

Advanced eGauge Operation

Covers advanced operations, such as working with Alerts and other advanced software features

- [USB thumb-drive functionality](#)
 - [Automated USB Export and Upgrade scripting file](#)
 - [USB Data Exports through LCD](#)
 - [Formatting a USB stick on Windows 10](#)
- [Alerts](#)
 - [Alerts and Email Gateway with SendGrid](#)
 - [Configuring eGauge Alerts](#)
- [Zero out data/spikes tutorial](#)
- [eGauge language options and adding new languages](#)
- [High Gain Mode](#)
- [Connecting via the eGauge proxy server using IE 10/11](#)
- [Monitoring High Voltage Systems](#)
- [Formula registers and remote devices](#)
- [Register ID and location](#)
- [Working With Max Demand Export Data](#)
- [Monetary registers](#)
- [Lua Scripting](#)
 - [Lua Scripting Overview](#)
 - [Creating and Using Lua scripts](#)

USB thumb-drive functionality

eGauge Core/Pro have USB and thumb-drive storage functionality

Automated USB Export and Upgrade scripting file

Introduction

Starting with [firmware](#) v3.4, eGauge firmware supports USB Mass-Storage devices on hardware with USB ports (e.g., EG4015 (Core) and EG4030 (Pro)). Both Windows VFAT and Linux EXT3 filesystems are supported. When the eGauge device detects a valid storage device, it will automatically bring up a USB Storage menu on the LCD screen. From there, it is possible to select various operations, such as:

- [Creating a backup](#) of all eGauge data
- Exporting data to a CSV (comma-separated-values) file.
- Updating the device firmware.
- Ejecting (unmounting) the storage device so it can be removed safely.

For flexibility and to simplify operations when it is necessary to perform the same USB Storage task on multiple devices, the firmware also supports a script file called `auto.run`. When this file is present in the top-level directory (root folder) of the USB storage device, it will be executed as soon as the USB storage device is detected by the firmware. The details of how this file is executed are specified below.

Scripting Language

The `auto.run` file consists of a sequence of text lines, each line containing a command. Commands are executed in sequence, one by one. If a command fails for any reason, script execution stops and the most recent error is displayed on the LCD screen. The operator needs to acknowledge the error by pushing the multiswitch button and, once that is done, the USB Storage menu is displayed.

In the absence of errors, script execution stops upon reaching the end of the script file or after execution of an "eject" command (see below). After reaching the end of the script, the USB

storage menu is displayed on the LCD. After an "eject" command, the LCD screen reverts to normal operation. The USB storage device is then no longer accessible until the USB storage device is removed and reinserted into the USB port or the device is rebooted (e.g., power cycled or rebooted via firmware).

Script files may contain line-comments that start with a hash character (#). The hash character and any following characters up to the end of the line are ignored.

Script Commands

In the following descriptions, square brackets are used to indicate optional parts of a command. Italics is used as place-holders for variable content.

backup [FILENAME_TEMPLATE]

Backup the eGauge device data to the file specified by FILENAME_TEMPLATE. This string may contain references to variables that are expanded as follows:

Variable Reference	Expands to	Example
\${DEVNAME}	eGauge device name	eGauge1234
\${SN}	eGauge serial number	1701260002
\${TIME}	current date and time	20180411-1310

If FILENAME_TEMPLATE is omitted, the template:

backup-\${DEVNAME}-\${TIME}.bin

is used by default.

export [PERIOD [GRAN [FILENAME_TEMPLATE]]]

Export eGauge device data to the file specified by FILENAME_TEMPLATE. The data is saved in comma-separated-values (CSV) format. The file-name template may use the same variables as the

backup command. In addition, the following variables are supported:

Variable Reference	Expands to	Example
<code>\${PERIOD}</code>	export time-period	ytd
<code>\${GRAN}</code>	export granularity	hour

PERIOD specifies the time period of the data to be exported:

Period specified	Time range covered
day	Most recent 24 hours of data are exported.
month	Most recent 31 days of data are exported.
year	Most recent 366 days of data are exported.
all	All data is exported.
dtd	day-to-date: export data from midnight up to current time
mtd	month-to-date: export data from start of month up to current time
ytd	year-to-date: export data from start of year up to current time

GRAN specifies the granularity (resolution) with which the data is exported:

Granularity specified	Resolution
sec	Second resolution (or best available).
min	Minute resolution (or best available).
quarter	15-minute resolution (or best available).
hour	Hour resolution (or best available).
day	Day resolution.

Note that the eGauge device database internally uses varying granularity to balance storage requirements with the ability to retain data for long time periods. Specifically, second-by-second data is typically retained only for the most recent hour, then dropping to minute-granularity, until at last day-by-day data is available for the longest amount of time (e.g., 60 years). The granularity specified in the export command is the finest resolution to be used during the export. If the database doesn't have the data available in the desired resolution, it will automatically exported in the best available resolution.

Also worth mention is that day-granularity data is captured every day at midnight UTC. The date and time exported in the CSV file is converted to the device's time-zone, which means the hour seen in the exported data may be one or the other, depending on whether or not daylight savings is in effect. For example, in US/Mountain time-zone, day-granularity data is captured at 6pm

(18:00) during the summer months and at 5pm (17:00) during the winter months.

```
update_fw [[PATH] force]
```

Update the device firmware using the file specified by PATH. If PATH is omitted, the file fw.bin will be used.

The update will be performed only if the file contains a firmware newer than the one that is currently installed. However, if option "force" is specified after the file path, the update will be performed regardless of the version currently installed on the eGauge device.

After a successful update, the LCD screen will indicate that the device needs to be rebooted. The operator needs to confirm this by pushing in the multiswitch button and that point, the eGauge device reboots itself to activate the new firmware.

```
eject
```

This command ejects (unmounts) the USB storage device and stops script execution. Once the command has finished, the LCD screen returns to normal operation and it is safe to remove the USB storage device from the USB port at that point.

Example Script

In the following example, let's assume the device name is "eGauge1234" and the commands are all executed on April 1st, 2018, at 1:14pm (13:14).

```
# This is a comment. It has no effect on execution.
```

```
# OK, let's do a backup first:
```

```
backup # will be saved in "backup-eGauge1234-20180401-1314.bin"
```

```
# Export data to "data-eGauge1234-ytd-hour.csv":
```

```
export ytd hour data-${DEVNAME}-${PERIOD}-${GRAN}.csv
```

```
# Eject USB storage and return eGauge device to normal operation:
```

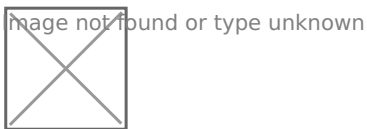
```
eject
```

USB Data Exports through LCD

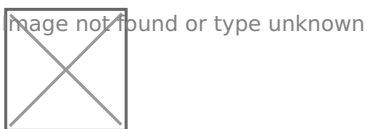
Data can be exported directly from eGauge 4xxx meters using a USB mass storage device. These exports automatically contain day-granular data. All data from epoch (the date set in "Date and Time Recording Started" under **Settings -> General Settings**) until the date and time the export was initiated is copied.

To select a specific granularity or time range (or otherwise automate USB exports), an auto.run file must be created and loaded on the USB mass storage device ([this article](#) has more information).

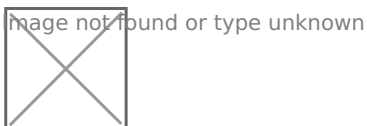
To begin the process, insert a mass storage device formatted as EXT3 or VFAT. The "Storage Menu" will appear. Menu navigation is performed using the multiswitch button. Moving the switch left or right will change the selected option. Pushing the multiswitch in will confirm the selection. "Export data" should be highlighted by default. To begin the export, push the multiswitch in:



Once the export has begun, the screen will read "export started". The export process takes a variable amount of time depending on the amount of data present on the eGauge.



Once the export has finished, the screen will read "export finished". The "OK" option will be highlighted. Push the multiswitch in to acknowledge.



After acknowledging the export, the "Storage Menu" will appear again. Move the switch left or right until "Exit & Eject" is highlighted. Push the multiswitch in to select this option.

Image not found or type unknown



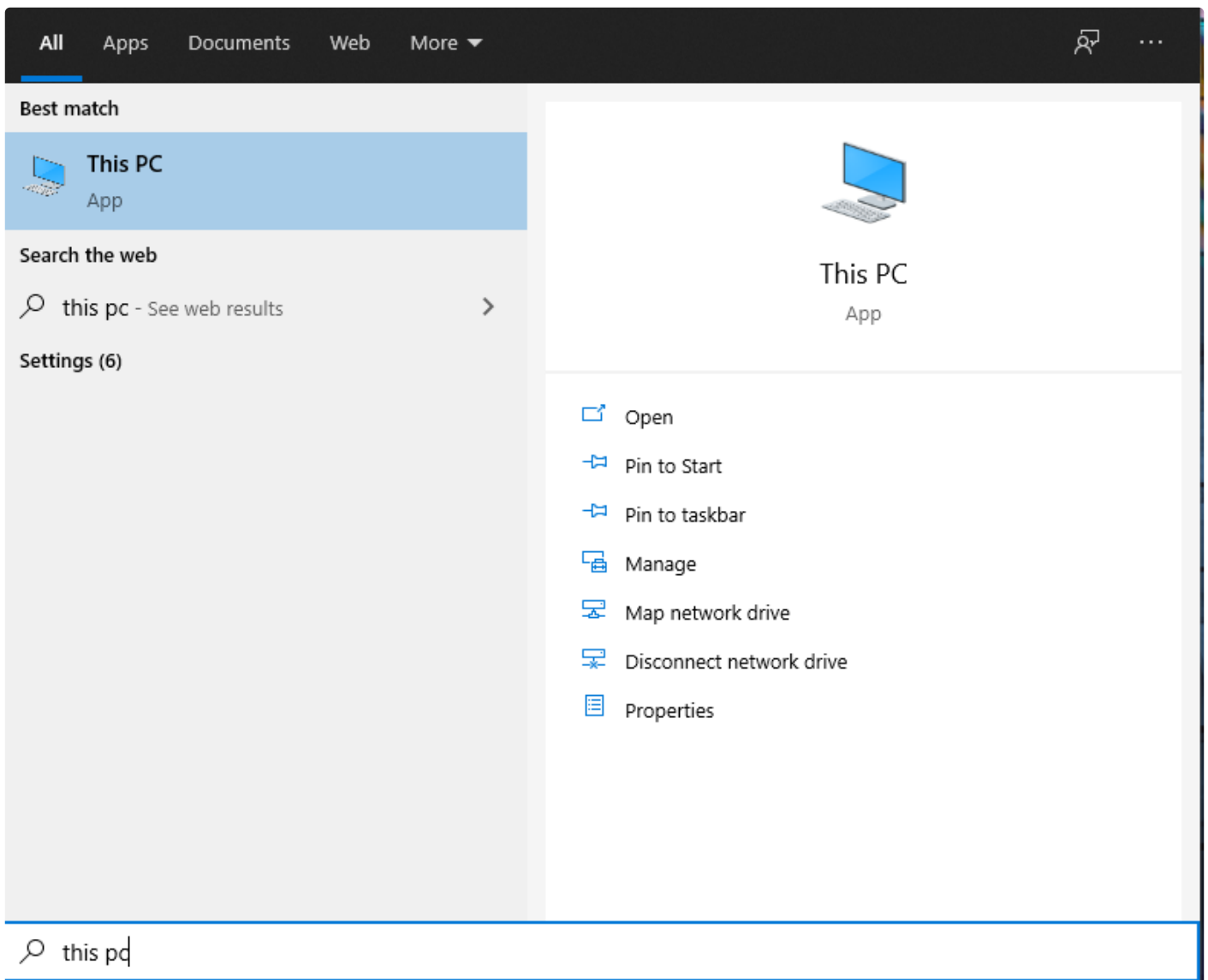
Once the main LCD menu appears, the USB mass storage device can be safely removed.

Formatting a USB stick on Windows 10

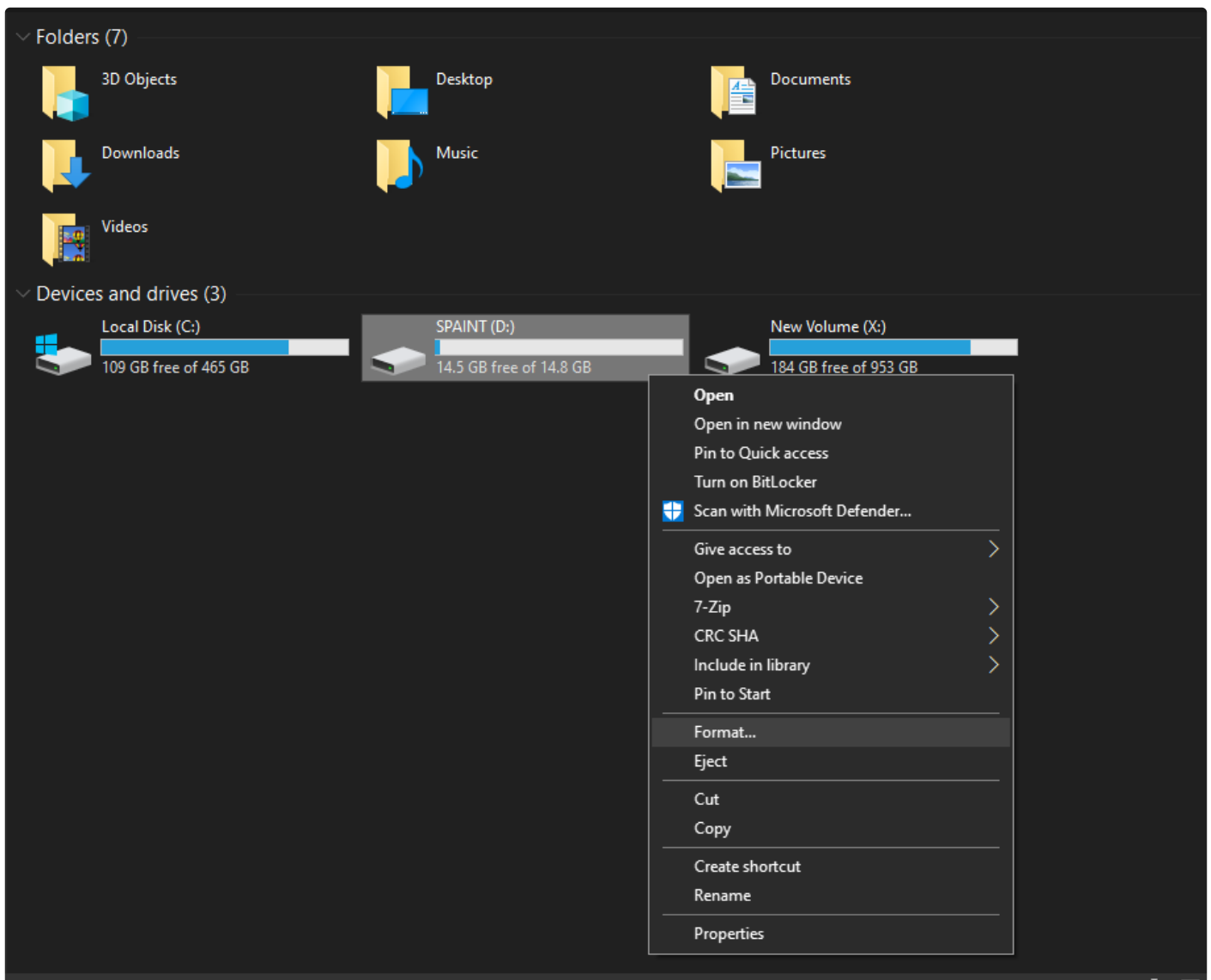
These instructions may not work for all versions of Windows and are provided as-is. eGauge Support cannot provide additional troubleshooting or support on formatting USB sticks. An adequate understanding of using the Windows operating system and file explorer may be necessary to complete these instructions.

