInputValue

Fetch the value of a hardware input.

Description

```
GetInputValue( inputno, mode )
```

The value returned will depend on the hardware mode of the input, which should have been set already using the SetInputMode command.

Parameters

`inputno`	Input number to read. Inputs 1-8 are on the ezeio. Use “DVCn+inputno” for expansion devices
`mode`	The desired read mode. See below.

Input numbers:

Input	Description
IN_1 .. IN_16	Input numbers (equivalent to 1 .. 16)
IN_VIN	Supply voltage (in mV)
IN_VBAT	Voltage on Battery Terminals (in mV)
IN_V5	5V regulated output voltage
DVC1+IN_1	(example) First input on first expansion device

Available read modes:

Mode	Reading
INVAL_RAW	Momentary value. Unit depends on input hardware setting.
INVAL_MIN	Smallest value in this sampling interval. Unit depends on input hardware setting.
INVAL_MAX	Largest value in this sampling interval. Unit depends on input hardware setting.
INVAL_AVG	Average value in this sampling interval. Unit depends on input hardware setting.
INVAL_INTERVAL	Pulse interval in ms
INVAL_FREQ	Pulse frequency in Hz x1000 (mHz)
INVAL_COUNT	Pulse count

Return value

Value of input.

The unit of the returned value depends on the input hardware setting:

Hardware setting	Input unit	Description
INMODE_10V	mV (milliVolt)	0-10240 mV
INMODE_30mA	uA (microAmpere)	0-30000 uA
INMODE_PULLUP	Ω (Ohm)	0-10000000 Ohm (0-1MΩ)
INMODE_THERMISTOR*	°K x100 (Kelvin)	0-60000 Kelvin x 100

Example usage

```
    new x;
    x = GetInputValue(IN_3, INVAL_RAW);  // Return the current value of the
input
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getinputvalue

Last update: **2019-09-02 21:38**

GetOutput

Get the state of an output

Description

```
GetOutput( outputno )
```

Return the current state of a native ezeio output. The output value is always in percent , 0-100. For digital outputs, any value 50 and higher will set the output to ON (100%).

Output 1-4 are the on-board ezeio outputs.
To reference outputs on ezeio expansion units, use “DVCn + output number”. See example below.

This command only works to control native ezeio outputs. To control outputs on third party modules, use the appropriately mapped registers. See documentation for the relevant driver.

Parameters

`outputno`	Output number

Return value

Output value in percent (0-100)

Example usage

```
new x;

SetOutput(4, 31);   // Set the on-board analog output to 3.1V.

x = GetOutput(4);   // Read the output value

// x now has the value 31
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getoutput

Last update: **2019-09-02 21:38**

GetRegister

Get the value of a register

Description

```
GetRegister( deviceno, registerno, {type} )
```

Fetches the value of a register

Parameters

deviceno	Device number (1-40)
registerno	Which register to fetch
type	(optional) a variable that will be set to the register type (NULL/UINT/INT/FLOAT)

Return value

The value of the register

Example usage

```
    new x, type;

    AllocateRegisters(3, 8);         // Allocate eight registers for device 3
    SetRegister(3, 1, 1234, INT);    // Set register 1 on device 3 to the value
1234

    x = GetRegister(3, 1);

    // x is now 1234

    x = GetRegister(3, 1, type);

    // type will be INT
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getregister

Last update: **2020-08-14 20:21**

GetRegisterStatus

Get status of a register

Description

```
GetRegisterStatus( deviceno, registerno, parameter )
```

Fetches the status of a register

Parameters

deviceno	Device number (1-40)
registerno	Which register to fetch
parameter	The parameter to request, one of REGSTAT_VALUE, REGSTAT_AGE, REGSTAT_INUSE, REGSTAT_TYPE

Return value

Depends on the parameter

REGSTAT_VALUE	Return value is the value of the register (same as GetRegister)
REGSTAT_AGE	Return value is the number of seconds since the register was updated (max 4095)
REGSTAT_INUSE	Return value is 1 if the register is in use/allocated, or 0 if not
REGSTAT_TYPE	Return value is one of NULL, INT, UINT, FLOAT

Example usage

```
    new x;

    AllocateRegisters(3, 8);         // Allocate eight registers for device 3
    SetRegister(3, 1, 1234, INT);    // Set register 1 on device 3 to the value
1234

    x = GetRegisterStatus(3, 1, REGSTAT_TYPE);    // x = INT
    x = GetRegisterStatus(3, 1, REGSTAT_AGE);     // x = 0 (since we just
updated it)
```

NOTE: First appeared in firmware 24052201

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getregisterstatus

Last update: **2024-05-22 21:44**

GetSystemItem

Fetch the value of a given system parameter

Description

```
GetSystemItem( item )
```

Returns the value of a given system parameter

Parameters

item	The system item to return (see below)

Parameter	Description
SYSITEM_GPSX	GPS latitude in degrees x 10^6
SYSITEM_GPSY	GPS longitude in degrees x 10^6
SYSITEM_GPSZ	GPS elevation in meters x 10
SYSITEM_GPSSIGNAL	Received signal level from GPS
SYSITEM_UPTIME	Number of seconds since last restart
SYSITEM_CELLINHIBIT	0=normal operation. >0 cellular modem is off, remaining seconds
SYSITEM_RSSI	Received signal level for cellular modem (0=no signal. 31=max signal strength)
SYSITEM_ETHLINK	0=No Ethernet connection. 1=Ethernet connection detected
SYSITEM_ETHCONN	0=Not using Ethernet connection. 1=Using Ethernet connection
SYSITEM_CELLCONN	0=Not using cellular connection. 1=Using cellular connection
SYSITEM_GPSLOCK	0=No current GPS position. 1=GPS position current
SYSITEM_YEAR	Real time clock, year
SYSITEM_MONTH	Real time clock, month
SYSITEM_DAY	Real time clock, day
SYSITEM_WDAY	Real time clock, weekday (0=Sunday, 6=Saturday)
SYSITEM_YDAY	Real time clock, day of year
SYSITEM_HOUR	Real time clock, hour
SYSITEM_MINUTE	Real time clock, minutes
SYSITEM_SECOND	Real time clock, seconds
SYSITEM_EPOCH	Real time clock, Epoch (number of seconds since 1970-01-01)
SYSITEM_RND	Random number. 32 bit signed integer
SYSITEM_OUT3RCPWM	Output 3 mode (0=0-100%, 1=1-2ms servo)
SYSITEM_RESETCAUSE	Cause of last hardware reset, bitmap or RESETCAUSE flags defined below
SYSITEM_SCRIPT_RESETCAUSE	Cause of last script restart, current script (see below)
SYSITEM_HIBERNATE	Nonzero if running as part of sleep cycle

Parameter	Description
SYSITEM_EXPn	ezeio-CAN Expander status (0=no comm, 1=comm ok), where n=1 through 7

Flags returned by SYSITEM_RESETCAUSE

constant	meaning
RESETCAUSE_EXT	Any external reset
RESETCAUSE_POR	Board was powered on
RESETCAUSE_BOR	Board detected brown-out (voltage dip)
RESETCAUSE_BOR	Software caused the reset
RESETCAUSE_WDOG	Internal watchdog caused the reset
RESETCAUSE_HIB	Board woke up from hibernation

Flags returned by SYSITEM_SCRIPT_RESETCAUSE

constant	meaning
SCRIPT_BOOT	Script started due to board reset
SCRIPT_ERROR	Script restarted due to script error
SCRIPT_USER	Script restarted by user
SCRIPT_CHANGE	Script restarted due to config change
SCRIPT_BUTTON	Script restarted with hardware button

Return value

Depends on parameter requested. See above.

Example usage

```
new x;

x = GetSystemItem(SYSITEM_YEAR);
// x is now 2019 (as of this writing)

if( GetSystemItem( SYSITEM_RESETCAUSE ) & RESETCAUSE_HIB ) {
  // this code will run if board came back from a hibernation cycle
}
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getsystemitem

Last update: **2025-02-11 23:18**

GetTimer

Get the current state of a timer

Description

```
GetTimer( timerno, mode=0 )
```

Return the current state of a timer.

Parameters

timerno	Timer number 0-3
mode	mode - see below

mode is one of:

TIMER_REMAIN	Returns remaining time on the timer in milliseconds (default)
TIMER_TIMEOUT	Returns the initial timeout set for this timer
TIMER_PARAM	Returns the user parameter value
TIMER_REPEAT	Returns the number of remaining repeats

Return value

Returns the timer parameter requested

Example usage

```
new x;

SetTimer(1, 100, 1234);          // set up timer 1
x = GetTimer(1, TIMER_PARAM);    // returns 1234
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/gettimer

Last update: **2019-11-19 00:08**

GetVal

Get value of a user-defined memory cell

Description

```
GetVal( address )
```

Fetch the value of one of the user defined memory cells. There are 512 32-bit cells available. The values are stored in NVM, so they will remain through power loss/reset.

UPDATE since firmware 22070801: Expanded to 2048 cells. Only the first 512 are stored in NVM.

Parameters

address	Address, 0-511

Return value

Returns the value of the cell.

Example usage

```
new x;

SetVal(100, 4711);  // Set cell with address 100 to value 4711
x = GetVal(100);    // Read cell 100

// x now has the value 4711
```

NOTE: First appeared in firmware 21071901

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/getval

Last update: **2022-07-13 17:03**

ispacked

Tests a string for how it is stored.

Description

```
ispacked( const string[] )
```

Tests a string for how it is stored. See Pawn documentation about string packed/unpacked storage. By default all strings are packed in the ezeio.

Parameters

`string`	reference to string to be tested

Return value

Returns 1 if the string is packed (single byte per character).

Return 0 if the string is using a cell (4 bytes) for each character.

Example usage

```
    new x;
    new s{6} = "Hello";

    x = ispacked(s);

    // x is now 1
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/ispacked

Last update: **2019-09-02 21:39**

K2C

Helper function to convert from Kelvin degrees to Celsius degrees

Description

```
Float:K2C( Float:K )
```

Return degrees Celsius from Kelvin.

Parameters

K	Degrees Kelvin

Return value

Degreees Celsius

Example usage

```
new Float:C = K2C( 280.15 );
// C = 7.0;
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/k2c

Last update: **2020-08-14 23:31**

K2F

Helper function to convert from Kelvin degrees to Farenheit degrees

Description

```
Float:K2F( Float:K )
```

Return degrees Farenheit from Kelvin.

Parameters

K	Degrees Kelvin

Return value

Degreees Farenheit

Example usage

```
new Float:F = K2F( 280.15 );
// F = 44.6;
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/k2f

Last update: **2020-08-14 23:36**

Linfit

Linear regression. Attempt to find best fit y=ax+b function given multiple x/y points.

Description

```
Linfit( Float:x[], Float:y[], points, &Float:a, &Float:b )
```

Find the best fitting y=ax+b function using least-square method.

Parameters

x[]	Array of x-values
y[]	Array of y-values
points	Size of the x and y arrays (number of points)
a	The variable to receive the a (slope) value
b	The variable to receive the b (offset) value

Return value

Returns 1 if successful. 0 otherwise (if less than 2 x/y points given).

Example usage

```
new Float:x[10], Float:y[10];
new Float:a, Float:b;

// assign 10 points to x/y

Linfit(x, y, 10, a, b);

// a and b now holds the calculated slope/offset values
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/linfit

Last update: **2020-08-14 23:27**

localtime

Split Unixtime into components

Description

```
localtime(timestamp, local=0, year, month, day, hour, minute, second, weekday, yearday)
```

Split the timestamp into components.

Parameters

timestamp	Unix timestamp (seconds since 1970-01-01, UTC)
local	false=result is UTC, true=result is local time
year	Will be set to the year
month	Will be set to the month
day	Will be set to day of month
hour	Will be set to the hour
minute	Will be set to the minute
second	Will be set to the second
weekday	Will be set to the day of week (0=Sunday)
yearday	Will be set to the day in the year

Return value

Returns the timestamp. The parameters year – yearday are written as described above.

Example usage

```
   new yy, mo, dd, hh, mi, ss;
   new t = GetSystemItem(SYSITEM_EPOCH);

   localtime( t, false, yy, mo, dd, hh, mi, ss );  // yy, mo, dd, hh, mi and ss are set to current UTC time.
```

NOTE: First appeared in firmware 22110901

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/localtime

Last update: **2022-11-09 19:51**

MAX

Macro that will return the larger of two values

Description

```
MAX( value1, value2 )
```

Returns the larger of two values

Works with both float and integer parameters.

Parameters

value1	input value
value2	input value

Return value

Returns the larger of value1 and value2.

Example usage

```
new n;

n = MAX(51, 53);     // 53
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/max

Last update: **2019-11-19 00:14**

MBRTU_Read

Read a single register from a Modbus/RTU device

Description

```
MBRTU_Read(unitid, cmd, reg, format=BT_UINT16, &error)
```

This command issues a Modbus/RTU read command and waits for a valid reply. If an error occurrs (timeout, or error reply from the device), the error parameter is set to a non-zero value.

Using this command in the user script will disable any system drivers using Modbus to avoid conflicts.

Parameters

`unitid`	Modbus/RTU device bus address (1-254)
`cmd`	The Modbus command, see below
`reg`	The register physical address
`format`	Register type, see below. Optional, defaults to UINT16
`error`	Variable to receive the result code. Optional.

Modbus commands

`MB_COIL`	Read coil (Modbus command 1)
`MB_DISCRETE`	Read discrete input (Modbus command 2)
`MB_HOLDING`	Read holding register (Modbus command 3)
`MB_INPUT`	Read input register (Modbus command 4)

Register types

`BT_INT8`	8 bit integer, signed
`BT_UINT8`	8 bit integer, unsigned
`BT_INT16`	16 bit integer, signed
`BT_INT16LE`	16 bit integer, signed, byte swapped
`BT_UINT16`	16 bit integer, unsigned
`BT_UINT16LE`	16 bit integer, unsigned, byte swapped
`BT_INT32`	32 bit integer, signed (2 Modbus registers)
`BT_UINT32`	32 bit integer, unsigned* (2 Modbus registers)
`BT_INT32LE`	32 bit integer, signed, word flipped (2 Modbus registers)

BT_UINT32LE	32 bit integer, unsigned*, word flipped (2 Modbus registers)
BT_INT32BLE	32 bit integer, signed, byte flipped (2 Modbus registers)
BT_UINT32BLE	32 bit integer, unigned*, byte flipped (2 Modbus registers)
BT_INT32LBE	32 bit integer, signed, byte and word flipped (2 Modbus registers)
BT_UINT32LBE	32 bit integer, unigned, byte and word flipped (2 Modbus registers)
BT_FLOAT	IEEE 754 Floating point (2 Modbus registers). See below.
BT_FLOATLE	IEEE 754 Floating point, word flipped (2 Modbus registers). See below.
BT_FLOAT64	IEEE 754 64 bit Floating point* (4 Modbus registers). See below.
BT_FLOAT64LE	IEEE 754 64 bit Floating point*, word reversed (4 Modbus registers). See below.

Return value

The function returns the decoded value as a Pawn cell.

*Note that the return value is always a 32 bit cell. This means that a large 32 bit unsigned value will be represented as a negative number in Pawn.

Floating point return values must be tagged as Float, otherwise they are rounded to an integer.

64 bit floating point values will be converted to the closest 32 bit value.

If there is an error, the return value will be zero, and the error parameter will be set to a non-zero value.

Example usage

```
   new err, x;
   x = MBRTU_Read( 5, MB_HOLDING, 101, BT_INT16, err );
   if(err == 0) {
     // x holds the value of holding register 40102 (logical addressing)
   }
```

Example usage with floating point

```
   new err, Float:x;
   x = Float:MBRTU_Read( 5, MB_HOLDING, 201, BT_FLOAT, err );
   if(err == 0) {
     // x holds the Float-value of holding registers 40202,40203 (logical
addressing)
   }
```

Modbus communication is non-trivial. Please refer to the Modbus documentation at https://modbus.org. eze System

can't assist with questions related to third party devices. We strongly recommend using our developed and tested drivers for any deployed systems. The functions described here should only be used for testing.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/mbrtu_read

Last update: **2023-10-02 18:51**

MBTCP_Read

Read a single register from a Modbus/TCP server

Description

```
MBTCP_Read(ip, unitid, cmd, reg, format=BT_UINT16, &error)
```

This command issues a Modbus/TCP read command and waits for a valid reply. If an error occurs (timeout, or error reply from the device), the error parameter is set to a non-zero value.

Note that the ip parameter only holds the last octet of the IP. It is assumed and required that the ezeio and the Modbus server is on the same /24 subnet. IP configuration is handled via the system settings.

Do not use Modbus/TCP on public networks. It is not a secure protocol.

Using this command in the user script will disable any system drivers using Modbus to avoid conflicts.

Parameters

`ip`	Last octet of the modbus server
`unitid`	Modbus/TCP unit address (0-255)
`cmd`	The Modbus command, see below
`reg`	The register physical address
`format`	Register type, see below. Optional, defaults to UINT16
`error`	Variable to receive the result code. Optional.

Modbus commands

`MB_COIL`	Read coil (Modbus command 1)
`MB_DISCRETE`	Read discrete input (Modbus command 2)
`MB_HOLDING`	Read holding register (Modbus command 3)
`MB_INPUT`	Read input register (Modbus command 4)

Register types

`BT_INT8`	8 bit integer, signed
`BT_UINT8`	8 bit integer, unsigned
`BT_INT16`	16 bit integer, signed
`BT_INT16LE`	16 bit integer, signed, byte swapped

BT_UINT16	16 bit integer, unsigned
BT_UINT16LE	16 bit integer, unsigned, byte swapped
BT_INT32	32 bit integer, signed (2 Modbus registers)
BT_UINT32	32 bit integer, unsigned* (2 Modbus registers)
BT_INT32LE	32 bit integer, signed, word flipped (2 Modbus registers)
BT_UINT32LE	32 bit integer, unsigned*, word flipped (2 Modbus registers)
BT_INT32BLE	32 bit integer, signed, byte flipped (2 Modbus registers)
BT_UINT32BLE	32 bit integer, unigned*, byte flipped (2 Modbus registers)
BT_INT32LBE	32 bit integer, signed, byte and word flipped (2 Modbus registers)
BT_UINT32LBE	32 bit integer, unigned, byte and word flipped (2 Modbus registers)
BT_FLOAT	IEEE 754 Floating point (2 Modbus registers). See below.
BT_FLOATLE	IEEE 754 Floating point, word flipped (2 Modbus registers). See below.
BT_FLOAT64	IEEE 754 64 bit Floating point* (4 Modbus registers). See below.
BT_FLOAT64LE	IEEE 754 64 bit Floating point*, word reversed (4 Modbus registers). See below.

Return value

The function returns the decoded value as a Pawn cell.

*Note that the return value is always a 32 bit cell. This means that a large 32 bit unsigned value will be represented as a negative number in Pawn.

Floating point return values must be tagged as Float, otherwise they are rounded to an integer.

64 bit floating point values will be converted to the closest 32 bit value.

If there is an error, the return value will be zero, and the error parameter will be set to a non-zero value.

Example usage

```
   new err, x;
   x = MBTCP_Read( 105, 1, MB_HOLDING, 101, BT_INT16, err );
   if(err == 0) {
     // x holds the value of holding register 40102 (logical addressing)
   }
```

Example usage with floating point

```
   new err, Float:x;
   x = Float:MBTCP_Read( 105, 1, MB_HOLDING, 201, BT_FLOAT, err );
   if(err == 0) {
     // x holds the Float-value of holding registers 40202,40203 (logical
addressing)
```

```
}
```

Modbus communication is non-trivial. Please refer to the Modbus documentation at https://modbus.org. eze System can't assist with questions related to third party devices. We strongly recommend using our developed and tested drivers for any deployed systems. The functions described here should only be used for testing.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/mbtcp_read

Last update: **2023-10-02 18:52**

memcpy

Copy byte buffer

Description

```
memcpy( dest[], source[], index, length )
```

Copy the `source` string to `dest`, starting at the `index` byte in dest, and counting `length` bytes.

Parameters

`dest`	destination buffer
`source`	string to be copied
`index`	offset in `source`
`length`	number of bytes to copy

Return value

Returns number of bytes copied.

Example usage

```
    new s1{20};
    new s2{20};

    strcopy(s1, "ABCdef");
    memcpy(s2, s1, 3, 3);

    // s2 now contains "def" - NOTE that no terminating NULL character is
copied
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/memcpy

Last update: **2019-11-18 22:37**

MIN

Macro that will return the smaller of two values

Description

```
MIN( value1, value2 )
```

Returns the smaller of two values

Works with both float and integer parameters.

Parameters

value1	input value
value2	input value

Return value

Returns the smaller of value1 and value2.

Example usage

```
    new Float:n;

    n = MAX(12.4, -3.14);     // -3.14
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/min

Last update: **2019-11-19 00:15**

mktime

Convert time and date into Unixtime value

Description

```
mktime(year, month, day, hour, minute, second, [local])
```

The date and time will be converted into a Unixtime value (seconds since UTC 1970-01-01).

By default (of `local` is false/zero or not included), the date is handled as UTC. If `local` is true, the local timezone is considered.

Parameters

`year`	Year input
`month`	Month input
`day`	Day of month input
`hour`	Hour input
`minute`	Minute input
`second`	Minute input
`[local]`	Set to true/1 if input is in local timezeone (optional, default=UTC/false)

Return value

Returns the number of seconds since 1970-01-01 00:00:00 (UTC)

Example usage

```
new utcIndependenceDay;

utcIndependenceDay = mktime(2022, 7, 4, 12, 0, 0); // returns 1656936000
```

NOTE: First appeared in firmware 22110901

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/mktime

Last update: **2022-11-09 19:50**

ModbusPortConfig

Configure the RS485 port Modbus/RTU settings

Description

```
ModbusPortConfig( speed, settings )
```

This command will configure the RS485 port.

Parameters

speed	Desired bit rate (1200 - 115200 bps)
settings	Bitmap combination of UART_WLEN_*, UART_STOP_* and UART_PAR_*

Settings

UART_WLEN_8	8 data bits
UART_WLEN_7	7 data bits
UART_STOP_ONE	Transmit one stop bit (receiving will always accept one or two stopbits)
UART_STOP_TWO	Transmit two stop bits (receiving will always accept one or two stopbits)
UART_PAR_NONE	No parity bit
UART_PAR_EVEN	Even parity
UART_PAR_ODD	Odd parity

Return value

No return value.

Example usage

```
ModbusPortConfig( 19200, UART_WLEN_8 | UART_STOP_ONE | UART_PAR_NONE );
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/modbusportconfig

Last update: **2023-10-02 17:47**

PDebug

Print to the debug terminal

Description

```
PDebug( const text[], ... )
```

Print a string to the debug terminal if it is open and active.

Parameters

`text`	String to print with optional formatting placeholders
…	Optional parameters

Formats a text string and outputs it to the debug terminal. If the debug terminal is not open and active, this command has no effect. Messages are not buffered. They will only be sent and printed if the terminal is open.

The text string use formatting similar to the C language 'printf' command.

Up to 20 messages will be sent in a burst. After that, the message rate is limited to one message per second. For example, if 25 messages are generated at once (within 1 second), only the first 20 will be sent. If 5 new messages are produced 10 seconds later, all 5 will be sent.

Each message will output on the debug terminal with a time stamp (minutes, seconds, 1/100's from startup) and the script that produced the message (USR/MOB/SDI etc..).

Return value

Returns 1 if the message was sent to the debug terminal. If no terminal is open, communication is not available or the rate limit is active, the return value is 0.

Example usage

```
new x = 10;
new s{} = "Some Text";
new Float:f = 123.45;

PDebug("Hello World %d", x);  // Outputs: 08:36.10 USR "Hello World 10"

PDebug("A String:%s, A Float:%f", s, f);  // Outputs: 08:36.11 USR "A
```

```
String:Some Text, A Float:123.45000"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/pdebug

Last update: **2019-09-02 21:40**

PID-functions

Set of helper functions to initiate and update a PID control loop

Description

```
PID_new( pid[], Float:set, Float:out ) // set up new PID
```

```
PID_dir( pid[], direction ) // set direction of control
```

```
PID_set( pid[], Float:set ) // adjust the target value
```

```
PID_tune( pid[], Float:Kp, Float:Ki, Float:Kd ) // set the P, I, D parameters
```

```
PID_limits( pid[], Float:min, Float:max ) // set the control range
```

```
PID_update( pid[], Float:in ) // update PID with feedback, and return new
control value
```

Parameters

`set`	setpoint / target
`out`	Initial output value
`direction`	1 (normal) or -1 (reverse)
Kp	Proportional component
Ki	Integral response component
Kd	Derivative response component
`min`	Smallest output value allowed
`max`	Largest output value allowed

Return value

The PID_update function returns the calculated output value.

Example usage

```
new p[PID];  // create a PID

main()
{
   // Initialize the PID with target and no output
```

```
    PID_new(p, 50.0, 0.0);

    // Limit output to +/- 10
    PID_limits(p, -10.0, 10.0);

    // Set the tuning parameters
    PID_tune(p, 0.2, 0.04, 0.01);

    // Set update speed to 100ms
    SetTickInterval( 100 );
}

@Tick(uptime)
{
    new Float:psi;
    new Float:v;

    // Read and scale a sensor input (0-100psi from 4-20mA sensor)
    psi = 100.0 * ((GetInputValue(1, INVAL_RAW)-4000)/16000);

    // Update the PID with the feedback value
    v = PID_update(p, psi);

    // Apply to output
    SetOutput(4, fround(v+50));
}
```

NOTE: The values used in the above example are not from a real world setup. Please make sure you understand how PID works and how to correctly tune the variables if you use any of the PID functions.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/pid_new

Last update: **2022-03-30 23:44**

Ping

Sends an ICMP (Ping) to another Ethernet connected device

Description

```
Ping( IP1, IP2, IP3, IP4 )
```

```
Ping( IP4 )
```

Send a 32 byte packet to other device and return the roundtrip time in milliseconds

Parameters

IP1	First IPv4 octet
IP2	Second IPv4 octet
IP3	Third IPv4 octet
IP4	Fourth IPv4 octet

If only one octet is given, it's assumed to be the last octet, and the rest of the address is the same as the ezeio.

Return value

Returns the roundtrip time in milliseconds, or 0 if there is an error.

Example usage

```
new ms;
ms = Ping(8, 8, 4, 4);
PDebug("Ping took %d ms", ms);
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/ping

Last update: **2026-01-08 01:13**

RegDirty

Find if a register value has been written to by any external process

Description

```
RegDirty( nDvc, nReg )
```

Returns 1 if the register has been written to by any other driver or system function. Returns 0 if not.

Parameters

nDvc	Driver number
nReg	Register number

This function mostly has utility in the development of shared drivers.

It is used to track changes to a register, and simplifies handling of writeable registers.

Every register has a 'dirty' status flag. This flag is set if the register is written to by any function other than SetRegister inside the same driver. By executing this function, the dirty flag is reset to 0.

Return value

Returns 1 (dirty) or 0 (not dirty)

Example usage

```
    if( RegDirty( nDvc, 5 ) ) {    // The register was touched
        // write value to external device
    } else {
        // read value from external device
        SetRegister( nDvc, 5, readvalue );
    }
```

This function first appeard in firmware 26010701

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/regdirty

Last update: **2026-01-08 01:42**

requestHibernate

Request from the ezeio to enter a hibernation cycle

Description

```
requestHibernate( cycles )
```

This will cause the ezeio to send a request to hibernate to the servers following the next 10 minute log collection. Please read the section about hibernation modes.

Parameters

cycles	Number of 10-minute cycles before re-connecting to the servers (0=cancel, 1-15)

Return value

No return value

Example usage

```
   requestHibernate( 6 );

   // If server config is set to "Controller Request", the ezeio will
hibernate for 6x10 minutes following this command.
```

NOTE: First appeared in firmware 21100601

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/requesthibernate

Last update: **2022-03-28 17:19**

SetBit

Helper function to set the state of a single bit in a byte-buffer.

Description

```
SetBit( buffer, bitno, value )
```

Set the state of a single bit in a buffer

Parameters

`buf`	Reference to the byte buffer
`bitno`	The bit number (0-indexed)
`value`	0 or 1 (any non-zero value is considered =1)

Return value

Returns the state of the bit (1 or 0) before it was set.

Example usage

```
   new buffer{80};

   SetBit(buffer, 18, 1);
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setbit

Last update: **2020-08-14 20:42**

SetCellBit

Helper function to set the state of a single bit in a cell value.

Description

```
SetCellBit( cell, bitno, value )
```

Set the state of a single bit in a cell variable

Parameters

`cell`	The cell variable to manipulate
`bitno`	The bit number (0-indexed)
`value`	0 or 1 (any non-zero value is considered =1)

Return value

Returns the state of the bit (1 or 0) before it was set.

Example usage

```
new v = 9;  // 9 = binary 1001
new b;

b = SetCellBit( v, 2, 1 );

// v is now 13 (binary 1101)
// b is 0 (previous value of bit 2)
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setcellbit

Last update: **2024-04-10 18:08**

SetDeviceStatus

Set the device status

Description

```
SetDeviceStatus( deviceno, status item, new value )
```

This function returns no value.

Parameters

`deviceno`	Device number to set
`status item`	The status item to set (see GetDeviceStatus)
`new value`	The value to set

See GetDeviceStatus for possible values for the Status item parameter.

Return value

0

Example usage

```
   SetDeviceStatus(1, DVCSTAT_COMMERR, 0);    // reset comm counter to 0 for device #1.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setdevicestatus

Last update: **2021-06-14 22:51**

SetDriverNo

System function to capture the driver number from the script

Description

```
SetDriverNo( n )
```

Saves the driver number internally

Parameters

This function is used by system drivers to capture the current driver number.

There is no case where this function should be used in a user script.

Example usage

```
    SetDriverNo( nDvc ) ;
```

This function first appeard in firmware 26010701

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setdriverno

Last update: **2026-01-08 01:43**

SetField/SetFieldFloat

Set the value of a field

Description

```
SetField( fieldno, value )
```

```
SetFieldFloat( fieldno, Float:value )
```

Set the integer or float value of a field and flag the value as changed.

