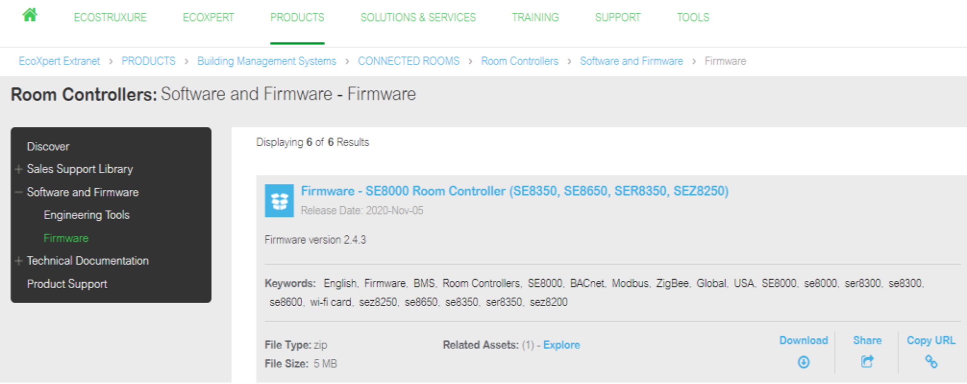
SE8K Programming Tutorial

Contents

Gather the tools

Firmware

The firmware can be found on the Schneider Electric Exchange site.



Room Controllers: Software and Firmware - Firmware

Displaying **6** of **6** Results

Firmware - SE8000 Room Controller (SE8350, SE8650, SER8350, SEZ8250)
Release Date: 2020-Nov-05

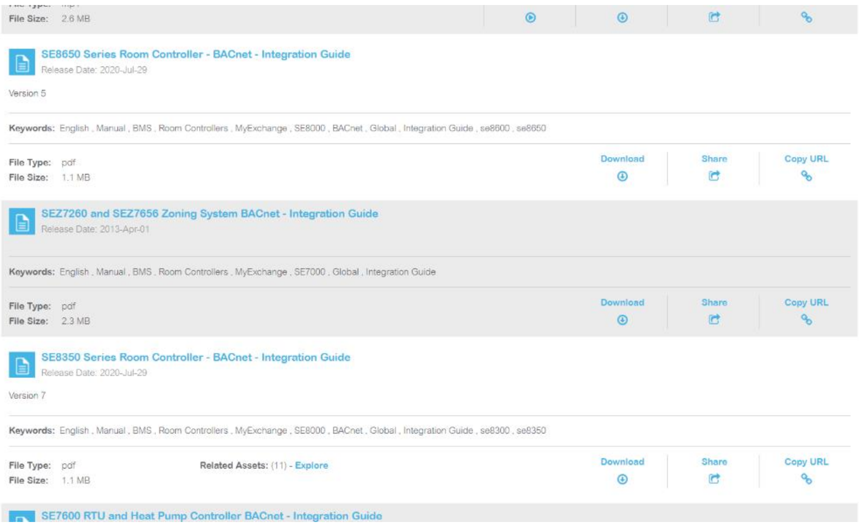
Firmware version 2.4.3

Keywords: English, Firmware, BMS, Room Controllers, SE8000, BACnet, Modbus, ZigBee, Global, USA, SE8000, se8000, ser8300, se8300, se8600, wi-fi card, sez8250, se8650, se8350, ser8350, sez8200

File Type: zip
File Size: 5 MB
Related Assets: (1) - Explore

Documentation

To program the SE8000 devices there are two main documents you need: the “BACnet Integration Guide” and the “User Interface Guide”. Both can be found on the Exchange site.



File Size: 2.6 MB

SE8650 Series Room Controller - BACnet - Integration Guide
Release Date: 2020-Jul-29

Version 5

Keywords: English , Manual , BMS , Room Controllers , MyExchange , SE8000 , BACnet , Global , Integration Guide , se8600 , se8650

File Type: pdf
File Size: 1.1 MB

SEZ7260 and SEZ7656 Zoning System BACnet - Integration Guide
Release Date: 2013-Apr-01

Keywords: English , Manual , BMS , Room Controllers , MyExchange , SE7000 , Global , Integration Guide

File Type: pdf
File Size: 2.3 MB

SE8350 Series Room Controller - BACnet - Integration Guide
Release Date: 2020-Jul-29