This **will permanently erase all data** on the selected drive or USB stick. Incorrect usage can cause permanent data loss on the wrong drive.

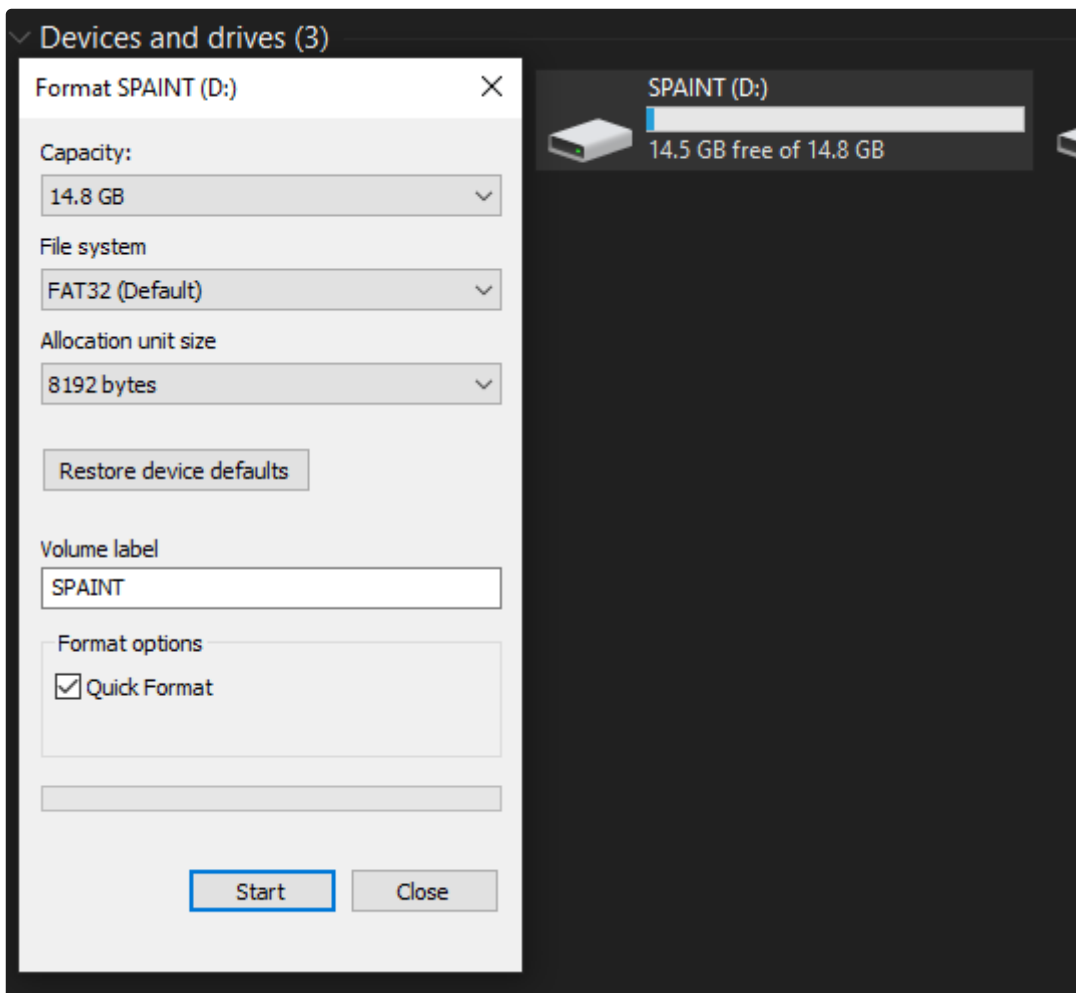
- 1) Connect a USB stick to a PC running the latest version of Windows 10.
- 2) Click on the Start Menu and type in "this pc" and open the "This PC" app.



3) In the "This PC" window, right-click the USB stick and choose "Format..."

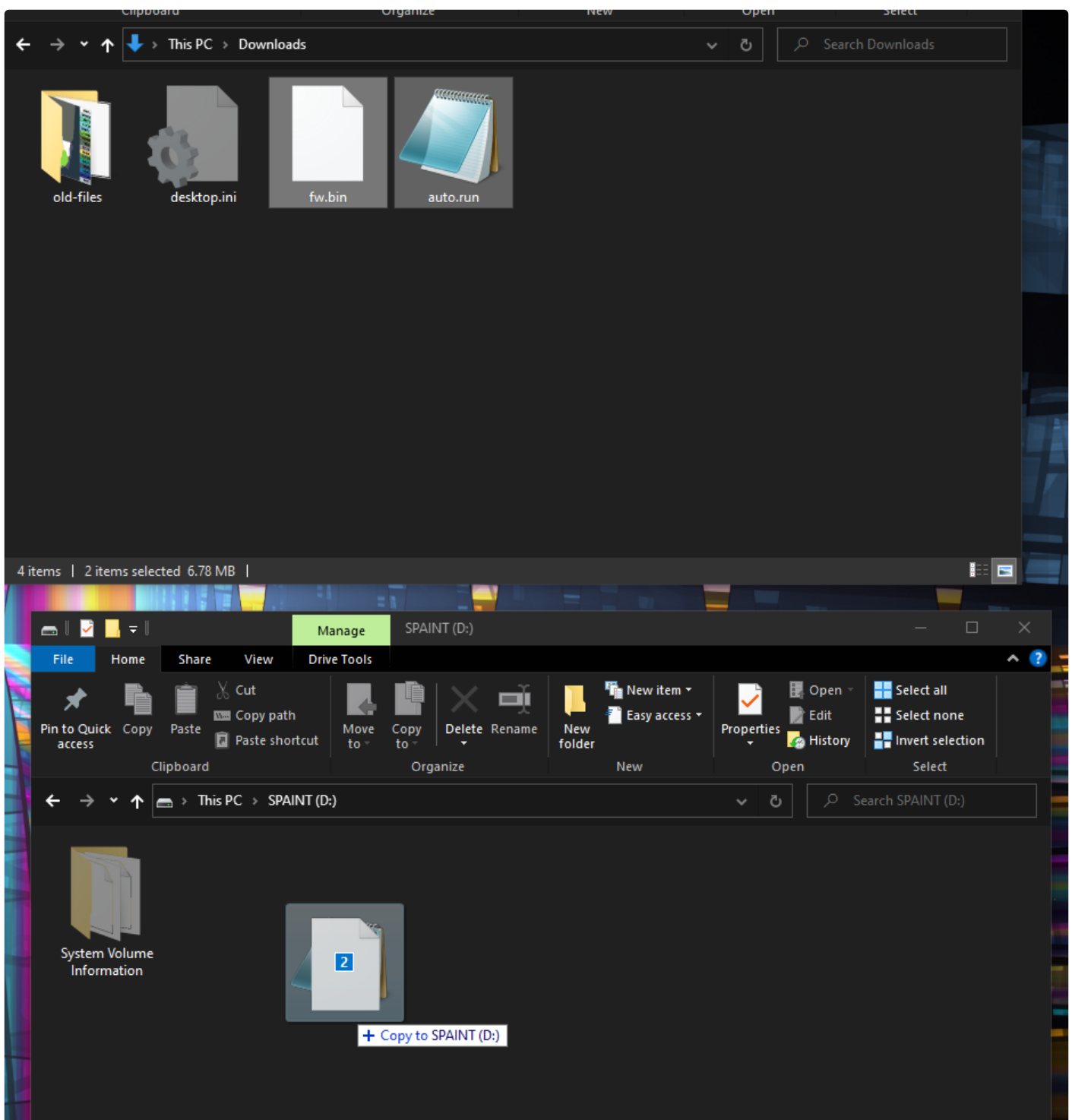


4) Ensure the "File system" is set to FAT32, and "Quick Format" is checked. The "Volume label" does not matter. Press "Start".

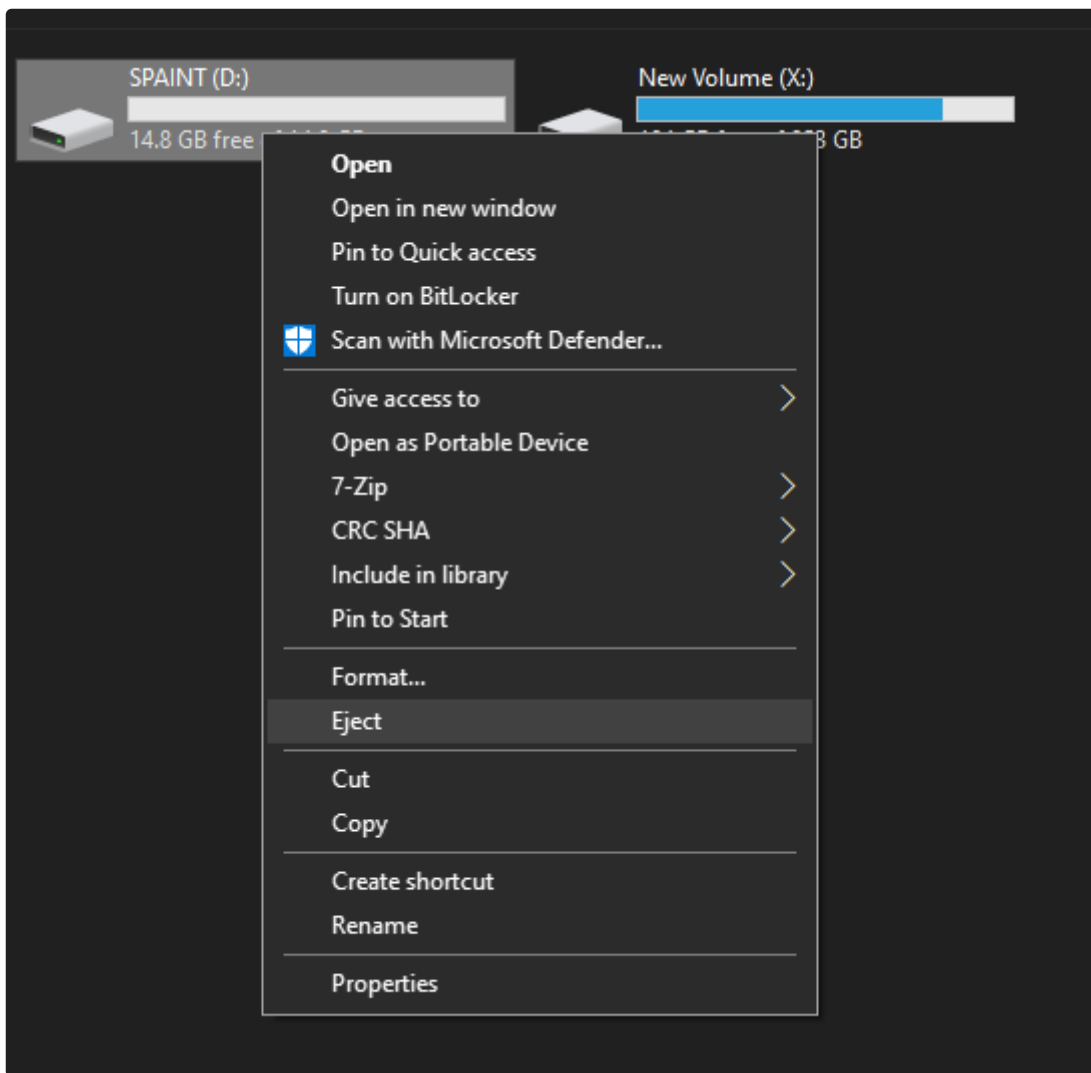


5) When complete, the USB stick is ready for use with an eGauge.

If copying files such as auto.run and fw.bin, open another explorer window to where the files are downloaded on your computer, and open the USB drive. Select the file(s) and drag the files to the explorer window of the USB stick.



6) After the files are copied, click the "back" button on the USB stick window, right-click the USB stick drive and choose "Eject".



7) Remove the USB stick. It is now ready for use with the eGauge meter.

Alerts

All about eGauge Alert functionality

Alerts and Email Gateway with SendGrid

Introduction

If delivering email to only one recipient, consider using the [eGuard Alert Service](#) to send device alerts as it is the easiest method for delivering email alerts.

You will need your own website's email address to send email from; Gmail, Yahoo, AOL and other public email services may not work with this method.

Compatibility Notice: Beginning in June 2023, only EG4xxx will be compatible with the SendGrid SMTP gateway. Legacy meters such as EG30xx and eGauge2 will fail to send alerts via the SendGrid SMTP gateway.

SendGrid is a free third party service unaffiliated with eGauge which provides a consistent outbound email delivery service. It can be used for eGauge alert delivery, and is more reliable than not specifying an "Email Gateway". Without an "Email Gateway" configured, alert emails may be rejected, dropped, or fail inconsistently.

It is intended as a commercial service for businesses rather than individual end users.

SendGrid allows **up to 100 emails daily** with the **free plan**.

SendGrid provides delivery information and monitoring such as email delivery failures, number of emails sent and other features.

Disclaimer

SendGrid is a third party email delivery service with no affiliation with eGauge Systems. eGauge Systems cannot guarantee email delivery, uptime, or security of using SendGrid.

Information on this page will be updated on best effort level. Information including pricing, set up instructions, screenshots, and locations of items are not guaranteed to be up to date or consistent.

eGauge Systems does not provide any support for SendGrid's services, and cannot assist with account creation, password recovery, or similar issues.

Pre-setup notes

[Ensure the eGauge meter used is on firmware v4.0 or greater, click here for information on checking and upgrading firmware.](#)

Email can be delivered from the eGauge meter using SMTP server `smtp.sendgrid.com` and a sendgrid API key.

If using alerts on multiple devices, it is advised to set up a unique API key for each one, so that if the key becomes compromised or the meter becomes inaccessible and alerts cannot be disabled, the API key can be revoked for only that meter.

API keys are passwords and are displayed only once after creation.