Note that the Math logic for the field is continuously evaluated, so if the math logic results in a different value, it will immediately overwrite the value set by this command. To only set the field from script logic, leave the Math blank.

Parameters

`fieldno`	Which field to fetch
`value`	Value to set the field to (integer for SetField, Float for SetFieldFloat)

Return value

None

Example usage

```
new x;

SetField(4, 123);

// Field 4 is set to the value 123.

SetFieldFloat(4, 123.45);

// Field 4 is set to the value 123.45.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setfield

Last update: **2020-09-16 23:15**

SetInputMode

Set the input mode on a native ezeio input.

Description

```
SetInputMode( inputno, mode )
```

This will set the hardware input mode on a ezeio input. The command only works on native ezeio inputs - on the ezeio controller or ezeio expansion I/O devices. Third party modules use different methods to change input configuration.

Parameters

`inputno`	Input number to change. Inputs 1-8 are on the ezeio. Use “DVCn+inputno” for expansion devices
`mode`	The desired mode. See below.

Available modes:

Mode	Hardware config	Input unit	Range
INMODE_10V	High impedance, >60kΩ	mV (Volt)	0-10000mV
INMODE_30mA	200Ω shunt to ground	µA (current)	0-30000µA
INMODE_PULLUP	Weak pull-up to +5V, <1mA	Ohm (resistance)	0-1000000Ω
INMODE_TEST		internal test mode	
INMODE_PULSE_EXCITE	Weak pull-up to +5V, <1mA	mHz (frequency)	0-400000mHz
INMODE_PULSE	High impedance, >60kΩ	mHz (frequency)	0-400000mHz
INMODE_THERMISTOR_10k_2	Weak pull-up to +5V, <1mA	°K x100 (Kelvin)	20000-70000 (200-700°K)
INMODE_THERMISTOR_10k_3	Weak pull-up to +5V, <1mA	°K x100 (Kelvin)	20000-70000 (200-700°K)
INMODE_THERMISTOR_10kB3380	Weak pull-up to +5V, <1mA	°K x100 (Kelvin)	20000-70000 (200-700°K)
INMODE_THERMISTOR_2k2	Weak pull-up to +5V, <1mA	°K x100 (Kelvin)	20000-70000 (200-700°K)
INMODE_THERMISTOR_100k	Weak pull-up to +5V, <1mA	°K x100 (Kelvin)	20000-70000 (200-700°K)
INMODE_RTD_PT1000	Weak pull-up to +5V, <1mA	°K x100 (Kelvin)	20000-70000 (200-700°K)

Return value

none

Example usage

```
   SetInputMode(3, INMODE_30mA);                    // Set input 3 on ezeio to measure current
   SetInputMode(DVC3+1, INMODE_THERMISTOR_10k_2); // Set input 1 on I/O expansion 3 to measure a thermistor
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setinputmode

Last update: **2019-09-02 21:41**

SetModbusTimeout

Set the bus timeout for Modbus/RTU communication

Description

```
SetModbusTimeout( ms )
```

Set the timeout in milliseconds that bus commands will wait for a device to reply.

Parameters

ms	Timeout in milliseconds

Return value

This function returns no value.

Example usage

```
   SetModbusTimeout(1000);  // Bus commands will wait up to 1s for replies.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setmodbustimeout

Last update: **2023-10-02 17:51**

SetOutput

Set an output

Description

```
SetOutput( outputno, value )
```

Directly set the value/state of a native ezeio output. The output value is always in percent , 0-100. For digital outputs, any value 50 and higher will set the output to ON (100%).

This command operates the output immediately.

Output 1-4 are the on-board ezeio outputs.
To reference outputs on ezeio expansion units, use “DVCn + output number”. See example below.

This command only works to control native ezeio outputs. To control outputs on third party modules, use the appropriately mapped registers. See documentation for the relevant driver.

Parameters

`outputno`	Output number
`value`	Value (0-100%)

Return value

none

Example usage

```
SetOutput(4, 31);   // Set the on-board analog output to 3.1V.

SetOutput(DVC2+4, 100); // Set output 4 on the second I/O expander to ON.
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setoutput

Last update: **2019-09-02 21:41**

SetRegister

Set the value of a register

Description

```
SetRegister( deviceno, registerno, {Fixed,Float,_}:value, type=INT )
```

Directly sets the value of a register.

Parameters

deviceno	Device number (1-40)
registerno	Which register to set
value	The new value
type	One of INT, UINT, FLOAT or NULL

Return value

none

Example usage

```
    AllocateRegisters(3, 8);         // Allocate eight registers for device 3

    SetRegister(3, 1, 1234, INT);    // Set register 1 on device 3 to the value 1234
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setregister

Last update: **2020-08-14 20:19**

SetSystemItem

Change value of a given system parameter

Description

```
SetSystemItem( item, value )
```

Change the value of a given system parameter

Parameters

item	The system item to set (see below)
value	The new value

Parameter	Description
SYSITEM_GPSX	GPS latitude in degrees x 10^6
SYSITEM_GPSY	GPS longitude in degrees x 10^6
SYSITEM_GPSZ	GPS elevation in meters x 10
SYSITEM_GPSSIGNAL	Received signal level from GPS
SYSITEM_CELLINHIBIT	0=normal operation. >0 cellular modem is off
SYSITEM_GPSLOCK	0=No current GPS position. 1=GPS position current
SYSITEM_OUT3RCPWM	0=0-100% duty cycle, 1=1-2ms pulse for RC servo

Return value

None

Example usage

```
   SetSystemItem( SYSITEM_CELLINHIBIT, 600 );  // Power off the cell modem for
10 minutes
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setsystemitem

Last update: **2020-08-18 22:11**

SetTickInterval

Set the rate of the calls to @Tick.

Description

```
SetTickInterval( interval_ms )
```

This will change the rate of which the system triggers the @Tick event.

Parameters

`interval_ms`	The interval between events. 50-10000ms.

Return value

none

Example usage

```
main() {
   SetTickInterval(100);  // Set tick rate to 100ms
}

@Tick(uptime) {
   // this code will be called ever 100ms
}
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/settickinterval

Last update: **2019-09-02 21:42**

SetTimer

Set a timer or interval

Description

```
SetTimer( timernumber, timeout, parameter=0, repeat=0 )
```

Set up a timer with a timeout (in milliseconds) and an optional user supplied parameter. There are four independent timers available to each script.

After the given timeout, the “@Timer” event handler function will be called.

To cancel a timer, simply set the interval to 0 like this: `SetTimer(0, 0)`

Parameters

`timernumber`	Timer number 0-3
`timeout`	Timeout in milliseconds, 1-3600000 (1h max)
`parameter`	User defined value
`repeat`	Number of times to repeat the timer (0-1000), or `TIMER_INF` for infinite

Return value

none

Example usage

```
    main()
    {
        SetTimer(0, 5000, 1234, TIMER_INF);  // Set up timer to call back
every 5 seconds
    }

    @Timer(timerno, timeout, param)
    {
        // Code to run every time the timer expires
    }
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/settimer

Last update: **2019-11-19 00:02**

SetVal

Set value of a user-defined memory cell

Description

```
SetVal( address, value )
```

Set the value of one of the user defined memory cells. There are 512 32-bit cells available. The values are stored in NVM, so they will remain through power loss/reset.

UPDATE since firmware 22070801: Expanded to 2048 cells. Only the first 512 are stored in NVM.

Parameters

address	Address, 0-511
value	New value

Return value

Returns the previous value of the cell.

Example usage

```
new x;

SetVal(100, 4711);  // Set cell with address 100 to value 4711
x = GetVal(100);    // Read cell 100

// x now has the value 4711
```

NOTE: First appeared in firmware 21071901

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/setval

Last update: **2022-07-13 17:04**

sleep

Pause script for a short duration

Description

```
sleep( milliseconds )
```

Pause the script for `milliseconds`.

USE WITH CAUTION

This command will prevent events to be executed and may cause unwanted delays in other logic.

Parameters

`milliseconds`	The duration for which to pause the script

Return value

This command does not return a value.

Example usage

```
main()
{
   SetOutput(1, 100);
   sleep(200);             // Allow output to be ON for 200 ms
   SetOutput(1, 0);
}
```

An alternative to using the sleep function is to use a timer, like this:

```
main()
{
   SetOutput(1, 100);      // Turn output on
   SetTimer(1, 200);       // Set up timer 1 to trip in 200ms
}
```

```
@Timer( timerno, timeout, param )
{
   SetOutput(1, 0);        // Turn output off
}
```

One benefit of using a timer is that you code continues to execute while the timer is running.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/sleep

Last update: **2020-08-15 00:12**

strcat

Concatenate strings

Description

```
strcat( dest[], const source[], maxlength=sizeof dest )
```

Copy the string `source` to the end of `dest`.

Parameters

`dest`	destination string
`source`	string to be added to dest
`maxlength` (optional)	max total length of the result

Return value

Returns the number of characters added to the string.

Example usage

```
    new s{20} = "Hello ";

    strcat(s, "World"); // returns 5

    // s is now "Hello World"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strcat

Last update: **2019-09-02 21:42**

strcmp

Compare strings

Description

```
strcmp( const string1[], const string2[], bool:ignorecase=false, length=1024 )
```

Compare string1 and string2

Parameters

`string1`	first string to be compared
`string2`	second string to be compared
`ignorecase` (optional)	if set to true (1), ignore the case
`length` (optional)	max number of characters to compare

Return value

Returns 0 if the strings are equal.
Returns 1 if string1 > string2.
Returns -1 if string1 < string2.

Example usage

```
    new x;
    new s1{20} = "ABC";
    new s2{20} = "abc";

    x = strcmp(s1, s2, true);  // ignore case

    // x is now 0
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strcmp

Last update: **2019-09-02 21:42**

strcopy

Copy string

Description

```
strcmp( dest[], source[], maxlength=sizeof dest )
```

Copy the `source` string to `dest`.

Parameters

`dest`	destination buffer
`source`	string to be copied
`maxlength` (optional)	maximum number of characters to copy

Return value

Returns number of characters copied.

Example usage

```
    new s{20};

    strcopy(s, "ABCdef");  // returns 6

    // s now contains "ABCdef"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strcopy

Last update: **2019-09-02 21:43**

strdel

Remove bytes from a string

Description

```
strdel( string[], start, end )
```

Remove some characters from a string

Parameters

`string`	String to be manipulated
`start`	Position of first character to be removed
end	Position of last character to be removed. Must be after `start`

Return value

Returns 1 if successful.
Returns 0 on error.

Example usage

```
    new s{20} = "CatDogAnt";

    strdel(s, 3, 6);  // returns 1

    // s is now "CatAnt"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strdel

Last update: **2019-09-02 21:43**

strfind

Searches a string for a substring

Description

```
strfind( const string[], const sub[], bool:ignorecase=false, index=0 )
```

Search the haystack `string` for a needle `sub`. Optionally ignore case and start at an offset in the haystack.

Parameters

`string`	String to be searched (the haystack)
`sub`	The string we're looking for (the needle)
`ignorecase` (optional)	If set to true, the search will ignore case
`index` (optional)	Start at an offset different than the start of the haystack string

Return value

Returns the offset of `sub` in `string` if successful.
Returns -1 if no occurrence of `sub` was found in `string`.

The returned offset is always relative to the start of the `string`, even if a search offset is given.

Example usage

```
new x;
new s{20} = "CatDogAnt";

x = strfind(s, "Dog");

// x is now 3
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strfind

Last update: **2019-09-02 21:43**

strformat

Formats a string with placehoders

Description

```
strformat( dest[], size, bool:pack, const format[], {Fixed,Float,_}:... )
```

Formats a string with C-style *printf* placeholders.

Possible placeholders are:

- %d : integer number
- %c : character
- %f : floating point number
- %x : hexadecimal number
- %s : string

Insert a number between the % and the letter to pad the field with spaces to a specific length.
Example: %4d

Parameters

`dest`	Destination buffer
`size`	Size of the destination buffer in 'cells'.
`pack`	Creates a packed string if set to true
`format`	String with placeholders
`...`	Variable number of parameters

Return value

Returns 1 if successful.
Returns 0 on error.

Example usage

```
new s{40};
new a = 5;
new b{20} = "cats";

strformat(s, sizeof s, true, "The dog likes %d %s.", a, b); // returns 1
```

```
    // s now contains "The dog chased 5 cats."
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strformat

Last update: **2019-09-02 21:43**

strins

Insert a substring in a string

Description

```
strins( string[], const substr[], index, maxlength=sizeof string )
```

Inserts a substring into a string.

Parameters

`string`	The string that will be manipulated
`substr`	The string that will be inserted
`index`	The location in `string` where the `substr` will be inserted. If 0, the substr will be prepended to string
`maxlength` (optional)	The size of dest in cells

Return value

Returns 1 on success, 0 on failure.

Example usage

```
    new s{40} = "CatAnt";
    new b{20} = "Dog";

    strins(s, b, 3);  // returns 1

    // s now contains "CatDogAnt"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strins

Last update: **2019-09-02 21:44**

strlen

Find the lengths of a string

Description

```
strlen( const string[] )
```

Counts the number of characters in `string`, not including the terminating null.

Parameters

`string`	The string that will be checked

Return value

Returns the number of characters in `string`

Example usage

```
new x;
new s{40} = "CatDogAnt";

x = strlen(s);

// x is now 9
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strlen

Last update: **2019-09-02 21:44**

strmid

Copy a subsection of a string

Description

```
strmid( dest[], const source[], start=0, end=cellmax, maxlength=sizeof dest )
```

Copy a subset of characters from `source` into `dest`.

Parameters

`dest`	The buffer for the result
`source`	The string that we will extract data from
`start`	The location in `source` where we start extracting data from
`end`	The location in `source` where we will stop copying.
`maxlenght` (optional)	The size in cells of the dest buffer

Return value

The number of characters copied into dest.

Example usage

```
    new s{20};
    new b{20} = "CatDogAnt";

    strmid(s, b, 3, 6);  // returns 3

    // s now contains "Dog"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strmid

Last update: **2019-09-02 21:44**

strpack

Create a packed copy of a string

Description

```
strpack( dest[], const source[], maxlength=sizeof dest )
```

This function copies a string from source to dest and stores the destination string in packed format. The source string may either be a packed or an unpacked string.

NOTE: This is a rarely needed command for ezeio programming.

Parameters

`dest`	The buffer for the result
`source`	The string that we will copy. May be packed or not.
`maxlenght` (optional)	The size in cells of the dest buffer

Return value

Number of characters copied

Example usage

```
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strpack

Last update: **2019-09-02 21:44**

strunpack

Create an unpacked copy of a string

Description

```
strunpack( dest[], const source[], maxlength=sizeof dest )
```

This function copies a string from source to dest and stores the destination string in unpacked format. The source string may either be a packed or an unpacked string.

NOTE: This is a rarely needed command for ezeio programming.

Parameters

`dest`	The buffer for the result
`source`	The string that we will copy. May be packed or not.
`maxlenght` (optional)	The size in cells of the dest buffer

Return value

Number of characters copied

Example usage

```

```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strunpack

Last update: **2019-09-02 21:44**

strval

Find the numeric value of a string.

Description

```
strval( const string[], index=0, limit=15 )
```

strval will evaluate all digits starting at the index position until it encounters a non-digit or evaluated `limit` characters whichever comes first.

Parameters

`string`	The buffer to be evaluated
`index`	The offset in the buffer to start looking for a number.
`limit`	The max number of digits to evaluate, starting after `index`.

Return value

The value in the string.
Returns 0 if the string starting at the index did not start with a valid number.

Example usage

```
new x;
new s{20} = "Cat47";

x = strval(s, 3);

// x is now 47
```

NOTE: The `limit` parameter requires firmware 24020101 or newer.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/strval

Last update: **2024-02-05 23:20**

SunPosition

Helper function to calculate the sun's position at a given time and position on the earth.

Description

```
SunPosition( &Float:Elevation, &Float:Azimuth, Float:Lat, Float:Lng, time )
```

The function calculates the sun's position (Elevation & Azimuth) at the given lat/long coordinate at the given time.

The function does not compensate for altitude, but assumes sea level.

Parameters

`Elevation`	Variable to receive the result elevation in meters
`Azimuth`	Variable to receive the result azimuth in degrees
`Lat`	Latitude in degrees
`Lng`	Longitude in degrees
`time`	UTC time as seconds since epoch (unixtime)

If Lat/Lng and time are not submitted, the current GPS position and current time is assumed

Return value

The function does not return a value, but sets the Elevation and Azimuth variables.

Example usage

```
   new Float:el;
   new Float:az;

   SunPosition( el, az, 38.67, -121.15, 1597439230 );

   // Sun position in Folsom, CA on August 14 2020 @ 14:07 local time (21:07
UTC)
   // el = 62.2 degrees, az = 211.4 degrees (rounded)

   // Using GPS/current time as defaults, omitting location and time
parameters:
```

```
SunPosition( el, az );
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/sunposition

Last update: **2022-03-03 20:31**

UpdateDeviceStatus

Update the status of a given device

Description

```
UpdateDeviceStatus( deviceno, commStatus, opStatus, appStatus, signal,
voltage, returnvalue=1 )
```

Updates the device status.

Parameters

deviceno	Device number to request status from (or 0 for all devices)
commStatus	Communication status, one of: DS_COM_UNKNOWN DS_COM_ERROR DS_COM_ISSUE DS_COM_OK DS_COM_BEST
opStatus	Operational status, one of: DS_OP_UNKNOWN DS_OP_ERROR DS_OP_PART DS_OP_FULL
appStatus	Application status, one of: DS_APP_UNKNOWN DS_APP_ERROR DS_APP_WARNING DS_APP_OK DS_APP_OK1 DS_APP_OK2 DS_APP_OK3 DS_APP_OK4
signal	Integer value, -127 to 127, typically signal strength where applicable
voltage	Voltage (if applicable) in mV (max 65V)
returnvalue	return value (default 1)

Only the first parameter, deviceno, is required. All other parameters are optional. Use _ (underscore) to skip parameters (eg `UpdateDeviceStatus( 3, DS_COM_OK, _, DS_APP_OK );` will set Comm and App status only on device 3)

Return value

Returns 1 by default, or the value of the `returnvalue` parameter.

Example usage

```
// Indicate normal condition
UpdateDeviceStatus( 3, DS_COM_OK, DS_OP_FULL, DS_APP_OK );

// Indicate marginal communication condition
UpdateDeviceStatus( 3, DS_COM_ERROR, DS_OP_UNKNOWN, DS_APP_UNKNOWN );

// Indicate application status
new batteryVoltage = 12345; // read the battery voltage from the device
UpdateDeviceStatus( 3, _, _, DS_APP_OK, _, batteryVoltage );
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/updatedevicestatus

Last update: **2025-07-10 18:21**

uudecode

Decode an UU-encoded string

Description

```
uudecode( dest[], const source[], maxlength=sizeof dest )
```

Since the UU-encoding scheme is for binary data, the decoded data is always “packed”. The data is unlikely to be a string (the zero-terminator may not be present, or it may be in the middle of the data).

A buffer may be decoded “in-place”; the destination size is always smaller than the source size. Endian issues (for multi-byte values in the data stream) are not handled.

Binary data is encoded in chunks of 45 bytes. To assemble these chunks into a complete stream, function memcpy allows you to concatenate buffers at bytealigned boundaries

NOTE: This is a rarely needed command for ezeio programming.

Parameters

`dest`	The buffer that will hold the decoded data
`source`	The uu-encoded source buffer
`maxlength` (optional)	The size of dest in cells. If the length of dest would exceed maxlength cells, the result is truncated.

Return value

The number of bytes stored in dest.

Example usage

```
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/uudecode

Last update: **2019-09-02 21:45**

uuencode

Decode an UU-encoded string

Description

```
uuencode( dest[], const source[], maxlength=sizeof dest )
```

This function always creates a packed string. The string has a newline character at the end.

Binary data is encoded in chunks of 45 bytes. To extract 45 bytes from an array with data, possibly from a byte-aligned address, you can use the function memcpy.

A buffer may be encoded “in-place” if the destination buffer is large enough. Endian issues (for multi-byte values in the data stream) are not handled.

NOTE: This is a rarely needed command for ezeio programming.

Parameters

`dest`	The buffer that will hold the encoded data
`source`	The plaintext source buffer
`maxlength` (optional)	The size of dest in cells. If the length of dest would exceed maxlength cells, the result is truncated.

Return value

Returns the number of characters encoded, excluding the zero string terminator; if the dest buffer is too small, not all bytes are stored.

Example usage

```
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/uuencode

Last update: **2019-09-02 21:45**

valstr

Convert an integer to a string

Description

```
valstr( dest[], value, bool:pack=true )
```

Parameter dest should be of sufficient size to hold the converted number. The function does not check this.

Parameters

dest	The buffer that will hold the string
value	integer value to be converted
pack (optional)	Controls if the resulting string is packed or not. Default is true (packed).

Return value

Number of characters writted to dest, excluding the terminating null character.