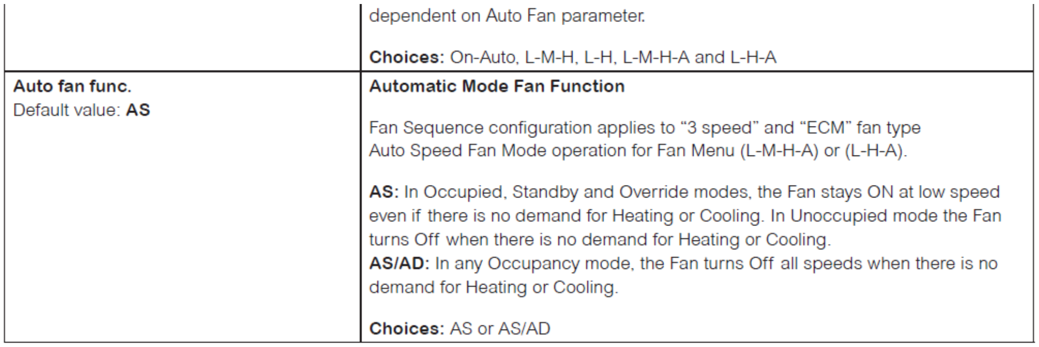
Version 7

Keywords: English , Manual , BMS , Room Controllers , MyExchange , SE8000 , BACnet , Global , Integration Guide , se8300 , se8350

File Type: pdf
File Size: 1.1 MB
Related Assets: (11) - Explore

SE7600 RTU and Heat Pump Controller BACnet - Integration Guide

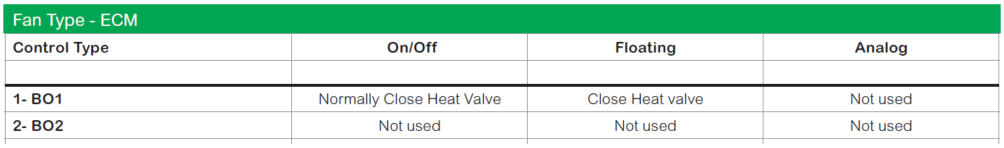
The BACnet Integration Guide will provide all the BACnet points that you will need to reference. It does not provide descriptions for what the points do, so you may need to look them up in the User Interface Guide if you are not familiar with them. For example, “AS” fan vs “AS/AD” fan found in the User Interface Guide:



	dependent on Auto Fan parameter. **Choices:** On-Auto, L-M-H, L-H, L-M-H-A and L-H-A
Auto fan func. Default value: **AS**	**Automatic Mode Fan Function** Fan Sequence configuration applies to “3 speed” and “ECM” fan type Auto Speed Fan Mode operation for Fan Menu (L-M-H-A) or (L-H-A). **AS:** In Occupied, Standby and Override modes, the Fan stays ON at low speed even if there is no demand for Heating or Cooling. In Unoccupied mode the Fan turns Off when there is no demand for Heating or Cooling. **AS/AD:** In any Occupancy mode, the Fan turns Off all speeds when there is no demand for Heating or Cooling. **Choices:** AS or AS/AD

Ensure the version of the documentation is appropriate to the firmware version you are using – older versions of the firmware or documentation may be missing some of the points.

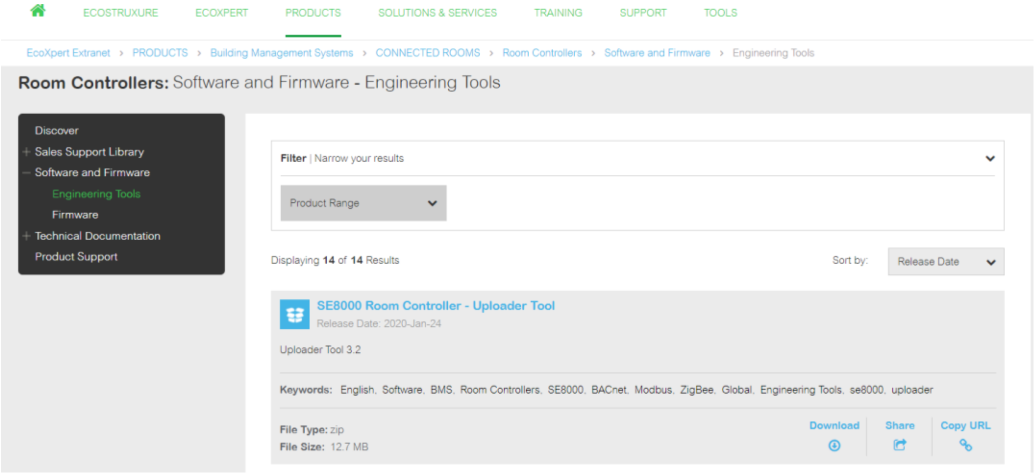
The Installation Guides are another useful document, as they provide charts that associate functionality and terms to inputs/outputs. Ex: BO1 on an SE8300 will close the cooling valve if the control type is set to “Floating” and Fan type set to “ECM”, BO3 on an SE8600 is Y1 (1st stage of cooling).



