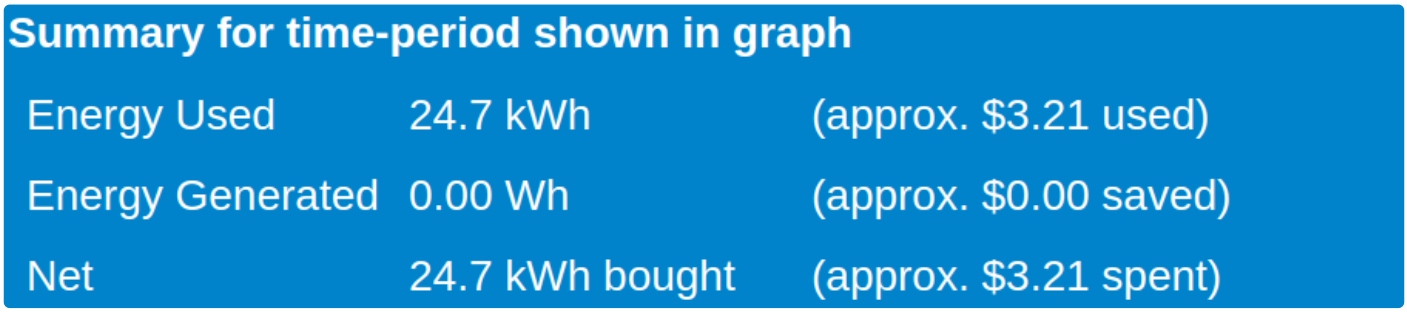
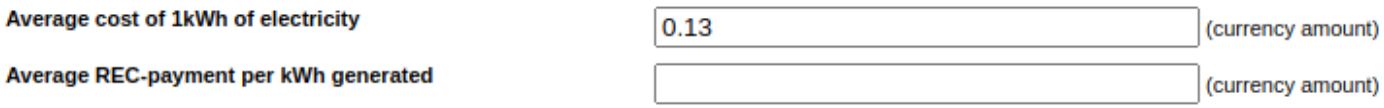
Monetary registers

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).



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.



This approach does not work for tiered billing systems (e.g., 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).

> 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

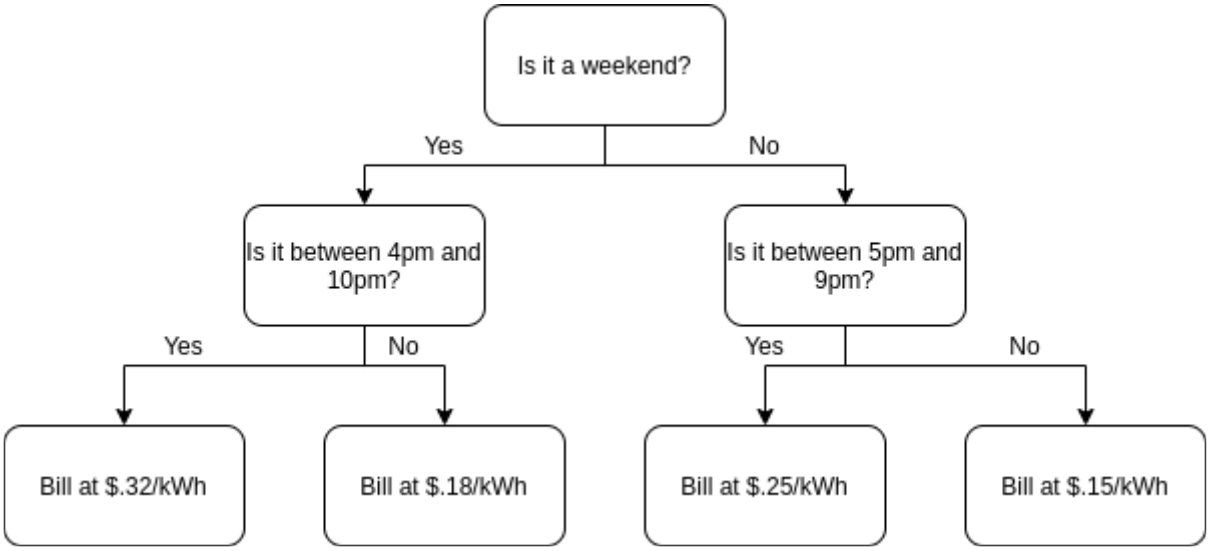
Moving on to a more complex example - it's possible to nest conditionals to perform more complex comparisons. Here's an example using time of day and day of week:

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

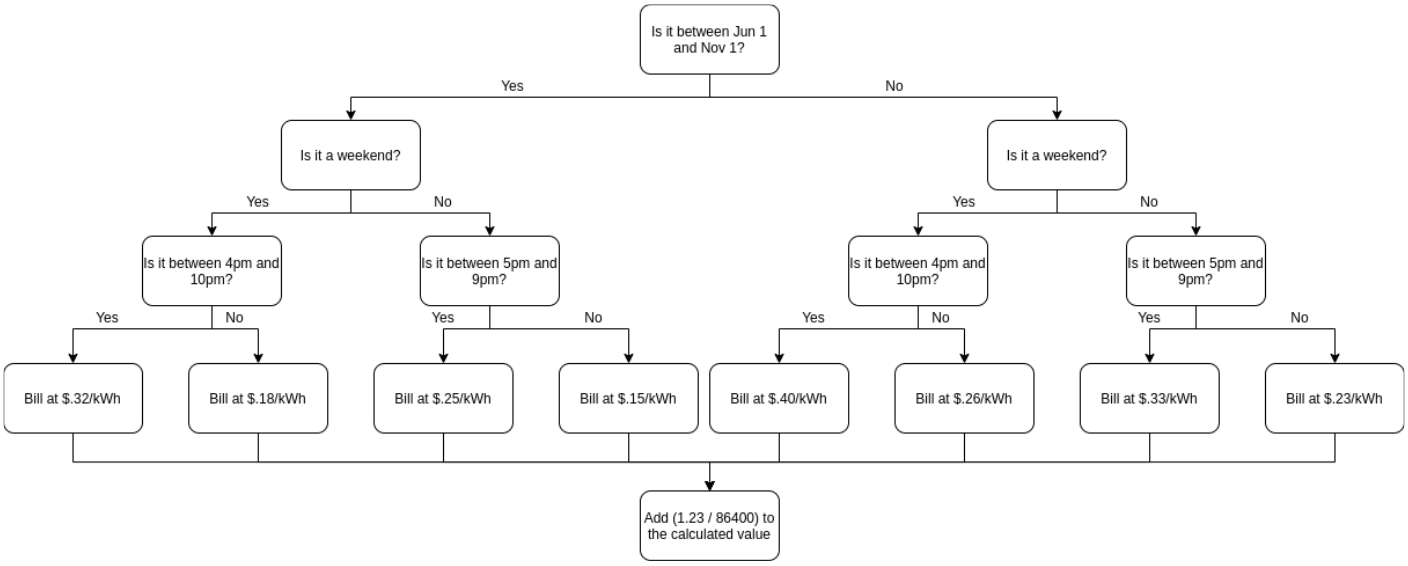
```
((((month()<10)*(month()>4)) ? ((wday()<=4) ? ((time()>17)*(time()<21) ? ($"Grid"*.25) : ($"Grid"*.15)) : ((time()>16)*(time()<22) ? ($"Grid"*.32) : ($"Grid"*.18))) : ((wday()<=4) ? ((time()>17)*(time()<21) ? ($"Grid"*.33) : ($"Grid"*.23)) : ((time()>16)*(time()<22) ? ($"Grid"*.40) : ($"Grid"*.26))))+(1.23/86400))/3.6e6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

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 (i.e., June 1) or < 10 (i.e., 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.

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

It's also possible to view the cumulative total (i.e., 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:

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