Example usage

```
new x = 1234;
new s{20};

valstr(s, x);  // returns 4

// s now contains "1234"
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/scriptref/valstr

Last update: **2019-09-02 21:45**

eze System Documentation

Go to eze.io

- IMPORTANT INFORMATION
- Introduction to the ezeio
- Registration
- Indicators
- Installing the ezeio
- Connecting Sensors and Peripherals
- User Interface
- Drivers
- Expressions
- Script reference
- API reference
- ezeio MkII I/O Expander (TEMPORARILY UNAVAILABLE)

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/sidebar

Last update: **2021-09-24 23:42**

User Interface

The eze.io user interface (UI) is accessible using most Internet browsers, such as Google's Chrome, Mozilla's Firefox, Microsoft's Edge, and Apple's Safari. So, there is no software to download and you can use any PC, laptop, or tablet. Navigating the interface on a small phone screen may be inconvenient due to the amount of information presented by the UI.

The UI is divided into two levels, manage and configure. The manage screen is designed for viewing data, simple controls, as well as user and system administration. Many users will have no need for access beyond this level and some will have restricted access within it. The configure screen is, as the name implies, for configuring individual ezeio controllers. The term configure is used because, in most cases, no programming is required. Adding sensors, relays and other devices can be done with a few clicks of a mouse. The next two sections of the manual will discuss the manage and configure screens in grater detail.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/start

Last update: **2021-09-27 22:55**

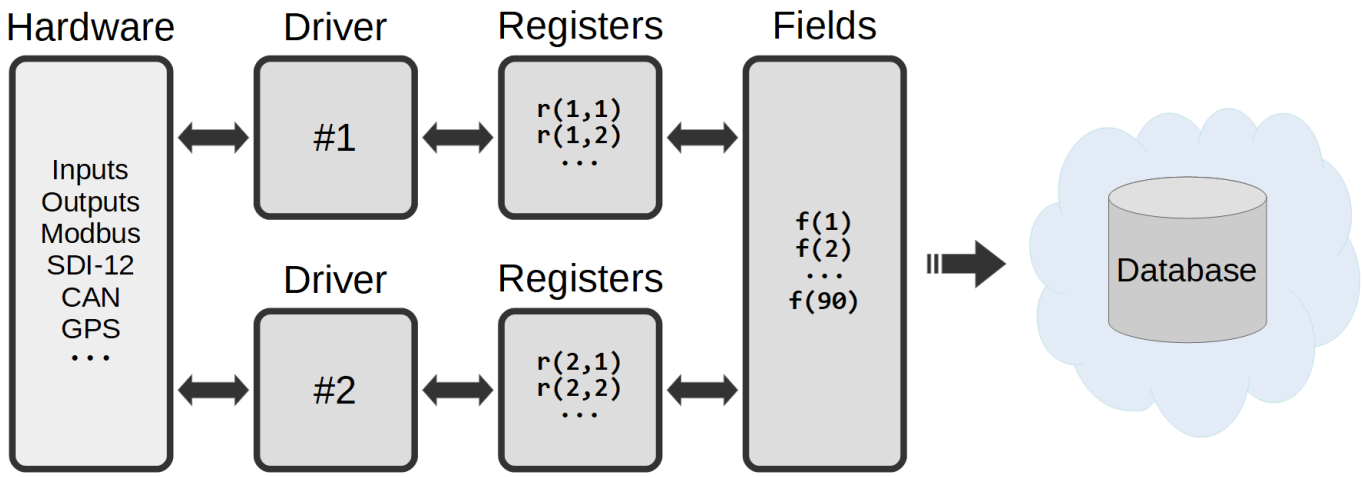
Configuration Screen

By default, the ezeio will connect to the cloud servers and synchronize data in both directions. This functionality is not optional, and normally requires no configuration. There are also a lot of other basic functions in the ezeio that always runs, such as scanning inputs, managing the power supply, keeping correct time etc.

The application functionality is controlled by the configuration system. This is similar to applications on a computer; the application controls if you are editing a spreadsheet or watching a movie, while the operating system runs in the background and provides the basics, like connecting to the Internet or taking input from the keyboard.

Similar to a computer, the ezeio functionality is dictated by Drivers. Drivers are in essence small programs that access resources (like inputs, outputs or communicates with a device via CAN or Modbus), and makes those resources available to the ezeio. In the ezeio, each driver allocates Registers (up to 150 per driver), and the registers are used in the ezeio to move data back-and-forth.

Data that needs to be recorded or sent to the servers must be mapped into Fields. Each ezeio can handle up to 90 (ninety) fields.



The ezeio can support up to 40 (forty) drivers at the same time, and each drivers typically works with a specific device or group of inputs/outputs. The resulting data is put into registers, where it can be easily referenced to by the fields.

The field data is automatically communicated to the servers.

Drivers are numbered 1 through 40. Usually each driver manages one device, such as a sensor or meter.

Registers are referenced by driver number and register number, so for example `r(4,12)` means it's the 12th register on device(driver) number 4.

Fields are simply referenced by their number; `f(1)` through `f(90)`.

From LOGIN to, LOGGING DATA, in 4 easy steps

As shown in the graphic above, the typical process starts with connecting hardware (sensor/device). The Connecting Sensors and Peripherals section of this manual covers the basics of physically connecting peripherals to the ezeio. (The Device driver may contain device specific notes for comm port wiring). Once your device is connected, login to eze.ioand follow these steps.

From the “System” page;

- click the configure button for the desired ezeio

From the “Devices” page;

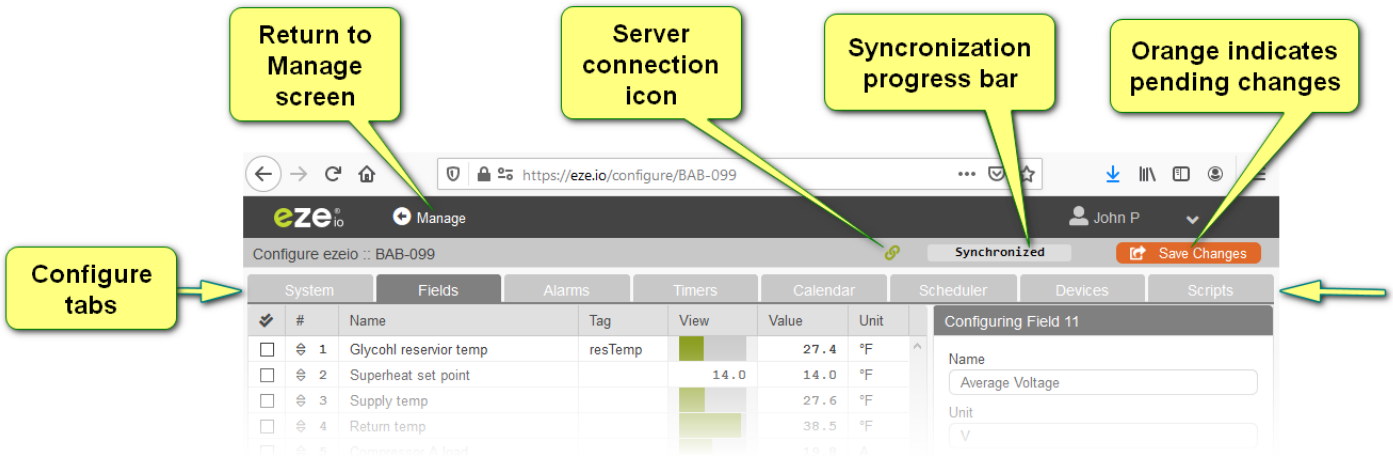
- Add a device driver, providing the programming to interpret the data or signal, allowing setting & preferences to be applied, and registers to be created on the ezeio. Save changes
- Select the registers you wish to add to Fields. *Fields can be created and linked manually* (see “Add Field” button)

From the “Fields” page;

- Configure the settings of each Field. Save changes

This simplified overview of the process is meant to serve as a generic guide. The process is the same for all external sources of data, whether that source is a simple sensor connected to one of the ezeio's input terminals or an engine control panel connect via Modbus (over a RS485 serial bus). The difference is in the specific settings within the “Device driver” and the amount of registers available from the device.

Configuration screen navigation



For users with full privileges, 8 tabs will be visible. Above these the banner contains (from left to right) the “Return to mange screen link” and the current users name. Clicking the user reveals the log out and view Terms of service options. Just below the banner, on the left, the serial number of the selected ezeio controller is displayed. To the right is the server connection icon, synchronization progress bar and save changes button.

Server connection icon

The ezeio controller connects periodically to the eze.io servers to send system and log data every 10 minutes (at a minimum). The server will also ping the ezeio at regular intervals. If all of these communications are normal, a green “linked ” symbol is displayed. If the server breaks the connection to the ezeio or the ezeio fails to respond to pings, an orange “broken link” symbol is displayed.

Synchronization progress bar

When saving changes, committing script edits or updating firmware, the progress bar will display the percentage of progress (graphically) in green as well as numerically. The delay in initiating these changes to the ezeio can vary from seconds to minutes. During this delay the progress bar will read “Update pending”. When no changes are pending the progress bar will read “Synchronized”.

Save changes button

Most changes made on the configuration level will cause the “Save changes button” to change from gray to orange with an animated symbol.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/start

Last update: **2021-09-27 22:55**

Alarms

Alarms are user configured event triggers (Each ezeio is limited to 200 alarms). An alarm is triggered by an “Alarm condition” and stays in alarm until the “Restore condition” is true (or until it is forced off). On their own alarms do not do anything, other than switch on or off like a digital flag. “Alarm actions” and “Restore actions” such as “Send a message” or “Set a Field Value” can be configured by the user to add the desired functionality. Each alarm can have up to 4 “Alarm actions” and 4 “Restore actions”. Alarms can be added or deleted using the buttons at the bottom of the page.

Alarm Table - The left panel of the Alarm page is a table listing all of the alarms, alarm / restore conditions and their current status. The table can be sorted by column.

Alarm Status - Several icons are used to indicate the status of an alarm. The table below lists describes the status icons in the progression of a typical alarm cycle.

	Restored	In ready state. Condition/s to alarm have not been achieved
	In Alarm Holdoff	Condition/s to alarm are true, alarm holdoff timer is counting down
	In Alarm	Condition/s for alarm are true, Alarm actions will be triggered
	Between conditions	Condition/s for alarm are no longer true, but restore conditions have not been achieved
	In Restore Holdoff	Condition/s to restore have been achieved, restore holdoff timer is counting down
	Restored	Condition/s to restore have been achieved, restore holdoff timer has expired. Restore actions will be triggered

ALARM PAGE

ALARM SETTINGS

Name - Appears on the alarm table. By default (if no subject has been formatted by a user) the alarm name is transmitted in “Send message” actions

Alarm condition settings - This is where the user describes the conditions that will trigger an alarm, using eze Systems' Expressions and Tags. This can be a single temperature threshold, the result of an equation or multiple conditions, such as the examples below.

Referencing a Field value	f(3) > 27	Value of Field #3 exceeds 27
Referencing a Field value (using it's asset tag)	f(<PSIout>) > 27	Value of Field #3 exceeds 27
Referencing a Register value	r(2,5) > 27	Value of register 5, on device #2 exceeds 27
Calculation based on two Field values (Water filter pressure drop)	f(2) - f(3) > 3	Differential of Fields #2 and #3 exceeds 3
Two conditions, both true (Oil pressure is low & engine running)	(f(6) < 30) && (f(7) > 600)	Logical AND, assesses the statements on each side of the &&
Field value and Schedule Status	(f(3) > 27) && sch(2)	Value of Field #3 exceeds 27 and Schedule #2 in active

Alarm Holdoff Time - The “Alarm condition” must persist for the amount of time specified in the “Alarm Holdoff”. If the alarm condition fluctuates between true and false, the Holdoff time will reset. The user can enter any time from 0.1 to 10,0000 seconds (1/10th of a second to 27.8 hours). During the hold off time, the status of the alarm will display a gray arrow pushing toward the alarm icon.

Alarm Action - As mentioned above, up to four Alarm Actions can be configured to perform a variety of tasks. Click on an Alarm Action's banner to expand and view its settings.

Active / Inactive - A check box is provided to allow a user to suspend an Alarm Action. This is very helpful when setting up and testing alarms or, for example when equipment is taken offline for service. By default a new Alarm action will be inactive.

Action Name - Use this to apply a unique name to the action. By default this name will be sent as part of the subject of a message when using the Send Message action, preceded by “ALARM” (or RESTORE *in the case of Restore Actions*).

Action Type - Several action types are available, Send Message being the most common. Others can be used to create automation. The list below will describe each Action Type and how to use it.

- **No Action** - This is the default state of an Alarm action. As discussed above, with no Alarm Actions configured an alarm is just a digital flag. An alarm does have a status that can referenced by other parts of the systems' configuration such as; User script, Fields and other Alarms.
- **Send Message** - Sends messages to multiple destinations / end points. Available messaging formats are; email, text (SMS), voice, push notification (app based messaging) and API. The group of destination is selected from the **Alarm Action List** drop down menu. These Destination Lists are created and managed from the Manage screen. See Destinations and Destination Lists for more information. The message field provided is intended for unique information relevant to this alarm and action. Live “Field” values, statuses and other information can be included via Message Templates, such as [F#] (Field Status) or [GPS] (GPS coordinates *if available*). Standardized message formatting can be applied to a Destination List, eliminating the need to add these details to individual alarm action messages.
- **Log Event** - Sends a message to the Event Log of the ezeio, found under the Systems tab. The

message can contain contain Message Templates as described above. This feature is helpful for trouble shooting / tracking issues. Each occurrence can be recorded without the annoyance of multiple emails or texts.

- **Set Field Value** - This action changes the value of a field. Examples of applications; control of analog and digital outputs, reset timers, counters or totalizers. Enter the Field number as the Index and the new value under “Value”
- **Set Register Value** - This action will write a value to device register, such as an output register of a Modbus I/O.
- **Force Alarm** - This action will force an alarm on
- **Force Restore** - This action will force an alarm off
- **Generate Script Event** - This action will trigger a event written in the User Script. For more information see the system callback function; **@action** under programming pattern, in the Script reference section of the user manual.

Condition Expression - Each Alarm Action and Restore Action has an optional Condition Expression that can be used as additional logic to control actions. This is normally left blank, triggering actions when the associated alarm becomes active. A common use of this option is to differentiate between two actions on the same alarm. For example; a door switch alarm could trigger a bell to ring during business hours and send a text message when outside of business hours.

Level - THIS FEATURE IS NOT ACTIVE.

RESTORE SETTINGS

Restore settings offer the same options as the Alarms setting including Restore Condition, Restore Holdoff time, and Restore Actions. The Restore Condition is required to return the alarm to a ready state where it can be triggered again. Without a “Restore Condition” a Force Restore action will be required to return the alarm to a ready state.

RETRIGGER ALARM SETTING

By default, an alarm will not retrigger until it has restored. This feature allows an alarm to retrigger the configured Alarm Actions a specified number of times at a regular interval, during the period of time the alarm remains active. Enter the “Count” (number of retriggers) and Delay (interval between re-triggers). Count can be from 1-100. To disable retriggering enter a 0 as the Count. Delay range is 0.1 to 10,0000 seconds (1/10th of a second to 27.8 hours).

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/alarms

Last update: **2022-10-18 15:04**

Calendar

The “Calendar” feature works in tandem with the “Scheduler”. It can be used as the primary way of applying schedules (seasonally for example) or as a way to mark exceptions to the weekly schedule (such as holidays or school minimum days). The calendar has no independent functionality, it is dependent on the “Scheduler” for creating and editing of the schedules for the “Day types”.

Calendar Navigation

The navigation bar at the top of the calendar shows the current month & year by default. The arrows at the far edges of the navigation bar allow the user to page month by month. Clicking on the month & year shown at the center will switch to a 12 month view. Clicking again will switch to a 12 year view. When the user clicks on one of the options displayed, the navigation will revert back to the previous view (12 month view or selected month view).

10 year rolling

This is a 10 year rolling calendar, so the system is only tracking the last digit on the year. For example: when you make changes to the year 2028, they will also appear on the years 2038, 2048 and so on. As your calendar rolls over and the “Day type” assignments are no longer relevant, the “Clear Month” button will clear all the markers from a given month.

Create and set day type

Eight day types are available for each ezeio's calendar. The user defined name and color of each day type can be changed at any time and changes will be reflected throughout the calendar by clicking “Save changes”. Schedules for the day types are created and edited in the “Scheduler”

Applying & removing day type assignments

Clicking the radio button to the right of the day type makes it active (turning your cursor into a magic wand). Simply click within a day on the calendar to assign that day type. Click again in the same day to remove. Select another day type and distribute it throughout the calendar. A day can be assigned any number of day types (0 to 8).

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/calendar

Last update: **2021-09-24 21:57**

Devices & Drivers

The “Device” page is where most links to hardware and data sources are created. This is done through the addition of drivers. Drivers are application programs that (in most cases) establish a communication link to hardware and provide an eze.io interface. This is similar to drivers, programs and apps that you would load on your phone, tablet or personal computer. For some such as “GPS receiver GN-803G” or the Expansion CAN port activation“ adding and saving changes is all that is required. Other driver's settings will range in complexity based on the requirements of the device or the level of customization provided by the driver.

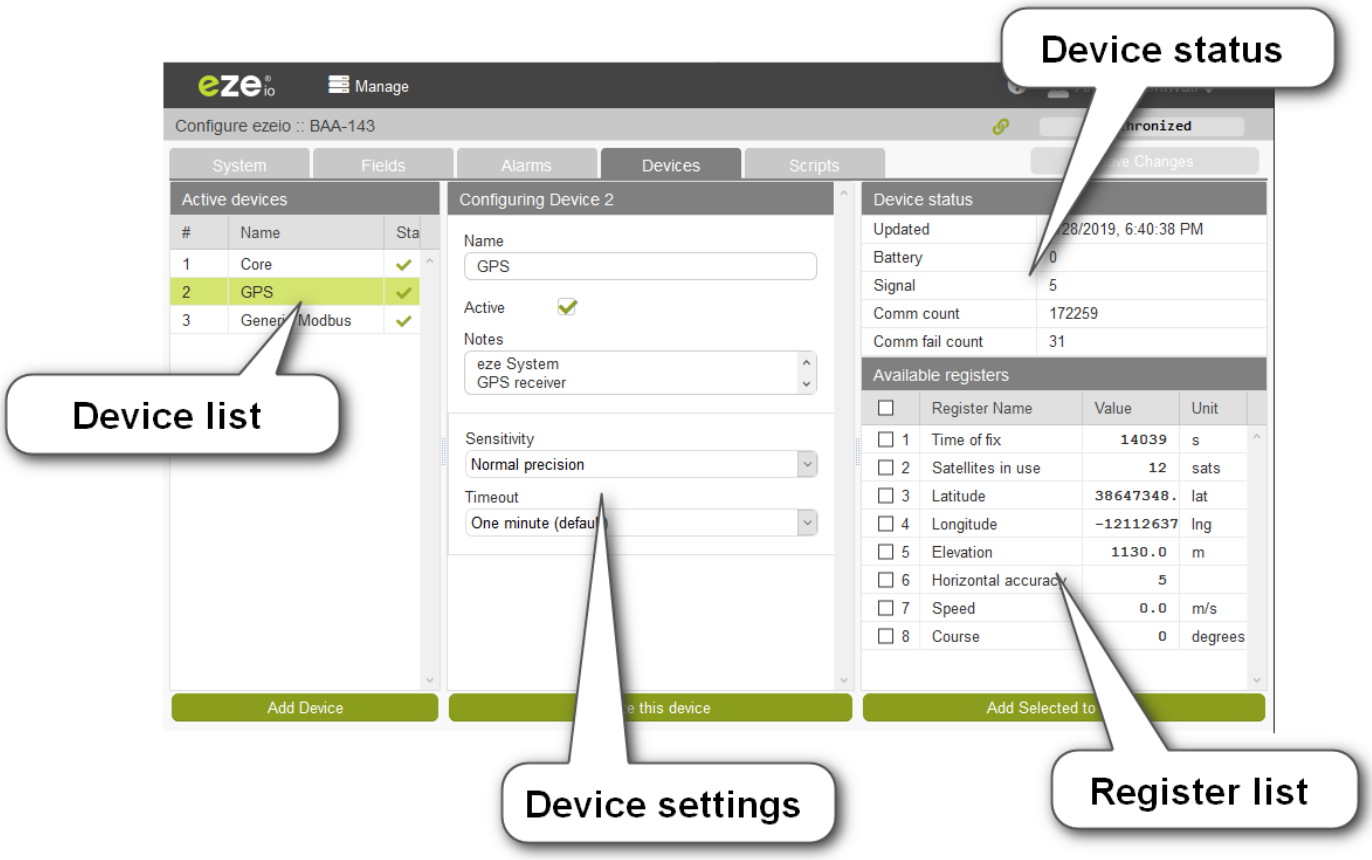
Adding a device

To add a device, please follow these steps:

1. (From the Configure screen) select the “Devices” tab
2. At the bottom of the left panel click the Add Device button (This will bring up a list of available device drivers)
3. Select the appropriate driver and click Add Driver

Most drivers have some settings that needs to be properly set before the driver can be functional. For example, all Modbus RTU devices require a device address to be set.

Make your selections for the new driver and click Save Changes.



Finally, enable your new driver by checking the Active box, and then save your changes again.

The driver will now start working, and you should see values populate in the “Available register” list.

Select the register values that you want to show as fields by checking the corresponding checkboxes, and click Add selected to Fields. This will create new Fields, complete with “Data expression” and some default settings.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/devicelist

Last update: **2024-05-08 19:29**

Fields

Fields are the user configured points that an ezeio logs in onboard memory and sends to the Cloud. Up to 90 can be added per ezeio. These are typically mapped to inputs or outputs, but can be used to create counters, set points, constants, sums, differentials, averages, and more. The possibilities are truly endless. Each Field is evaluated every tenth of a second by the ezeio's Expression engine. The result can be visualize on Dashboards, trigger alarms, drive automation as well as being logged for historical reference. Each Field has a number of required and optional settings detailed below.

Fields Table

This table provides a view similar to the controller status table on the manage level. One of the purposes of the Fields page is to configure this version this version that is accessible by users with out “Edit ezeio privileges. The order the Fields are shown on both tables is manages here. By default the order is descending numerically. Below are descriptions of the table columns and their functions.

Multiple selection check boxes

Multiple Fields can be selected for modification by checking the corresponding box. (once selected) Left clicking the header at the top of this column will reveal several options, including; “set show on System tab” and “delete selected”. For most, clicking the desired option will apply the change. Clicking “Delete selected will bring up a dialog box to confirm.

Field number order

Clicking the header of this column toggles between ascending and descending number order (up & down arrows may be visible). If set to alphabetical listing by Field name, clicking the banner will also return the list to number order. Fields may also be moved up or down the order individually, using the up & down arrow icons running down the left side of the column. Simply, Left click-Hold and drag to the desired location, then release. *Clicking in the name or number banner will return to alpha / numeric order.*

Name

Clicking in the header of the Name column will reorder the Fields in alphabetical order. Clicking in the header a second time will flip the listing to ascending order. When Fields are added from the Devices page, the default name is a pairing of the user defined device name and the driver defined register name. When adding a Field manually the name must also be added manually by the user. Whether generated automatically or entered manually the name can always be edited by a user with adequate privileges'.

Asset Tag

If an Asset Tag has been entered in the config settings of a Field, the tag/s will appear here.

View

This column can display the value of a Field in graphical, text, numeric, or unit interpretations of the

Fields value. See “Display in format in view” for available formats.

Value

This column displays the result of the Fields “Data expression”.

Unit

This column displays the unit entered or selected, in the Fields settings, for the corresponding value.

Configuring Fields

As discussed at the beginning of this page a Field is a user defined data point logged by an ezeio. What is logged, how it is logged, and how it is presented, is controlled by a Fields configuration. Below are descriptions of the available settings and there functions.

Name

The name/text entered here appears on the Fields table and ezeio Status table. It can also be sent in alarm messages by using the “Message template tag” [FN#]. The tag [F#] also includes the Field name, along with the Field's value and unit.

Unit

This user defined text is shown on the Fields table, controller “Status” table, linked Widgets and “Message template tags” [FU#] & [F#].

Asset Tag

Enter nicknames or codes here, separated by space. These “Tags” are an alternate means of referencing one or more Fields and utilized in message templates, some dashboard Widgets and anywhere expressions are used (Fields, alarms, alarm & restore actions, timers).

Tags names should always start with a letter, and only include letters, numbers, underscore (_) and hyphen (-). Spaces, international letters or other special characters are not supported in tags.