Fan Type - ECM			
Control Type	**On/Off**	**Floating**	**Analog**
1- BO1	Normally Close Heat Valve	Close Heat valve	Not used
2- BO2	Not used	Not used	Not used

Software

The uploader tool can be found on the Exchange. This program will allow you to connect to the controller using a micro USB cable, and upload firmware, scripts, and screensaver images (requires specific formatting of the image).



ECOSTRUXURE
ECOXPERT
PRODUCTS
SOLUTIONS & SERVICES
TRAINING
SUPPORT
TOOLS
EcoXpert Extranet › PRODUCTS › Building Management Systems › CONNECTED ROOMS › Room Controllers › Software and Firmware › Engineering Tools

Room Controllers: Software and Firmware - Engineering Tools

Displaying **14** of **14** Results

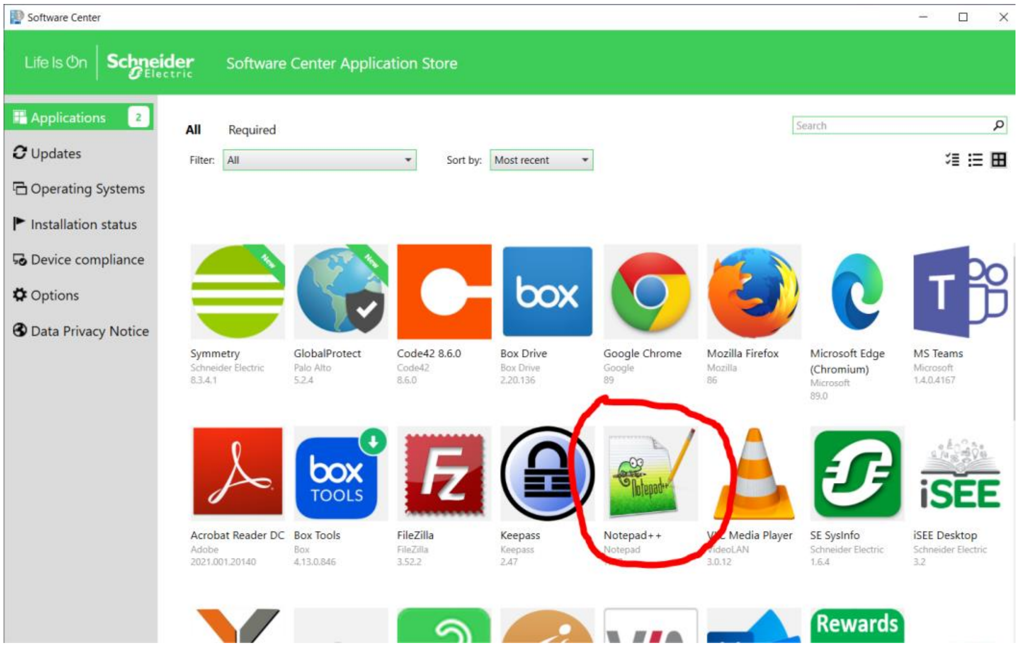
SE8000 Room Controller - Uploader Tool
Release Date: 2020-Jan-24

Uploader Tool 3.2

Keywords: English, Software, BMS, Room Controllers, SE8000, BACnet, Modbus, ZigBee, Global, Engineering Tools, se8000, uploader

File Type: zip
File Size: 12.7 MB

IDE/Text Editor of Choice. I recommend Notepad++ - it is a lightweight program that has syntax highlighting to help you write the code (top menu: Language > L > Lua), and is available for download from the Software Center.

