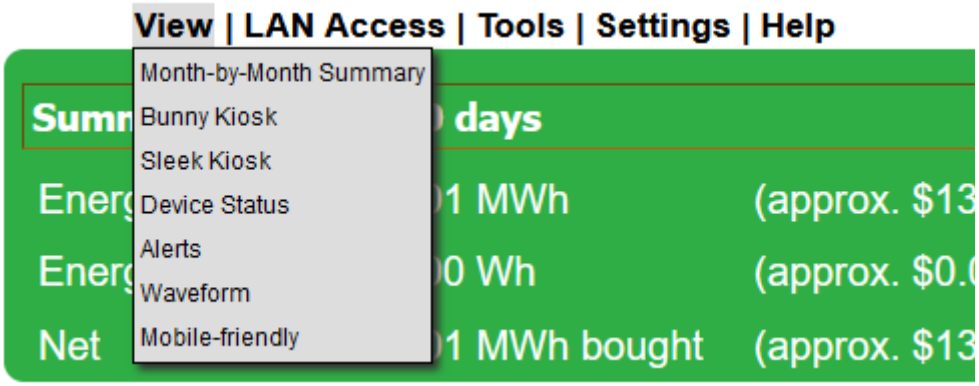
Creating and Using Lua scripts

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

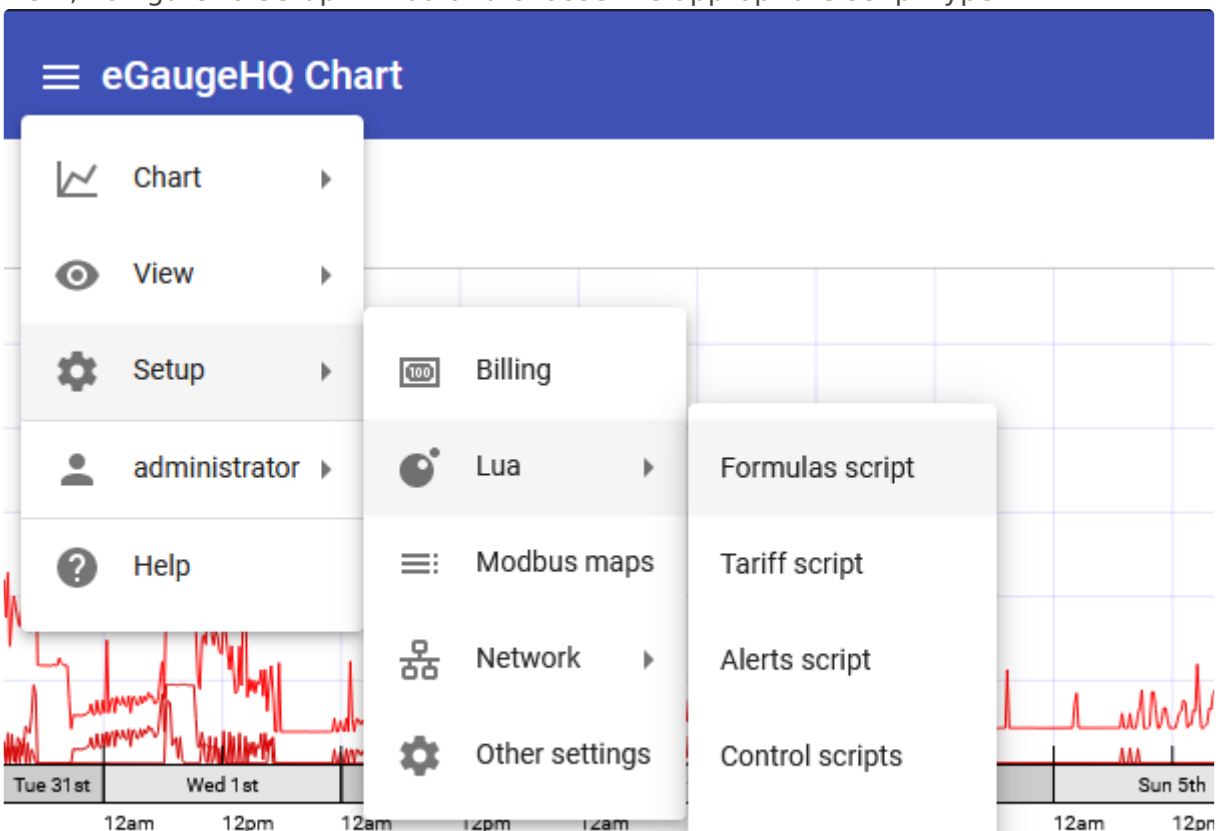
eGauge meters in firmware 4.1 and later have built-in Lua scripting functionality. The eGauge Lua script editor may be accessed from the Mobile Friendly interface.

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

```lua
--[[ function returning the max between two numbers --]]
function lua_max(num1, num2)

   if (num1 > num2) then
      result = num1;
   else
      result = num2;
   end

   return result;
end

--[[ function returning the min between two numbers --]]
function lua_min(num1, num2)

   if (num1 < num2) then
      result = num1;
   else
      result = num2;
   end

   return result;
end
```

They may then be used in a formula register:

lua max function	=	Whole number	lua_max(1,20)
lua min function	=	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 scrript.

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 providert 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 confrol 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

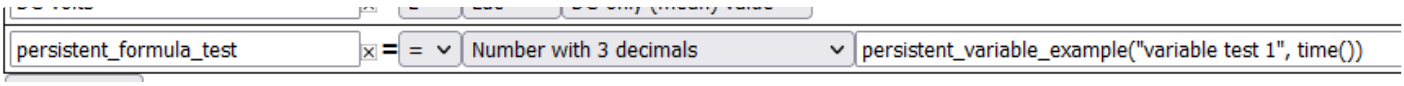
See the main Lua Scripting Overview section on Persistent variables.

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

The following Formula script creates and updates a peristent 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.