Asset tags can be user defined and/or generated by a device driver. They can be unique such as “HVACload” or generic such as “load”. Asset tags should be kept short, starting with a letter, and containing no special characters or spaces. The asset tags are entered on the Field's configuration, for example as: HVACload and referenced in an expression or message template as: <HVACload>

It is common for hundreds controllers to be used for the same purpose across an account. The configuration could be identical or utilize different devices for the same purpose. For example: a fleet of engine driven pumps. The size, design and manufacture may vary, but they all use fuel, oil, coolant, and pump fluid. Assigning an asset tag for each of the common data points allows them to be easily reference regardless of their Field number.

Within an expression, the asset tag is synonymous to the Field number expression. For example, if

HVACload is the asset tag in Field #3, <HVACload> would replace f(3) in an expression. Both represent the Field's value.

When reference by an Aggregating dashboard Widget, all the Fields (from controllers in the dashboards group and below it on the group tree) a with the same generic asset tag will be included or evaluated in the aggregation.

Within a message template the asset tag replaces the Field number in these “Message template tags” [FV#], [FU#], [FN#] and [F#]. Example: [F<HVACload>]

Decimals selector

Clicking plus or minus will add or remove decimal points to the value of the Field. If the result of the Field's data expression has more resolution than can be displayed with the selected number of decimal points the number will be rounded.

Minimum & Maximum

The range established here is used by Dashboard Widgets and the “linear gauge” display format in the view column of the Fields table.

Display format in view This feature converts numeric values into text, graphics, or commonly used formats such as feet and inches or hours and minutes. The formatted value is displayed in the “View” column of the “Field” and “Status” tables. Also available on Dashboards with the “Formatted Field Value” widget.

Name	Description	Example
Blank	No graphic or text shown	
Number	same value as shown in “Value” column	Tip: required setting for manually inputting value
Feet as Feet and inches	Converts decimal values to inches	6.75 = 6 feet, 9 inches
Pounds as Pounds and ounces	Converts decimal values to ounces	6.75 = 6 lbs., 12 oz.
Seconds as Minute and seconds	Minutes as a decimal are converted to Minutes and Seconds	72.5 = 72min 30s
Hours as Hours and minutes	Hours as a decimal are converted to Hours and Minutes	72.5 = 72h 30min
Seconds as Days, Hours, Minutes and Seconds	Convert seconds to	6958745 = 80d 12h
Degrees as direction	#s from 1 to 360 are shown as compass directions	235 = Southeast
String		
EPOCH as local time	Converts EPOCH (*RTC()*) seconds to time stamp	1609419599 = 12/31/2020 23:59 PM
Switch	on / off switch, click to toggle,	On = 100%
Linear gauge	displays percentage of min - max range as green bar graph	

Slider	Click-hold-drag to adjust	Range of slider is 0-100 (When driving an anlaog output 0 = 0V, 100 = 10V

Data expression The value of a field is determined by the Field's “Data expression”. The expression works just like a calculated cell in a spreadsheet. The simplest expression is a constant, like `1234` or `89.02`, but more commonly the field expression will be used to map the Field to a data source by referencing a register, for example `r(3,4)`. If there is a Device driver #3, with 4 or more registers, the value of its 4th register will populate the Field.

All field expressions are re-evaluated at a constant rate of ten (10) times per second.

Expressions can include math, like `r(3,4)+45`. *As you would expect, the field value will now be 45 greater than the register value.*

You may also use values from multiple registers: `r(3,4)-r(2,18)` *This example a creates a differential.*

Fields can also reference other fields: `f(19)`, or `abs(f(2)-f(3))` *The second example will show a differential as an absolute value.*

A more advanced example calculates temperature (in °C) from a thermistor (r(1,1) in Ohms), using the Steinhart-Hart equation:
`1/(0.00112530885+0.00023471186*ln(r(1,1))+0.00000008566*ln(r(1,1)))-273.15`

Another useful example is to use a field as a run-time counter. Because the field value is re-calculated 10 times per second, we can increment the field value based on some other condition. For example, if we want to monitor the time a switch on field #3 is ON (>50), we can use the following expression on a new field: `f(THIS) + (f(3)>50)*0.1`.

The logic condition (f(3)>50) will evaluate to either 0 (false) or 1 (true). Because the field updates 10 times per second, we multiply this with 0.1 before adding it to the current value of our counter field.

Leaving a Field's data expression blank and checking the box to make it writable allows the value to be set by several different means.

- Selecting **Switch**, **Slider**, or **Number** from the “Display format in view” allows the value to be set from the “View” column of the Field or Status tables
- Linking the “On/Off Switch” or ”Field Value with Editor” widgets to the Field allows it to be set from a Dashboard
- Using the SetField command allows the Field to be set through the User script
- The command `field[FIELDNO]=value`, allows a Field's value to be set via API

Please refer to the expression reference for the syntax and all the available functions.

Reverse expression

This setting is optional and left blank for most applications. Reverse expressions only effect writable “Fields” or registers. For more information see reverse expressions

Log interval

The default log interval of 10 minutes is included with basic service. Two upgrades to the logging service are available, “Logging down to 1 minute” (includes 1, 2, & 5 minute intervals) and “logging down to 5 seconds” (includes 5 & 15 second intervals).

Log method

Several different logging methods are available.

Method	Description
Snapshot	Captures the momentary value at the log interval
Mean (average)	Calculates the average of all the samples taken over the interval
Minimum	Records the lowest sample value over the interval
Maximum	Records the highest sample value over the interval
Extreme	Calculates the average and records the sample furthest from the average
Trend	Calculates a slope value for the samples taken within the processing interval (see “Processing speed”)

Process speed for Log Method

When the log method is Mean, Min, Max, Extreme or Trend, the momentary field value is calculated over the time given by this setting.

Example:

Assume Log interval is set to 1 minute, Log method is Maximum and Process speed is set to 10 seconds. The momentary (real-time) field value will be the maximum value detected over a rolling 10 second window. The logged value is computed from the momentary field value, as the max over the log interval (1 minute).

This setting has no effect if the log method is set to Snapshot.

Is active (check box)

This box is checked by default. Un-checking it will disable logging for the Field

Median filter (1s) (check box)

Checking this box will filter out any high or low values that do not persist for more that 1 second. This effectively introduces a 1 second delay. This delay effects alarms and other logic.

Field values are updated every 1/10th of a second. The filter buffer holds 21 of these values/samples. The samples are sorted from highest to lowest and the middle (median) is selected to update the field. Every 1/10th of a second the oldest sample is removed, a new sample is added and the median is selected.

Writable (check box) Checking this box allows users to manually change the value of the Field. Once checked and saved, the cell in the Field's view column will change from light grey to white. Select a setting on the “Display format in view” dropdown menu that provides a means of manipulating the value, such as; Number, Switch, or Slider. With number selected, clicking in the cell will produce a pop-up box for the user to enter and submit a new value. This feature functions differently whether the Field is mapped to a connected device or not. When mapped to writable register on a peripheral device, the new value will be written to the register. If no mapping (no “Data expression” entered) the Field value can be referenced by Alarms, Timers, Script or application drivers.

Peripheral device example: Write a new RPM setpoint to an engine controller

No “Data expression” example: Manually change the threshold in an “Alarm condition setting”.

Continuous write (check box) The value of a Field is calculated every 1/10th of a second. If the Field is mapped to a writable register, “Continuous write” will attempt to write the value shown to every time the Field is calculated.

Show on system tab (check box)

For some applications may be desirable to show a sub-set of the Fields on the Manage level. Checking this box will make the Field appear on the controller's status table, found under the Manage screen's Systems tab.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/fields

Last update: **2024-05-20 15:37**

Scheduler

This tool can be used in many ways for many purposes, such as operational hours and irrigation schedules. The “Scheduler's” features allow for a great amount of flexibility. The schedules are passive in nature having no directly connected actions. The result of a schedule is a status of; “inactive” (value = 0) or “active” (value = 1). This status can be used in any expression (such as a Field's Data expression, an Alarm condition) or in the user script.

Features, capabilities, rules

- A max of 30 schedules are allowed
- Each schedule can have multiple start and stop times within a 24 hour cycle
- Each schedule can be active on any number of days of the week and effected by any number of “day types”
- A schedule can only be effected by “day types” in one way. Not on day type, Or on day type, or And on day type

Not on, Or on, And on **logic**

logic option	Description	Example
Not on	Overrides schedule if day type/s present	Follow Tues-Thurs schedule, but not on Holiday day type
Or on	Applies same schedule on day type/s selected	Saturdays or Holiday day type
And on	Active on the day/s of the week and selected day type/s	

Schedule creation & settings

Schedule table

Alarms are listed by number and name on the Schedule table. Click on the desired schedule to view or edit settings.

Use the “Add Schedule” button at the bottom of the table to create a new schedule.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/scheduler

Last update: **2021-09-24 21:57**

Scripts

The features of the ezeio MkII system provide tremendous configurability to create monitoring, alarming, controlling and automation solutions, but for some applications, this may not be enough. To allow users to create custom functionality, the ezeio™ system supports the PAWN script language. PAWN has a C-like syntax and executes completely inside the ezeio. The “Scripts” page allows users to write, compile and commit code to the ezeio. Extensions to the PAWN language and custom functions relating to the ezeio have been added (see Script reference). The custom functions allow access to, and in some cases, control over an ezeio's configuration and hardware. The “Scripts” can be used to do small tasks, like calculations too complex or verbose for a Field's Data expression. Or automation can created using state machine logic and/or PID loop. The custom PAWN functions that link the ezeio's configuration to the “User scripts” allows the allocation of tasks to be shared. Fields, alarms, schedules, timers and system statuses can drive or influence decisions made by the script. This can serve to minimize the amount of code and configuration required. This split in tasks can also be used to manage access. Simple set points and switches can be operated with minimal privilege, while configuration changes are allowed with “Can edit ezeio in group” and finally scripts could be restricted to 1 or 2 individuals.

More information on how PAWN is used for the ezeio MkII can be found in this document under "Script reference". More information on the PAWN language in general can be found on online. A PAWN scripting introduction is available at https://ezesys.com/files-docs , as a PDF download.

Script list (sidebar)

Upon entering the “Scripts” page, you will see a single script name “AUTO_USER” highlighted and checked as active in your side bar (or scripts list).

Add script “+” button - To add a new script, click the “+” plus symbol in the upper left corner of sidebars header.

Selecting a script - Click on a scripts name in the list to see its code in the editor window.

Script editor window

The header of the builder window contains the delete (trash can), restart, Compile, Commit buttons and the name field.

Name - Click in the name field to modify the name and click the “Compile” button, to save the change.

Delete - Clicking the trashcan button will bring up a “Delete script” dialog box. Clicking delete will permanently remove the script highlighted on the list.

Restart - Clicking the button with the counter clockwise arrow symbol will restart the script currently loaded in the editor

Compile - “Compile” button is used to check the check the code for errors and warnings. The results are

shown in the summary in the top of the “Terminal” window. The code will not run or be downloaded to the ezeio.

Commit - When you are ready to run the script on the ezeio, click the “Commit” button. You will receive a green confirmation message

Script terminal window

Compiling or commiting a script will generate details on the size of the code and any errors or warnings will be listed numerically by line. When a script is successfully commited to an ezeio the terminal window will display all debug messages.

The partition between the “Editor” and “Terminal” windows can be raised or lowered to display more code or results

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/script

Last update: **2021-09-24 21:57**

System

The System tab provides meta data, performance metrics, and user defined settings individual ezeio controllers. The window is divided in half with System information on the left and System settings on the right. We recommend naming the each ezeio and setting the time zone, all additional system settings are optional.

System Settings

System name

Time zone

Listed alphabetically by, **continent or ocean**, then **city** (preceded by state in some cases). Also shown is the hour/s ahead or behind UTC.

Note

This text is displayed in the Controller table on the on the Manage level. It can also be sent in alarm messages by using the Message template tag [EZENOTE].

System location

The displayed location of an ezeio is determined by one of the four methods detailed below. A GPS receiver can be purchased from eze System, Inc. if high accuracy is required for mobile applications.

Method		**Accuracy**
Cell tower proximity	Auto location default if no GPS receiver connected	0-5 miles
IP location (Ethernet)	Auto location default if no GPS & no cellular connection	Region
GPS sensor	Auto location default if connected	1 to 10 meters
Manual location	Overrides Auto location defaults if box is unchecked	user defined

Condition to enable fast log

The default logging rate of an ezeio MkII is 10 minutes. Service upgrade options allow for data logging rates as fast as 5 seconds. This provides high data resolution, but also create a massive amount of data to sort through. This feature allows fast logging to be selective based on conditions. For example; you may wish to closely monitor water quality readings above a specific flow rate or only during business hours. *This feature utilizes the same Expression engine employed for Fields and alarms.* Simply enter an expression like f(3) > 15 or sch(2) and all Fields with Logging intervals set higher than 10 minutes will begin recording at their selected rate. They will continue fast logging as long as the condition is true.

Low-power hibernation mode

The ezeio MkII is very efficient and under normal operation the average power consumption is less than one Watt for the ezeio itself. For some solar powered installations the quality and quantity of sun light can vary dramatically, making even a 1 Watt draw unsustainable. The hibernation feature reduces the

ezeio's average power consumption to less than 0.1W. Note that the peak power requirement does not change.

Hibernation negates many of the ezeio's capabilities such as automation and alarms.

To enter hibernation, the server must send a command to the ezeio telling it to start a hibernation cycle. The ezeio will wait until the next 10-minute data log had been collected, and then power down. **Note that no processing is taking place while the ezeio is powered down.** Every 10 minutes, the ezeio will power up for a few seconds to collect a new data sample and then automatically power down again. When one hour has passed since the hibernation command was issued, the ezeio will power up normally and connect to the servers and upload the collected data. This completes a hibernation cycle.

To continue hibernating, the server must issue a new hibernation command to the ezeio. This server-driven strategy is design to avoid any system failure or misconfiguration that would cause the ezeio to become unavailable by continuously hibernating. Note that the ezeio must be able to reliably communicate for hibernation to work. If the ezeio can't connect at least once every hour, it will exit hibernation and continue to run normally.

The following hibernation modes are available

Normal Mode	No hibernation commands are sent. If already in a hibernation cycle, the ezeio should connect normally within 1 hour.
1 Hour Hibernation	The servers will issue a hibernation command automatically each time the ezeio checks in.
Controller Request	The ezeio can issue a request using special drivers or the script command requestHibernate. If set to Controller Request, the servers will grant the request, and issue the hibernation command.

Important notes regarding hibernation

- Local logic (alarms, scripts etc) are NOT running during hibernation.
- Communication with the ezeio during hibernation is not possible.
- When cancelling hibernation, it may take up to 1h for the ezeio to reconnect.
- Power will flow from the barrel jack to the 'batt' and +V terminals during hibernation.
- +5V and any outputs will be powered off during hibernation.

Hardware Settings

Pulse threshold

The default auto setting will work to read most pulse outputs. This setting finds the average voltage of the pulses emitted by the connected meter and that average is used as the threshold. You should not change this setting unless you know the voltage range of the pulses. If you have trouble registering a pulse on the ezeio, review the documentation provided by the meter manufacture to confirm the wiring and whether an excitation voltage is required.

Pulse debounce

Dry contact pulse mechanisms can bounce when making contact generating multiple pulses. To eliminate these unwanted/erroneous pulses we apply a filter that eliminates additional pulses occurring within a specified window of time. The default filter is 5 milliseconds. Check your meters documentation to see if the pulse rate and expected flow require the use of a different debounce filter.

Modbus TCP settings

Server & Client ports Default setting is 502 on both server and client. This is the standard port number established by Modbus.org. To disable the Modbus/TCP server, set the server port to 0 (zero).

Ethernet settings

IP The local IP of this ezeio device. Leave blank to use DHCP automatic assignment.

DNS Only significant if the IP is set. This should be the IP address of a valid DNS server.

This can be left blank if the ezeio should only use the Ethernet port for LAN communication (Modbus/TCP) and not for server communication.

Gateway Only significant if the IP is set. This should be the IP address of the local gateway device.

This can be left blank if the ezeio should only use the Ethernet port for LAN communication (Modbus/TCP) and not for server communication.

Network mask Only significant if the IP is set. IPv4 network mask for the local network.

For Modbus/TCP, only /24 or smaller networks are supported. The last octet is used in the driver section to address the Modbus/TCP servers.

Unregister ezeio

The Unregister ezeio button removes an ezeio from the account it is registered to. It can then be added to another account. Since an ezeio can be moved between groups within an account, by simply dragging and dropping, this feature is only needed when the new location desired is outside of the current account. Deleting the ezeio has no effect on its configuration or services. Links to dashboard Widgets and reports in the group will be broken.

Once deleted, you will need the serial number and registration code to add the ezeio to another account. These are located on the left side of the ezeio and also in the Identity section of the System information.

System information

This section occupies the left side of the System window. It is mainly informational, but there are some tools available to users with **Edit ezeio** privileges.

Identity

Serial - The serial of an ezeio is factory set and cannot be changed.

Name - The ezeio's name is user defined and can be changed by any user with **Edit ezeio** privileges'.

Registration code This code is factory set and cannot be changed. The reg code is used only when registering for a new account or adding an ezeio to an existing group/account.