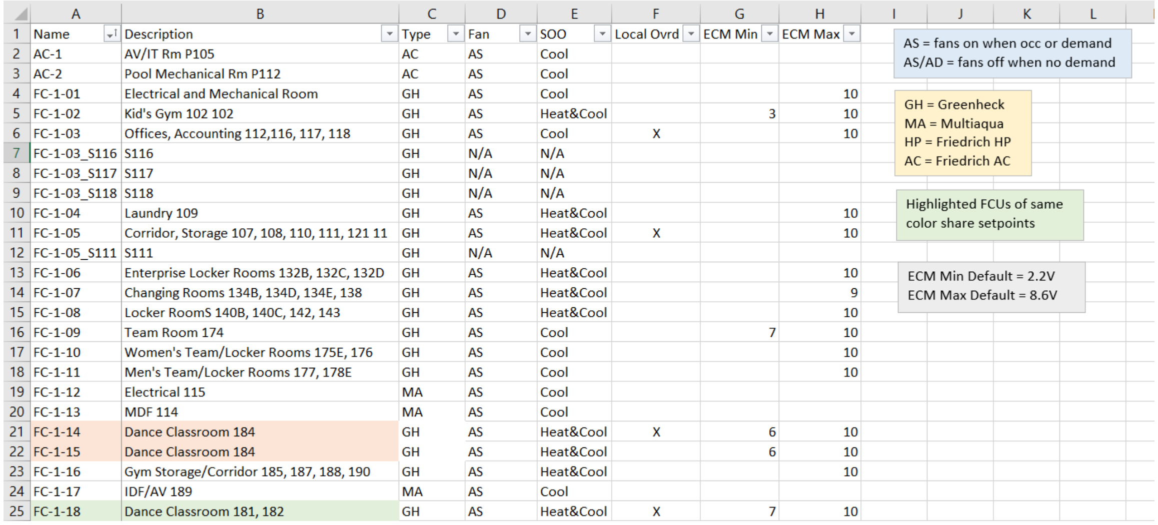


Plan

Plan out what you need the device to do. Compare the requirements from the submittals to the configuration choices available. Determine what requirements can be achieved by just setting configuration options versus programming device behavior. Determine any configuration values that can be left as their defaults.

If configuration changes are all that is needed, the script will only need an initialization section. This will ensure that in the case of power loss, the device will retain its configuration settings when power is returned (for planning purposes, the script is maintained for up to 14 days without power).

If there are many devices with different configurations or sequences, consider creating a “cheat sheet” to help you keep track of the differences.



	A	B	C	D	E	F	G	H
1	Name	Description	Type	Fan	SOO	Local Ovrd	ECM Min	ECM Max
2	AC-1	AV/IT Rm P105	AC	AS	Cool			
3	AC-2	Pool Mechanical Rm P112	AC	AS	Cool			
4	FC-1-01	Electrical and Mechanical Room	GH	AS	Cool			10
5	FC-1-02	Kid's Gym 102 102	GH	AS	Heat&Cool		3	10
6	FC-1-03	Offices, Accounting 112,116, 117, 118	GH	AS	Cool	X		10
7	FC-1-03_S116	S116	GH	N/A	N/A			
8	FC-1-03_S117	S117	GH	N/A	N/A			
9	FC-1-03_S118	S118	GH	N/A	N/A			
10	FC-1-04	Laundry 109	GH	AS	Heat&Cool			10
11	FC-1-05	Corridor, Storage 107, 108, 110, 111, 121 11	GH	AS	Heat&Cool	X		10
12	FC-1-05_S111	S111	GH	N/A	N/A			
13	FC-1-06	Enterprise Locker Rooms 132B, 132C, 132D	GH	AS	Heat&Cool			10
14	FC-1-07	Changing Rooms 134B, 134D, 134E, 138	GH	AS	Heat&Cool			9
15	FC-1-08	Locker RoomS 140B, 140C, 142, 143	GH	AS	Heat&Cool			10
16	FC-1-09	Team Room 174	GH	AS	Cool		7	10
17	FC-1-10	Women's Team/Locker Rooms 175E, 176	GH	AS	Cool			10
18	FC-1-11	Men's Team/Locker Rooms 177, 178E	GH	AS	Cool			10
19	FC-1-12	Electrical 115	MA	AS	Cool			
20	FC-1-13	MDF 114	MA	AS	Cool			
21	FC-1-14	Dance Classroom 184	GH	AS	Heat&Cool	X	6	10
22	FC-1-15	Dance Classroom 184	GH	AS	Heat&Cool		6	10
23	FC-1-16	Gym Storage/Corridor 185, 187, 188, 190	GH	AS	Heat&Cool			10
24	FC-1-17	IDF/AV 189	MA	AS	Cool			
25	FC-1-18	Dance Classroom 181, 182	GH	AS	Heat&Cool	X	7	10