Setup

1. Set up API keys:

[Initial account creation and setup wizard for first email](#)

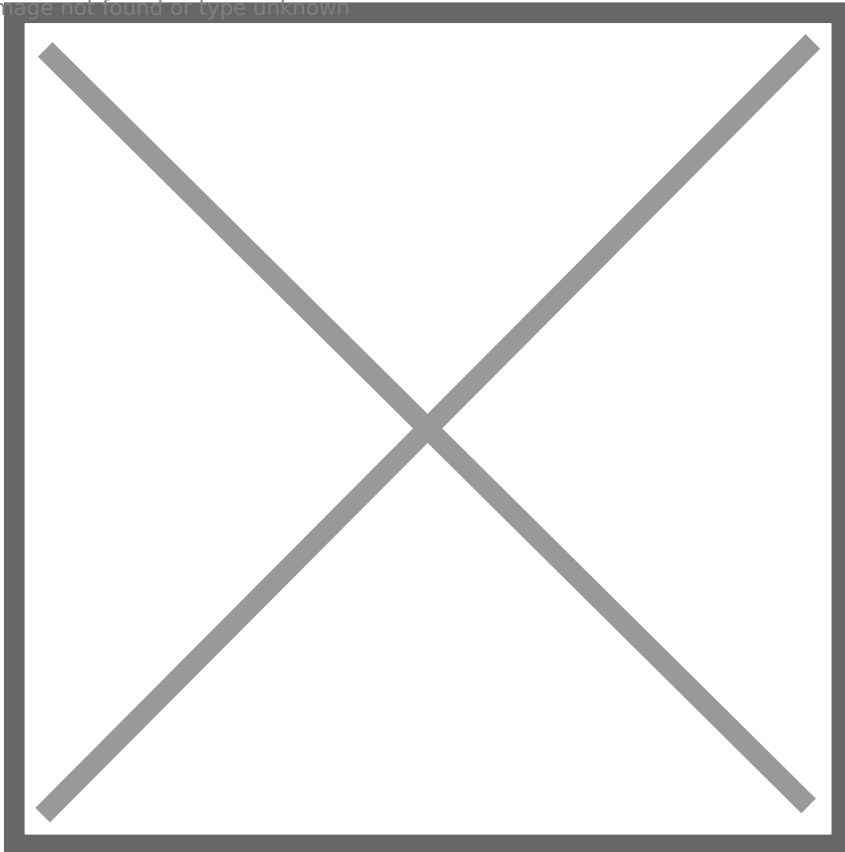
or

[Adding a second API key, or adding without the wizard from the main dashboard](#)

2. [Verify a sender identity](#)
3. [Configure the eGauge meter](#)

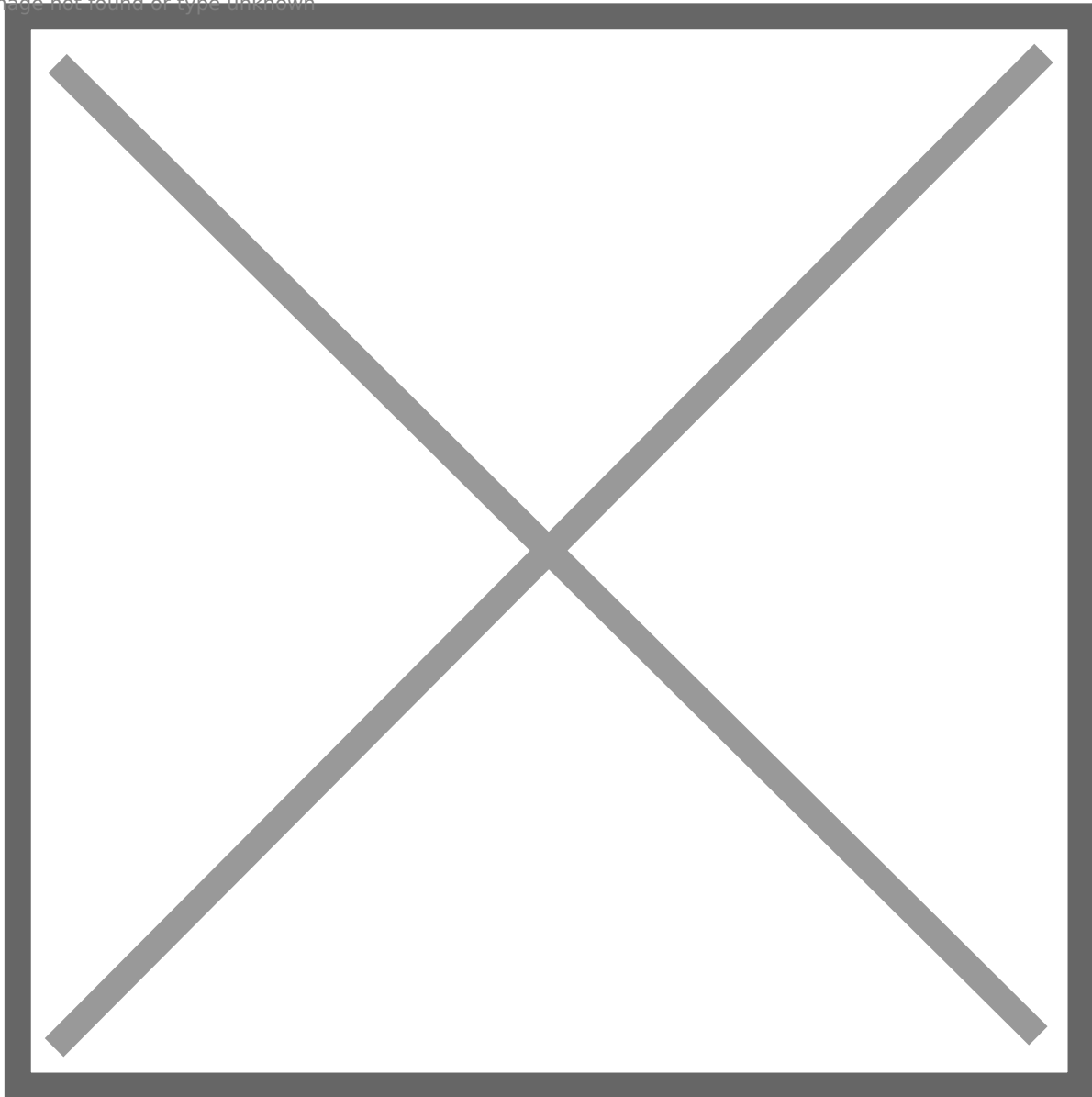
Initial account creation and setup wizard for first email

1. Create an account at <https://sendgrid.com/>
2. Click "Start" to the right of "Integrate using our Web API or SMTP relay"



3. In "Choose a setup method", choose "SMTP relay"

Image not found or type unknown



4. Give the API key a name, like "eGauge Alerts Emails" and press "Create Key"



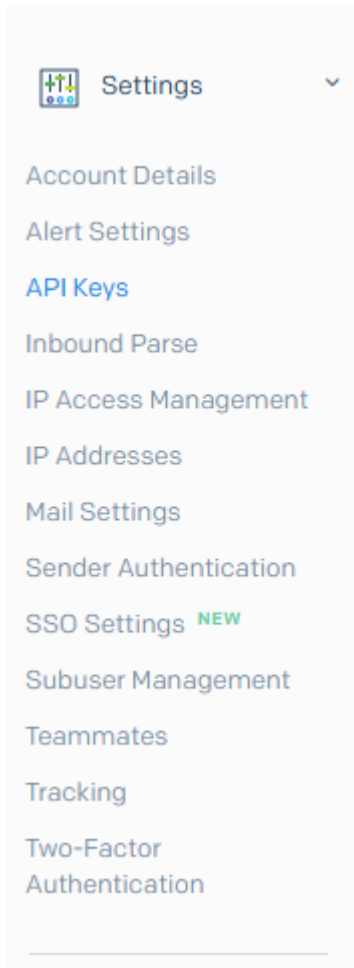
5. You will now be given the SMTP server (smtp.sendgrid.net), the username (apikey), and password (hidden). This is the only time the password will be displayed, it should be saved somewhere securely like an encrypted keychain if it will be used more than once. Keep this page open or copy the password as it will be used in a later step.



Image not found or type unknown

Adding a second API key, or adding without the wizard from the main dashboard

1. In the main dashboard on the left-side menubar, expand Settings and click API keys.



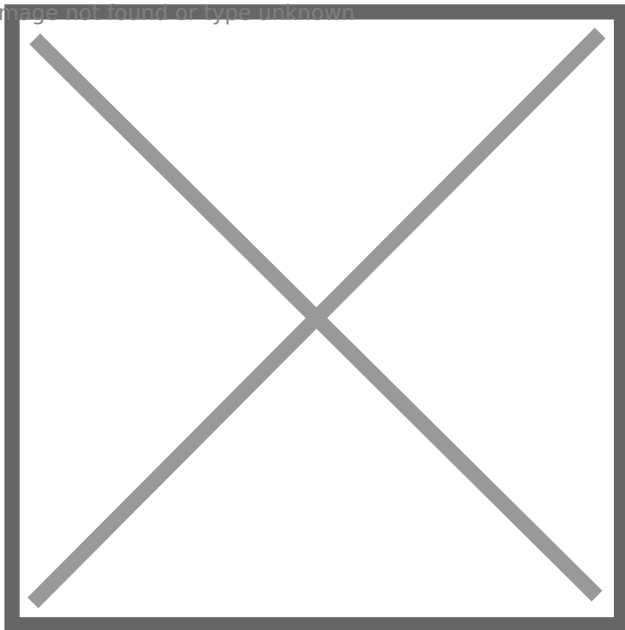
2. Click "Create API Key" in the upper right-hand corner:

API Keys

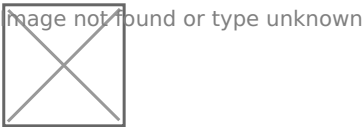
Create API Key

NAME	API KEY	ACTION
[REDACTED]	[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]	[REDACTED]

3. Choose "Restricted Access" as the API key permissions, expand the "Mail Send" section, and click the dot on the right side of the bar next to "Mail Send" to grant that permission



4. Click "Create and View", and copy the API key displayed on the next screen. This will be used to configure the eGauge meter in a later step.



Verify Sender Identity

You will need to set up Sender Authentication so the receiving email server trusts and accepts the email the eGauge meter delivers. Follow the instructions for [Single Sender Authentication](#) (simple: sends an email link to verify email address) or [Domain Authentication](#) (advanced: requires modifying DNS entries on your website).

Without sender authentication, you will see an error such as `The from address does not match a verified Sender Identity` when sending a test email without verification completed. It will contain a link with information to set up sender authentication.

The email address you verify will be used in the `Custom "From" address` setting in the eGauge meter alert settings.

Configure the eGauge Meter

1. Navigate to Settings -> Alerts, ensure "Alert Provider" is set to "SMTP Gateway" and click on "View/Edit Gateway & Alert Destinations". Enter the mail server hostname (`smtp.sendgrid.com`), username (`apikey`), and the API key / password that was created previously. For the Custom "From" Address, enter an email identity verified in the previous step.

SMTP Gateway Settings

Email Gateway

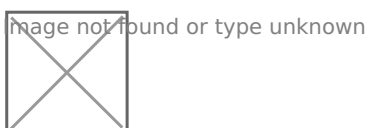
Hostname of mail server [?]	<input type="text" value="smtp.sendgrid.com"/>
Username for mail server [?]	<input type="text" value="apikey"/>
Password for mail server [?]	<input type="password" value="....."/>
Custom "From" address [?]	<input type="text" value="my-verified-sender@mydomain"/>

2. Enter the email address(es) that should receive alerts from the meter under Alert Destinations.

Alert Destinations

Message Format [?]	Email address or phone number [?]	Min. Alert Prio [?]	
Email ▾	email-to-be-notified@theirdomain	0 ▾	Send Test Message
Email ▾		0 ▾	Send Test Message
Email ▾		0 ▾	Send Test Message
Email ▾		0 ▾	Send Test Message

3. Press "Save" at the bottom of the page, and then "Send Test Message" to the right of each Alert Destination to ensure the delivery works.



Configuring eGauge Alerts

Overview

The eGauge can be configured to send alerts based on a variety of trigger conditions. Alerts must be configured through the eGauge interface, and the eGauge needs to be powered on and connected to the internet in order to send alerts. There are three possible alert destinations: SMTP (email or SMS-capable phone numbers via an email-to-SMS gateway if the cellular provider supports), the eGuard alert service, or a custom URI for a JSON POST (advanced users).

SMTP emails credentials may be supplied, and the eGauge will use this email account to generate alerts. Some services such as Gmail may restrict logins to browsers, or disallow the login if devices in different locations are attempting to all log in to send alerts. For large larger deployments a [service such as SendGrid](#) may be used.

The [eGuard Alert Service](#) is a more simplified email alert delivery service and only requires you have an eGauge.net account.

The following article covers basic alert configuration and provides some sample alerts. Additional information is available on an eGauge-specific basis by navigating to <http://DEVNAME.egaug.es/fundoc.html?alert> where **DEVNAME** is the [device name](#) of your specific eGauge. To take advantage of all alert features, the eGauge should be on the [latest firmware](#).

For meters shipped after January 1, 2024 this information may be found at: <http://DEVNAME.egaug.io/fundoc.html?alert>

Contents

[Alert Basics](#)

[Configuring the Alert Service Provider](#)

[SMTP Gateway](#)

[Using SendGrid](#)

[eGuard Alert Service](#)

[Custom Alert Destinations](#)

[Configuring Alerts](#)

[System Alerts](#)

[User Defined Alerts](#)

[Viewing and Acknowledging Alerts](#)

[User-defined Alert Examples](#)

[Example and description of POST data](#)

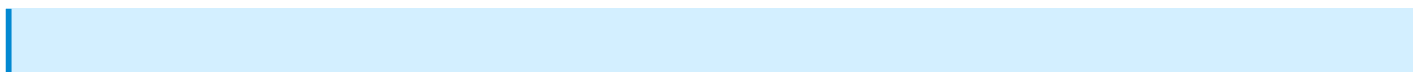
Alert Basics

Alerts may be configured from the **Settings → Alerts** page and viewed from **View → Alerts**. There are two types of alerts: system alerts and user-defined alerts. System alerts can report conditions such as when the device configuration is changed or when the connection to a remote device has been established. User-defined alerts are built arbitrary conditions that, when true, trigger the alert. For example, you could define an alert that triggers when solar production for a period is below a certain threshold value, or an alert that triggers when the register monitoring Oven usage has been above a certain value for a certain time. More examples are **available here**.

Configuring the Alert Service Provider

Choose the View/Edit Gateway & Alert Destinations button from the top of the Alerts page to configure how alerts are sent. The page will request credentials in order to make changes if none have been previously cached.

SMTP Gateway



Legacy meters (eGauge2 and EG30xx series) support TLS 1.1, while newer meters (such as EG4xxx) support TLS 1.2.

Alert Service Provider

Alert Provider [?](#) 

SMTP Gateway Settings

Email Gateway

Hostname of mail server [?](#)

Username for mail server [?](#)

Password for mail server [?](#)

Custom "From" address [?](#)

The SMTP Gateway Alert Service Provider allows the eGauge to send alerts directly to email addresses, SMS-enabled phones, or a mixture of the two. This functionality requires an internet connection, but *does not* require the eGauge to be connected to the proxy server at d.egauge.net. The following fields are required:

Hostname of mail server

Normally, eGauge attempts to deliver email directly to the destination address. Similarly, it attempts to deliver SMS directly to an SMS-gateway. However, if a firewall prevents the device from directly establishing such connections, as is commonly the case for consumer-grade Internet-service, you will have to set the value of this setting to the hostname of a mail server which can forward the messages to the final destination. The mail server may either be a host on the same LAN (e.g., within a company or school network) that will accept email delivery without authentication or it may be an external mail server where you have a valid user account. By specifying the username and password for that account, the device is then able to deliver email through that mail server (ie, the alert messages from the eGauge will originate from your username on that mail server). As an example, if you have a Gmail account, you can set the hostname to smtp.gmail.com. By specifying your Google account's username and password, you can then have alerts delivered via Gmail.

Username for mail server

When non-empty, this setting specifies the username the device uses to authenticate itself to the mail server. If empty, mail is delivered without authentication or encryption. Note that this option is required for almost all mail servers.

Password for mail server

This setting specifies the password the device uses to authenticate itself to the mail server. It is used only if Username is not empty.

Caution: on legacy meters (eGauge2 and EG30xx series) the password is transmitted to the eGauge over an unencrypted channel. Only change this password from a computer that's connected to the same LAN as the eGauge and only after clicking on the **LAN Access** link on the eGauge main page. As an added security measure, create a dedicate email account at the mail server for sending eGauge alerts.

Using SendGrid

Compatibility Notice: Beginning in June 2023, only EG4xxx will be compatible with the SendGrid SMTP gateway. Legacy meters such as EG30xx and eGauge2 will fail to send alerts via the SendGrid SMTP gateway.

SendGrid credentials are entered in the SMTP Alert Service Provider fields. For more information on using SendGrid or a similar service, please refer to [this article](#).