Most resent comm - The date and time (local web browser time) of the last communication with the serve is displayed. The time elapse since this contact is also displayed for an “at a glance” reference of comm status. The elapse time is displayed as the most relevant unit/s of time. Starting with seconds (s) and progressing to months and days (mo, d).

Uptime - This is the elapse time since last reset/reboot.

Reboot - Clicking “Reboot” will bring up a dialog box to confirm your intention and explain the result. The result is similar to power cycling the ezeio. It will immediately reboot. After rebooting the ezeio will begin the process of reconnecting, through the onboard cellular modem or through the local network via Ethernet port.

Supply voltage This displays 3 voltage readings. First is the DC in (barrel jack at the top right of the ezeio). Next is the Batt connection (far right on the green screw terminals). Last is the 5 volt supply (center of the green screw terminals). If the barrel jack is not used, disregard the first reading.

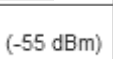
Firmware version - The firmware version represented by a date code ending with a 2 digit code. The format is YYMMDDxx.

Update - Clicking on the Update link, brings up a dialog box for selecting a firmware version. Selecting a firmware version to see the Release notes. The release notes will detail the added features included in this version. If you wish to proceed with the update, click the Update firmware button, otherwise, click Cancel. Updating firmware is optional in most cases and eze System does not force mass firmware updates into customers deployed ezeios. You may be instructed by eze System support or your reseller to update firmware to take advantage of new features.

Cellular information

All data shown in this section is informational.

Signal (RSSI) - Icons here indicate the connection type and signal strength. Two connected links with a letter R, indicates for connected and roaming (normal state for an ezeio). An X icon indicateThe familiar increasing, green 5 bar graph indicates optimal signal strength.

	Icon	Indication	Discription
Link		Connected & Roaming	This is the normal connection and service type for an ezeio
Link		Connected	Connection active in local service area
Link		Searching	The ezeio is searching for cellular service
Link		No connection	Failed to connect to cell tower
Received signal		No connection	No network found or signal too weak
Received signal		Weak signal	Connected, but signal strength is marginal
Received signal		Strong signal	Connected, excellent signal strength
Signal level		Strength of received signal	Lower negative number is stronger signal. Range: -100 dBm = 0%, -30 dBm = 100%

SIM ICCID - This 19 digit number is the SIM cards serial number.

Hardware IMEI - (International Mobile Equipment Identity) The Model information following the IMEI identifies the modem type and version.

Subscriber IMSI - (International Mobile Subscriber Identity) The Network number following the IMSI, is the combined MCC (Mobile Country Code) & MNC (Mobile Network Code) which identify the country and carrier. Depending on the country, there may be agreements with multiple carriers. This means the MNC may change whether the unit is stationary or mobile.

PIN status -

IP address -

Data used -

Ethernet information

IP address -

Data used -

Logic status

Input setting -

Reset logic - Clicking “Reset logic” will bring up a dialog box to confirm your intention and explain the result. Registers, Fields, counters, will be cleared. The restarting sequence allows data to populate registers and fields before logic and automation, dependent on those values, restart. Alarms that were in alarm state and who's conditions are still true will retrigger.

Button status - The button on the top of the ezeio can be pressed during start up for different effects.

Script status

User script -

Modbus -

SDI-12 -

CAN -

Core -

System log

The system log tracks changes in connectivity between the ezeio and the eze.io servers. Time stamps are shown in UTC. Below is a list of system log details and their explanations.

Type	Details	Explanation
Connection	CONNECT ETH	Connection established via Ethernet
Connection	CONNECT CELL	Connection established via cellular network
Connection	DISCONNECT ETH	The Ethernet connection was terminated.
Connection	DISCONNECT CELL [reason]	The cellular connection was terminated.
	TMAIN - Modem stopped responding	
	TLINK - No heartbeat message from servers received in 10 minutes	
	TCONN - No acknowledgement received from the servers	
	ETH - The Ethernet connection was activated	
	ATT - The cell modem was disconnected from the cell network	
	SOCK - The TCP socket connection was closed from the remote side	
	CELLOFF - The cell modem was powered off	
Connection	LINK-LOSS [reason]	The communication was lost / timed out
	VERSION - The firmware ID is invalid	
	SERIAL - The serial number is not recognized	
	FINGERPRINT - The security certificate is invalid	
	CLEAN - The link changed (Eth→Cell or Cell→Eth) and the old link status was erased	
	ERROR - Registration could not complete	
	BADDATA - Received data was corrupted/incomplete	
	BADKEY - A step was skipped during registration	

	MSGLEN - Received data had an invalid size	
	MONITOR - The server could not allocate necessary resources	
	TCPEOF - More data was expected, but did not arrive	
	TIMEOUT - No new data received in 10 minutes	
	SHUTDOWN - The server was shut down/restarted	
	USER - The user sent a disconnect request	
Boot	POR	Power-on reset
Boot	EXT,POR Down:1m 21s	External loss of power followed by elapse time
Boot	EXT,POR [SLEEP] Down:22s	followed by elapse time
Boot	SW	Software caused a reset
Boot	SW Down:1s	Rebooted, followed by elapse time

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/system

Last update: **2025-05-07 22:07**

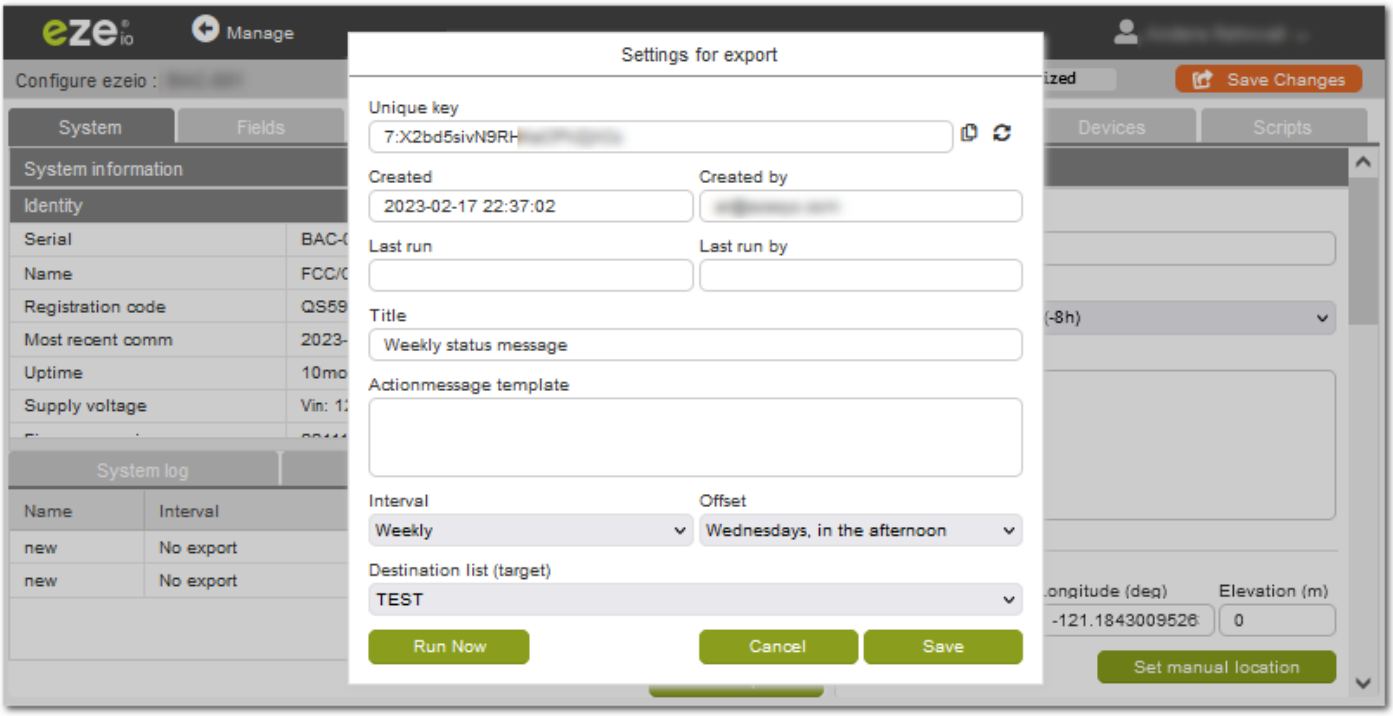
Export

The export feature will periodically send a message.

Export messages can be sent hourly, daily, weekly or monthly.

Export messages use the same Destinations and Destination lists as alarms do, so before setting up automatic exports, please create at least one destination, and add it to an active destination list.

Export messages are triggered from the cloud servers, but will not be sent if the controller related to the export has not been online for a full interval cycle. They will resume if the controller comes back online.



Setting up a new periodic export message

Navigate to Configure→System and select the Export tab.

At the bottom of the screen, click the `New export` button. A new export is immediately created and added to the list, and a dialog shows the settings for the export.

Export settings

Automatic metadata

Each export setting is automatically assigned a unique key identifier. This identifier can be used in the sent messages to identify what the source of the message was. There is no requirement to use the key. It's only there for advanced users that need the highest level of security. If the key has been compromised, click the recycle icon to generate a new key.

When the export setting is created, the “Created” and “Created By” fields are filled in with the current UTC time as well as the current user. This data can not be edited.

The Last Run box will be populated with the most recent UTC time when the export was triggered.

If the export was triggered manually (see below for Run Now), the Last run by box will show what user triggered the export. If the export was automatically triggered, this box will show “AUTO”.

User settings

The `Title` is a user-assigned name of the export. It is recommended to give it a reasonable name for future reference.

The `Actionmessage` template allows for custom messages to be inserted in the message sent. This field supports all the message template tags as described under Message Tags.

The `Interval` can be selected as one of hourly, daily, weekly or monthly. If set to No export, the export will not be automatically triggered.

The `Offset` is relevant for the daily, weekly and monthly interval settings, and defines when during the day the export will be triggered.

The export will be sent to a destination list. Only destination lists accessible under the current account are listed. If no destination list is selected, no export will be triggered.

Run now button

The Run now button will simply trigger the export once. This is useful for testing to make sure the message can be successfully sent, and has the relevant information.

After saving the export settings, the automatic scheduler will automatically trigger the export at the given interval/offset.

From:
https://doc.eze.io/ - **ezeio documentation**

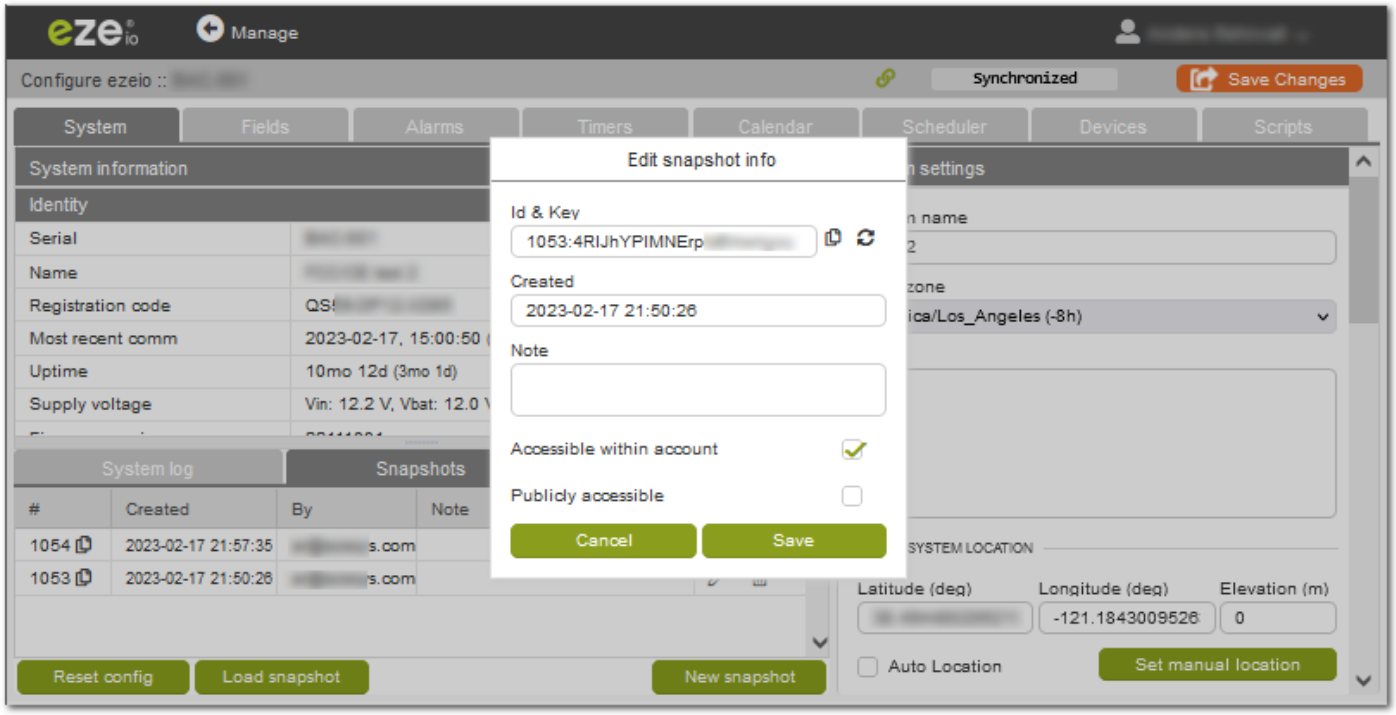
Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/system/export

Last update: **2023-02-18 00:15**

Snapshots

Snapshots are a convenient way to save a backup of a configuration, or to copy a configuration from one controller to another.

A snapshot stores all configuration for a single controller at the moment the snapshot was created.



When the snapshot is created, it is automatically given a unique ID and key, looking something like "`753:4RlJhYPlMNErptc6mwnyvu`". Using this combination at a later time, this snapshot can be loaded on the same, or a different controller.

Create new snapshot

To create a new snapshot, navigate to the Snapshot tab under Configuration→System, and click the `New Snapshot` button at the bottom.

A pop-up will confirm that a new snapshot is being created. After accepting this, the snapshot is immediately created and added to the list, and a dialog opens to allow adjusting the settings.

Snapshot settings

Each snapshot is automatically assigned a ID and key. The combination can be easily copied to your local clipboard using the copy-icon, or by highlighting the text and pressing CTRL-C.

The Note-field allows for a short description to be added to the snapshot. This is optional, but obviously recommended to simplify identifying the snapshot later.

Two checkboxes determine where the snapshot can be used:

Accessibility within account	This much be enabled in order for the snapshot to be accessible.
Publicly accessible	If enabled, the snapshot can be accessed by anyone that has the ID/Key. If disabled, the snapshot is only available within the same account where it was created.

List of snapshots

The list shows all the snapshots that has been created for this controller.

Up to 20 snapshots can be saved for each controller.

On each row, there are three icons:

Copy	Copies the snapshot ID/Key into the local clipboard
Edit	Opens the settings dialog for this snapshot
Trash	Deletes the snapshot from the system (after confirmation)

Loading a snapshot

To load a snapshot, simply click the `Load Snapshot` button.

A dialog will request the ID/Key for the snapshot to be loaded. We recommend using the clipboard paste function (CTRL-V) to enter this information, as it is easy to make a mistake trying to manually enter the code.

Loading a snapshot completely overwrites any previous configuration on the controller where it is applied.

If the code entered is valid, and the snapshot is accessible in the current account context, the system will show the details of the snapshot, and give a final option to confirm that it will load.

The loading is very quick, and the controller will automatically initiate a synchronization to apply all the new settings.

If the snapshot was created from a controller under a different account, any configuration referencing distribution lists will not be applied. If distribution list references were cleared, these will be listed in the Notes field.

After loading a new snapshot, please check the Notes field for any issues.

Reset config

The `Reset config` button at the bottom of the Snapshot list will erase all configuration from the controller, restoring it to the way it came from the factory.

Data recorded by the controller, as well as service settings will not be affected by this function. Only the configuration is reset.

Note that this is NOT NECESSARY to do prior to loading a snapshot, as loading a snapshot automatically erases all prior configuration.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/system/snapshots

Last update: **2023-02-17 23:05**

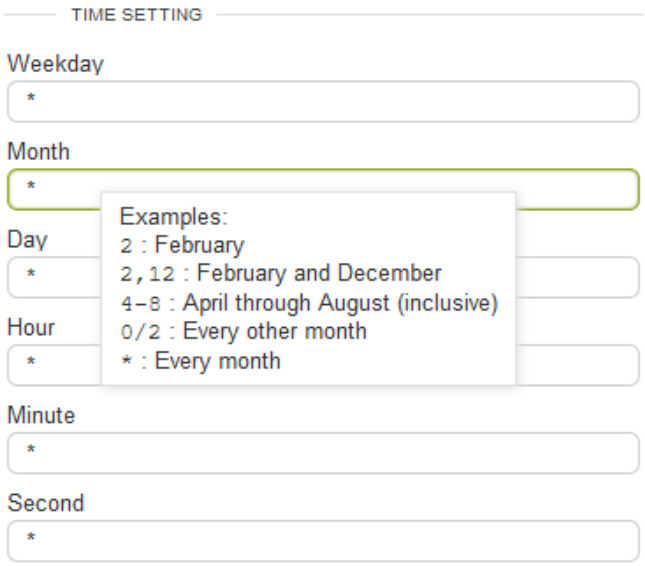
Timers

The timer feature is based on the software utility “cron”. The purpose is to trigger reoccurring actions such as “Set register” or “Force alarm”. Cron expressions are entered into each of the “Time setting” fields to describe the periodic interval for an action. The cron expressions work as filters. At the beginning of every second, the timer evaluates each filter's true or false status. If all of the cron expression are true compared to the current date & time the action will be triggered.

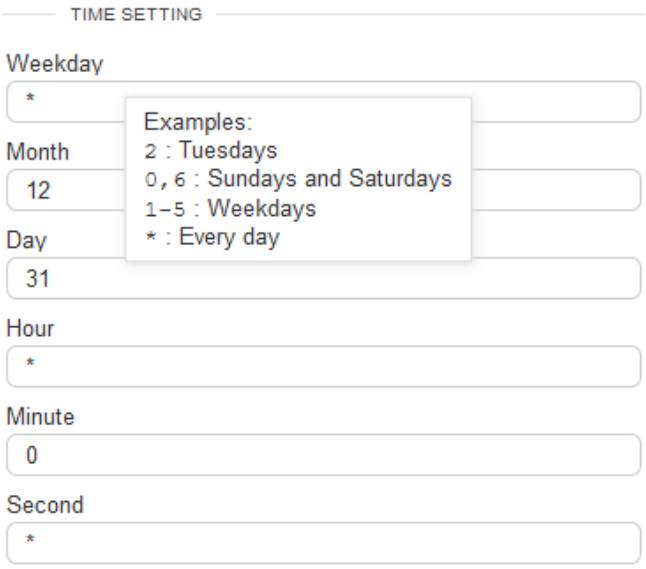
“Time settings” span from seconds to months and also allow for days of the week to be specified. The cron expressions allow tremendous flexibility. Numbers can be entered to be true once during a specific second, minute, day, or month, or special characters such as commas, dashes, slashes and asterisks, allow the timer to skip, span, alternate or always be true. For more information see https://en.wikipedia.org/wiki/Cron

Timers run on the ezeio, therefore reference the time zone set in the configuration of the individual ezeio.

Example 1: If all filters have an asterisk (*), then the timer will be triggered every second



Example 2: If minute is set to 0, day is set 31 and month to 12, the timer will trigger 60 times during the first minute of every hour on December 31st



Unit of time filters

When a new timer is created, the default setting in each time setting field is “0”. *This timer configuration*

will never trigger as there is no month zero. Hover over each field for tips on scheduling multiple triggers per unit of time. More information is available in the next section “special characters”.

The basic format of each unit of time is described below.

Day of the week - Enter the day/s of the week you wish to be true (0 = Sunday) or enter an asterisk (*) to make true every day of the week.

Month (1 to 12) - Enter the month/s of the year you wish to be true (1 = January) or enter an asterisk (*) to make true every month.

Day (of the month) - Enter the day/s of the month you wish to be true (1 through 31) or enter an asterisk (*) to make true every day of the month.

Hour (of the day) - Enter the hour/s of the day you wish to be true (0 to 23) or enter an asterisk (*) to make true every hour of the day/s.

Minute (of the hour) - Enter the minute/s of the hour/s you wish to be true (0 to 59) or enter an asterisk (*) to make true every minute of the hour/s.

Second (of the minute) - Enter the second/s of a minute/s you wish to be true (0 to 59) or enter an asterisk (*) to make true every second of the minute/s.

Special characters

As mentioned above special characters can be used to describe multiple increments or spans within a unit of time. Examples are shown in the table below.

Character	Function / Meaning	Example
Asterisk (*)	* means true	Minute * = true every minute of within an hour
Comma (,)	Separate multiple units of time	5,37,49 True if = 5 or 37 or 49
Dash (-)	Spans from the beginning value through the ending value	7-53 True from 7 through 53
Forward slash (/)	Beginning at x and then every y	2/3 = 2, 5, 8, 11, 14, 17 or Feb, May, Aug, Nov

Timer Actions

These actions allow the direct manipulation of “Fields”, registers, alarms, and user script, based on the reoccurring trigger and settings described below.

Action	Description	Index	Parameter	Value
Set Field	Replace Field's value with value specified in action	Field #	N/A	New value
Adjust Field	Reduce or increase Field's value by amount specified in action	Field #	N/A	Adjustment value

Set Register	Replace registers' value with value specified in action	Device #	Register #	New value
Adjust Register	Reduce or increase registers' value by amount specified in action	Device #	Register #	Adjustment value
Force Alarm	Immediately triggers alarm actions associated with an alarm	Alarm #	N/A	N/A
Force Restore	Restores alarm and triggers restore actions associated with an alarm	Alarm #	N/A	N/A
Script Action				

Example 1: Reset counter daily with “Set Field”.

- Weekday = *
- Month = *
- Day = *
- Hour = 23
- Minute = 59
- Second = 59
- Command = Set Field
- Index (Field #) = 6
- Parameter = (Not applicable)
- Value = 0

Example 2: Send a message with data and/or statuses every day at 3:25pm using “Force alarm”.

- Weekday = *
- Month = *
- Day = *
- Hour = 15
- Minute = 25
- Second = 0
- Command = Force Alarm
- Index (alarm #) = 3
- Parameter = (Not applicable)
- Value = (Not applicable)

From:
https://doc.eze.io/ - **ezeio documentation**

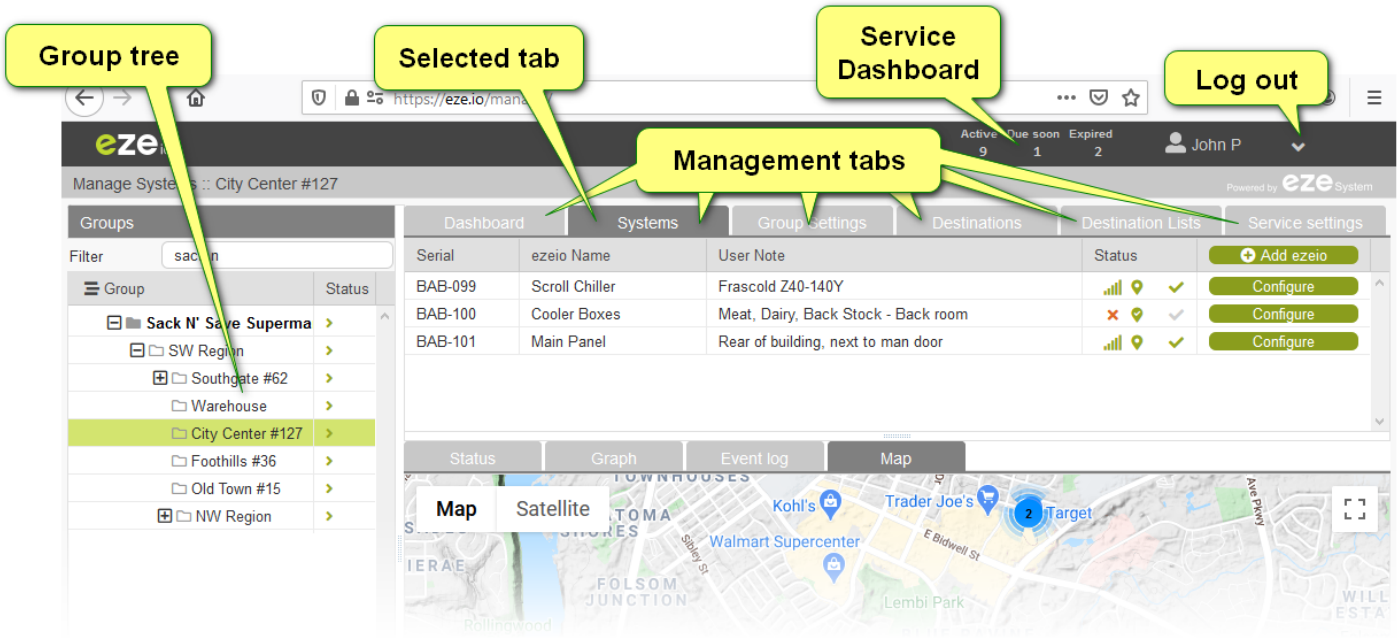
Permanent link:
https://doc.eze.io/ezeio2/userinterface/configuration/timers

Last update: **2021-09-24 22:03**

Manage Screen

The Manage screen is where you will land when logging into the system for the first time. This is the environment where you can view and make changes in an account or group context (*Changes to individual ezeio controllers are made through the Configure screen*). The left sidebar contains your Group tree. Along the top of the main panel you will find a row of tabs. The tabs visible to you are based on the privileges granted to you by the account/group administrator/s. The full array of tabs is shown in the image below.

In this image the Systems tab is selected, displaying a list of controllers and a map with their locations plotted. To navigate within the Manage screen, simply click on a different tab or select another group from the Group tree. To navigate to the Configuration screen, click the green configure button adjacent to the desired controller (shown in the image bellow).



From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/start

Last update: **2021-09-27 22:55**

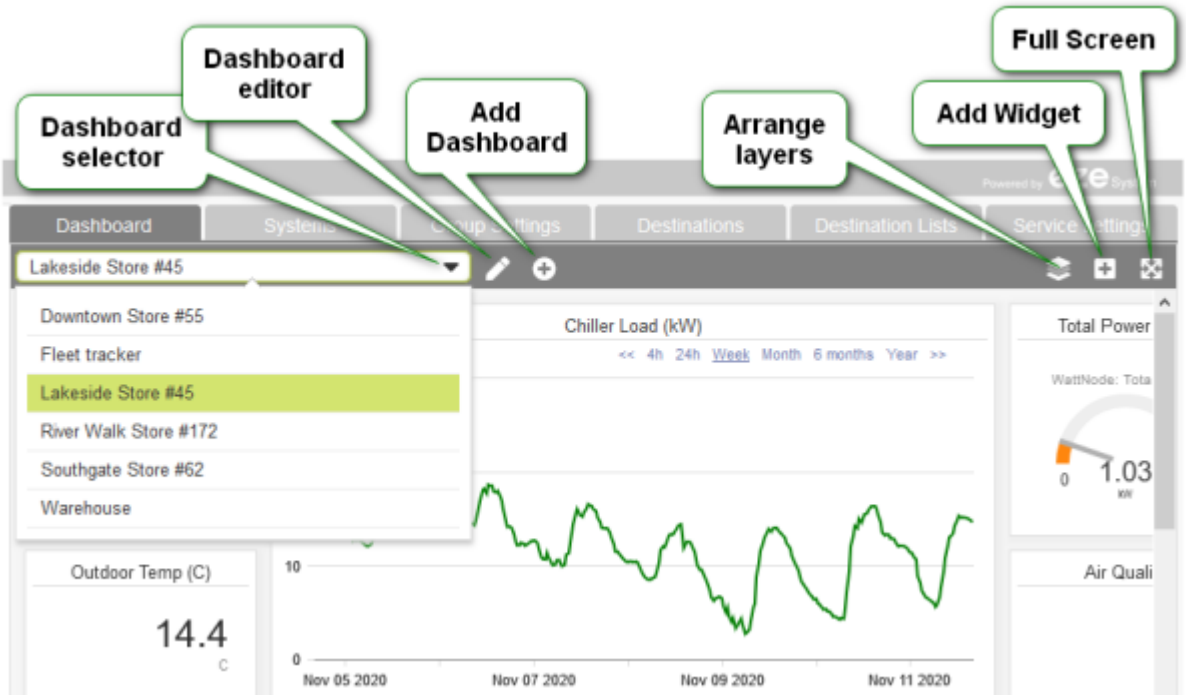
Dashboards

Located on the “Manage” level of the eze.io user interface, Dashboards are designed as a place to curate collections of data, graphically displayed with “Widgets”. Similar to the design of an automobile dashboard, a Dashboard can offer live data, status & warning indicators, controls, settings and historical data. The size, color, type and placement of various Widgets can be used to create an intuitive interface. Multiple Dashboards can be used to group by location, asset, team, personal preference, or use one for an overview and additional Dashboards to provide a deeper look into specific systems/equipment. See the Widget page for more general information on Widgets and a directory of individual Widget pages.

Dashboards Features

- Create multiple “Dashboards” in each Group or Account
- Display data from multiple devices, Including data from subgroups.
- Control from Dashboard via; Switch, Button and set point (Field value with editor) Widgets
- Restrict user access and/or editing privilege per Dashboard
- Show map of ezeio controllers in Group and Sub Groups
- Show external applications, such as weather, company website or Internet connected video camera
- Expand to show full screen
- Print Dashboard

Dashboards Tools

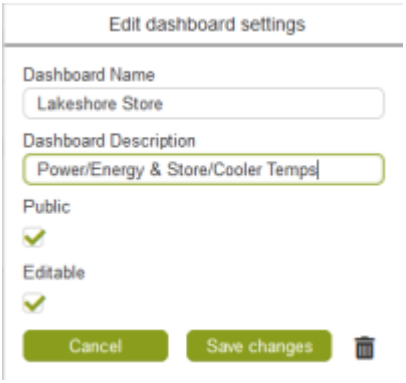


Dashboard Selector

This drop-down menu lists (in alphabetical order) all the Dashboards available to the user.

Dashboard Editor

Clicking the pencil icon will bring up the **Edit Dashboard Settings** dialog box (shown below). The following settings can be edited through this dialog box.



- Edit the currently selected Dashboard's **Name** and/or **Description** using the fields supplied.
- Unchecking the **“Public” check box** restrict viewing access of the Dashboard to the creator.
- Unchecking the **Editable check box** restricts editing of the Dashboard to the creator. These check boxes are checked by default. In this default state, viewing and editing access is based on a users privileges'.
- The **trash can icon** will bring up a **Delete Dashboard?** dialog box. Click the **“Delete”** button to confirm or the **“Cancel”** button to return to Edit Dashboard Settings.

Print

Click the printer icon in the Dashboard's header to bring up the print dialog box. Select the desired printer or PDF creator if you have one installed.

Arrange

Widgets can be stacked one on top if the other. To take full advantage of this feature you may need to arrange the layer order of some widgets. Click on the **“Layer icon”** to see a list of Widgets by layer order. Simply Click-Hold-Drag the Widget to a high or lower position on the list (top of list = top layer).

Add Dashboard

Click the circle with plus sign icon to add a new Dashboard. Edit the new Dashboard's settings or start building the Dashboard by adding widgets (see Add widgets below).

Add Widgets

Click the **Add Widget icon** (square with plus sign) see a list of available Widgets. The standard list includes; line graphs, bar graphs, dial gauges, digital value, switches, buttons, maps, and a note Widget.

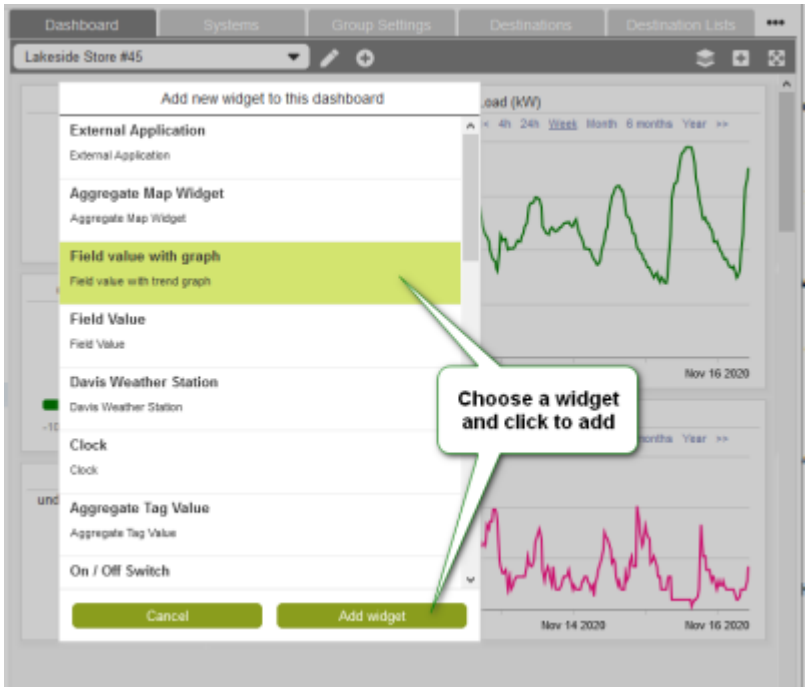
Expand

Click the Expand icon to view the Dashboard tab as full browser width. Click it again to return to the normal view with the all of the *“Manage” tabs and the “Groups” side bar.

Widgets

Widgets are individual applications that run within eze.io's Dashboards. The **“Add Widget”** button described in the previous section, will bring up a list of available Widgets. This list will continue to grow, as new Widgets are developed. Below are a few categories of Widgets.

- **Historical Data** - Line graph, Stacked Bar Graph
- **Instantaneous Data** - Field value, Field value as gauge, Field value table