AS = fans on when occ or demand
AS/AD = fans off when no demand

GH = Greenheck
MA = Multiaqua
HP = Friedrich HP
AC = Friedrich AC

Highlighted FCUs of same color share setpoints

ECM Min Default = 2.2V
ECM Max Default = 8.6V

LUA basics

Language basics

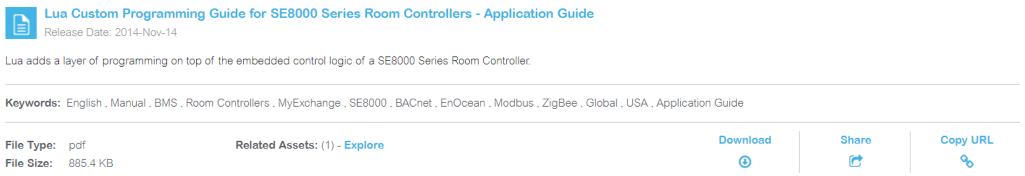
Lua was designed as a scripting language for C++, and thus it should look familiar to anyone used to the “C family” of languages (C++, Java, Javascript, etc). Some of the more noticeable differences are “if” statements structured as “if then end”, and string concatenation is with “..” - Ex:

```
print(“My variable: “ .. myVariable)
```

if myVariable has, say, a value of 3, the statement printed will be

```
My variable: 3
```

Lua tutorials are easily found on Youtube and other internet sites. Also, see the “Lua Custom Programming Guide” document for useful functions, practices, and technical information.



Lua Custom Programming Guide for SE8000 Series Room Controllers - Application Guide
Release Date: 2014-Nov-14

Lua adds a layer of programming on top of the embedded control logic of a SE8000 Series Room Controller.

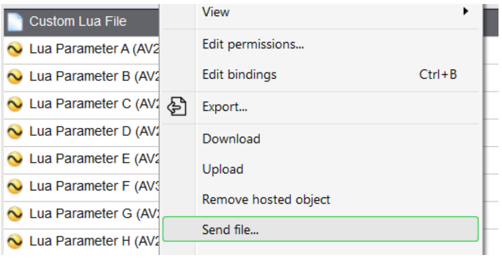
Keywords: English . Manual . BMS . Room Controllers . MyExchange . SE8000 . BACnet . EnOcean . Modbus . ZigBee . Global . USA . Application Guide

File Type: pdf
File Size: 885.4 KB

Related Assets: (1) - Explore

Script Options

Previously there was an option of one "large" script uploaded by connecting a USB cable from your computer to the controller and using the uploader tool found on the Exchange, or ten "small" scripts - Generic BACnet Program objects within EBO, limited to 420 characters each. However, with the latest version of the firmware, the ten small scripts within EBO are replaced with the ability to send the single large file to the device via EBO. The device has an object called "Custom Lua File" - if you right-click it there is a "send file" option, which will let you upload a script from your computer.

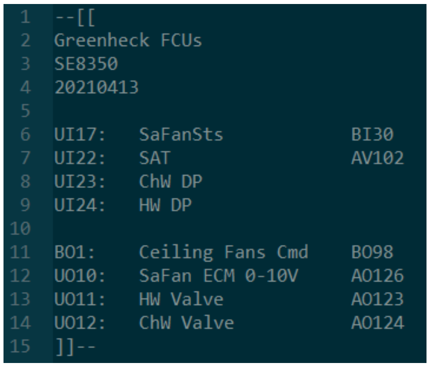


Maximum size of the script is 15 KB, but due to memory overhead used, the programming guide recommends keeping it to less than 13 KB.