Setting Alert Destinations

Alert Destinations

Message Format [?]	Email address or phone number [?]	Min. Alert Prio [?]	
Email ▼		0 ▼	Send Test Message
SMS to AT&T phone ▼		0 ▼	Send Test Message
Email ▼		0 ▼	Send Test Message
Email ▼		0 ▼	Send Test Message

Message Format: select the appropriate SMS carrier or email format.

Email address or phone number: enter the appropriate destination for the alert.

Min. Alert Prio (Minimum Alert Priority): minimum level of alerts this destination should receive (see below).

Up to four alert-destinations can be defined. Alerts are prioritized. For each alert-destination, a **minimum priority** can be defined. Only alerts whose priority is equal to or greater than the minimum priority are reported to an alert-destination. Once an alert-destination has been notified, only alerts of higher priority result in a new notification to that destination until the alert has been acknowledged or deleted via the alerts page, or after 24 hours have passed.

eGuard Alert Service

Legacy meters (eGauge2 and EG30xx) require **HTTPS certificate validation to be disabled** to activate the eGuard Alert Service. This is due to a bug with an older SSL library

used on legacy meters and the eGauge.net certificate provider. This should only be used if the alert information being sent is not sensitive.

Alert Service Provider

Alert Provider [?](#)
Minimum priority to report [?](#)

The eGuard alert service provides an alternative to configuring the eGauge with SMTP credentials. This is especially useful for users with a large number of devices. The meter must be in an eGuard group controlled by the user. More information on eGuard is [available here](#). Also note that eGuard features built-in alerts - those are covered in [this article](#).

To use the eGuard Alert Service, simply select "eGuard Alert Service" and click the "Activate" button. A new window will open, and you will be prompted to log in to eGuard. Once logged in (or if you are already logged in), eGuard will confirm you want to register this device for alerts. Click "Register for Alerts" to confirm.

Register for Alerts

Click the button below to register device alerts for [REDACTED] with this account, [REDACTED]

Minimum priority to report: Setting this to a value other than zero will omit any alerts with a priority set lower than that value. This can be useful when certain alerts are not required (eg, set all system alerts to zero, set minimum priority to report to 1, then set all user-defined alerts to 1).

Custom Alert Destinations

Alert Service Provider

Alert Provider [?](#) URI: Options:
Minimum priority to report [?](#)

Custom alerts may be utilized by advanced users to send JSON-formatted data as a POST to a user-provided URI (alert destination).

Alert Provider: Must be set to "custom".

URI: The URI to send the JSON POST to. Should be unique to the device, such as with a GET token to uniquely identify the device making the POST.

Options: A comma separated list of options available below:

- deflate: Use "deflate" content-encoding when posting alerts.
- gzip: Use "gzip" content-encoding when posting alerts.
- secure: For HTTPS connections, fail if the alert provider server's certificate cannot be verified as being valid.

Do not use multiple compression schema, i.e., do not use gzip AND deflate on the same device.

Minimum priority to report: All alerts with a priority level equal to or greater than this will be POSTed to the URI when triggered. To prevent some or all system alerts from being reported, this may be set to "1" or greater. When alerts below the minimum priority level are triggered, they are only logged on the device locally and do not create a POST.

An example of the JSON post contents is [available here](#).

Configuring Alerts

Alerts are reported with a delay of approximately 30 seconds and are automatically acknowledged 24 hours after reporting them. These rules ensure you will be promptly informed of any alert conditions for a device without a deluge of SMS or email messages. Alerts of higher priority are reported even if there are pending alerts of a lower priority.

There are two types of alerts: System alerts and User-defined alerts

System Alerts

System alerts are predefined but you can choose the priority with which they are reported. This allows control over which recipients receives which system alerts (if any) and which alerts are more important. To set an alert priority, use the dropdowns in the "Prio" column. If there are certain system-alerts that you do not wish to have reported at all, select priority 0 and ensure that all alert destinations have a minimum alert priority of at least 1. Note that alerts with a priority of 0 will still be logged on the Alerts page, but no notifications will be sent for those alerts if all alert destination priorities are higher than 1.

System Alert Priorities

Name	Prio [?]
Proxy-connection established	0 ▼
Proxy-connection lost	0 ▼
Device-configuration changed	0 ▼
Date and/or Time changed	0 ▼
Device running hot	0 ▼
Device temperature OK	0 ▼
Remote-device connected	0 ▼
Remote-device lost	0 ▼
Failed to push data	0 ▼
Device up and running	0 ▼
Device rebooted by firmware	0 ▼
Network interface changed	0 ▼
Database error	0 ▼
Web server down	0 ▼
Web server up	0 ▼
Remote device fault	0 ▼
Remote device fault cleared	0 ▼
Failed to connect to server	0 ▼

Proxy-connection established/lost: tracks when a connection to the proxy server at d.egauge.net is opened or closed. If this occurs frequently it can indicate an unstable network connection.

Device-configuration changed: reports when a device's configuration is changed, and which account has made the modification.

Date and/or Time changed: reports when the device date or time is changed (either by the user or automatically).

Device running hot: reports if the eGauge's internal temperature reaches a significantly high temperature.

Device temperature OK: reports when the eGauge's temperature returns to a safe level.

Remote-device connected: tracks when a connection is established to a remote device (including Modbus devices and remote eGauges).

Remote-device lost: tracks when a connection to a remote device is lost (including Modbus devices and remote eGauges).

Failed to push data: reports if a data push is set, and the eGauge is unable to successfully push data.

Device up and running/Device rebooted by firmware: tracks when the meter is rebooted, and when the meter comes back online from a reboot or power outage.

Network interface changed: tracks when the meter switches from an Ethernet (ETH) to HomePlug (PLC) connection. This may happen immediately after a reboot and can generally be ignored.

Database error: typically reports when the device configuration is changed. The occasional database error is considered normal, but if this alert triggers multiple times per day and no configuration changes are being made it may indicate an issue. If this happens, contact eGauge support at support@egauge.net.

Web server down/Web server up: reports when the eGauge's internal webserver stops and starts. This will generally happen as the result of a reboot, and may happen as a normal occurrence during regular operation (for example, watchdog resets). If this alert triggers multiple times per day for several days it may indicate an issue.

Remote device fault/Remote device fault cleared: reports when Sunspec fault codes trigger on a remote Modbus device, and when those fault codes are cleared.

Failed to connect to server: reports when certain outbound connections fail, typically alert POSTs when using a custom alert destination.

User Defined Alerts

User-defined alert patterns allow the flexible detection and reporting of various conditions. For example, an alert could be defined which, on a second-by-second basis, checks whether a register value is outside of its permitted range (e.g., whether a voltage or frequency is above or below a certain threshold).

User-defined Alerts

Name ?	Prio ?	Trigger Condition ?
<input type="text"/>	<input type="text" value="0"/>	<input type="text"/> < <input type="text"/>
Chk Freq ? Daily	Msg ?	
<input type="text"/>	<input type="text" value="0"/>	<input type="text"/> < <input type="text"/>
Chk Freq ? Daily	Msg ?	
<input type="text"/>	<input type="text" value="0"/>	<input type="text"/> < <input type="text"/>
Chk Freq ? Daily	Msg ?	

The alert fields are described below:

Name: The name of the alert. This should be short but informative enough to convey the nature of the alert.

Trigger Condition: The trigger condition consists of three parts: left-hand-side (lhs), comparison operator, and right-hand-side (rhs). The comparison operator may be one of less-than (<), less-or-equal (<=), equal (=), not-equal (!=), greater-or-equal (>=), or greater-than (>). The lhs is compared to the rhs based on this operator and, if true, the alert is triggered.

Chk Freq (Check Frequency): select the frequency with which the trigger condition is to be checked. eGauge evaluates all alert conditions whenever the device starts up and hence may

evaluate the conditions more frequently than requested. Apart from the first time a condition is checked on start up, hourly conditions are evaluated during the first minute of each hour, daily conditions during the first hour after midnight, weekly conditions during the first hour of Sunday, monthly conditions during the first hour of the first day of the month, and annual conditions during the first hour of the first day of the year. "Every second" conditions are evaluated each second, "Every minute" conditions once a minute.

Choose the lowest check frequency possible as evaluating too many conditions too often may slow down the device. If a slow-running condition (eg, a condition using the `peak_risk()` function) is evaluated, evaluation of other conditions may be delayed until the evaluation of that condition is completed.

Msg (Message): use this field to define a custom-message to be displayed along with the alert name. If left empty, a default message is included which shows the value of the lhs, the operator, and the rhs of the trigger-condition. A well-written message will explain the alert - for example, on a "Low Production" alert the message might be "Caution: Low production on Inverter 1 (north side)". The placeholders `%l` and `%r` can be used in the message field to include the calculated value for the lhs and rhs of the equation.

Examples of user-defined alerts are available in the [User-defined Alert Examples](#) section near the end of this document. Click [here](#) to jump to that section.

Viewing and Acknowledging Alerts

You can view and acknowledge alerts on your device under **View → Alerts**. By default, a list of triggered alerts will be visible. For more information on each alert and the option to acknowledge or delete an alert, click the "View Privileged Details" button.

Pending Alerts:

Ack	Prio	Time	#	Name	Last Reported
<input type="checkbox"/>	0	08/21/20 08:42am	1	Network interface changed	not yet reported
<input type="checkbox"/>	0	08/21/20 08:42am	1	Proxy-connection established	not yet reported
<input type="checkbox"/>	0	08/21/20 08:42am	1	Device up and running	not yet reported
<input type="checkbox"/>	0	08/21/20 08:42am	1	Network interface changed	not yet reported
<input type="checkbox"/>	0	08/21/20 08:42am	1	Device rebooted by firmware	not yet reported
<input type="checkbox"/>	0	08/21/20 08:42am	2	Device-configuration changed	not yet reported
<input type="checkbox"/>	0	08/21/20 08:42am	1	Database error	not yet reported

[Refresh](#)[View Privileged Details](#)

Ack (Acknowledged): indicates if this alert has been acknowledged. Once acknowledged, the alert will be reported again should it reoccur and its priority is sufficiently high. Alerts are automatically acknowledged after 24 hours. To ensure new alerts are reported, alerts should be acknowledged when they are received.

Prio (Priority): the priority of the corresponding alert.

Time: date and time of the most recent occurrence of the alert.

#: number of times this alert has occurred (note that this isn't necessarily the number of times the alert has occurred since the device was installed).

Name: name of the alert.

Last Reported: date and time when the alert was last reported to at least one of the alert-destinations.

To view detailed alert information as well as acknowledge and delete reported alerts, click the "View Privileged Details" button. Valid credentials are required to see this information and acknowledge/delete alerts.

Pending Alerts:

<input type="checkbox"/>	Ack	Prio	Time	#	Name	Last Reported	Detail
<input type="checkbox"/>	<input type="checkbox"/>	0	08/25/20 01:43pm	5	test	not yet reported	1086.86
<input type="checkbox"/>	<input type="checkbox"/>	0	08/21/20 08:42am	1	Network interface changed	not yet reported	Switched from eth0 to qca0.
<input type="checkbox"/>	<input type="checkbox"/>	0	08/21/20 08:42am	1	Proxy-connection established	not yet reported	
<input type="checkbox"/>	<input type="checkbox"/>	0	08/21/20 08:42am	1	Device up and running	not yet reported	software reset
<input type="checkbox"/>	<input type="checkbox"/>	0	08/21/20 08:42am	1	Network interface changed	not yet reported	Switched from none to eth0.
<input type="checkbox"/>	<input type="checkbox"/>	0	08/21/20 08:42am	1	Device rebooted by firmware	not yet reported	Web-initiated reboot
<input type="checkbox"/>	<input type="checkbox"/>	0	08/21/20 08:42am	1	Database error	not yet reported	short read from epoch.dat (136 < 152)

[Refresh](#)[Acknowledge Checked Alerts](#)[Delete Checked Alerts](#)[Edit Alert Settings](#)

The **Detail** column contains additional information about each alert. For example, one instance of the "Network interface changed" alert provides the additional detail that the network interface was changed from "none" to "eth0" (this happens immediately after a reboot) while the other (newer) instance of the "Network interface changed" alert provides the additional detail that the network interface changed from eth0 to qca0. This happened very quickly (within the same minute), and is normal behavior for a meter coming back online after a reboot.

To modify alerts, check off any alerts you wish to delete or acknowledge, and click the appropriate button. Deleting alerts here will remove them from the reported alert page until it occurs again.

User-defined Alert Examples

For available functions on your particular firmware version, visit <http://DEVNAME/fundoc.html?alert> where DEVNAME is your eGauge device name.

General Notes

`$(REG NAME)` returns the instantaneous value of the register REG NAME, while `"REG NAME"` points a function at a specific register (but doesn't necessarily return the register's value). When using functions such as `avg()` or others listed in the function documentation, **do not** include the dollar sign.

When creating a message (Msg), there are several shortcuts which can be used to include values from the alert condition itself:

- %l will return the value of the left-hand-side
- %L will return the formula of the left-hand-side
- %r will return the value of the right-hand-side
- %R will return the formula of the right-hand-side
- %% will return a single percent sign (eg, %l %% would read as <value from left side of the comparison> %)

Basic Examples

In the following example, "Grid Average" will return the daily average of the Grid register if that value is less than or equal to 5000W, while "Grid Instantaneous Usage" will return the instantaneous reading of the Grid register if that value is less than or equal to 1000W.

User-defined Alerts

Name ?	Prio ?	Trigger Condition ?
Grid Average	1	avg("Grid",1440) <= 5000
Chk Freq ? Every minute	Msg ? %l	
Grid Instantaneous Usage	1	\$"Grid" <= 1000
Chk Freq ? Every minute	Msg ? %l	

The next example will trigger if the value of the L1 voltage register is greater than or equal to 130V (which could indicate a dangerous condition for devices connected to that service).

User-defined Alerts

Name ?	Prio ?	Trigger Condition ?
High Voltage	1	\$"L1 Voltage" >= 130
Chk Freq ? Every second	Msg ? Danger: high voltage on L1 at %l V.	

More complex math can also be performed on either side of the alert expression. For example, the following alert obtains the average voltage from two references, and triggers if that value is greater than or equal to 130V.

User-defined Alerts

Name ?	Prio ?	Trigger Condition ?
High Average Voltage	1	(\$"L1 Voltage" + \$"L2 Voltage") / 2 >= 130
Chk Freq ? Every second	Msg ? Danger: high average voltage at %l V.	

It's also possible to calculate cumulative values (kWh) over a period and trigger an alert based on those values. In the following example, let's assume an outdoor hot tub has a 6kW pump/heater, which cycles every 3 hours for 30 minutes at a time. Thus, every 6 hours there should be 6 kWh of energy used. Any less could indicate a pump or heater failure, and the hot tub could freeze.

(avg("Hot Tub Pump/Heat",360)*6) / 1000 will take the average power (W) read on the register Hot Tub Pump/Heat over the last 360 minutes (60 minutes in an hour, 6 hours). Then, the average power is

multiplied by 6 hours to get Wh, the total energy used over the 6 hour period. Finally, we divide by 1000 to convert Wh to kWh.

That value is then compared to the value on the right-hand-side, in this case 6. If the alert is triggered the alert will be sent.

User-defined Alerts

Name ?	Prio ?	Trigger Condition ?
Low Hot Tub Usage	1	(avg("Hot Tub Pump/Heat",360) * 6) / 1000
Chk Freq ? Every minute	Msg ?	Warning: Hot Tub usage over the past 6 hours is only %l kWh!