- **Aggregated Data** - Aggregated Map (shows multiple ezeio locations), Aggregated Tag Value, Circular Gauge (Tag Value & Circular Gauge allow multiple “Field” values to be calculated as sums, average, min or max)
- **Control** - On/Off Switch, Push Button, Field Value with Editor
- **External** - External application (can be pointed to a web address to show a website, video, Internet video camera.
- **System Info** - Clock
- **Text** - Note widget



From:
https://doc.eze.io/ - **ezeio documentation**

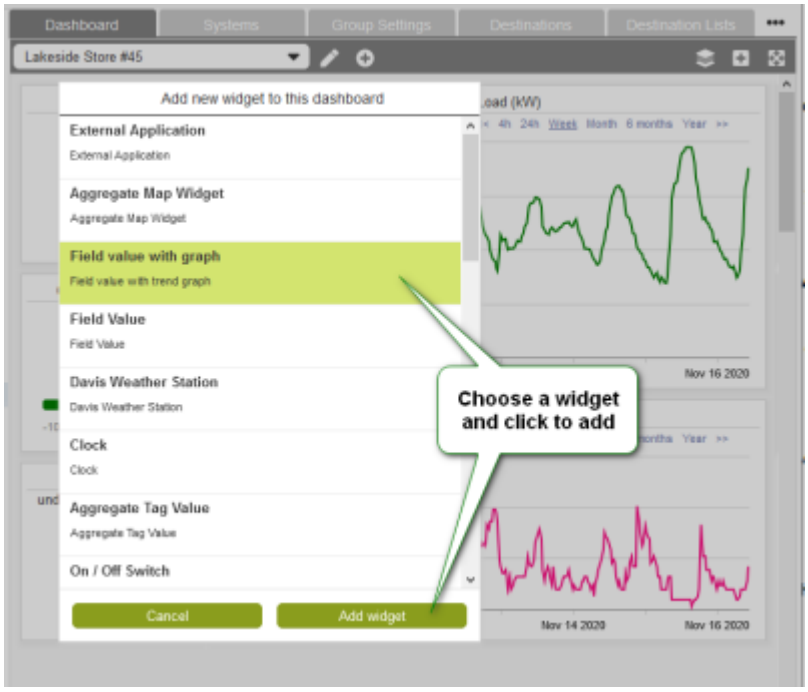
Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards

Last update: **2021-09-24 21:56**

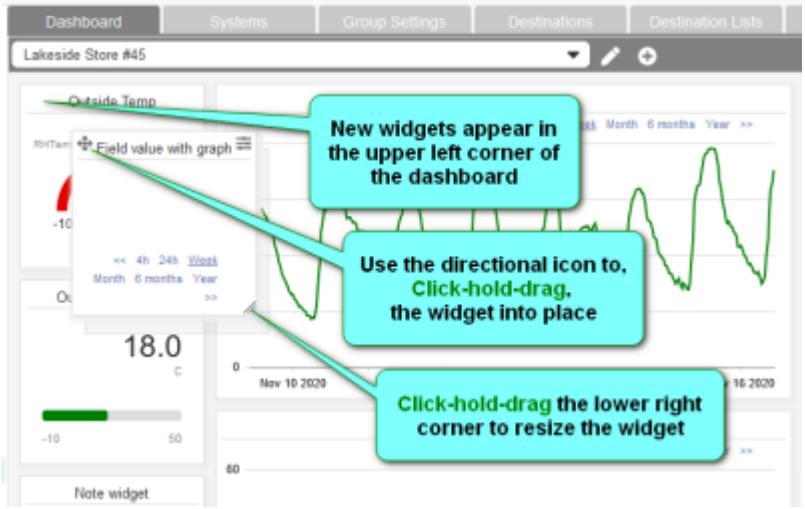
Widgets

Widgets are individual applications that run within eze.io's Dashboards. The **“Add Widget”** button described in the previous section, will bring up a list of available Widgets. This list will continue to grow, as new Widgets are developed. Below are a few categories of Widgets.

- **Historical Data** - Line graph, Stacked Bar Graph
- **Instantaneous Data** - Field value, Field value as gauge, Field value table
- **Aggregated Data** - Aggregated Map (shows multiple ezeio locations), Aggregated Tag Value, Circular Gauge (Tag Value & Circular Gauge allow multiple “Field” values to be calculated as sums, average, min or max)
- **Control** - On/Off Switch, Push Button, Field Value with Editor
- **External** - External application (can be pointed to a web address to show a website, video, Internet video camera.
- **System Info** - Clock
- **Text** - Note widget



Common widget features and functions



Moving widgets

The direction icon in the upper left corner of all widgets is used like a handle to drag them into position. While hovering over the icon, simply click-hold and drag. As you drag, widgets will snap into position in the invisible grid.

Resizing widgets

The height and width of a widget can be resized by dragging the lower right corner. While hovering over the lower right corner, simply click-hold and drag. A shadow indicates the new size of the widget, snapping to the next grid line as you drag.

Arranging / Layering

Widgets can be placed one above the other partial or totally obscuring the lower layer widgets. The default layer order is based on the order the widgets were created. Use the layer tool on the dashboard header to rearrange the layer order.

Settings

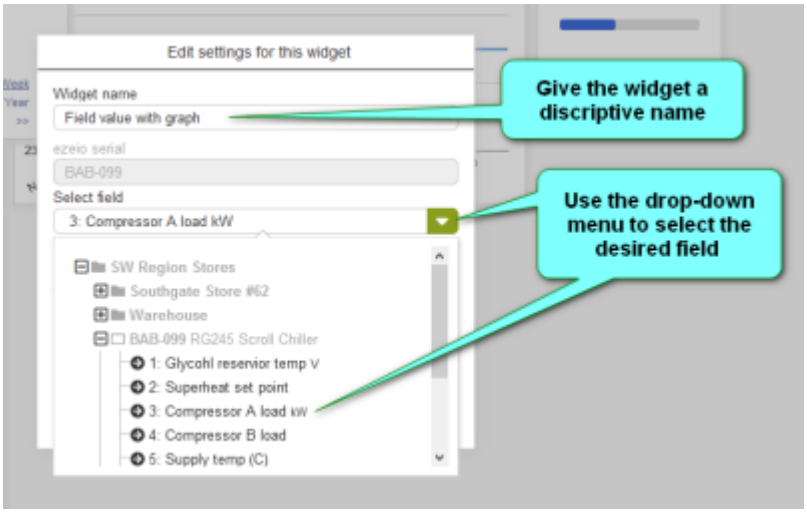
Click on the icon in the upper left corner of the a widget to access it's settings.

Printing

An image of single widget can selected for printing or PDF by clicking the printer icon (upper left) in the “Edit settings” dialog box of all widgets

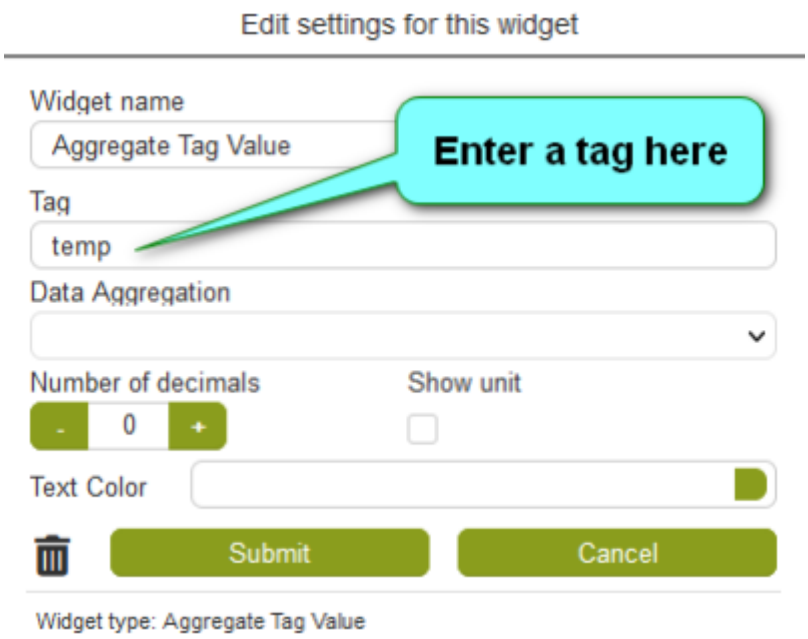
Mapping

When linking a widget to data source/s two methods are used, selecting via a drop-down-menu and or referencing asset tags.



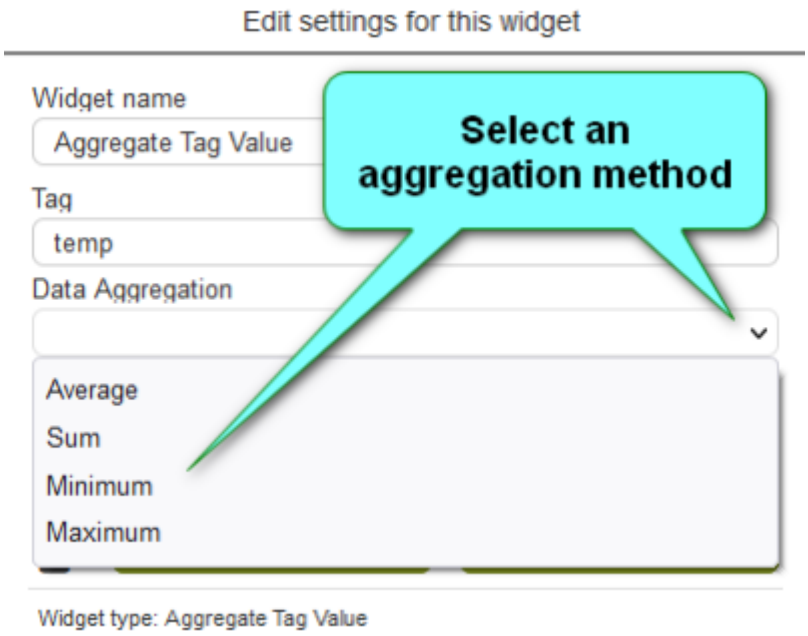
Select via Drop-down-menu

Most widgets allow the data source/s to be selected via a drop-down-menu, as shown in the image above. A tree of Groups and their ezeio controllers can be expanded to reveal individual fields available as data sources for a widget.



Referencing asset tags

This method is used mainly for aggregate value widgets. These widgets use asset tags (assigned to Fields) as means of incorporating multiple data points into a single value.

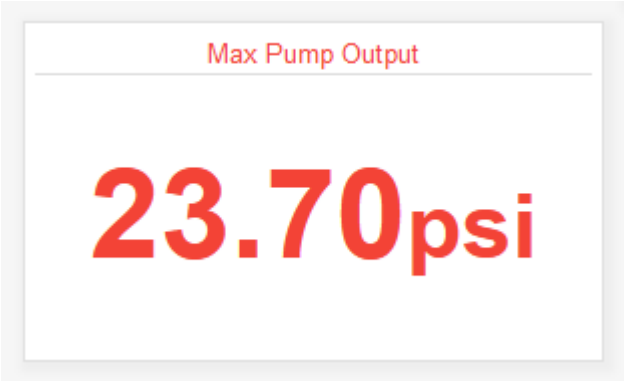


From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets

Last update: **2021-09-24 21:56**

Aggregate Tag Value



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

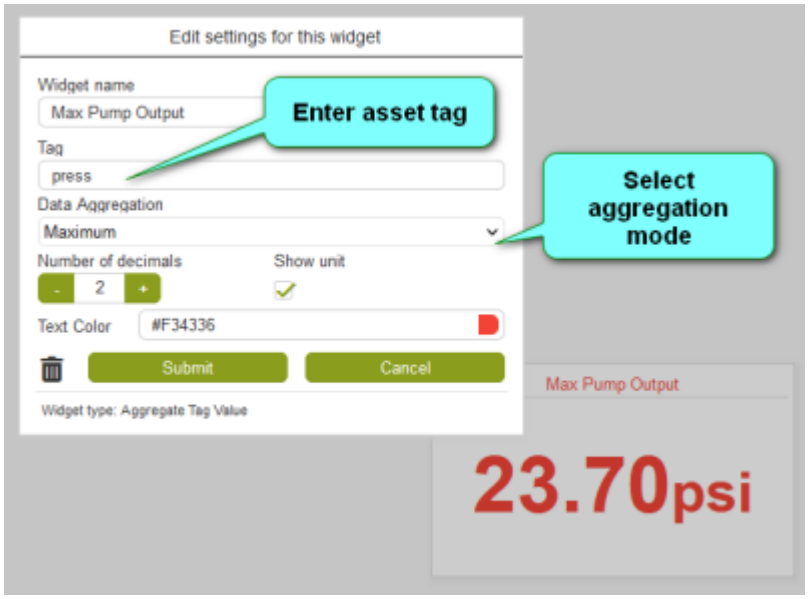
The eze.io Aggregated Tag Value widget processes multiple data points with the same tag, to produce a singe value. Aggregation options include; average, sum, minimum, and maximum. This widget is designed for live/instantaneous values. These values could be rapidly changing such as a power reading (kW) or slow like a tank level.

Minimum height - 2 grid units

Features / Options

- Numeric value shown with unit
- 4 Aggregation setting (Avg., Sum, Min., Max.,

Settings



Widget name

Appears top and center of widget

Tag

Enter the "Asset tag" that was assigned to the Fields to be aggregated. For more information on this linking method see the mapping section on the widgets page.

Data Aggregation

A drop-down-menu allows the user to select one of the four data aggregation options described below

- **Average** - Assign the same tag to two or more Fields and the widget will display the average
- **Sum** - All Fields with the same tag will added together to produce a total
- **Minimum** - The lowest value out of all of the Fields with the tag will be displayed
- **Maximum** - The highest value out of all of the Fields with the tag will be displayed

Text color

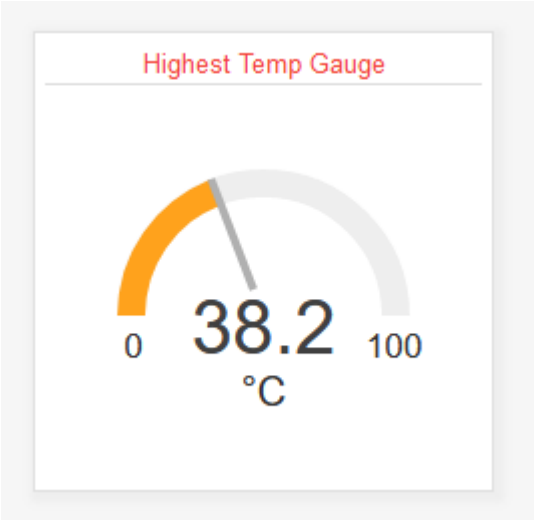
Click on the color swatch to access the color palette and select a color for the range bar

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/aggtagvalue

Last update: **2021-09-24 21:56**

Circular Gauge Widget



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io Circular Gauge widget processes multiple data points with the same tag, to produce a singe value. That value is presented as a dial gauge, similar to the analog gauges on an old car or instrument panel. Aggregation options include; average, sum, minimum, and maximum. This widget is designed for live/instantaneous values. These values could be rapidly changing such as a power reading (kW) or slow like a tank level.

Minimum height - 2 grid units

Features / Options

- Numeric value & range displayed
- Up to 5 range segments (max of 3 colors)
- Field name and engineering unit shown
- 4 Aggregation setting (Avg., Sum, Min., Max.)

Settings

Widget name

Appears top and center of widget

Tag

Enter the "Asset tag" that was assigned to the Fields to be aggregated. For more information on this linking method see the mapping section on the widgets page.

Data Aggregation

A drop-down-menu allows the user to select one of the four data aggregation options described below

- **Average** - Assign the same tag to two or more Fields and the widget will display the average
- **Sum** - All Fields with the same tag will added together to produce a total
- **Minimum** - The lowest value out of all of the Fields with the tag will be displayed
- **Maximum** - The highest value out of all of the Fields with the tag will be displayed

Gauge Colors

The full range of the gauge can be divided into five sections displaying one of three colors. Each range

has a layer hierarchy. “FULL RANGE” is the back most layer, next is the “MIDDLE RANGE” and at the top is the “INNER RANGE”. As the pointer sweeps through the range, the colored area to the left will change to the color assigned to that section of the range. If the ranges overlap, then the top most layer/color is displayed.

Full Range

The full range of this widget is driven by this setting, unlike other widgets where the range is set by the Field's min-max settings. The gauge can range from negative to positive, but it will only sweep from left to right. Enter the lower value on the in the “From” box and the higher value in the “To” box. Then click on the color swatch to access the color palette and select a color to represent one or both of the extremes.

Middle Range

As described above, enter the range and select a color to represent the 'MIDDLE RANGE“ & middle color layer.

Inner Range

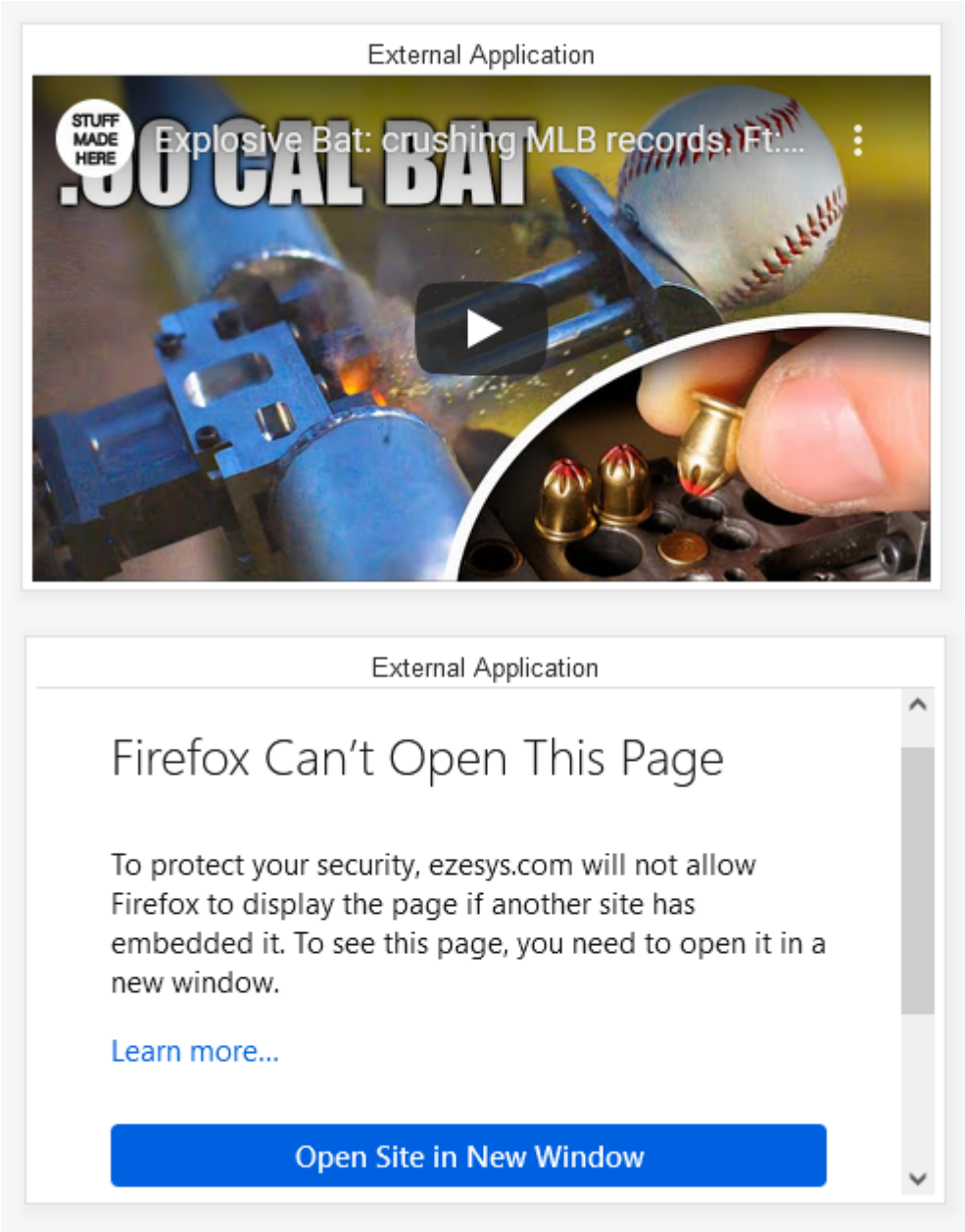
This is the top most layer, so when the value is within this range the color selected will be shown. Enter the range and select a color as described in “Full Range”.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/circulargauge

Last update: **2021-09-30 23:00**

External Application Widget



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io External Application widget allow an embeddable web page to be place inside a Dashboard. If

the webpage does not allow embedding it will not display (see example above).

Some unembeddable webpages have breakout code which forces them to expand to the full browser window. This causes the newly added webpage to take over the browser tab. Breakout code is not a commonly used, but the condition is only recoverable by eze System's technical staff.

Settings

Widget name

Appears top and center of widget

Enter URL

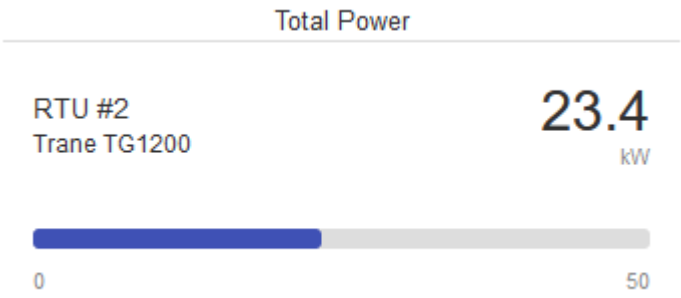
Copy the URL from the source or enter it manually.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/externalapp

Last update: **2021-09-25 00:04**

Field Value



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io Field Value widget displays the current value of a single field as a numeric value. A horizontal bar and the bottom of the widget shows value as the percentage of the Field's range. This widget is designed for live/instantaneous values. These values could be rapidly changing such as a power reading (kW) or slow like a tank level.

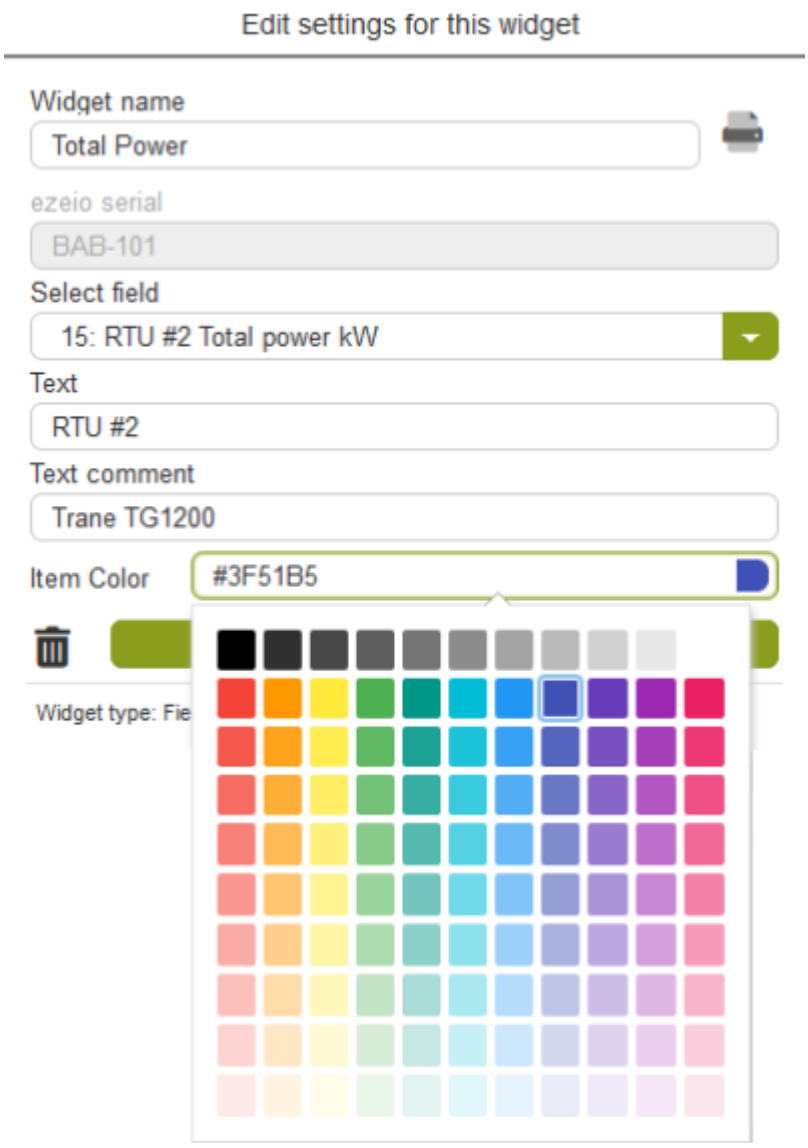
Minimum size (value, name, unit only) - 2 x 2 grid units

Minimum size (full feature) - 3 x 6 grid units

Features / Options

- Numeric value shown with unit
- Percentage of range shown as horizontal level bar
- 3 user defined text fields displayed

Settings



Widget name

Appears top and center of widget

Select field/s

A drop down menu provides the means to link to a specific “Field”, from the desired ezeio. For more information see mapping on the widgets page.

Text

Enter text to be displayed to the left of the Field value (1st line)

Text comment

Enter text to be displayed to the left of the Field value (2nd line)

Item color

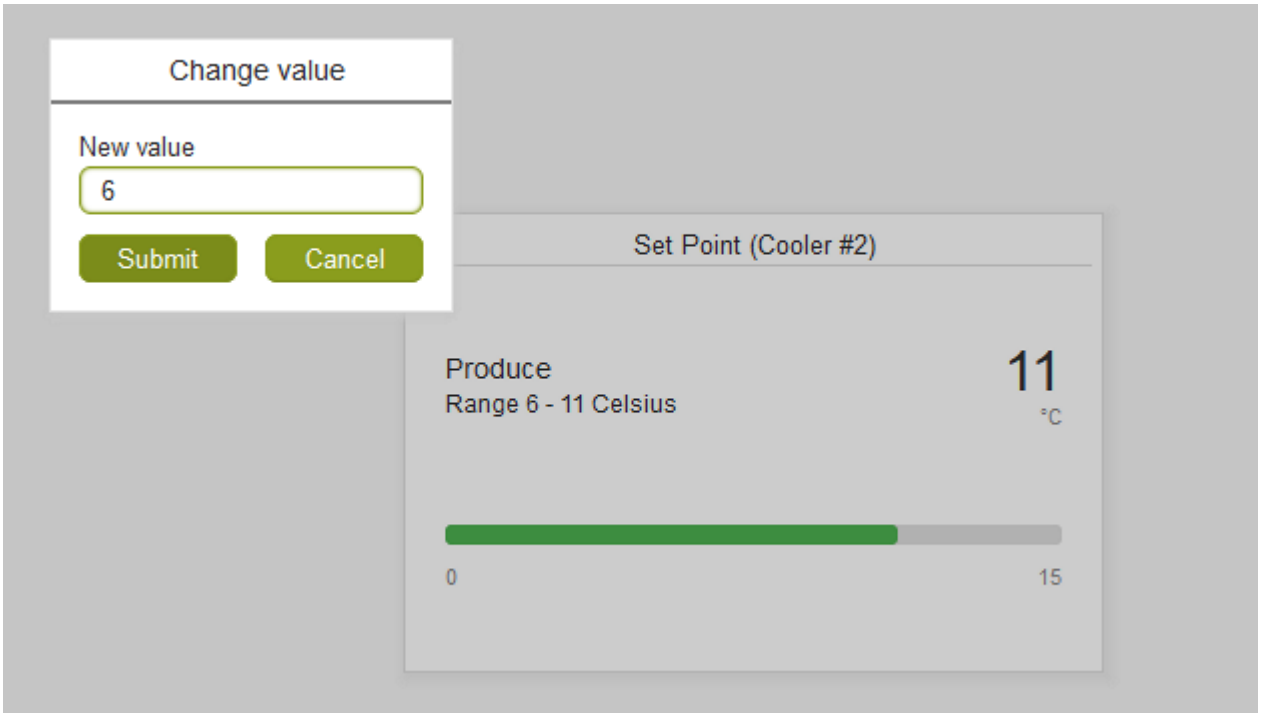
Click on the color swatch to access the color palette and select a color for the range bar

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/fieldvalue

Last update: **2021-10-01 16:09**

Field Value with Editor



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io Field Value with Editor widget is, as the name suggests, a Field value widget that allows users with sufficient privilege's to change the value shown. Simply click on the numeric value and the “Change value” dialog box will appear. This functionality can be used to for many different purposes. For example, the Field value it is linked to could be a variable in an alarm condition. There are a few configuration settings required on a field to make accommodate this functionality. For more information see the configuring Fields section of this manual.

Minimum size (value, name, unit only) - 2 x 2 grid units

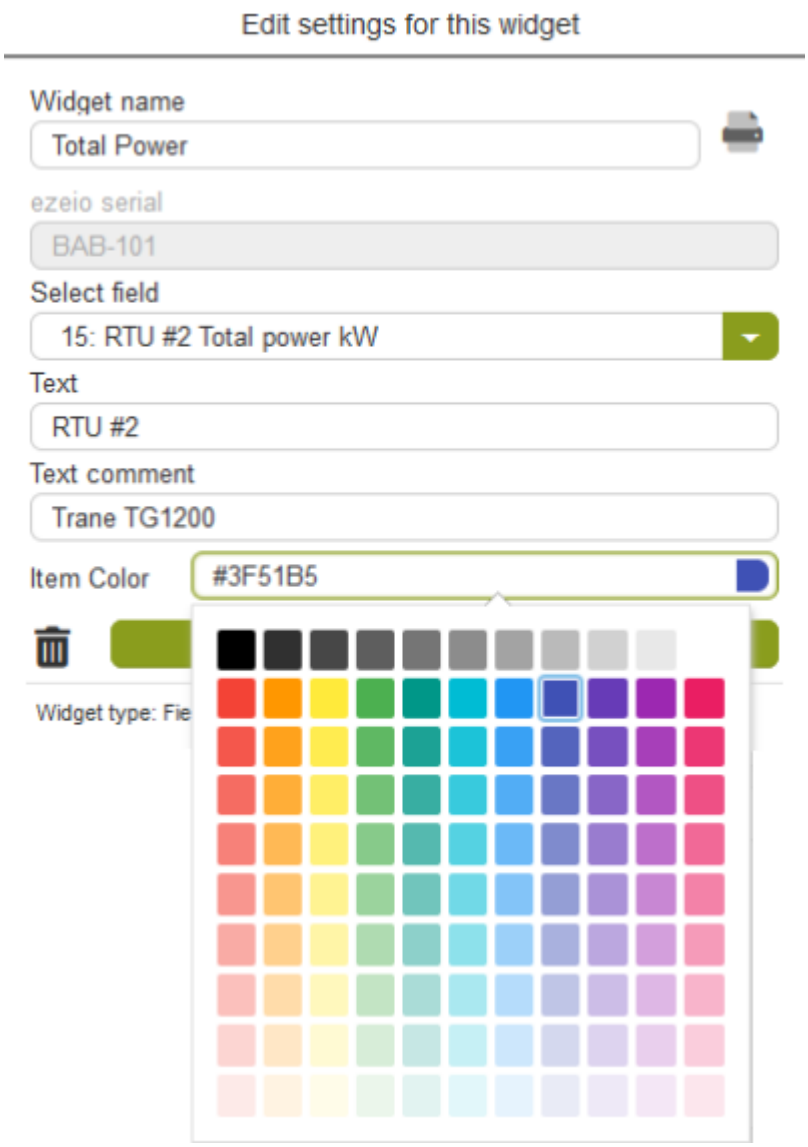
Minimum size (full feature) - 3 x 6 grid units

Features / Options

- Numeric value shown with unit
- Percentage of range shown as horizontal level bar

- 3 user defined text fields displayed

Settings



Widget name

Appears top and center of widget

Select field/s

A drop down menu provides the means to link to a specific “Field”, from the desired ezeio. For more information see mapping on the widgets page.

Text

Enter text to be displayed to the left of the Field value (1st line)

Text comment

Enter text to be displayed to the left of the Field value (2nd line)

Item color

Click on the color swatch to access the color palette and select a color for the range bar

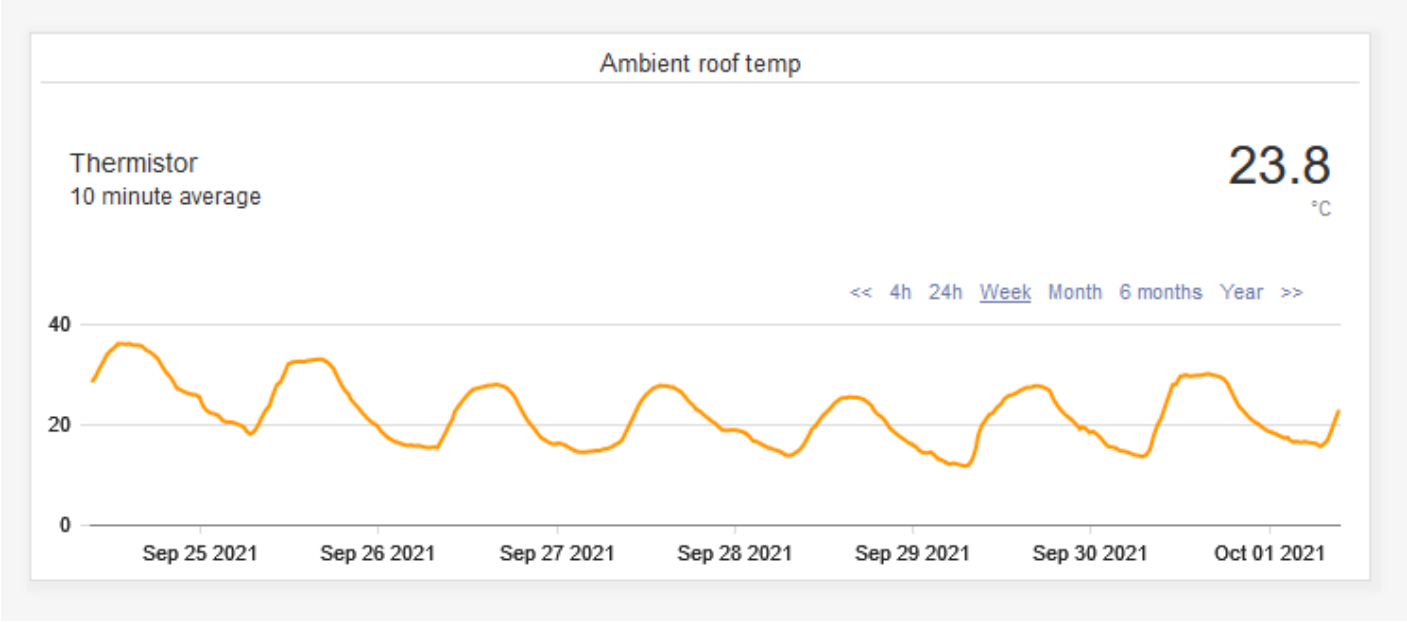
From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/fieldvalueeditor

Last update: **2021-10-01 16:10**

Field Value with Graph

Also see Line Chart widget



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io “Field value with Graph” widget tracks displays a single Field value with a line chart. The numeric value can be removed to show just the graph. The view is a sliding widow of time with most recent data on the right. Viewer can select time frames from 4 hours to 1 year. The window can also be shifted forward or backwards along the time line. This widget is best suited for data that rises and falls over time such as temperature.

Minimum height (grid units) - 4

Features / Options

- 6 Viewer selectable time frame windows
- Show show or hide numeric value
- Arrow forward or backwards along the time line
- Show zero (opt in). Scale of chart is based on range of data values unless “Show zero” is checked“.

Settings

Widget name

Appears top and center of widget

Select field/s

A drop down menu provides the means to link to a specific “Field”, from the desired ezeio.

Text

Enter descriptive text if desired

Text comment

Enter descriptive text if desired

Item color

Click on the color swatch to access the color palette and select a color for the graph.

Graph only

Check the “Graph only” box if you would like to hide the numeric value.

Show zero

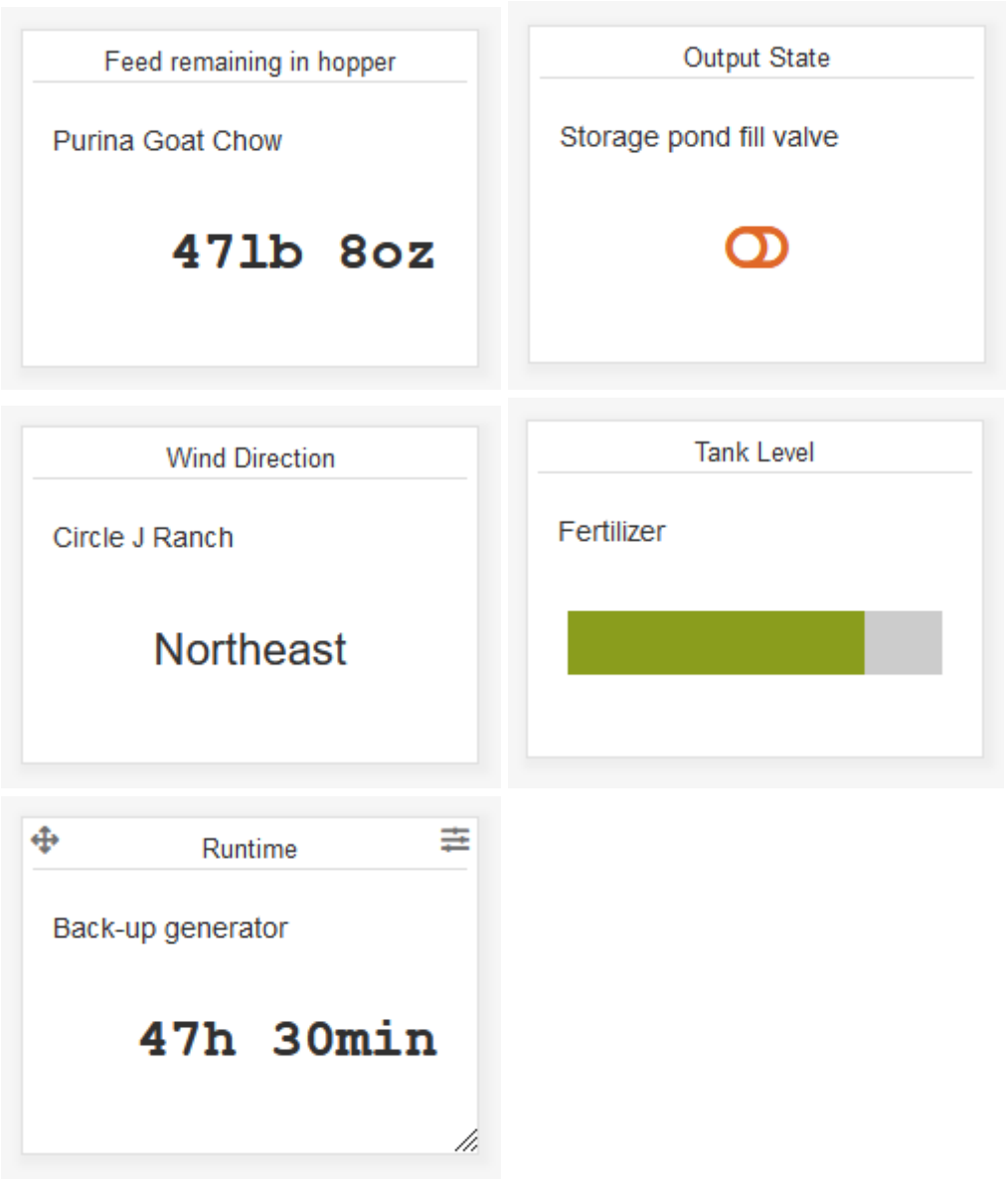
By default, the chart will scale to the range of the data, in order to optimize resolution. “Show zero” forces the chart's scale to start at zero or include zero. If values are negative, zero would be at the top of the chart. Example: If your data ranges from 10,500 to 12,500 and you select “Show zero”, your chart will render the 2000 point deviation as a nearly flat line.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/fieldvalve_graph

Last update: **2021-10-01 17:01**

Formatted Field Value Widget



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io Formatted Field Value widget displays the Field value as it is shown in the “View” column of the “Fields” and “Status” tables. This formatting converts the numeric value to one of several options available from the “Display format in view” setting of each Field.

Minimum size - 3 grid units

Settings

Widget name

Appears top and center of widget

Select field/s

A drop down menus provide the means to link to a specific “Field”, from the desired ezeio. For more information see mapping on the widgets page.

Text

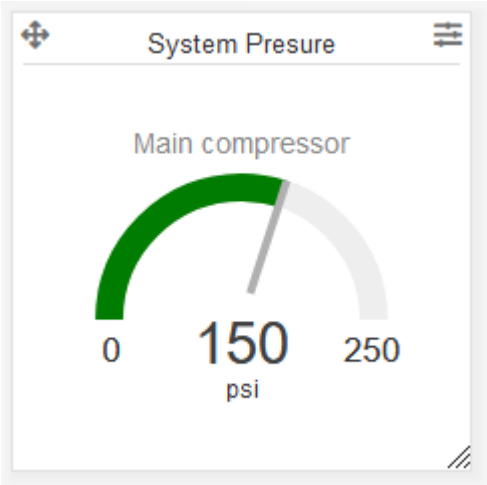
Enter text to be displayed to the left of the Field value

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/formatedvalue

Last update: **2021-09-24 23:07**

Field Value as Gauge



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io Field Value as Gauge widget displays the current value of a single field as a dial gauge, similar to the analog gauges on an old car or instrument panel. The numeric value is shown in the center of the widget and a pointer indicates where the value is position in the define range. The area to the left of the sweeping pointer is filled with a user defined colors. Three colors can be assigned to a maximum of 5 segments of the range. This widget is designed for live/instantaneous values. These values could be rapidly changing such as a power reading (kW) or slow like a tank level.

Minimum height (grid units) - 2

Features / Options

- Numeric value & range displayed
- Up to 5 range segments (max of 3 colors)
- Field name and engineering unit shown

Settings

Widget name

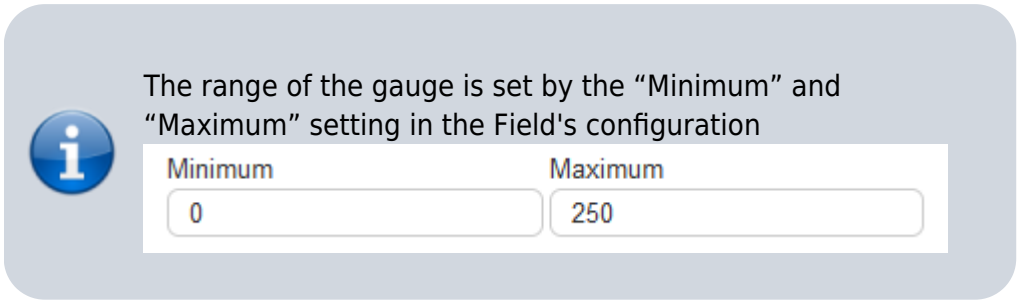
Appears top and center of widget

Select Field

A drop down menu provides the means to link to a specific “Field”, from the desired ezeio. For more information see mapping on the widgets page.

Item Color

Select the main or lowest color layer by clicking on the color swatch. This opens the color palette. Select the desired color and the palette will close. This color can be displayed at the beginning and end of the gauges range.

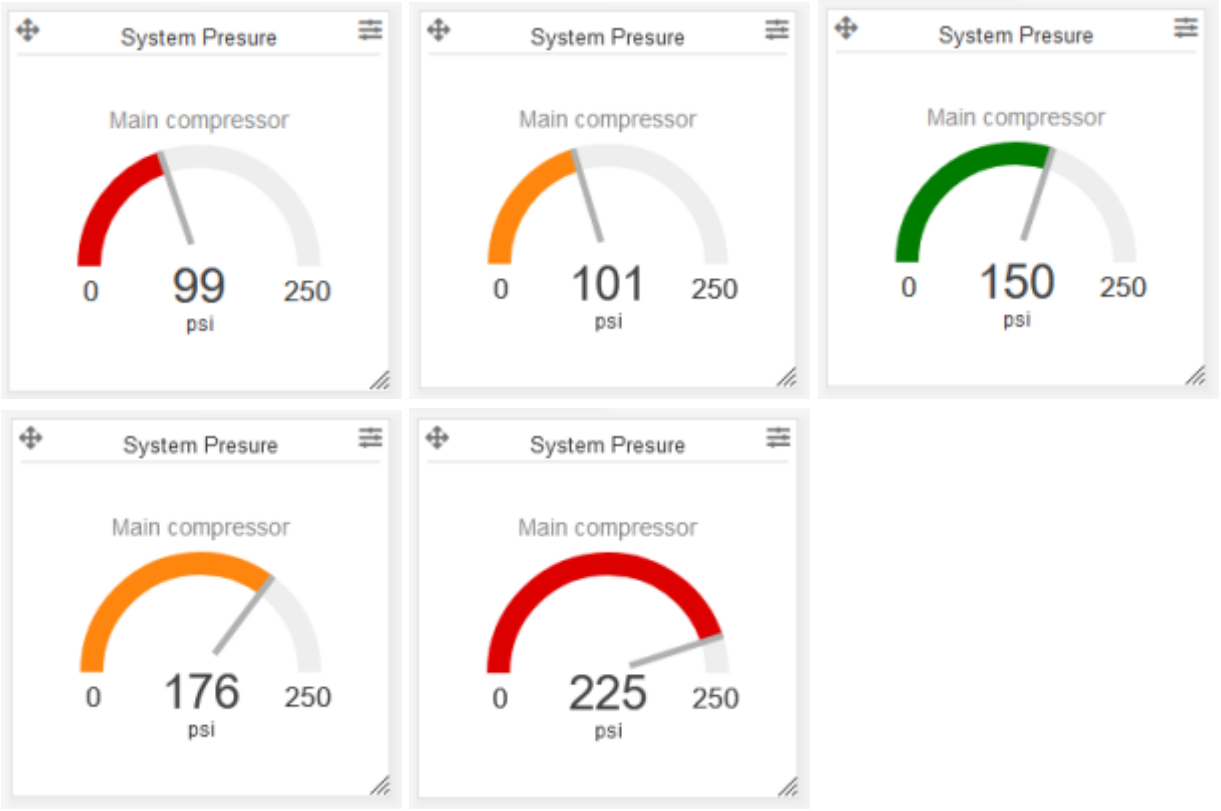


Middle Range

Define the middle range and select a color (using the same method described above). In terms of color layers, the middle range lies between the item color (back) and the “Inner Range” color (top). In the example below this range is represented in orange.

Inner Range

Define the inner range and select a color. In terms of color layers, the inner range color is the top. In the example below this range is represented in green.



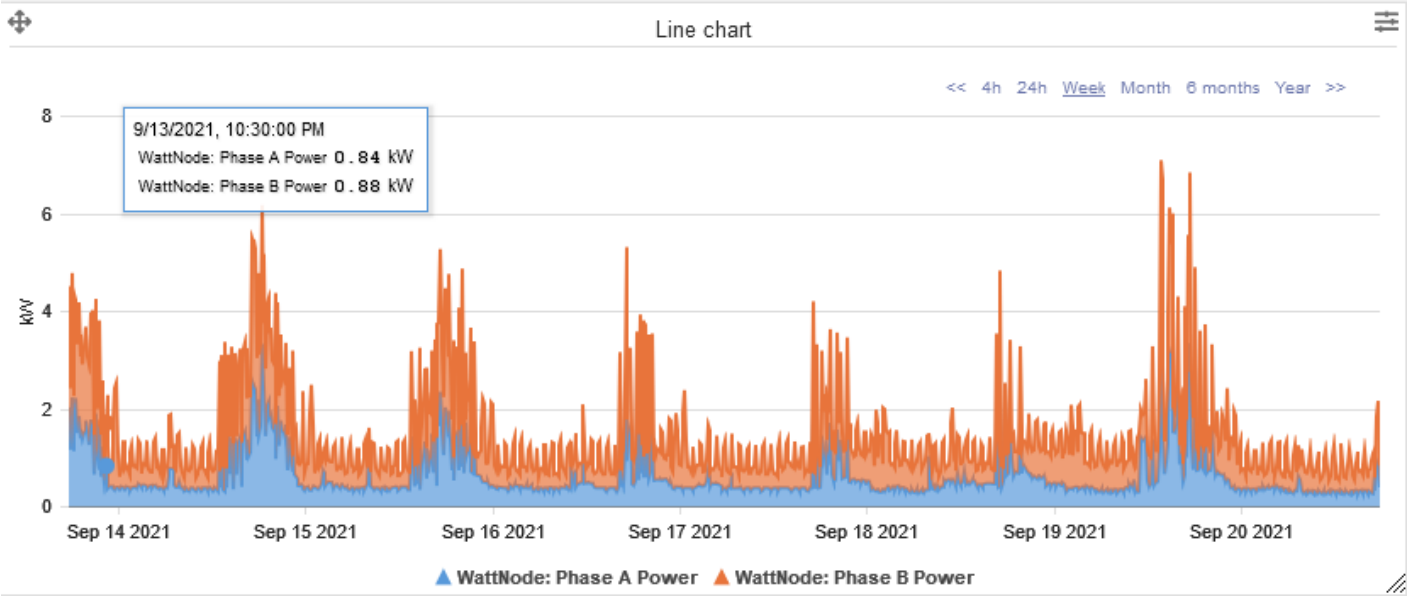
In the example above, two ranges are entered. *The third range (full sweep of the gauge) comes from the field's min and max setting*. The widget setting's allow three colors to be associated with the three ranges. The results is three to five segments of the dial to displaying one of three colors as it completes a full sweep.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/gauge

Last update: **2021-10-01 16:09**

Line Chart



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io line chart widget tracks the changes of up to 5 data points. The view is a sliding widow of time with most recent data on the right. Viewer can select time frames from 4 hours to 1 year. The window can also be shifted forward or backwards along the time line. This widget is best suited for data that rises and falls over time such as temperature.

Minimum height (grid units) - 3

Features / Options

- Up to 5 data points
- 6 Viewer selectable time frame windows
- Arrow forward or backwards along the time line
- 8 Chart types (Line, area, step, etc.)
- Selectable timeline cutoff (“Do not show data before this date”)
- Show zero (opt in). Scale of chart is based on range of data values unless “Show zero” is checked“.

Settings

Widget name

Appears top and center of widget

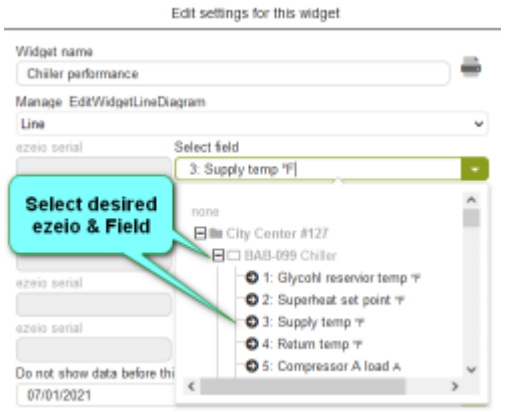
Select Chart Type

Drop down menu offer the graphical styles listed below

Invalid Link
Line
Invalid Link
Spline
Invalid Link
Step
Invalid Link
Area
Invalid Link
Spline Area
Invalid Link
Step Area
Invalid Link
Stacked Area
Invalid Link
Stacked Area 100%

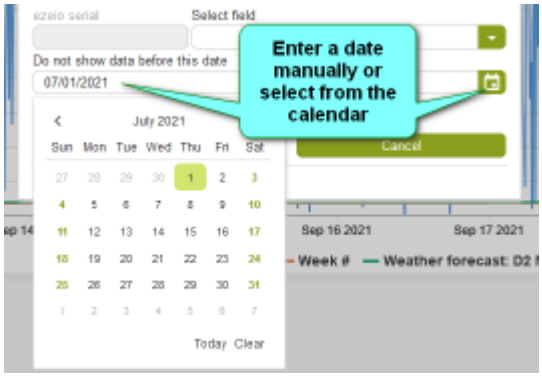
Select field/s

Five drop down menus provide the means to link to a specific “Field”, from the desired ezeio.



Do not show data before this date

Sets a cutoff date, so older data is not displayed on the chart



Show zero

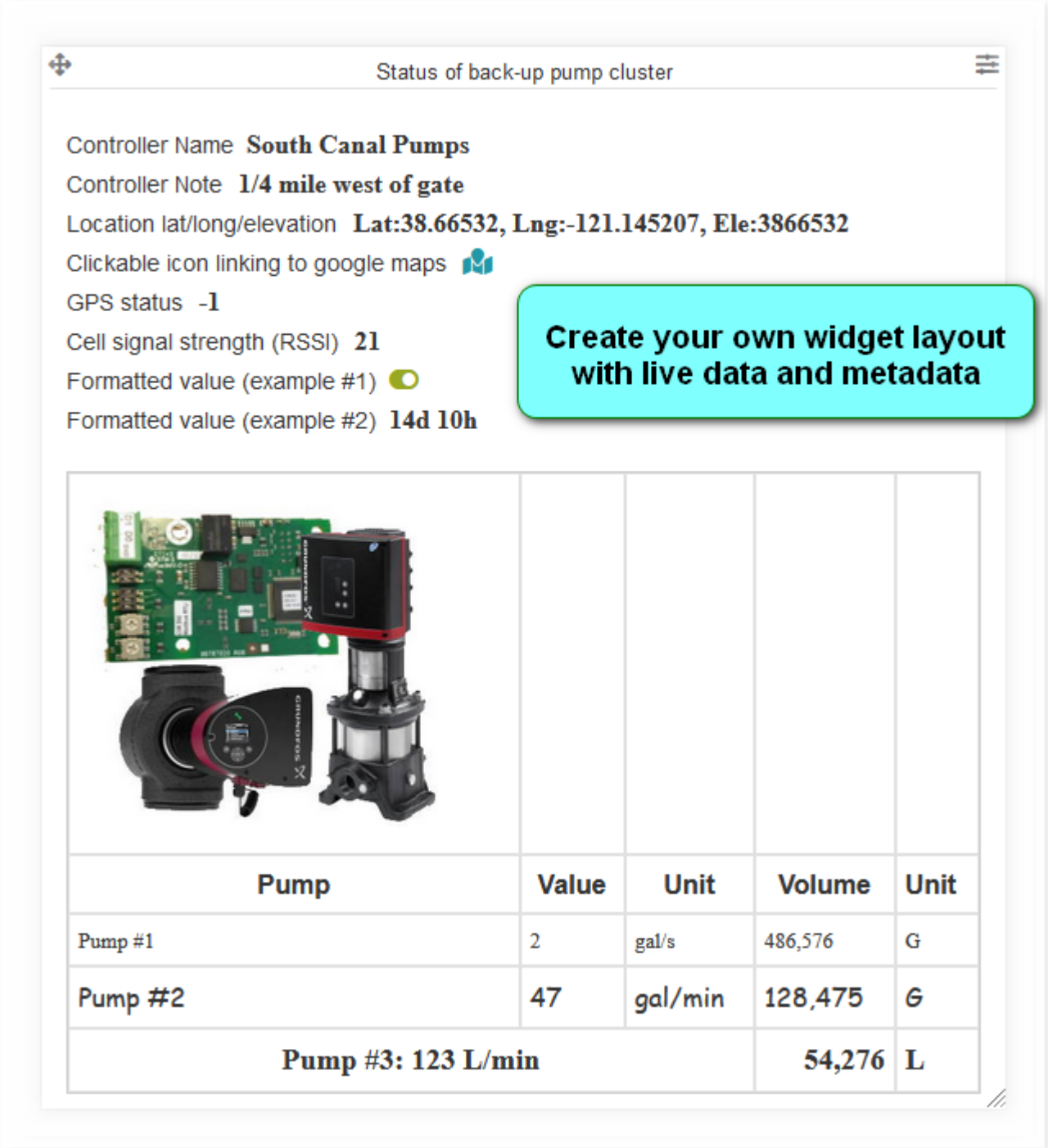
By default, the chart will scale to the range of the data, in order to optimize resolution. “Show zero” forces the chart's scale to start at zero or include zero. If values are negative, zero would be at the top of the chart. Example: If your data ranges from 10,500 to 12,500 and you select “Show zero”, your chart will render the 2000 point deviation as a nearly flat line.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/linechart

Last update: **2021-09-24 21:56**

Note Widget



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found on the Widgets section of the manual.

Description

The eze.io note widget allows users to input text and insert tags to show live and meta data. The tools provided are similar to a document writer application. As shown n the example above images and tables can be inserted.

Minimum height - 2 grid units

Features / Options

- Display user defined text
- Edit: size, font, color, etc.
- Display live and meta data
- Insert images
- Insert Table
- Insert map links

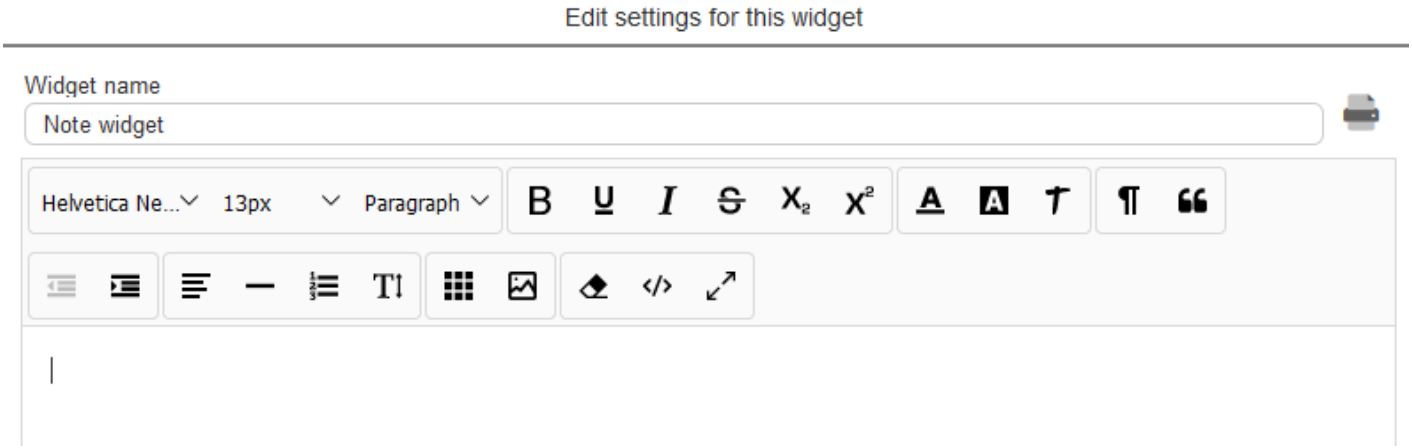
Settings

Macro $1 and Macro $2

The editor accepts two macros. Any occurrences of '$1' will be replaces by the text in the Macro $1 box. Same thing applies to $2. If no text is entered for the macro, no replacement happens in the text body. The macro replacement happens before any other processing of the text body.

Document Editor

The “Note Widget” has a limit of 4000 kb of code. A counter in the lower left of the editor shows the current size of the code. Appling styles to text will quickly consume the available memory.

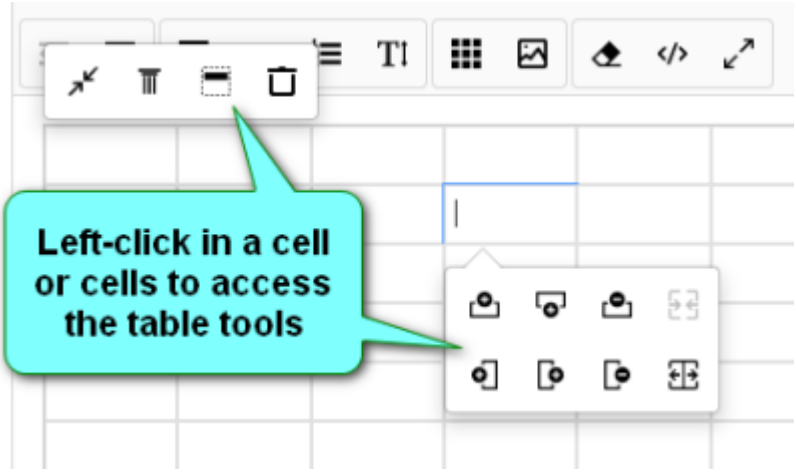


The document editor has all the typical editing tools, including subscript, superscript, line height and code view.

Insert image

Insert table

Document editor allows the user to insert a table that stretches (by default) to fil the width of the widget. Click on the table tool and select an initial number of columns and rows (max 10 x 10).



- Minimize table
- Fix Column width
- Add header row
- Delete
- Add or remove rows
- Add or remove columns
- Split or merge cells

Available Tags

The tags utilized by this widget have a similar naming convention to those used in “Message Templates” and both require the tag to be encapsulated by square brackets “[ ]”. Since Dashboards can access multiple ezeio controllers, the tag must include a serial number.

Tag	Description	Origin
[ABC123,EZENAME]	Name of controller	System settings (user defined)
[ABC123,EZENOTE]	Note	System settings (user defined)
[ABC123,LASTCOM]	Time since last communication	System status
[ABC123,FIRMWARE]	Firmware version	System status
[ABC123,GPS]	GPS location as lat/long/elev	System info, based on cell proximity or IP if no GPS receiver
[ABC123,GPSLINK]	Shows a clickable icon linking to google maps	System info, based on cell proximity or IP if no GPS receiver
[ABC123,GPSMAP]	Inserts small map showing current location	GPS receiver if available
[ABC123,GPSSIGNAL]	GPS status	GPS receiver if available
[ABC123,RSSI]	Cell signal RSSI	Cellular modem
[ABC123,F,1]	Field name + Value + Unit	Fields (user defined)
[ABC123,FV,1]	Field's value (formatted)	Fields (user defined)
[ABC123,FR,1]	Field's value (unformatted)	Fields (user defined)
[ABC123,FU,1]	Field's unit	Fields (user defined)
[ABC123,FN,1]	Field name	Fields (user defined)
[ABC123,AREAID]	Group ID of geographic area including the controller	GPS location
[ABC123,AREANAME]	Group name of geographic area including the controller	GPS location
[ABC123,AREADESC]	Group description of geographic area including the controller	GPS location
[ABC123,PREVAREAID]	Previous group ID of geographic area including the controller	GPS location
[ABC123,PREVAREANAME]	Previous group name of geographic area including the controller	GPS location
[ABC123,PREVAREADESC]	Previous group description of geographic area including the controller	GPS location

It's convenient to use the macros in place of the controller identifier. Example:

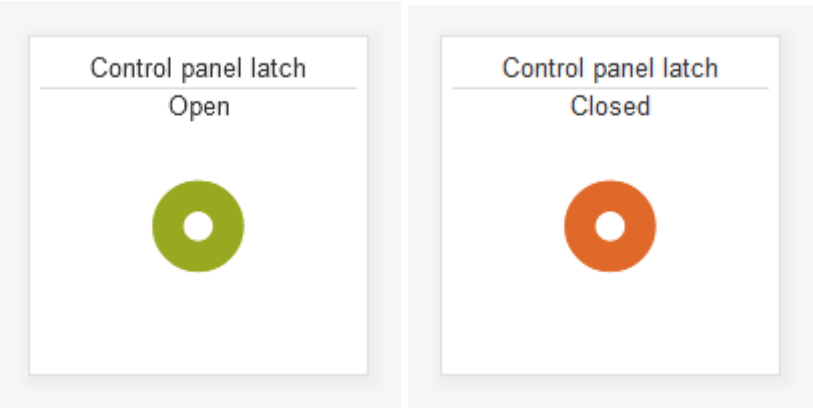
```
My controller serial is $1 and its name is [$1,EZENAME]
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/note