Writing the script

Header & version

A commented-out section at the top of the script. I recommend a short descriptor/name, the type of controller the script is for, and a list of inputs and outputs. It is also useful to include the BACnet points that reference the values of the I/O points, to make it easier to remember those points when you are writing the script.

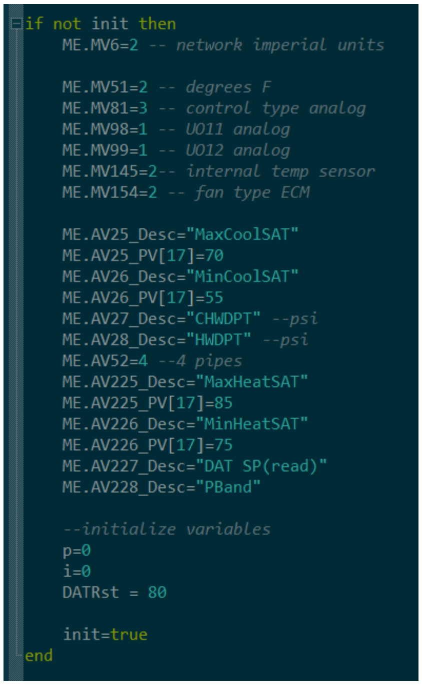


```
--[[
Greenheck FCUs
SE8350
20210413

UI17:   SaFanSts            BI30
UI22:   SAT                 AV102
UI23:   ChW DP
UI24:   HW DP

BO1:    Ceiling Fans Cmd    BO98
UO10:   SaFan ECM 0-10V     AO126
UO11:   HW Valve            AO123
UO12:   ChW Valve           AO124
]]--
```

Some method of tracking the version of the script is important, as you want to ensure everyone working on the project is using the same version after any changes are made (I use the date in YYYYMMDD format).

Initialization

The first bit of actual code in the script. This is run once, when the device powers up, and is where you will have configuration settings. It looks for a variable (in the screenshot I use “init”) - if the device just powered on, that variable will have no value, and the code under the “if not init then” statement will run. At the end of the section give the variable a value of “true” so that the initialization does not run again next time the script runs (scripts run once per second).



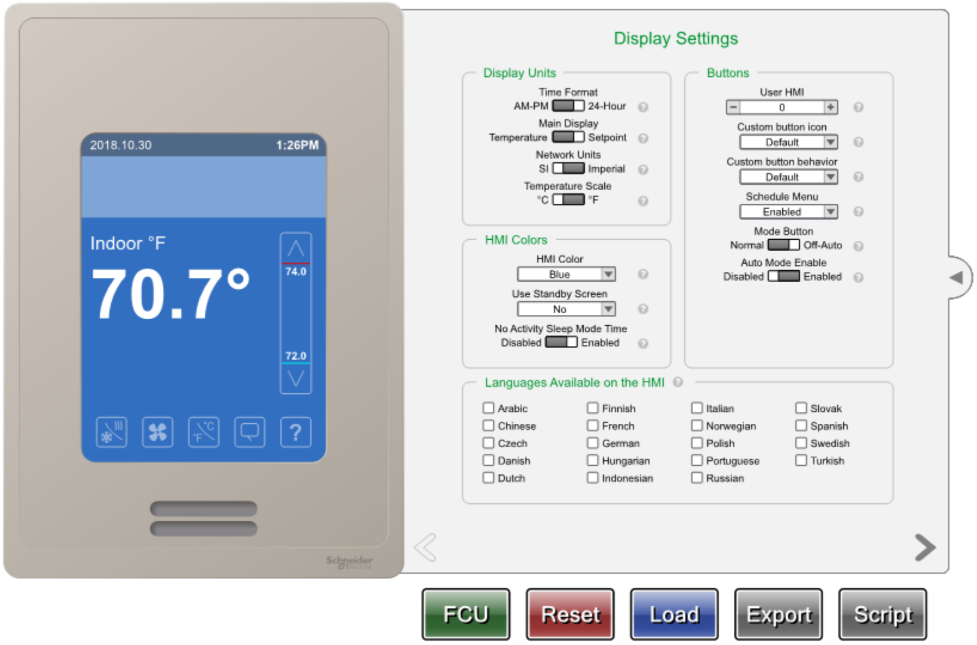
```
if not init then
    ME.MV6=2 -- network imperial units

    ME.MV51=2 -- degrees F
    ME.MV81=3 -- control type analog
    ME.MV98=1 -- UO11 analog
    ME.MV99=1 -- UO12 analog
    ME.MV145=2-- internal temp sensor
    ME.MV154=2 -- fan type ECM

    ME.AV25_Desc="MaxCoolSAT"
    ME.AV25_PV[17]=70
    ME.AV26_Desc="MinCoolSAT"
    ME.AV26_PV[17]=55
    ME.AV27_Desc="CHWDPT" --psi
    ME.AV28_Desc="HWDPT" --psi
    ME.AV52=4 --4 pipes
    ME.AV225_Desc="MaxHeatSAT"
    ME.AV225_PV[17]=85
    ME.AV226_Desc="MinHeatSAT"
    ME.AV226_PV[17]=75
    ME.AV227_Desc="DAT SP(read)"
    ME.AV228_Desc="PBand"

    --initialize variables
    p=0
    i=0
    DATRst = 80

    init=true
end
```