Ternary operator

The syntax of the “?” ternary operator (also referred to as a conditional or conditional test) is `condition?value_if_true:value_if_false` and can be nested. This is a fundamental component of many alerts, especially more complex alerts.

Boolean expressions

Simple boolean expressions may be used within an alert:

`(5 > 4)` will return 1. Conversely, `(5 < 4)` will return 0.

The boolean value can be multiplied by another value (including a register value). For example, `($"Grid" < 7000) * $"Grid"` returns the value of “Grid” if “Grid” is greater than 7000 W, and returns 0 if the value of "Grid" is less than 7000W.

To break this down: if `($"Grid" < 7000)` is true, it will return a 1. `1 * $"Grid"` returns the value for the "Grid" register. If `($"Grid" < 7000)` is false, it will return a 0. `0 * $"Grid"` is 0. Remember, `$"REGNAME"` returns the instantaneous value of the register.

Let's look at how the `time()` function can be used with the ternary operator and boolean expressions to trigger an alert at a specific time:

User-defined Alerts

Name ?	Prio ?	Trigger Condition ?
Daytime Usage High	1	(time() > 8) * (time() < 18) ? \$"Grid" : 0
Chk Freq ? Every minute	Msg ?	Warning: Grid usage is %l W
Nighttime Usage High	1	(time() > 18) * (time() < 8) ? \$"Grid" : 0
Chk Freq ? Every minute	Msg ?	Warning: Grid usage is %l W

These two alerts work together to trigger if the Grid value is greater than 5000W during daytime hours and greater than 3000 during nighttime hours.

time() returns the current time as a number from 0 up to (but not including) 24 with minutes as a fractional value. For example, 11:30am would be 11.5. We use two booleans here:

```
time() > 8 * time() < 18
```

If the time is > 8 (8am) and less than 18 (6pm), the booleans work out to 1 * 1 or 1.

If either boolean is false, the output from the booleans is 0. 0 * 1 or 1 * 0 both equal 0.

This gives us a ternary expression of either `1 ? $"Grid" : 0` or `0 ? $"Grid" : 0` (remember ternary expressions work out as `condition?value_if_true:value_if_false`). Thus, if the booleans evaluate to 1, the left side of the formula returns the value of \$"Grid". If the booleans evaluate to 0, the left side of the formula returns 0.

Moving on to the alert expression: if the booleans work out to zero (ie, if the time range is not correct), the left side of the alert returns 0. This can never be greater than 5000, so the alert never triggers. If the booleans work out to 1, the left side of the alert returns the value of the "Grid" register. If the value of the "Grid" register is >= 5000, the alert triggers.

Example and description of POST data

```
{
  "now": "1568419537.35",
  "alerts": [
    {
      "id": 1804290019,
      "priority": 7,
      "occurrences": 12,
      "first_occurrence": 4462.5,
      "last_occurrence": 389.31,
      "name": "Device-configuration changed",
      "detail": "By owner."
    },
    {
      "id": 1804290035,
```

```
"priority": 0,  
"occurrences": 1,  
"first_occurrence": 0,  
"last_occurrence": 0,  
"name": "Device rebooted by firmware",  
"detail": "Howdy do?"  
}  
]  
}
```

Generic

- "now" is a 64-bit UNIX timestamp, possibly with a fractional (sub-second) part, formatted as a decimal integer string.
- "alerts" is a list of reported alerts:
 - "id" is a number that uniquely identifies an alert. It is used, for example, to acknowledge or clear an alert.
 - "priority" is the user-assigned priority level of the alert (0 being the lowest priority and 7 the highest priority).
 - "occurrences" gives a count of how many times the alert has occurred since it was last cleared.
 - "first_occurrence" and "last_occurrence" specify how many seconds ago the alert occurred for the first time and the last time, respectively, relative to "now". That is, the specified number should be subtracted from "now" to get the absolute UNIX timestamp of when the alert occurred first and last, respectively.
 - "name" is the name of the alert that occurred.
 - "detail" provides additional detail on the alert that occurred.

Zero out data/spikes tutorial

Devices on firmware older than [version 3.01](#) may experience a phenomenon where register configuration changes that require a reboot cause unusually high readings to appear in the minute following the change. These readings are referred to as spikes, since they tend to "spike" far above the actual device readings. Spikes are almost always at least 100x the expected reading, and may be even higher.

Since spikes can cause cumulative readings to display incorrectly, the eGauge has a tool that can **permanently clear data** from the device database. Extreme caution should be exercised when using this tool. Once data is erased, it cannot be recovered. The video tutorial below provides a walkthrough of the tool's functionality.

Note that this tool may be used in other situations where clearing data is necessary (for example, removing bad data due to a configuration issue). However, this tool will **always permanently clear data** when used.

<https://www.youtube.com/embed/moh7krPBW58?wmode=opaque>

eGauge language options and adding new languages

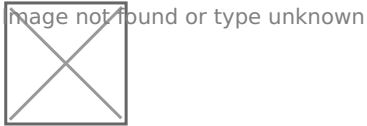
The EG30xx and EG4xxx have support for different locales. The eGauge will use the language associated with the locale that is requested by the user's browser. If the locale does not exist or is not supported, it will default to English.

New locales and improvements upon existing ones are welcome. Please see our internationalization (I18N) page at <https://www.egauge.net/i18n/> for full details. This page also shows the completion percentage of any supported language. Anyone is welcome to download and improve upon the PO files and submit them to i18n-team@egauge.net for review and implementation.

A specific locale can be forced to display in the browser by changing the browser's language/locale setting. Check your browser's help menu for information on changing your its language settings.

High Gain Mode

High-gain mode is available on the EG4xxx hardware only. To enable high-gain mode, navigate to **Settings -> Installation**. Check the "Use high-gain mode" option just above the dropdown menus for CT selection.



High-gain mode selection box

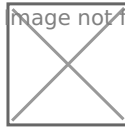
High-gain mode amplifies the incoming signal from all CTs by a factor of ten. From a practical standpoint, this means a CT with a higher amperage rating can be used to accurately measure much lower amperages. For example, with high-gain mode enabled a 100A CT could be used in place of a 10A CT.

High-gain mode does not require any changes to standard installation or wiring procedure. When high-gain mode is enabled, the CT amperages displayed in the drop down menus will decrease by a factor of ten (for example, an 800A CT will be listed as an 80A CT). For the simplest configuration, it is recommended to set actual CT amperages first and then enable high-gain mode. Note that high-gain mode is a global setting - when enabled, it applies to all configured CTs. As with most other installation settings, enabling high-gain mode will only impact data recorded from that point moving forward.

Connecting via the eGauge proxy server using IE 10/11

Users with Internet Explorer 10 and 11 may require some additional steps to function properly with the eGauge proxy server. These steps are covered in the document below.

image not found or type unknown



[IE 10/11 PROXY SETTINGS](#)

Monitoring High Voltage Systems

Warning: Systems greater than 600V phase-to-phase are not fully supported by eGauge Systems. Supported CTs are rated for up to 600Vac, systems at higher voltages will need to source alternate CTs.

Amperage-output CTs (e.g., 5A output, or 100:5 ratio, etc) can cause serious shock or electrocution. Use appropriate protection when installing and handling equipment.

Amperage-output CTs must not be connected directly to the eGauge meter and may cause damage.

Monitoring High Voltage Systems

First, a disclaimer: although it's technically possible to monitor a high voltage system (greater than 480Vac phase to phase or 277Vac phase to neutral, or 600Vac phase to phase using EV1000 sensors), eGauge Systems doesn't officially support this and doesn't offer hardware capable of doing so directly. eGauge support can assist with eGauge configuration, but cannot offer any guarantees regarding accuracy. Note that a wiring mixup during installation can damage or destroy the eGauge - this damage is **not covered under warranty**.

In order to calculate real power, the eGauge meter needs two measurements - a voltage reference and an amperage reference (technically the meter also captures amperage and voltage waveforms, which lets the meter calculate power factor and thus real power).

Voltage Reference

eGauge Systems cannot guarantee meter accuracy when third party potential transformers are used. It is recommended to use the [EV1000 high voltage sensor](#) when measuring a system with higher voltage than the eGauge meter rating, or on a different side of a transformer from where the eGauge voltage taps are connected.

In most installations, the eGauge can tap directly into the line voltage at the site (up to 277Vac Ph-N). Connecting the eGauge directly to higher voltages will damage or destroy the meter. However, it is possible to use stepdown transformers to reduce higher voltages to something the eGauge can safely read. These stepdown transformers are referred to as Potential Transformers (PTs) or Instrument Transformers. Most will take a common service voltage (eg, 480V, 600V, 4160V) and

reduce it to 120Vac.

With the release of the [EV1000 High Voltage Sensor](#), it's now possible to monitor high voltage services without the use of bulky stepdown transformers. The EV1000 can be used to measure up to 707Vac, making it ideal for 600V services. Note that the EV1000 is only supported on EG4xxx and newer meters - the EV1000 cannot be used with EG30xx and older meters.

Amperage Reference

To obtain an amperage measurement, the eGauge uses Current Transformers (CTs) with a 333mV output. These CTs clamp directly around conductors, and as such are rated for the voltage used by those conductors (a 600V rating is typically adequate for most installation scenarios). 333mV output CTs are the only CTs which can be connected directly to the eGauge. Connecting any other CT output type (eg, 4-20mA, 5A) will **damage or destroy the meter**.

Necessary Hardware

eGauge Systems currently stocks and supports one PT ([Functional Devices TR50VA008](#), 480V to 120V) and several CT lines rated for use on up to 600V services. The [EV1000 High Voltage Sensor](#) can be used on measurements up to 707Vac (for example, the phase to neutral voltage on a 347/600V service is 347V, so the EV1000 would work well here). For a service at a higher voltage, the customer will need to identify and purchase:

1. Potential Transformers or Instrument Transformers which can take a higher voltage down to <277V (preferably 120V). The transformer should have a relatively low VA rating to minimize phase shift (we suggest <=50VA, although up to 100VA is acceptable). Stepdown transformers intended for *high accuracy measurements* are generally referred to as *Instrument Transformers*. The transformer may require a larger enclosure or additional mounting hardware as well - they tend to be bulky and heavy. The total number of conductors on a service will determine the number of transformers required. A three phase four wire (wye, with neutral) service needs three transformers; a three phase three wire (delta, no neutral) needs two transformers.

2. A CT which can be installed around conductors carrying the service voltage (or overrated depending on code and installation requirements). The CT should either have 333mV output or 5A output (more on that below). Solid core CTs are recommended, although rope CTs are acceptable.

There aren't many (if any) 333mV output CTs rated for use on a 600V+ systems. The 5A output profile is much more common. A 5A output CT **cannot be connected directly to the eGauge**. However, it is possible to read from a 5A output CT using one of a supported CTs. Essentially, the output from the 5A CT is shorted (wires connected together). One of the standard 333mV output CTs is clipped around this new loop, and the eGauge measures the output current of the CT (0-5A). A custom scale factor is used to for the eGauge to treat this as a reading from the larger CT. In a similar manner, a custom scale factor is set on the line voltage readings to force the eGauge to treat the 120Vac as the voltage on the primary side of the transformer.

eGauge support can assist with this part of the device configuration, but cannot offer specific CT or transformer suggestions.

Summary

It is technically possible to measure power used/generated on services over 600Vac (phase to phase), although not officially supported. The customer will need to find suitable hardware (CTs and PTs). eGauge support can offer advice on the suitability of a given piece of hardware but cannot provide specific hardware recommendations. eGauge support can also assist with configuration and testing, although it's not possible to guarantee any specific level of accuracy.

Formula registers and remote devices

Physical registers represent any single data point recorded by an eGauge meter. These can come from locally-obtained raw measurements (eg, [amperage](#) or [voltage](#)), locally obtained calculations (eg, real power and other [power register subtypes](#)), and even registers imported from remote devices ([remote eGauge](#) or [third party devices](#) connected via Modbus TCP/RTU).

Formula registers are a type of physical register which can be used for a variety of calculations. Typically, a formula register will read data from one or more physical registers on a device, perform some calculation, and then store a new value. This is commonly used for advanced calculations such as [power factor](#) or [reactive power](#).

However, when using a formula register with a remote register, some additional consideration must be taken to avoid incorrect values when the remote register is not available (eg, the remote device has been disconnected from the network, dropped offline, etc). When this happens, the remote register may return a "NaN" value (Not a Number) instead of a numerical value, which can "break" the formula register. Consider the simple example below:

Remote and Local Power	⊗	=	▼	Power [W]	▼	["\$Local_Power" + "\$Remote_Power"]
------------------------	---	---	---	-----------	---	--------------------------------------

In this example, a formula register is used to add together the physical registers "Local_Power" and "Remote_Power". "Local_Power" is a power register measured by the meter itself, while "Remote_Power" is a power register imported via Modbus from an external source.

Normally, this is simple addition: if "Local_Power" = 10 and "Remote_Power" = 20, the value of "Remote and Local Power" would be 30. However, if the remote device which provides a value for "Remote_Power" goes offline, the eGauge is now reading 10 + NaN, which returns NaN. This behavior isn't desirable - it would be much better to at least return the value of "Local_Power" (10).

To achieve this, we can use a **conditional** and a **function** to convert any returned "NaN" into a 0.

A **conditional** is expressed as X?Y:Z, where **X** is something that evaluates to True or False, **Y** is the value returned if **X** is true, and **Z** is the value returned if **X** is false. For example, (3=4)?0:1 would return 1, because 3=4 is false (3 does not equal 4).

The **function** isnan() takes one value in the parenthesis and returns a "True" if the value is NaN and "False" if the value is a number. For example, isnan(1) would return "False", because 1 is **not** a NaN. isnan(sqrt(-1)) would return "True", because sqrt(-1) (square root of -1) isn't a number (and is

therefore NaN).

As you can see, `isnan()` will return either a "True" or "False". We can feed that into our conditional and dictate which value is returned. The basic form for this is:

```
isnan($"registername")?0:$"registername"
```

If the value of the register "registername" is a NaN, `isnan()` returns "True" and the conditional returns 0. If the value of the register "registername" is **not** a NaN, `isnan()` returns "False" and the conditional returns the value of "registername".

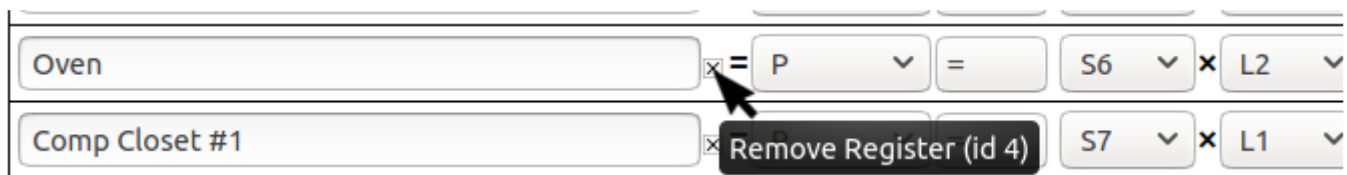
Using our original example:

Remote and Local Power	⊗	=	▼	Power [W]	▼	(\$"Local_Power" + isnan(\$"Remote_Power")?0:\$"Remote_Power")
------------------------	---	---	---	-----------	---	--

Register ID and location

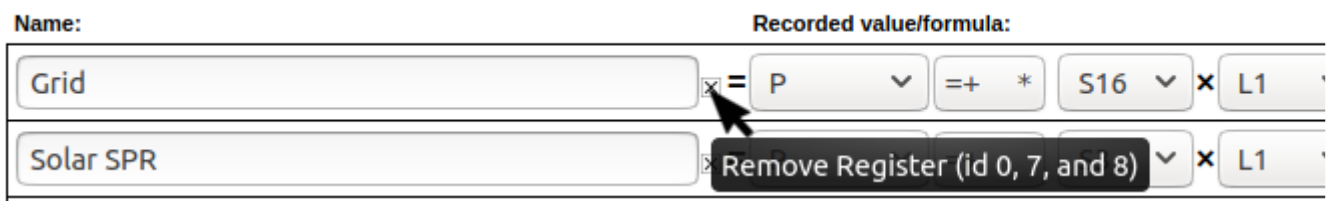
The eGauge uses "registers" to record data. Each register is an independent data point. For example, one register may be recording the voltage of L1, and another register can be recording the power of CT1*L1. Data is only recorded and stored if a register is configured to record it.