Last update: **2025-08-26 23:11**

Push Button Widget



Description

This widget can be used as an actuator or an indicator. The circular button's color changes from between green and red to indicate the state. User defined text can also be added to clarify the state and purpose of the button.

Actuator If connected to a Field that is writable, pushing the button will switch the state of the Field (to a value of 100 or 0, depend on the current state).

Indicator If connected to a field that is not writable, the zero or non-zero value will determine which of the two user defined text statement is displayed.

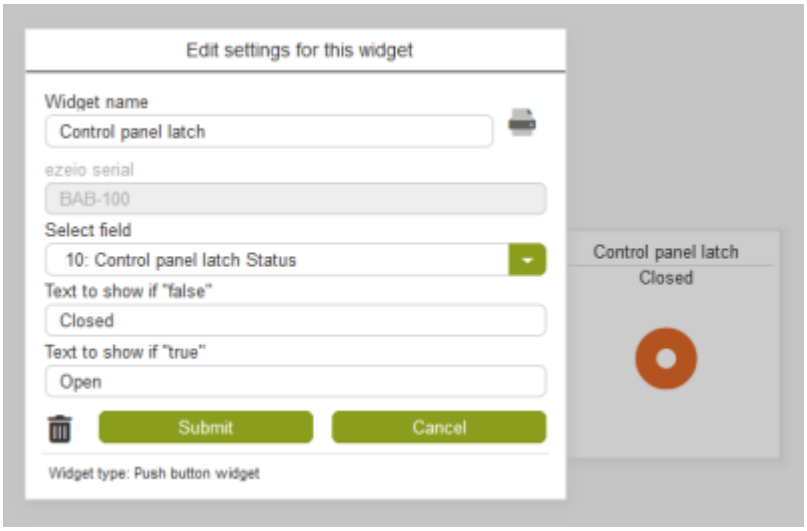
Pushing the button

When pressed the widget will attempt to set the Field value to 100. A spinning graphic may appear while the system attempts to set the state. When successful, the button color will change to green and the “if true” text will appear. The Field will remain at 100 until another command, action or event changes the value.

True or False

- Field values other than zero are evaluated as **True**. This includes negative numbers and decimals.
- Zero is evaluated as **False**

Settings



Select field

A drop down menu provides the means to link to a specific “Field”, from the desired ezeio. For more information see mapping on the widgets page.

Text to show if "false"

As stated above, zero is evaluated as **False**. Enter the desired text to display if false.

Text to show if "true"

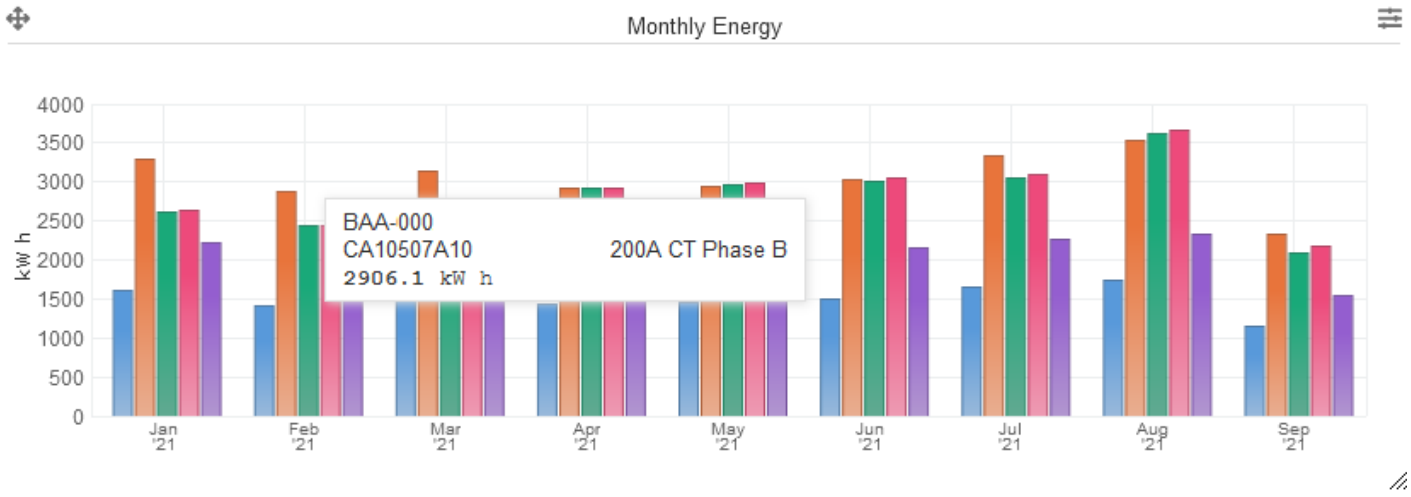
As stated above, any non-zero Field value is evaluated as **True**. Enter the desired text to display if true.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/pushbutton

Last update: **2022-02-02 23:19**

Bar Graph



General Dashboard and Widget info

Features common to all Widgets and general Dashboard info can be found in the Dashboards section of the manual.

Description

The eze.io Bar Graph widget displays total accumulation over a user selected time frame, ranging from hourly to monthly. Up to 5 data points can be tracked. This widget is best suited for comparing or tracking usage, for example water or energy.

Minimum height (grid units) - 3

Features / Options

- Up to 5 data points
- 7 User selectable time frames
- Selectable timeline cutoff (“Do not show data before this date”)

Settings

Widget name

Appears top and center of widget

Select Bar Diagram Type

Invalid Link
Side by side
Invalid Link
Stacked

Select field/s

Five drop down menus provide the means to link to a specific “Field”, from the desired ezeio. For more details see the “Point to Field” section of the Widget page.

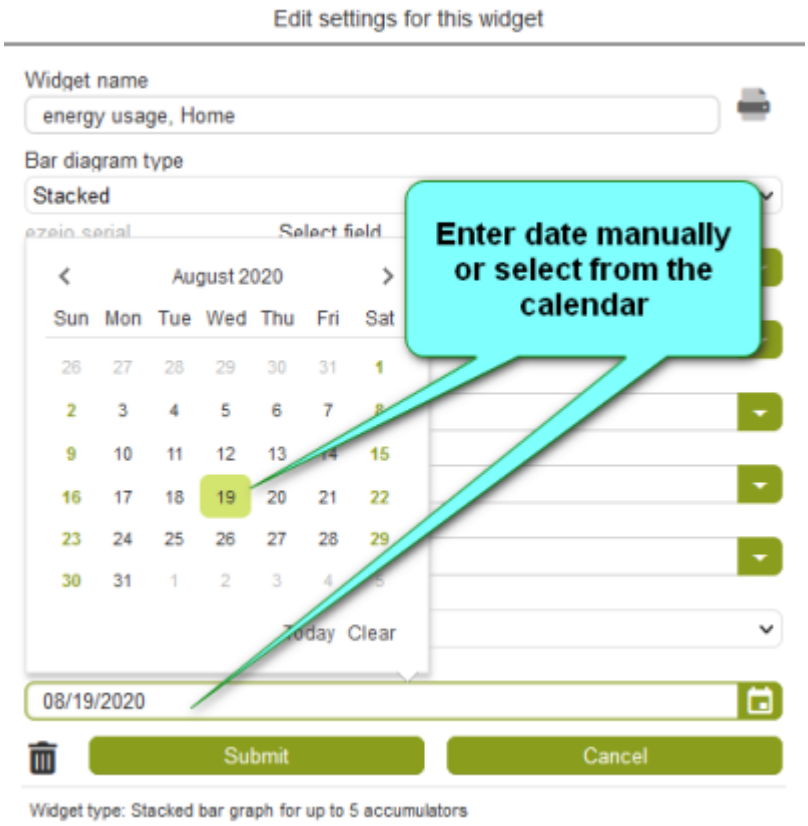
Select Bar Width and Time Scale

Drop down menu offer the graphical styles listed below

- Hourly, for 24 hours
- 4 Hours, for 4 days
- Daily, for 10 days
- Daily, for 30 days
- Weekly, for 1 year
- 4 weeks, for 2 years
- Calendar month, for 2 years

Do not show data before this date

Sets a cutoff date, so older data is not displayed on the chart



From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/dashboards/widgets/stackedbargraph

Last update: **2021-09-24 21:56**

Destinations & Destination Lists

The purpose of these two resources is to allow efficient management of message templates and where they are sent. Once created, a Destination List can be selected for an unlimited number of “Send message” alarm actions. If changes are required, such as on-boarding a new employee, simply edit the Destination List and the effect is account wide.

Although visible from the groups under an account, Destinations and Destination List tabs are an account resource. If the users of a group need to create and manage their own Destinations and Destination Lists, their group must be upgraded to an account.

To access Destinations and Destination List a user must be invited to an account and have the privilege “Can add/change destinations”.

Destinations

Destinations are the endpoints for messages sent via “Alarm Actions”. These endpoints are typically various methods of connecting with human recipients, but API calls to other software programs is also an option. The “Destination” tab for each “Account” is like a address book, but since each destination is a single endpoint an individual human recipient may have multiple entries to list; email, cell phone, and text.

Endpoint Type	Action	Example
Send email	Directs message to the email address entered	john.smith@mybiz.com
Post http(s) Message	Directs Message to the server address entered	https://my.server.com/script
Send SMS	Directs text message to the phone number entered	**US:** +12125551234 **AU:** +6130717523456
Call phone with voice message	Directs a voice message to the phone number entered	**US:** +12125551234 **AU:** +6130717523456
Send to Pushover	Directs a text message to an individual through the Pushover app, using their Pushover key	:abc123def456abc123def456abc123xy
Send to Nagios	Nagios is a open source monitoring system. See https://www.nagios.org	https://my.nagiosserver.com/nrdp + access token
Send command	Send a command to a different ezeio. See control API for possible commands.	Serial number of ezeio to send command to

Destination Lists

It is typical for alarm messages to be sent to multiple destinations. Destinations Lists allow you to create groups like “Field Techs”, “2nd Shift”, or “Office staff”. These groups are then selected from the drop-down-menu of the Alarm and/or Restore Actions. *A Destination* can *be assigned to multiple lists*.

Creating Destination Lists

Add List - The **+ Add List** button at the top of the left-hand panel, under the Destination List tab, is where you create and name a new list. You may also choose to activate the list now.

Add List Step - The **+ Add List Step** button at the top of the right-hand panel allows you to add Destinations to a list (new or old). The only requirements to adding a List Step are, select the Destination from the drop-down-menu and click Save changes. Optionally you may want to customize the Message Template. For more information on see Message Templates and Message Template Tags.

Managing Destination Lists

When an employee is added, promoted or leaves the company, this requires changes to potentially hundreds of alarms. Destination Lists allow you to make that change in one place. Add or remove a Destination from the List Step, save the changes and the change will be reflected on all the alarms using that Destination List.

Editing Destination Lists - The pencil and trash can icons under the Destination Lists' Edit column allow users, with adequate privileges, to change the name, active/inactive status or delete a Destination list.

Editing List Steps - The pencil and trash can icons under the List Steps' Edit column allow users, with adequate privileges, to swap Destinations, edit Message Templates, or delete the List Steps.

Message Templates

The Message Template allows a user with adequate privileges, to customize the message assigned to each List Step (Destination) using plain text and Message Template tags. The Default message templates are tailored to the Destination Type, making the default SMS messages shorter than email. You can also choose to give more or different information to some Destinations (recipients). Message Template Tags allow the message to contain current field values, device status, system info, meta data, even a Google maps link with the units location.

Default SMS Message Template

Message structure	Message item	tags
Sender	server phone #	Not customizable
Body	Message Type, Action Name	Not customizable
Body	Alarm Action Message, Time	[ACTIONMESSAGE] [ISOTIME]

Default email Message Template

Message structure	Message item	tags
Sender	ezeio S/N, Name	Default to `{serial}@eze.io`, see below
Subject	Message Type, Action Name	[SOURCENAME] [ACTIONAME]
Body	Alarm Action Message, Date, Time	[ACTIONMESSAGE] [ISODATE] [ISOTIME]
Footer (Fixed)	https://ezeio & ezeio serial #	See below

Customizing the email subject

Customize an email Message Template by adding your own Subject text or tags followed by the special character "|". Text and tags before the "|" will be formatted as the subject line, everything after the "|" will be formatted as the body of the email.

Example: `[SOURCENAME] [GROUPNAME] | [ACTIONMESSAGE] [USTIME] [GPSLINK]`

Customizing the email sender

By default, the email sender will be `{serial}@eze.io`, where `{serial}` is the serial number of the ezeio controller generating the email. In cases where the email is sent to some automated system that only accept specific senders, the email sender can be customized by using the Token field in the destination setting.

If the token ends with an at (@) character, the text before the at-character will be used as the sender.

Example : If the token is set to `marvin@`, the sender address will be `marvin@eze.io`. The domain cannot be customized.

The token supports message tags, so `[EZENAME]@` will result in a sender address using the name of the ezeio. Please be aware that the resulting address must conform to valid email addresses (https://datatracker.ietf.org/doc/html/rfc2822)

Customizing the email/SMS footer

By default the system will automatically add a short footer with the serial number to email and SMS messages, and a preamble to voice messages. You can eliminate this by adding a caret (^) as the very last character in the message template.

POST format

When using the POST destination, the message will be formatted in JSON as follows (whitespace added for readability):

```
{
  "time": "2022-06-11T16:05:42Z",
  "subject": "ALARM Alarm name",
  "eventid": 90,
  "type": "ALARM",
  "source": "SEND",
  "sourceid": 1,
  "sourceindex": 1,
  "sourcename": "Alarm name",
  "actionname": "This is the action name",
  "param1": "0.000000",
  "param2": "0.000000",
  "param3": "0.000000",
  "param4": "0.000000",
  "message": "The message body from the action",
  "text": "",
  "meta": {
    "serial": "BAA-999",
    "group": "55",
    "accountgroupid": "2",
    "name": "Name of controller",
    "note": "Text from the note field",
    "tzofs": "0"
  },
  "adc": {
    "1": 2,
    "2": 1,
    "3": 2,
    "4": 279,
    "5": 5232,
    "6": 2,
    "7": 16960,
    "8": 2,
    "Vin": 12350,
    "Vbat": 12268,
    "V5": 5061
  },
  "out": {
    "1": 0,
    "2": 100,
    "3": 0,
    "4": 0
  },
  "pos": {
    "x": 0,
    "y": 0,
    "z": 0,
```

```
    "signal": 0
  },
  "fields": {
    "1": {
      "name": "Field 1",
      "unit": "",
      "assettag": [],
      "raw": 9659017,
      "value": "9,659,017"
    },
    "2": {
      "name": "Field 2",
      "unit": "",
      "assettag": [],
      "raw": 17,
      "value": "17"
    }
  }
}
```


Additionally, a “X-Hash” header is added to the message. The X-Hash is calculated as a SHA1 hash of the message payload text with the destination token appended at the end. This may be useful to validate the authenticity of the message.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/destinations

Last update: **2026-01-12 20:23**

Ivanti integration

About Ivanti

Ivanti Neurons is a enterprise scale ITSM system, primarily designed to manage IT services.

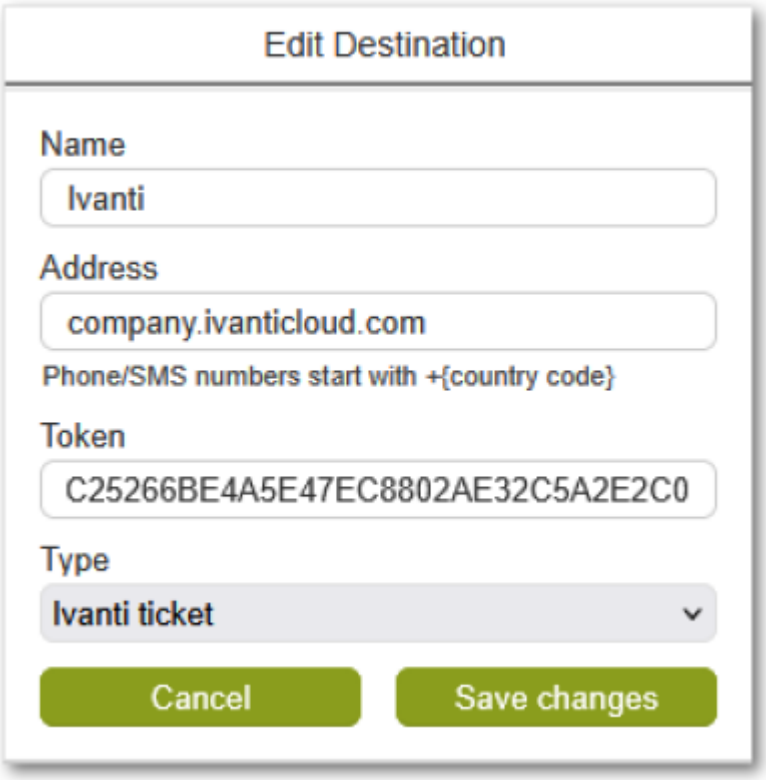
Prerequisites

The Ivanti REST API endpoint must be accessible from the public Internet and the server must have a valid SSL certificate.

Configuring an Ivanti destination

In the Address field, enter only the Ivanti API domain name, without protocol prefix or trailing slashes.

In the Token field, enter the Ivanti API access token (32 characters)



Setting up ezeio alarms for Ivanti

The ezeio supports three (3) types of Ivanti actions:

1. Creating a new ticket
2. Updating an open ticket (journal entry)
3. Adding a resolution to an open ticket

Note that the ezeio does not automatically change the ticket status to closed when adding a resolution.

When an Ivanti message is sent in an Alarm Action, and there is no open ticket for this alarm, a new ticket is opened.

If the message is sent in a Restore Action, the Resolution field is populated and the ticket is closed in the ezeio system, but remains open in Ivanti.

If the message is sent as the result of an Alarm or Restore action of an alarm that has a parent reference, the message is added to the existing ticket as a Journal entry.

Template notes

The default alarm template for Ivanti tickets looks like this:

```
@I:[EZENAME1]     // This is the site ID, extracted from the first part of the
ezeio name
@T:mec            // The 'AlarmType' field
@O:ITL            // The 'Source' field
@S:Open           // The 'Status' field
[ACTIONMESSAGE]
```

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/destinations/ivanti

Last update: **2026-01-06 21:07**

Nagios integration

About Nagios

Nagios is an open source software designed for monitoring networked systems. While its primary purpose is to monitor the status of computer networks and IT-related services, it can also display information from the ezeio system.

Prerequisites

The Nagios service must allow passive checks using NRDP. Please see the Nagios documentation for details.

https://assets.nagios.com/downloads/nagiosxi/docs/Configuring_Inbound_Checks_With_XI.pdf

The NRDP endpoint must be accessible from the public Internet and the server must have a valid SSL certificate.

ezeio mapping to Nagios hosts/services

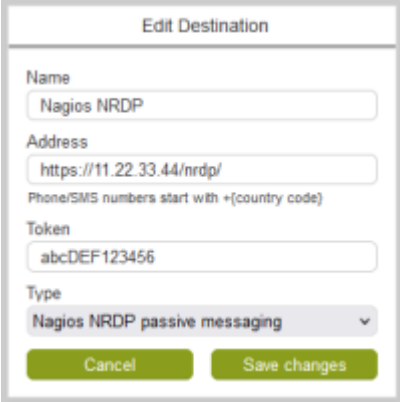
The ezeio name (configurable under Configure→System→System Name) will be mapped to the host name in Nagios. Please keep the name short and avoid special characters as those may cause issues in Nagios.

When setting up alarms, the ezeio alarm name (Configure→Alarms→Name) will be mapped to the Nagios Service name, and the action message will be attached to the service status message. Alarms are always given a “critical” status in Nagios, while restore messages are given the “ok” status in Nagios.

When setting up periodic data updates, the ezeio field name (Configure→Fields→Name) will be mapped to the Nagios Service name and the field value will be attached to the service status.

Setting up the Nagios Destination

Navigate to the Destinations tab, and add a destination of type “Nagios NRDP”.



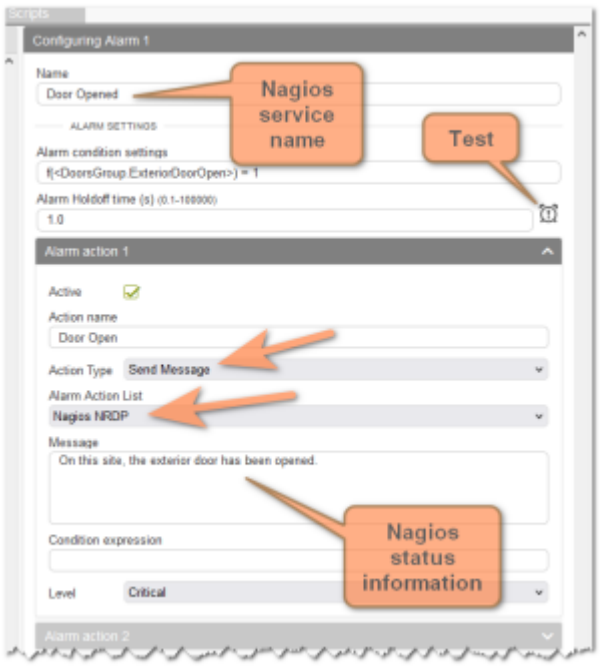
The address shall be the URL of your nagios service, typically ending with “/nrdp/”.

The token is generated from Nagios, and shall be copied as-is.

Don't forget to also create a destination list and add the new destination to it.

Sending alarms/alerts to Nagios

To send an alarm to the Nagios server, simply create an action of the “Send Message” type, and use the destination list with the Nagios destination.



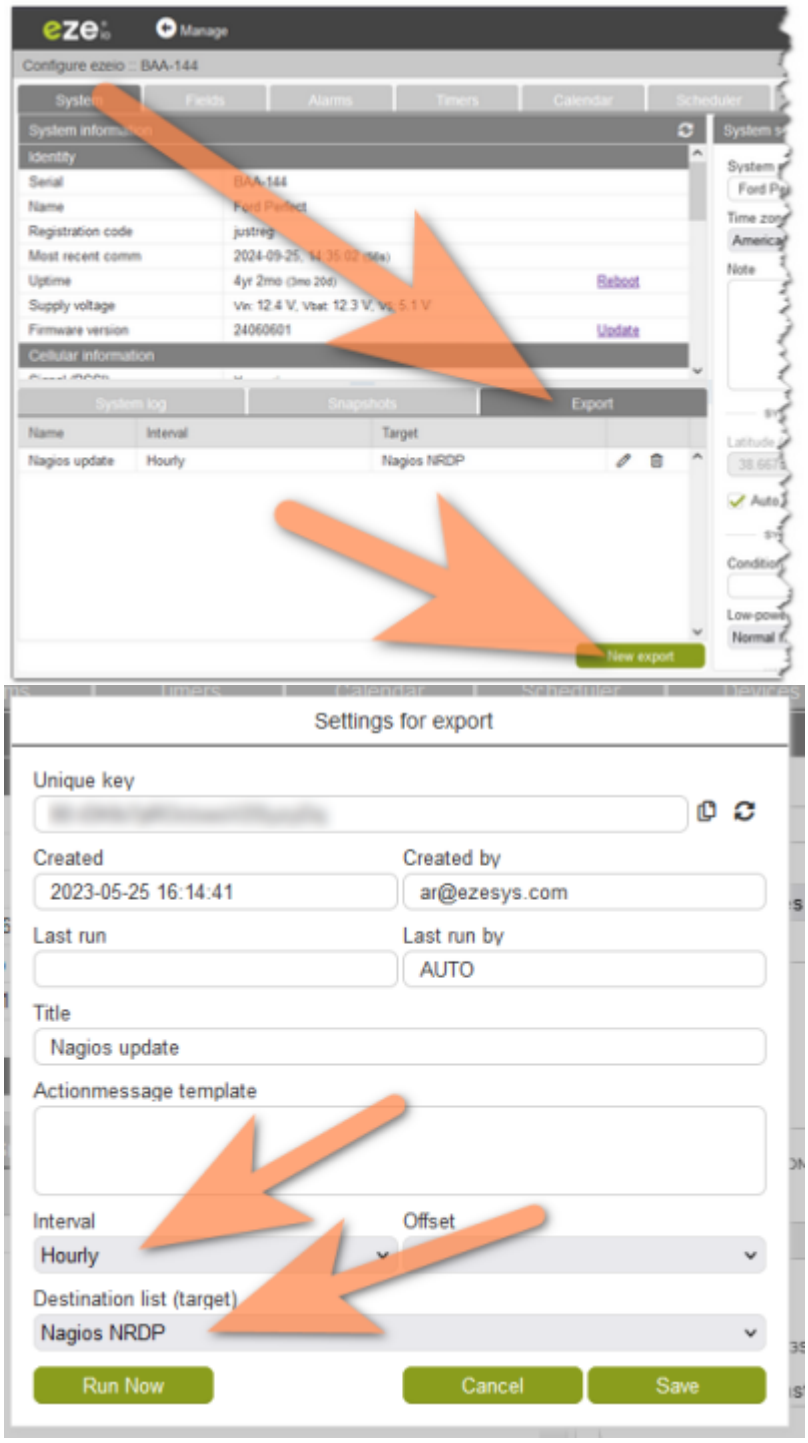
Note that you should also set up a restore action to send the reset message.

Periodically sending data to Nagios

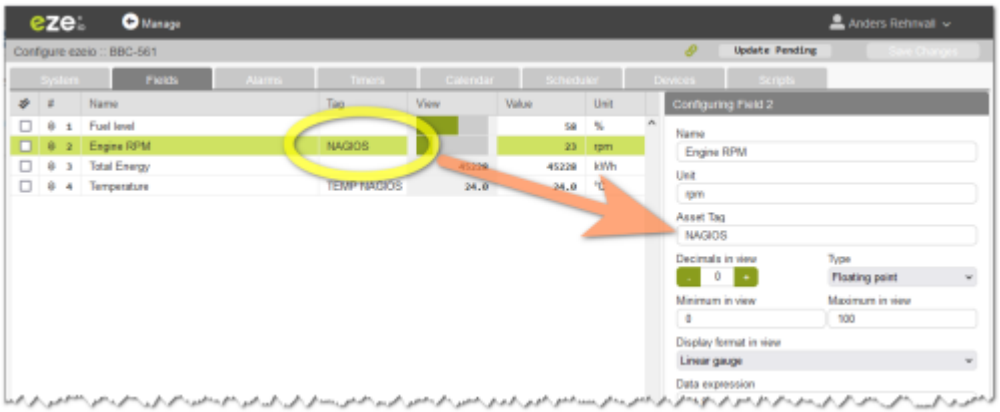
To set up periodic (hourly) updates to Nagios for logging purposes navigate to

Configure→System→Export.

Click New Export at the bottom of the screen, and set up an hourly export selecting the Nagios destination.



To select which fields to include in the export, navigate to the field settings and add the tag “NAGIOS” to each field that should be exported. Note that you can add multiple tags to a field, separated by space.



From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/destinations/nagios

Last update: **2024-09-25 21:37**

Group Settings

Overview

The ezeio system is designed to allow a user (potentially) to have access to multiple groups and/or accounts from a single login. Registering your first ezeio creates your account. Additional groups and sub-accounts can be created to organize ezeio controllers, dashboards, users, services and more.

Groups

A “group” in the ezeio system works just like a folder on your computer. Groups are arranged in a tree-structure, where each group may contain other groups, on its branch. There is no limit to how many groups you can create, or how many groups can be contained in a higher-level group.

In addition to containing other groups, each group may also contain ezeio controllers and users. There is no restriction on how many ezeio controllers a group may hold, if any. This is similar to how you can store files in folders on your computer (the files being ezeio units and users, while the folders are comparable to groups).

A group is not required to contain an ezeio. You may choose to use a group to manage user access to it's subgroups. For example: a chain of stores may have regional management and maintenance personal without a physical office or need for an ezeio. In this case, their access and privileges' would extend to the subgroups below, that contain the ezeios in the individual stores.

A single ezeio controller can only be located in a single group, although it's data and some controls can be linked to dashboards widgets in groups higher up the branch of the group tree.

You may use the group structure to manage your controllers in any way that makes sense for your organization. If you only have a small number of ezeio's it may be sufficient to just have a single group, but as you add units you probably want to organize them in groups by location, client or function.

Groups can be moved within an account, where as the root level account is fixed. Since users and dashboards belong to the group/account), relocating the configuration from a root level account would require, recreating dashboards and inviting users to the new group. For this reason we strongly recommend creating at least one group below your account to set-up your ezeio controllers.

Accounts

Accounts are specially designated groups that isolate branches of group tree. The “Root level Account” separates your parent organization from all others on eze.io, like a bank account. Sub-accounts create divisions within your account, like checking, savings, and the kids collage fund. This allows account level services such as SMS messages, voice messages and API calls to be purchased and consumed by a selected group or branch. Where as groups provide organizational structure, accounts add security, privacy and exclusivity. If your organization/company provides products or services to other organizations sub-accounts allow you access to your clients, without allowing them access to each other.

Root level Account

A root level Account is automatically created by clicking the “*Don't have an account? Sign up here*” link. This process requires the serial number and registration code of an unregistered ezeio and email address not associated with a user. Most account holders on eze.io will only need to do this once, as additional controllers, users, groups and sub-accounts are added from within. The root account cannot be deleted.

If using an existing user's email to generate a new account, the system assumes you are a using the wrong process to add an ezeio to an existing account and redirects you to the sign in.

Sub-accounts

Any group can be promoted to an account, making it a sub-account of the root account or of another sub account. Users with the required privileges can “Right-Click” on a group and select “Make account” to start the promotion process. You will then be asked to enter the billing information for this account. After submitting the billing info, you are instructed to logout (and back in) for the changes to be visible. Click on the new account and view the Account settings under the “Group settings” tab. At the bottom you will see check boxes for allowing credit card or invoice payment. If these are left unchecked users invited to this account will not have the ability to purchase services or view pricing.

Separating account level services and destinations

Message credits (text and voice) and API calls are “Account level services”. This means, if you only have one account these services can be consumed by any group or ezeio. Creating sub-accounts allows services to be restricted to a specific branch.

Destinations and Destination lists are also accessible account wide. These are the contacts (end points) and groups of contacts for alarms messages within an account. These destinations can contain sensitive personal information such as cell phone numbers. At a minimum the “Destination lists” may be irrelevant to other groups in an account or lead to alarm messages sent to the wrong recipients.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/groupsettings

Last update: **2021-11-18 17:43**

Manage API keys

`Manage API keys` can be accessed by clicking at the side bar to the right of the Group Settings screen.

The user must have the following privileges enabled to access API key settings
Can edit group
Can access API keys

API access is a paid & metered service. API calls are metered per account. The service can be added and purchased under Service Settings.

Multiple keys can be active for the same account at the same time. All keys share the same service and are metered together against the allowed number of calls.

Please see API reference for details about how to make API calls, and what API features are available.

To add a new API key, click the (+) symbol in the header. The new key will be added to the list immediately.

Click the pencil symbol to view and edit the details. The trashbin symbol will delete the key.

Each key has the following settings:

API key name	A user-defined name
Privileges	Enables the key for accessing the various API functions. Each key can have any combination of privileges enabled.
Key expires	The key will automatically stop working at the given date.
Inherit to subgroups	If checked, the key will allow access to groups below the current group. If not checked, only data in the current group can be accessed
Callback URI	Some API functions will generate a call-back. If this feature is used, enter the desired URI here.
Key ID	This is the first part of the credentials needed to user the API call (=username)
Key	This is the second part of the credentials. (=password)

The Key ID and Key are auto-generated and can't be changed.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/groupsettings/apikeys

Last update: **2025-01-07 18:13**

Manage Users

A “user” is simply a person that access the ezeio system. Every user is primarily identified by his/her email address. Each user is assigned privileges suitable for the role of the user. For example, an administrator may be able to add new users, assign privileges and manage service payments, a technician may be able to change configuration of devices, while a read-only user may only be able to view data but have no ability to alter configuration or control.

Each person accessing the ezeio system should have a **personal** login.
NEVER SHARE YOUR PASSWORD

Each user will use a personal email address and a password to access the system. For added security there is also the option to use multi-factor authentication.

Select “strong” passcodes with a minimum of eight characters, digits and special characters. Do not use regular words or combinations that are easy to guess.

The list of users with access to a specific group/account is shown on the `Manage Users` screen. Access `Manage Users` from the bottom of the `Group Settings` screen.

Account creation and first user

The first user to register a controller for your organization will automatically receive all privileges. This will also automatically create a group and mark this group as the “Account”. Account groups are shows in bold in the group tree, and cannot be deleted or moved the same way a normal group can. Also, the Account must always have a user associated with it, which happens automatically when the account group is created.

Adding a new user

To add a new user, first select the group in the tree to the left. The new user will only be able to access the selected group and any groups *below* the group where the user is added.

Click the button `Invite User` at the bottom of the screen, and be sure to enter the correct email address, as well as set the desired privileges for the user.

The new user will receive an email with instructions on how to confirm the invitation. You will also receive an email when the user has confirmed the invite.

Forgot passcode

Should a user forget the passcode, simply use the “Forgot Password?” link on the log-in screen. This will ask the user to enter their email address, and send instructions to this address for how to reset the password.

If the user has also forgot the email address, we recommend contacting an administrator of your ezeio account.

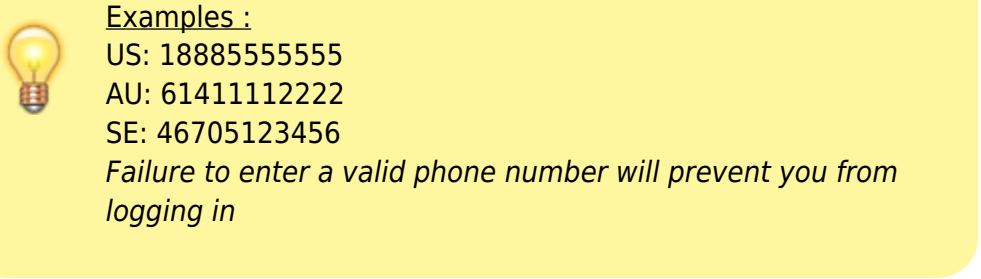
If you are and administrator and forget your email address, please contact eze System.

Note that passcodes are not stored in the ezeio system, and eze System can't extract any information about the passcode.

User settings

Each user has the following settings:

Email Address	This is the email address used to log in. It can not be changed.
First Name	User's first name
Last Name	User's surname
Phone	User's direct phone number. If Multi-factor log in is enables, this must be a cellphone capable of receiving SMS
Company	User's organization
Receive Communication Alerts	Select if this user should receive messages about expiring services and communication status messages
Multi-Factor log in method	Select if this user shall use SMS verification to log in
Privileges	The privileges assigned to this user (see below)



Privileges

The privilege settings controls what functions are available to each user, in each group.
Typically the privileges are inherited from the top group where the user has access, so that the user has the same privileges in all the groups. It is possible to add or remove privileges in sub-groups, but typically this is adding unnecessary complexity for the administrator. We recommend starting simple by leaving privileges inherited in all sub-groups.

Privileges

Can view group	Required to allow the user to see the group
Can view ezeio in group	Allows the user to view (not change) configuration settings
Can view other users	Allows user to see other users
Can view script	Allows viewing (not changing) script code
Can manage services	Allows adding/disabling and paying for services
Can send control commands	Allows the user to send control commands and change real-time field values
Can edit group	Required to change any setting in the group
Can edit ezeio in group	Allows the user to change configuration
Can edit other users	Allows the user to change settings for others and invite new users
Can edit script	Allows user to change the script code for controllers
Can edit dashboards and widgets	Allows user to add/change dashboards and widgets (if Dashboard settings allow editing)
Can add/change destinations	Allows user to add/change alarm destinations

If a privilege is not shown, it means you do not have access to it.

Note that some privileges have prerequisites. For example, a user with “Can add/change destinations” will not be able to do that unless the same user is also allowed the “Can edit group” privilege.

Please consider the effects of allowing privileges to each user, and only enable features to trusted users.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/groupsettings/users

Last update: **2023-11-21 16:48**

Groups (group tree)

The Groups sidebar occupies the left side of the Manage screen. It is visible unless full screen is selected for Dashboards. This feature allows the user to locate, search, and select groups (and accounts), through the use of the search field and group tree.

Search field

Search for an ezeio, a group, or an account, by entering a partial or whole name. Or enter a complete serial number to find a specific ezeio controller.

Serial Number

Entering an ezeio's serial number (with or without dash) will automatically select the group (or account) containing the ezeio (no need to press Enter). The group will be highlighted in the group tree. If the Systems tab is selected, the ezeio will be highlighted/selected in the controller table.

Group Name or ezeio Name

Entering a string of characters that is present in the group/ezeio you are looking for, will narrow the branches shown (in the group tree) to only those with groups that match the search. Pressing the Enter key at any point will bring up a dialog box with ezeio names that match the results. If only one match is found, the group containing that ezeio will be selected.

No Match

If no match is found, the group tree will be empty. In the case of a serial number search, the text “No match found” will appear below the group tree's header.

Group Tree

The group are like folders on your computer. The group tree shows them as branches cascading down from their parent group or account. The + and - icons indicate expandable/collapsible branches. Accounts are identified by the solid grey folder icons, where as group folder icon is a grey outline .

Adding groups

First select the parent group/account you would like your new group to reside under. Then click the green Add Group button at the bottom of the side bar. A dialog box with appear allowing you to enter a name for your new group and a description if you wish. Finish by clicking the Add Group button of the dialog box.

Moving groups

Groups can be dragged to a new location within an account or sub account structure. Groups cannot be

moved between accounts. Left click-hold and drag to the desired parent group. Release while hovering over the intended group and release. A dialog box will appear asking “Are you sure you want to move _ to _”. *The group will be positioned as a sub-group of the group referenced in the dialog box*. Click on **Move Group** to proceed.

Deleting groups

Groups can be delete if they contain no ezeio controllers. This is done from the Manage Group and Account panel under the Group Settings tab.

Any users that were invited to a group that has been deleted will have there access and privileges deleted. They will still have a user record, but if the deleted group was the only group they were invited to, they will have no access. If it is not your intention to remove the users access, we recommend inviting them to another group before deleting the group you want to remove.

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/grouptree

Last update: **2021-09-24 21:56**

Message Templates

Message templates controls the content of messages sent as Alarms/Restores or periodic exports.

A template can include static text as well as dynamic data using the 'tags' defined below.

Every *destination type* (email, SMS etc..) comes with a default message template.

For example, the default template for emails looks like this:

```
[ACTIONMESSAGE]
Message generated [ISODATE] [ISOTIME]
```

The default template for SMS messages looks like this:

```
[ACTIONMESSAGE] @ [ISOTIME]
```

Because SMS messages are typically limited in size, the default template contains less text and no additional line breaks.

When an alarm is generated, and the action is to send a message, the system will fetch the “Message” of the alarm action, and insert this in place of the [ACTIONMESSAGE] tag in the default template. Any tags in the Action Message are also substituted.

For example, the alarm's action message may look like this:

```
It is [FV<TEMPERATURE>] [FU<TEMPERATURE>] now!
```

When this is sent as an email using the default template, the result will be something like:

```
It is 25.6 °C now!
Message generated 2022-12-31 15:00:02
```

The templates for each destination can be changed directly under Destination List settings.

Message Templates Tags

The following tags are available in message templates:

Tag	Purpose	Example
[MESSAGEID]	A unique message id	*ABC123-2-14-7474*
[ZULUTIME]	Zulu/UTC time when message was triggered	*2019-12-31T23:59:52Z*
[ISODATE]	Date when message was triggered, in controller time zone, ISO format	*2019-12-31*

Tag	Purpose	Example
[USDATE]	Date when message was triggered, in controller time zone, US format	*12/31/19*
[ISOTIME]	Time when message was triggered, in controller time zone, ISO format	*23:59:59*
[USTIME]	Time when message was triggered, in controller time zone, US format	*11:59:59 PM*
[UNIXTIME]	Time when message was triggered, UTC, as seconds since 1970-01-01	*1676675802*
[UNIXTIMEMS]	Time when message was triggered, UTC, as milliseconds since 1970-01-01	*1676675802000*
[WEEKDAY]	Weekday when message was triggered, in controller time zone	*Mon .. Sun*
[MONTH]	Month when message was triggered, in controller time zone	*Jan .. Dec*
[DAY]	Day when message was triggered, in controller time zone	*31*
[EZEID]	ezeio serial number	*ABC-987*
[EZENAME]	ezeio name	From Configure→System→Name
[EZENAME#]	ezeio partial name, #=1-5	From Configure→System→Name, split by comma, semicolon or pipe
[EZENOTE]	ezeio note	From Configure→System→Note
[EZENOTE#]	ezeio note line, #=1-5	From Configure→System→Note, split by line
[TYPE]	Message type	*ALARM, REALARM, RESTORE, INFO, EXPORT*
[SOURCE]	Message source	*SEND, LOG, SCRIPT*
[GROUPNAME]	Name of the group where the ezeio is assigned	From Group Settings→Group Name
[GROUPNAME#]	Partial name of the group, #=1-5	From Group Settings→Group Name, split by comma, semicolon or pipe
[SOURCENAME]	Name of the source alarm	From Configure→Alarms→Name
[SOURCENAME#]	Partial name of the source alarm, #=1-5	From Configure→Alarms→Name, split by comma, semicolon or pipe
[ACTIONNAME]	Name of the action source action	From Configure→Alarms→Alarm action→Name
[ACTIONNAME#]	Partial name of the action source action, #=1-5	From Configure→Alarms→Alarm action→Name, split by comma, semicolon or pipe
[GPS]	GPS coordinates (if available)	*Lat:38.671447, Lng-121.152385:, Ele:105.2*
[GPSLINK]	GPS coordinates Google maps link	https://maps.google.com/maps?q=…
[GPSSIGNAL]	Signal quality of the GPS receiver	*14*
[RSSI]	Cellular signal RSSI	*0* (no signal) .. *31* (full signal)

Tag	Purpose	Example
[CSV]	Attach a CSV file with all field statuses (applies only to email destinations)	
[P#]	Message (#=1-4) parameters as integers (see script command “Event”)	
[PD#]	Message (#=1-4) parameters as floats (see script command “Event”)	
[FV#]	Field value, #=1-90 formatted	*318.2*
[FR#]	Field value, #=1-90 unformatted	*318.2*
[FU#]	Field unit, #=1-90	*psi*
[FN#]	Field name, #=1-90	*Pressure*
[FT#]	Field asset tag (first only), #=1-90	*PRESSURE*
[F#]	Field status, #=1-90, as “Field name: Value Unit”, same as [FN#]: [FV#] [FU#]	*Pressure: 318.2 psi*
[FL<*AssetTag*>]	Field list by asset tag. Lists all fields having the given asset tag, as “Field name: Value Unit”	*Pressure: 318.2 psi*, multiple lines
<*AssetTag*>	Can be used in place of the Field number, Example: [FV<outsidetemp>]	
<*AssetTag*:SUM>	Works with FV and FR. Returns the sum of all field values with the given AssetTag, Example: [FV<outsidetemp:SUM>]	*172.2*
<*AssetTag*:MAX>	Works with FV and FR. Returns the highest value of all field values with the given AssetTag, Example: [FV<outsidetemp:MAX>]	*45.6*
<*AssetTag*:MIN>	Works with FV and FR. Returns the lowest value of all field values with the given AssetTag, Example: [FV<outsidetemp:MIN>]	*38.2*
<*AssetTag*:AVG>	Works with FV and FR. Returns the mean (average) value of all field values with the given AssetTag, Example: [FV<outsidetemp:AVG>]	*43.02*
<*AssetTag*:BOR>	Works with FV and FR. Returns the binary OR value of all field values with the given AssetTag, Example: [FV<outsidetemp:BOR>]	*47*
<*AssetTag*:ALL>	Works with FV and FR. Returns all field values with the given AssetTag separated by comma, Example: [FV<outsidetemp:ALL>]	*38.2, 44.1, 44.3, 45.6*

Tag	Purpose	Example
[TEXT]	Event text (see script command “Event”)	
[ACTIONMESSAGE]	The message/template from the action (typically used in the destination list)	
[RECIPIENT]	The name of the destination (recipient) for this message (from Destination→Name)	
[DESTINATION]	The address of the destination (from Destination→Address)	
[DESTINATIONTOKEN]	The token of the destination (from Destination→Token)	
[DVC#NAME]	Device name, #=1-40	Name from Configure→Devices→Device→Name
[DVC#COMMSTAT]	Device communication status, #=1-40	*n/a, ERROR, WARNING, WARNING2, WARNING3, OK, OK2, OK3*
[DVC#OPSTAT]	Device operational status, #=1-40	*n/a, ERROR, WARNING, OK*
[DVC#APPSTAT]	Device application status, #=1-40	*n/a, ERROR, WARNING, OK, OK2, OK3, OK4, OK5*
[DVC#STATUS]	Device overall status, #=1-40	*OK* or if there's an issue *COMM:ERROR*
[DVCSTATUS]	List all devices overall status	*#1 Pressure sensor: OK* *#2 Energy meter: OP WARNING* *#3 Geiger counter: APP ERROR*
[DVCWARNINGS]	List all device warnings and errors (if any)	*#2 Energy meter: OP WARNING* *#3 Geiger counter: APP ERROR*
[DVCERRORS]	List all device errors (if any)	*#3 Geiger counter: APP ERROR*

Template automation

It is also possible to create simple loops to search for field asset tags using the special `[FORTAG tag delimiter]…[FORTAG]` syntax. This is best explained with an example of template code:

```
Fieldname,Value,Unit
[FORTAG MYTAG \n][FN],[FV],[FU][FORTAG]
```

The [FORTAG MYTAG \n] construct will iterate over all fields, selecting those that match the asset tag “MYTAG”. For each matched field, Field name (`[FN]`), Field value (`[FV]`) and Field unit (`[FU]`) is added, followed by the delimiter `\n` (new line).

Note that no field number is necessary for the Fx-tags, as the field number is automatic from the tag-search loop.

The output of the above will be in CSV format, thus something like this:

```
Fieldname,Value,Unit
Temperature,53.2,C
```

```
Pressure,12.1,psi
```

Within a FORTAG loop, the following tags are valid:

[FV]	Field value formatted	*318.2*
[FR]	Field value unformatted	*318.2*
[FU]	Field unit	*psi*
[FN]	Field name	*Pressure*
[FT]	Field asset tag (first only)	*PRESSURE*
[F]	Field status, as “Field name: Value Unit”, shorthand for [FN]: [FV] [FU]	

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/messagetemplates

Last update: **2025-06-30 22:27**

Service Settings

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/service

Last update: **2021-09-24 21:56**

Systems

Located on the “Manage” level of the eze.io user interface, the Systems tab is where the ezeio controllers reside within a group. From this tab you can add or remove controllers, view their status, graph historical data and access event logs.

Controller Table

This table lists the controllers assigned to the group with columns for serial number, name, notes, status, and the button linking to the configuration screen for each ezeio.

The information in the first three columns can be added to alarm messages by use of message template tags. Serial Number = [EZEID], Name = [EZENAME], User Note = [EZENOTE]

- **Serial Number** - This number is assigned by the manufacture and cannot be changed.
- **ezeio Name** - This name is user defined and edited through the ezeio's System settings under, the System tab in each controllers configuration.
- **User note** - This text is user defined and edited through the ezeio's System settings, under the System tab in each controllers configuration. This provides the opportunity to deliver additional information about the system, sensors or asset the ezeio is connected to. This information can also be sent as part of an alarm message using the Message Template Tag [EZENOTE].
- **ezeio Status** - Icons in this column indicate status of the connection, location, and synchronization with the server.

Controller specific data & group map

The lower half of the Systems page is where you can view, graph and download the data made available from a selected ezeio's Fields. The map tab shows the location of all controller in the group.

Status Tab

Clicking on the status tab (if not already shown by default) will reveal a table of exposed Fields. Columns from left to right are; 10 minute log graphing check box, Fast log graphing check box, Field name, View, Value, and unit.

- **10 minute log graphing check box** - All Fields are logged at 10 minute intervals by default. Check the box next to desired data point to include in graph.
- **Fast log graphing check box** - If configured for fast logging, the check box will be outlined in dark gray. Check the box next to desired data point to include in graph.
- **Name** - Driver or user defined name of Field.
- **View** - Is a alternative way of displaying a Field value.
- **Value** - This number can represent many things including a measurement, status, set point.
- **Unit** - Driver or user defined unit of measure.

Graph Tab

This feature provides an interactive visualization of logged data in line graph form. The user can specify the Fields (data channels), date range, and aggregation method.

Before clicking on the Graph tab, select the Fields you want to include in the graph by checking the associated box on the far left. If you have the optional “Fast Logging” set on any of your “Fields”, the associated checkbox in the second column from the left will be outlined in dark gray.

After selecting your Fields, click the Graph tab. The default view displays 10 days plus the current days readings.

Graph time stamps are based on the users browser time. The most resent log entry will be minutes prior to the time shown on your computer, tablet or phone.

Downloaded .csv files are always Coordinated Universal Time (UTC)

- **Zoom** - Click-Hold and Drag from upper left to lower right to zoom in on an area of the graph. Users can zoom in multiple times. Click the “circle back” in the upper right of the graph to return to the full date range.
- **Date range** - The current date range is shown in the lower left of the graph tool. Beside it is a calendar icon. Click here to bring up to the data range selector. Click on the right and left arrow to move backwards or forward month by month. Click on the calendars title to bring up a list of months, Click again to bring up a list of years. Once you have navigated to the correct starting and ending months, select first the start day of your desired range, then the end day. When your selection is made, click “Done” to retire the calendar, then click Update to apply the changes to the graph. The “Today” link can be used as a quick preset to narrow the graph down to the current day.

Event Log Tab

Map Tab

From:
https://doc.eze.io/ - **ezeio documentation**

Permanent link:
https://doc.eze.io/ezeio2/userinterface/manage/systems

Last update: **2023-07-19 00:13**