Each register has an ID which can be found in the Settings -> Installation page by hovering the mouse over the [x] delete button to the right of the register name:



This shows register "Oven" is recording in ID #4.

If there are multiple power sub-types selected, multiple IDs will appear in order:



Grid is recording net power (=), positive-only (+) and apparent power (*). They are recording in ID #0, #7, and #8 respectively.

Adding a new register will use the first available unused ID, so if register ID #0 is removed and a new register is added, the new register will record in ID #0. Historical data is not erased when a register is removed, so historical data from ID #0 will always show in ID #0 even if the register is deleted and a new register records data in it.

To permanently erase data from a register, you can use the Zero-out data tool in the Tools menu.

Working With Max Demand Export Data

Overview

As of [firmware version 3.1.10](#), the eGauge supports rolling max demand exports through the [CSV export function](#) built into the main graph page. These exports contain a rolling average over the past X minutes with a data point presented at the interval requested when creating the export.

Many utilities bill based on rolling max demand averages in addition to total kWh consumption. While it's possible to obtain a rolling max demand average using minute-granular data from a meter, the process is involved and the data overhead is quite large (data for every minute over the period must be downloaded, then the user must calculate a set of rolling averages for each interval, and record the maximum average for an interval). For example, a rolling max demand export over the past 30 days would return 2880 data points (one point every fifteen minutes), while a minute-granular export for the same period would contain 43200 data points (one point every minute).

eGauge Systems cannot assist with utility reconciliation or questions about the billing process used by a specific utility. You'll need to reach out to your utility for assistance with this.

Verifying Settings

The max demand export calculates a rolling average which is X minutes long. X is defined by the value selected for "Length of a demand interval" under **Settings -> Preferences**.

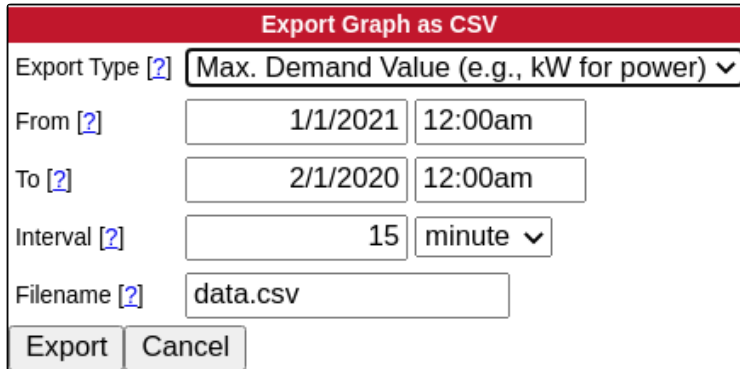
Length of a demand interval.

15 ▼ Minutes

This value can be adjusted in one minute increments, from 15 to 60 minutes. Your utility should make their demand interval public (although it may be necessary to contact the utility directly to get this information). eGauge Systems cannot advise as to the correct value to select for this option.

Performing An Export

To perform a max demand export, use the dropdown menu in the top left corner of the main graph page. The export window should look something like the following image. Note that the export type is set to "Max. Demand Value" and the interval is set to 15 minutes.



Export Graph as CSV	
Export Type [?]	Max. Demand Value (e.g., kW for power) ▼
From [?]	1/1/2021 12:00am
To [?]	2/1/2020 12:00am
Interval [?]	15 minute ▼
Filename [?]	data.csv
<button>Export</button> <button>Cancel</button>	

The export interval setting **is not the same as the length of a demand interval setting**. The former determines how often a data point is returned (eg, one data point for every 15 minute period) while the latter determines the length of the demand interval (eg, how many data points are included when calculating an average).

In other words, a single data point in a max demand export can be described as "the average peak demand over the past N minutes for the time period from XX:XX to YY:YY with a data point every Z minutes", where N is the "Length of a demand interval" setting, XX:XX is the starting time of the export, YY:YY is the ending time of the export, and Z is the export interval.

Example Data

In the example below, two data sets were obtained from the same meter. The first data set (columns A and B) is a minute-granular average value export. The second data set (columns G and H) is max demand data from the same period, with a 15 minute demand interval and a 15 minute export interval.

	A	B	C	D	E	F	G	H
1	Minute-granular data			Rolling average from minute-granular data			Max Demand data	
2								
3	Date & Time	Usage [kW]		Average from	Rolling Average		Date & Time	Usage [kW]
4	2021-01-28 13:59:00	55.089283333		13:45-14:00	56.7900211111		2021-01-28 14:00:00	56.7900009
5	2021-01-28 13:58:00	56.49275		13:44-13:59	56.5595088889		2021-01-28 13:45:00	52.3419991
6	2021-01-28 13:57:00	56.900633333		13:43-13:58	56.2174022222		2021-01-28 13:30:00	54.0769997
7	2021-01-28 13:56:00	57.765366667		etc	55.8321444445		2021-01-28 13:15:00	51.6069984
8	2021-01-28 13:55:00	60.395766667			55.4234922222		2021-01-28 13:00:00	48.5709991
9	2021-01-28 13:54:00	56.6279			54.9259544444		2021-01-28 12:45:00	48.7529984
10	2021-01-28 13:53:00	57.475966667			54.6246155555		2021-01-28 12:30:00	47.9659996
11	2021-01-28 13:52:00	57.564466667			54.2360833333		2021-01-28 12:15:00	49.1230011
12	2021-01-28 13:51:00	58.2063			53.8801266666		2021-01-28 12:00:00	49.1749992
13	2021-01-28 13:50:00	55.730083333			53.2236766666		2021-01-28 11:45:00	53.9049988
14	2021-01-28 13:49:00	55.256983333			52.6654633333		2021-01-28 11:30:00	56.8689995
15	2021-01-28 13:48:00	54.837833333			52.0963011111		2021-01-28 11:15:00	57.1749992
16	2021-01-28 13:47:00	52.859533333			51.6740511111		2021-01-28 11:00:00	54.1669998
17	2021-01-28 13:46:00	61.186466667			51.6769844444			
18	2021-01-28 13:45:00	55.460983333		13:30-13:45	51.5543422221			
19	2021-01-28 13:44:00	51.6316		13:29-13:44	51.2030555555			
20	2021-01-28 13:43:00	51.36115		13:28-13:43	51.0654799999			
21	2021-01-28 13:42:00	51.121766667		etc	50.9269777777			
22	2021-01-28 13:41:00	51.635583333			50.7804288888			
23	2021-01-28 13:40:00	52.9327			50.7078866666			
24	2021-01-28 13:39:00	52.107816667			50.4607111111			
25	2021-01-28 13:38:00	51.647983333			50.6825277777			
26	2021-01-28 13:37:00	52.225116667			50.5729288889			
27	2021-01-28 13:36:00	48.35955			50.5417255555			
28	2021-01-28 13:35:00	47.356883333			51.1233977777			
29	2021-01-28 13:34:00	46.71955			51.6241911111			
30	2021-01-28 13:33:00	48.504083333			51.9212866666			
31	2021-01-28 13:32:00	52.903533333			52.0759266666			
32	2021-01-28 13:31:00	59.346833333			52.2548322222			
33	2021-01-28 13:30:00	50.191683333		13:15-13:30	51.6948977778			
34	2021-01-28 13:29:00	49.567966667		13:14-13:29	52.2043133334			
35	2021-01-28 13:28:00	49.283616667		13:13-13:28	52.4247444445			
36	2021-01-28 13:27:00	48.923533333		etc	53.0138922222			
37	2021-01-28 13:26:00	50.54745			53.3614977778			
38	2021-01-28 13:25:00	49.225066667			53.5938622223			
39	2021-01-28 13:24:00	55.435066667			53.9187677778			
40	2021-01-28 13:23:00	50.004			53.7153766667			
41	2021-01-28 13:22:00	51.757066667			54.0772077778			
42	2021-01-28 13:21:00	57.084633333			53.8461433333			
43	2021-01-28 13:20:00	54.868783333			53.5020177778			
44	2021-01-28 13:19:00	51.175983333			53.1382655556			
45	2021-01-28 13:18:00	50.823683333			53.3412022223			
46	2021-01-28 13:17:00	55.587116667			53.3585811112			
47	2021-01-28 13:16:00	50.947816667			52.6938211112			
48	2021-01-28 13:15:00	57.832916667			52.3805677779			

Column E contains a rolling 15 minute average calculated from the minute granular data in column B (the "Length of a demand interval" value). The blue outlines represent 15 minute intervals (the export interval value).

The colored values in column E are the highest value peak) for each 15 minute interval (one per blue box). They are color coded to tie to the same values in the max demand data (column H). Note that these aren't perfect matches - the process used by the eGauge is slightly different, so there will be small differences due to rounding and granularity.

Monetary registers

Overview

By default, the eGauge applies a flat rate cost per kWh used or generated to the summary areas at the top of the main graph page. In the example below, the site has used 24.7 kWh at a cost of \$.13 per kWh, for a grand total of \$3.21 of usage (values are rounded to the nearest cent).

Summary for time-period shown in graph		
Energy Used	24.7 kWh	(approx. \$3.21 used)
Energy Generated	0.00 Wh	(approx. \$0.00 saved)
Net	24.7 kWh bought	(approx. \$3.21 spent)

For Usage, the value entered in the "Average cost of 1kWh of electricity" under **Settings -> Preferences** is multiplied by the value of the "Usage" totaling register. For Generation, the value entered in the "Average REC-payment per kWh generated value is *added to* the "Average cost of 1kWh of electricity" value, and then multiplied by the value of the "Generation" totaling register.

Average cost of 1kWh of electricity	<input type="text" value="0.13"/>	(currency amount)
Average REC-payment per kWh generated	<input type="text"/>	(currency amount)

This approach does not work for tiered billing systems (ie, any billing system where the cost per kWh varies in response to the date, time, or month). However, it is possible to create **monetary registers** to accommodate these billing systems. Generally, one monetary register is required for Usage and one for Generation.

Creating a monetary register

Monetary registers are a special type of formula register. Formula registers are physical registers (that is, they occupy one register slot in the eGauge database) and are specified using the register type "=". A monetary register uses the unit type "Monetary [{currency}/s]". The register also features a formula field (empty in the example below).

Monetary Register	x	=	▼	Monetary [{currency}/s]	▼	
-------------------	---	---	---	-------------------------	---	--

As with all physical registers, a monetary register will only record values from the time it is created moving forward.

Creating a monetary register formula

It's strongly recommended to use a plain text editor (notepad, nano, vim, etc). Other software may add formatting characters which can cause issues.

At the most basic level, a monetary register multiplies the value of an arbitrary power register or set of power registers against one or more cost per kWh values. The monetary register is expressed in terms of currency units per second (eg, dollars, pounds, euros, etc). To convert from currency per kWh to currency per second, the final step of any monetary register calculation is to divide by 3.6e6 (or 3600000). Here's a basic example:

```
("Grid"*.25)/3.6e6
```

This example would effectively be the same thing as setting the "Average cost of 1kWh of electricity" to .25, except you can use *any arbitrary register* in the calculation (meaning you're not limited to using the Usage totaling register). For example, you could add two registers together:

```
((("Subpanel1"+"Subpanel2")*.25)/3.6e6
```

or multiply by some additional value:

```
((("Grid"*2)*.25)/3.6e6
```

Monetary register formulas can also utilize some of the built-in eGauge functions (specifically `time()`, `wday()`, `mday()`, and `month()`) for tiered billing structures. For more information on these functions, refer to the function documentation on your specific meter by appending `/fundoc.html?` to the end of your meter URL (for example, `DEVNAME.d.egauge.net/fundoc.html?` where `DEVNAME` is the [device name](#) of your meter).

Available functions will vary from meter to meter depending on meter firmware version. It may be necessary to [update meter firmware](#) to get access to certain functions. On meters connecting to eGauge.io this information will be available at: `DEVNAME.egauge.io/fundoc.html`

Basic Examples

Let's start with a basic example - tiered billing with a higher peak rate between 5pm and 9pm.

Standard 12am to 5pm; 9pm to 12am - \$.15 / kWh
Peak 5pm to 9pm - \$.25 / kWh

The eGauge features a full variety of comparison operators (=, <, >, <=, >=), so we'll use those along with the time() function.

The time() function is based on the time zone set under **Settings -> Date & Time**

A comparison operator will return a 1 if true, or a 0 if false. Thus, we can use the following to test for the time range between 5pm and 9pm:

```
(time()>17)*(time<21)
```

If both conditions are met, the result is 1*1. If either condition is false, the result is 0*1 or 1*0 (0 either way).

Next, we'll use a conditional to return a value based on the time comparisons (expressed as C ? T : F where C is the condition, T is the value returned if true, and F is the value returned if false).

Spaces in the formula are optional, and are ignored. In the examples below, spaces are used to improve readability.

```
((time()>17)*(time<21) ? ("Grid"*.25) : ("Grid"*.15))/3.6e6
```

This is the complete formula - we'll get the value of the "Grid" register multiplied by .25 and divided by 3.6e6 if the time is between 5pm and 9pm, and the value of the "Grid" register multiplied by .15 and divided by 3.6e6 if the time is outside of that range.

Intermediate Example

Moving on to a more complex example - it's possible to nest conditionals to perform more complex comparisons. Here's an example using time of day and day of week:

Peak - 5pm-9pm M-F \$.25

Peak Weekend - 4pm-10pm S-S \$.32

Off Peak - 12am - 5pm; 9pm-12am M-F \$.15

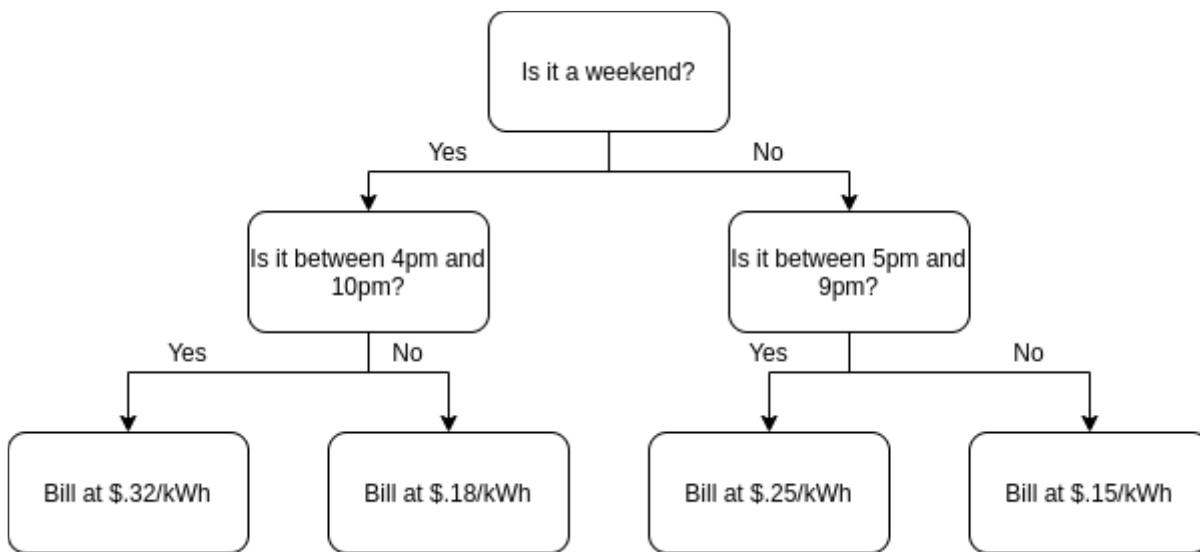
Off Peak Weekend - 12am-4pm; 10pm-12am S-S \$.18

```
((wday()<=4) ? ((time()>17)*(time<21) ? ("Grid"*.25) : ("Grid"*.15)) : ((time()>16)*(time<22) ? ("Grid"*.32) : ("Grid"*.18)))/3.6e6
```

Here, we've created a conditional which uses a comparison to see if wday()<=4 (corresponding to a weekday). If it is a weekday, the first nested conditional runs to check the time and return a

value based on the time; if it's a weekend, the second conditional runs a different time check and returns a different value based on the time. Finally, we divide by 3.6e6 at the end, which is always required.

The following flowchart may better illustrate the logic behind this formula:



Formula registers have a finite length, meaning that extremely intricate billing schemes may not be supportable.

Advanced Example

Building off of the previous example, let's say the utility also has an additional per-kWh charge from November thru May, and an additional flat daily rate:

Peak - 5pm-9pm M-F \$.25

Peak Weekend - 4pm-10pm S-S \$.32

Off Peak - 12am - 5pm; 9pm-12am M-F \$.15

Off Peak Weekend - 12am-4pm; 10pm-12am S-S \$.18

Winter Peak Charge - Nov-May, \$.08 per kWh

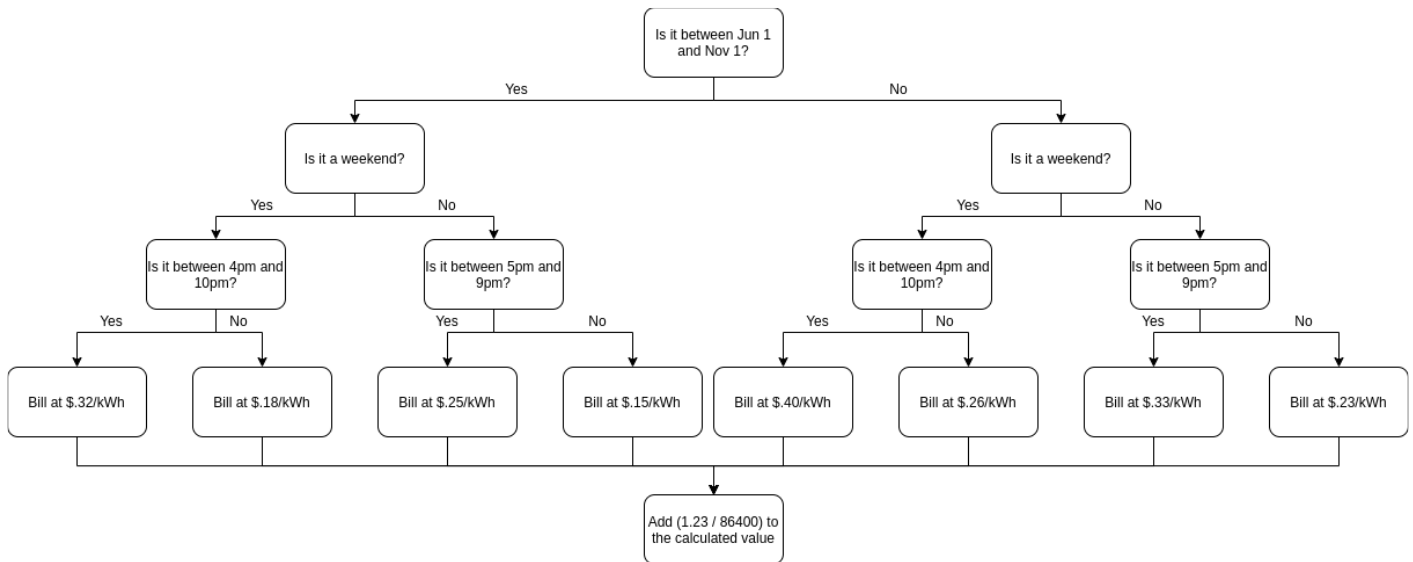
Daily surcharge of \$1.23

```
(((month()<10)*(month()>4)) ? ((wday()<=4) ? ((time()>17)*(time<21) ? ("Grid".25) : ("Grid".15)) : ((time()>16)*(time<22) ? ("Grid".32) : ("Grid".18))) : ((wday()<=4) ? ((time()>17)*(time<21) ? ("Grid".33) : ("Grid".23)) : ((time()>16)*(time<22) ? ("Grid".40) : ("Grid".26))))+(1.23/86400))/3.6e6
```

We're building off of the previous formula in this example. We've used another conditional to test for the month of the year - if the month is > 4 (ie, June 1) or < 10 (ie, Nov 1) the meter uses the original formula in the Intermediate Example. If the month *isn't* within that range, the meter uses a modified version of the formula in the Intermediate Example (with rates adjusted upward by \$.08 per kWh).

Once a cost per kWh value is calculated, we take the calculated value multiplied by the actual register measurement and add 1.23/86400. We have a daily charge of \$1.23, but the monetary value register expects *dollars per second*, not dollars per day. To convert \$/s to \$/d, we can divide by the number of seconds in a day (86400).

The following flowchart may better illustrate the logic behind this formula:



Using the monetary register

Once a monetary register has been created and saved, navigate back to **Settings -> Preferences**. Scroll down to "Money used register" or "Money earned register" (based on whether the monetary register is used for consumption or generation billing) and select the appropriate register from the dropdown menu.

Money used register

Monetary Usage Register ▼

Money earned register

▼

Make sure to click "Save" at the bottom of the page to apply these changes.

It's also possible to view the cumulative total (ie, cost) recorded by a monetary register by displaying that register in the [Mobile-Friendly Dashboard UI](#). To do this, create a new "Summary Table" dashlet (or add a new register to an existing dashlet). Leave the Type as "Register value", name the dashlet as desired, select the monetary register in the "Register to display" field, and leave the unit as "auto". It may be necessary to flip the polarity of the reading to get a positive dollar value (using the "Flip sign of the value" option). This should result in something similar to the following:

Since midnight



Grid Cost

\$0.06

Lua Scripting

Lua Scripting Overview

Lua scripting is an advanced topic. eGauge Support cannot review code and has limited support for troubleshooting Lua scripts.

Introduction

Support for eGauge Script (eScript) was introduced in firmware v1.1, allowing limited scripting through the use of [formula registers](#). eScript was designed for calculating basic and common mathematical formulas such as basic arithmetic operations, comparisons, and conditionals, along with the ability to call library routines (functions).

Firmware v4.1 introduced [Lua v5.3](#) support. To ensure safe operation, Lua execution is guarded with timeouts to protect against infinite loops or unacceptably long execution times. Additionally, a restricted set of standard libraries is supported.

To maintain backwards compatibility, firmware transparently translates eScript expressions to Lua scripts before execution and the Lua execution environment provides all the functions available within eScript.

Lua Primer

Lexical Conventions

Names may consist of any letters, digits, or underscores but may not start with a digit. Numeric values use the same conventions as in most other languages, including the ability to write hexadecimal values with a prefix of `0x`. The two boolean literal values are `true` and `false`. A variable with an undefined value is `nil`. Strings may be enclosed in double or single quotes. Characters within a string may be escaped with a backslash. Comments start with a double-dash (`--`) and extend to the end of the line. Statements may be terminated with a semicolon but, generally, semicolons are not required (even when there are multiple statements on a single line). The only exception to this rule is that a semicolon is required if the following statement starts with a left parenthesis.

Variables

Variables are *global* unless explicitly declared as local. For example:

```
x = 10      -- global variable
local y = 42 -- local variable
```

Operators

Lua provides a normal set of operators for arithmetic operations and comparisons. A bit unusual is that the inequality operator is `~=` rather than the more typical `!=`. Logical operators use keywords like in Python: **and**, **or**, and **not**. The length of a string or a table (see [Structured Data](#)) is obtained by prefixing the string or table name with a hash mark. `#'hi'` would return 2, for example. Strings can be concatenated with the double-dot operator. `'yell'..'ow'` would yield `'yellow'`, for example.

Control Structures

The syntax for the various control structures is:

```
for index=initial,step,final do block end -- numeric for loop
```

```
for var in iterator do block end -- iterator for loop
```

```
while cond do block end -- while loop
```

```
repeat block until cond -- do while loop
```

```
if cond then block else block end -- conditional statement
```

You can use **break** to exit a loop but, unlike in C and other languages, there is no `continue` that would allow you to skip to the next iteration.

Structured Data

Lua uses tables to represent both arrays and hash tables (Python dictionaries, Javascript objects). For example:

```
local a = {'a', 'b', 'c'}
```

assigns an array containing the strings `'a'`, `'b'`, and `'c'` to local variable `a`. Unlike in most other languages, the array base index is 1, so `a[1]` would yield the first element or `'a'` in our example.

Dictionary literal values can be written like this:

```
local d = {'k1'='v1', ['k2']='v2'}
```

and indexed with square brackets such that `d['k2']` would yield `'v2'`, for example. If the keys for a dictionary happen to be valid Lua names, the square brackets and quotes around the key strings can be omitted. For example, the above example could be simplified to:

```
local d = {k1='v1', k2='v2'}
```

For such key values, it is also possible to access their values using member-name syntax. For example, `d.k1` would yield `'v1'`, just as `d['k1']` would.

Migrating from eScript to Lua

Most of eScript has a direct equivalent in Lua. eScript has full support for a single numeric type (IEEE754 double precision float) and limited support for strings. As configured for the eGauge firmware, Lua has the same numeric type but also supports 32-bit integers and has full support for strings, boolean values, and tables.

The most important differences between eScript and Lua are as follows:

- In eScript, the value of a register is obtained with `$"register_name"`, whereas in Lua, the equivalent expression is `__r("register_name")`.
- Lua boolean values cannot directly be used as numeric values, whereas eScript uses 0 to represent false and any non-zero value for true.
- Lua does not provide a direct analog for the conditional operator

```
cond ? if_true_expr : if_false_expr
```

Instead, Lua uses the logical expression

```
cond and if_true_expr or if_false_expr
```

This works quite similarly to the eScript conditional because of the way the **and** and **or** operators are defined. Specifically, **and** returns **false** if the left-hand side is **nil** or **false** or the right-hand side's value otherwise. Operator **or** returns the value of the left-hand side if it is not **nil** or **false** and the value of the right-hand side otherwise. Both operators short-circuit evaluation and operator **and** has higher precedence than **or**.

- eScript automatically propagates NaN (Not-a-Number) values. For example, if any function is called with a NaN value, the returned result is also NaN. Similarly, if the condition of the conditional operator is NaN, then the result of the conditional expression is also NaN.

Given the similarities between eScript and Lua, most eScript expressions trivially translate to Lua. The non-trivial translations are shown in the table below:

eScript expression	Lua equivalent
<code>a < b</code>	<code>__lt(a, b)</code>
<code>a <= b</code>	<code>__le(a, b)</code>
<code>a > b</code>	<code>__gt(a,b)</code>
<code>a >= b</code>	<code>__ge(a,b)</code>
<code>a = b</code>	<code>__eq(a,b)</code>
<code>c ? a : b</code>	<pre>(function(__c) if __c ~= __c then return __c end return __c~=0 and (a) or (b) end)(c)</pre>

In words, the comparison operators get translated to calls to helper functions `__lt()`, `__le()` and so on. These helper functions check if either `a` or `b` is a NaN and return NaN, if so. If not, they perform the comparison and return 0 for false and 1 for true. The translation of the conditional operator is more complicated because care has to be taken to handle NaN properly and to evaluate `a` and `b` only when necessary. This is accomplished in Lua with the help of an anonymous inline function which checks whether the condition is NaN and returns NaN if that is the case. Otherwise, the function checks if the condition has a non-zero value and, if so, returns the value of `a`. Otherwise, it returns the value of `b`.

Lua Environment provided by eGauge Firmware

Standard Environment

For safety reasons, the eGauge firmware provides a restricted Lua environment (the Lua sandbox). Basic functions and variables are limited to the following subset (see [Lua 5.3 manual](#) for a detailed description):

```
_VERSION, assert, error, getmetatable, ipairs, load, next, pairs, pcall,
print, rawequal, rawget, rawlen, rawset, select, setmetatable, tonumber,
tostring, type, xpcall
```

The following standard Lua libraries are available:

Additions Provided by eGauge Firmware

Basic and Alert functions

All functions available to eScript are also available to Lua scripts. See the online documentation of an eGauge meter for a complete list (**Help ? Basic functions** or **Help ? Alert functions**). When these functions are called from Lua, they will also propagate NaN values, just like for eScript. That is, if any of the functions are called with a NaN argument, the return value will also be NaN.

Module **json**

This module provides the ability to encode a Lua object to a string and safely convert a string back to an object again.

```
string = json.encode(value):
```

This function accepts a Lua value and serializes it to the corresponding JSON string, which it returns. Tables that contain cyclical references cannot be JSON-encoded and will result in an error. The maximum size of the JSON-encoded string is currently limited to 4095 bytes. Lua tables may be indexed by a mix of number, boolean, and string values, whereas JSON objects are always indexed by strings. This function converts Lua tables that are not empty and whose indices consist entirely of numbers in the range from 1 to N (where N is the length of the table) to JSON arrays and everything else to JSON objects. In the latter case, numeric and boolean indices are converted to their equivalent strings.

```
value = json.decode(string):
```

This function accepts a string and deserializes it to the corresponding Lua object. Only double-quotes are allowed for string quoting. White space consisting of blanks or tabs is ignored.

Module **persistent**

This module provides variables whose values persist across power outages and device restarts.

```
obj = persistent:new(name, initial, description):
```

Declares a persistent variable with the specified *name*. Unlike Lua names, this name can be an arbitrary string. The name must be unique as any Lua script declaring a persistent variable of the same name will access the same underlying object. If this is the first time the persistent variable has been declared, its value is set to *initial*. The purpose of the variable must be described by the string passed for *description*. The return value is an object representing the persistent variable.

```
obj:get()
```

Returns the current value of the persistent variable represented by object *obj*.

```
obj:set(value)
```

Sets the value of the persistent variable represented by object *obj* to *value*. Any value acceptable to `json.encode()` may be set.

Lua Environment for Control Scripts

Control scripts have access to the standard environment available as described in the previous section.

They also have access to all basic and alert functions as well as to the [coroutine](#) library. Several low-level functions as well as several convenience modules are available as well, as described below.

Low-level Functions

Most of these functions are normally not used directly. They provide the low-level mechanisms required to implement the higher-level abstractions provided by the modules described in the sections below.

```
tid, err = __ctrl_submit(attrs, method, args...)
```

Submit a call to the method named *method* on the device identified by *attrs*, passing arguments *args*. The method name must be a string, *attrs* a table of name/value pairs, and *args* must be a sequence of zero or more Lua argument values that are compatible with the argument types expected by the named method. The method name may be a fully qualified method name consisting of an interface name, immediately followed by a dot (`.`), immediately followed by the proper method name or a proper method name on its own. In the latter case, the method is invoked through the first interface registered for the device that implements the named method. Otherwise, the method is invoked through the named interface.

`__ctrl_submit` returns two values: a transaction id *tid* and an error string *err*. On success, *tid* is a non-negative number which uniquely identifies the newly created call object and *err* is `nil`. On error, *tid* is a negative error code (see `ctrl.Error` below) and *err* is an optional string that may provide an explanation of why the call failed. The error string, if provided, is typically translated to the locale set on the meter.

```
status, result = __ctrl_result(tid):
```

Get the result for the method call identified by transaction id *tid*. The *tid* must be non-negative and must have been returned by a previous call to `__ctrl_submit`.

`__ctrl_result` returns two values: an integer *status* and *result*. The *status* is zero on success, in which case *result* is the result returned by the method call, converted to a Lua value. On error, *status* is a negative error code (see `ctrl.Error` below) and *result* is `nil`. In particular, if a method call is still in progress, error code `ctrl.Error.AGAIN` (-8) is returned. In this case, the caller should wait for a little bit and then retry the `__ctrl_result` call again until it succeeds.

```
status, err = __ctrl_cancel(tid):
```

Attempt to cancel the method call identified by transaction id *tid*. This *tid* must be non-negative and must have been returned by a previous call to `__ctrl_submit`.

`__ctrl_cancel` returns two values: an integer *status* and an optional error string *err*. The *status* is zero on success, in which case *tid* is guaranteed to be an invalid transaction id, until it is reused and returned by another call to `__ctrl_submit`. On error, *status* is a negative error code (see `ctrl.Error` below) and *err* is an optional string that may provide an explanation of why the call failed. The error string, if provided, is typically translated to the locale set on the meter.

```
ret, err = __ctrl_get_devices(attrs):
```

Get a list of devices that match the optional attributes specified by table *attrs*. If the *attrs* is omitted or `nil`, a list of all known (registered) devices is returned.

`__ctrl_get_devices` returns two values: table *ret* and an optional error string *err*. On success, *ret* is a list of tables and *err* is `nil`. Each table in the returned list corresponds to a matching device in no particular order. The table contains the name/value pairs registered for that device.

On error, *ret* is `nil` and *err* may be a non-`nil` string explaining what went wrong, typically translated to the locale set on the meter.

```
ret = __ctrl_get_interface(name):
```

Get a particular interface or a list of all known interfaces. Argument *name* must be a string that names the desired interface or omitted or `nil` to get a list of all interfaces.

`__ctrl_get_interface` returns a single value *ret*. If there was an error or if the requested interface cannot be found, `nil` is returned. Otherwise, *ret* is a single interface (if *name* was specified) or a list of interfaces (if *name* was omitted or `nil`). Each interface is described by a table with the following members:

- *name*: The name of the interface as a string.
- *methods*: A list of methods implemented by the interface.

Each method is described by a table with the following members:

- *name*: The name of the method as a string.
- *arg_types*: The DBus type signature of the arguments as a string.
- *ret_type*: The DBus type signature of the return value as a string.
- *doc*: A description of what the method does as a string. This string may contain references to the arguments which are enclosed in `<arg>/</arg>` tags. For example, the string "`<arg>foo</arg>`" would refer to the method argument named "foo".
- *arg_names*: A string of comma-separated argument names, in order of appearance. This is used only for documentation purposes as, other than in the *doc* string, the argument names have no significance. For example, the string "`foo,bar`" would indicate that the method expects two arguments, which are referred to in the *doc* string as argument names *foo* and *bar*, respectively.

```
ret = __sleep(time):
```

Suspend execution of the call for *time* seconds. The specified time may be a fractional amount, not just an integer amount. If the Lua program calling `__sleep` has other runnable coroutines, the other coroutines are executed. If there are no runnable coroutines left, execution of the program is suspended for the minimum amount of time required until the first coroutine becomes runnable again.

The function returns an integer *ret* which is zero on success or negative if an error occurred.

```
ret = sleep(time):
```

This is an alias for `__sleep` and may be used by end-user Lua programs as a convenience.

Libraries should always call `__sleep` instead to ensure the intended function is executed even if a Lua program redefines the name `sleep` to another value.

Module **ctrl**

This module provides a higher-level interface to invoke control methods. It is generally preferable to use this module rather than the low-level functions documented by the previous section.

```
dev = ctrl:dev(attrs, obj):
```

Create a control device object with attributes *attrs*. The optional *obj* argument could be specified to implement an extended control device class, but is usually omitted (or you could pass `nil` to it) to create a new object. This operation does not communicate with the remote device identified by *attrs* and therefore returns immediately.

```
iface = dev:interface(name):
```

Create an interface object for the interface identified by *name* of device *dev*. The returned interface object will contain a proxy method for each method defined by the named interface. The proxy methods can be called like any other method and automatically forward the call to the control device and then wait for the result to become available from the device. As such, these operations can take a long time to complete and will call `__sleep()` as needed. Because of this, other coroutines may be executed while a proxy method call is in progress.

```
status, result = dev:call(method, args...)
```

Call the method named *method* on device *dev*, passing arguments *args* to it and return the result. The method name must be a string and *args* must be a sequence of zero or more Lua argument values that are compatible with the argument types expected by the named method. The method name may be a fully qualified method name consisting of an interface name, immediately followed by a dot (`.`), immediately followed by the proper method name or a proper method name on its own. In the latter case, the method is invoked through the first interface registered for the device that implements the named method. Otherwise, the method is invoked through the named interface.

Two values are returned: an integer *status* and *result*. The *status* is zero on success, in which case *result* is the result returned by the named method, converted to a Lua value. On error, *status* is a negative error code (see `ctrl.Error` below) and *result* is `nil`.

Error:

This table declares symbolic names for various control errors, namely:

- UNSPEC (-1): An unspecified error occurred.
- INVALID (-2): An invalid or incompatible argument was passed to a method.
- NODEV (-3): The device attributes specified an invalid device path.
- ATTRS (-4): The attributes do not match the selected device.
- NOMETHOD (-5): The method name could not be found.
- NOENT (-6): The specified transaction id could not be found.
- BUSY (-7): The device is busy (too many pending calls).
- AGAIN (-8): The call is still pending.

Module **tasks**

This module provides a convenient interface for creating several tasks that may be run quasi concurrently and then executing them until they have all completed.

`tasks:add(fun, args...):`

Add a task which, when executed, runs function *fun* with arguments *args* until the function returns. The function may call `__sleep` or `coroutine.yield` to suspend execution temporarily and give other tasks a chance to execute.

`tasks:run():`

Execute previously added tasks until they have all completed.

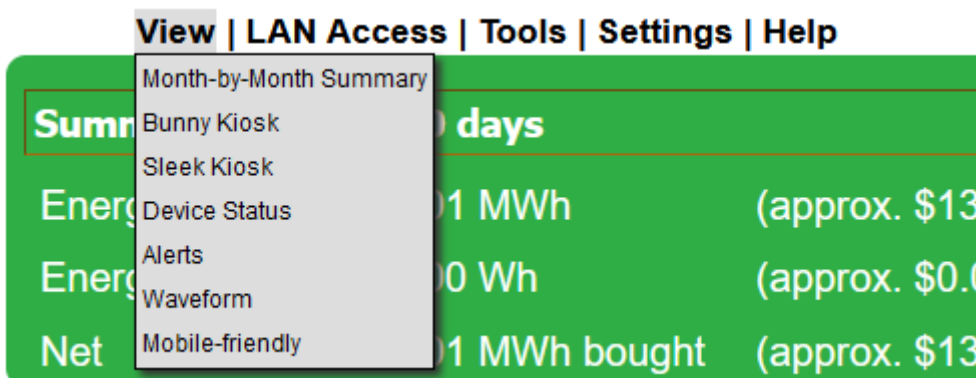
Creating and Using Lua scripts

Lua scripting is an advanced topic. eGauge Support cannot review code and has limited support for troubleshooting Lua scripts.

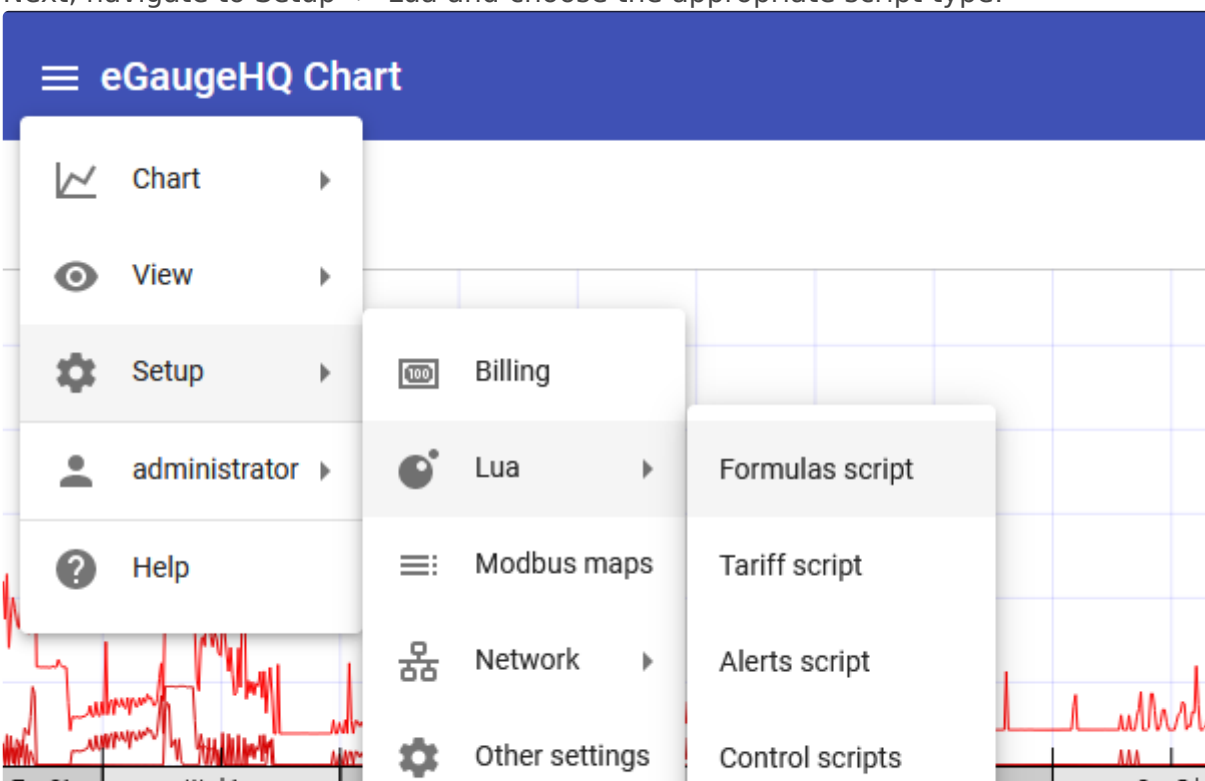
eGauge meters in firmware 4.1 and later have built-in Lua scripting functionality. The eGauge Lua script editor may be accessed from the Mobile Friendly interface.

Refer to the [Lua Scripting Overview](#) article for full details about the meter's Lua scripting interface.

If using the classic interface, click on View -> Mobile-Friendly:



Next, navigate to Setup -> Lua and choose the appropriate script type:



Formulas script

Creating lua functions in the Formulas script editor will allow the functions to be used in a formula register. For example, here are two functions made in the Lua Formulas script editor that will return the min or max of two numbers:

Lua script editor

Editing: teamd

```
1  --[[ function returning the max between two numbers --]]
2  function lua_max(num1, num2)
3
4      if (num1 > num2) then
5          result = num1;
6      else
7          result = num2;
8      end
9
10     return result;
11 end
12
13
14 --[[ function returning the min between two numbers --]]
15 function lua_min(num1, num2)
16
17     if (num1 < num2) then
18         result = num1;
19     else
20         result = num2;
21     end
22
23     return result;
24 end
```

They may then be used in a formula register:

lua max function	x	=	=	Whole number	lua_max(1,20)
lua min function	x	=	=	Whole number	lua_min(1,20)

And we may see the registers work as defined by the script functions:

1:40:06pm

lua max function	20
lua min function	1

Tariff script

Advanced time-of-use or tiered billing may be performed in this Lua script.

A tariff script should provide (at least) a `cost(register, negate, schedule)` function which calculates the incremental cost based on the energy-use recorded by register. *negate* can be set to true if the register's value counts down for power consumption. *schedule* is optional and can be set to the (non-default) name of the schedule to use when calculating the cost.

A formula register of type "monetary" would be used with the `cost()` function.

For an example billing script, from the classic interface:

1. Navigate to Settings -> Billing.
2. Choose Xcel Colorado as the tariff provider.
3. Click "OK" to save and go back to the main settings page.
4. Go back to Settings -> Billing.
5. Change the tariff provider to "custom"
6. Click the "Customize tariff script" button, which will open a copy of the Xcel Colorado billing tariff in the Lua script editor.

Alerts script

Alerts scripts work the same as Formula scripts, but are used in eGauge meter alerts, configured in Settings -> Alerts.

Control script

See the main [Lua Scripting Overview](#) Control Scripts section for additional Lua Control environment information.

There is high risk of damaging external equipment using control scripts. Only skilled Lua developers familiar with the eGauge meter and software should attempt to use Lua control scripts.

Control scripts can be used to control supported equipment such as the eGauge Power Relay Module (PRM3).

For example, the following script reads the instantaneous value of a register called "Temperature" and controls a PRM3 relay contact. If the temperature is lower than 21 C, relay number 0 of the PRM3 is closed (activated), otherwise it opens (turns off) relay number 0. It then sleeps for 15 minutes before checking again.

In the real world, the control script should be more advanced


```

dev = ctrl:dev({interface='relay'})
relay = dev:interface('relay')

while true do
    print("Temperature is currently: " .. __r("Temperature"))
    if __r("Temperature") < 21 then
        relay:close(0)
    else
        relay:open(0)
    end
    sleep(60*15)
end

```

Persistent variables

See the main [Lua Scripting Overview](#) section on Persistent variables.

Persistent variables are variables that are preserved between reboots or power cycles.

The following Formula script creates and updates a persistent variables with a given name and number passed to it in a formula register, and prints debug to the output log:

```

function persistent_variable_example(name, number)

    obj = persistent:new(name, number, "Variable stored by formula function persistent_variable_example")

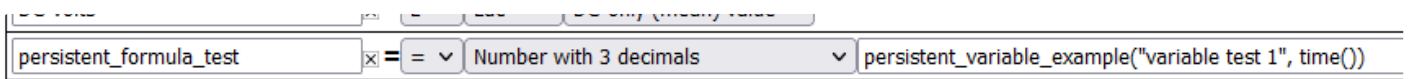
    current_value = obj:get()
    print(name .. " currently has value " .. current_value)

    print("updating " .. name .. "to new value " .. number)

    obj:set(number)
end

```

A register is configured to run the formula script:



This creates or updates a persistent variable called "variable test 1" with the current time.

The Formula script editor shows the print debug as the variable is updated once a second as the formula register is run:

```
15:26:28.168 variable test 1 currently has value 22.043333333333
15:26:28.168 updating variable test 1 to new value 22.043611111111
15:26:29.171 variable test 1 currently has value 22.043611111111
15:26:29.172 updating variable test 1 to new value 22.043888888889
15:26:30.168 variable test 1 currently has value 22.043888888889
15:26:30.168 updating variable test 1 to new value 22.044166666667
15:26:31.168 variable test 1 currently has value 22.044166666667
15:26:31.168 updating variable test 1 to new value 22.044444444444
```

While this particular example is rather pointless, persistent variables may be used in any Lua scripts. Control scripts are executed continuously, and a formula register would not need to be created to run it.