It is useful to have comments indicating what each configuration setting does.

In this section you can also assign default values, or more permanent ones – values assigned at a normal priority will prevent end-users from changing the value at the controller, though the BMS will still be able to override it.

The order in which values are assigned/configured is important, and the script may generate errors. For example, assigning an extreme value to a temperature setpoint may generate an error it that point is not first configured to the appropriate Fahrenheit/Celsius scale.

As previously mentioned, if the desired sequence of operation can be achieved solely through changing configuration options, then the initialization section is the only thing needed in the script. Also, if that is all you need (or you’re using the standard FCU program), take a look at the SE8000 Lua TGML Configurator found on the Exchange Community – it’s a graphic that helps you set up the device, or create a script that you can upload.



Display Settings

Display Units

Time Format: AM-PM / 24-Hour

Main Display: Temperature / Setpoint

Network Units: SI / Imperial

Temperature Scale: °C / °F

HMI Colors

HMI Color: Blue

Use Standby Screen: No

No Activity Sleep Mode Time: Disabled / Enabled

Buttons

User HMI: 0

Custom button icon: Default

Custom button behavior: Default

Schedule Menu: Enabled

Mode Button: Normal / Off-Auto

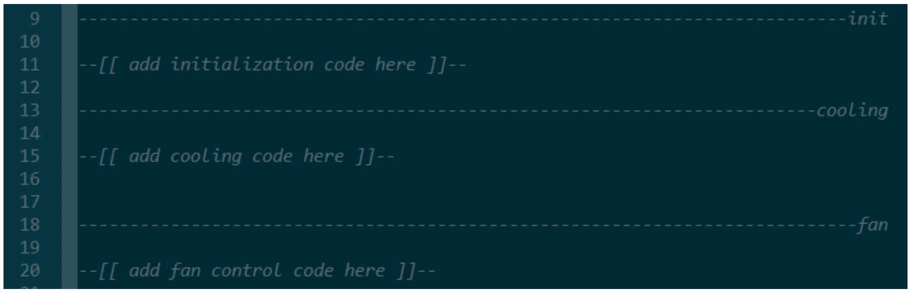
Auto Mode Enable: Disabled / Enabled

Languages Available on the HMI

□ Arabic
□ Chinese
□ Czech
□ Danish
□ Dutch
□ Finnish
□ French
□ German
□ Hungarian
□ Indonesian
□ Italian
□ Norwegian
□ Polish
□ Portuguese
□ Russian
□ Slovak
□ Spanish
□ Swedish
□ Turkish

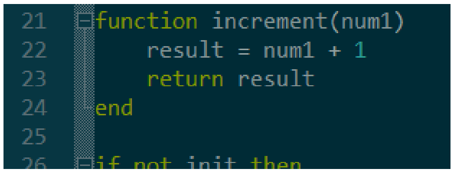
Functions

Break up the code into logical sections. For example, have all code related to heating grouped in one area, and all code related to the fan operation in another. For devices that have overall different sequences of operation, but some commonalities, this makes it easier to reuse code for the common parts, as well as managing and troubleshooting later.



```
--------------------------------------------------------------------------init

--[[ add initialization code here ]]--

-----------------------------------------------------------------------cooling

--[[ add cooling code here ]]--


---------------------------------------------------------------------------fan

--[[ add fan control code here ]]--
```

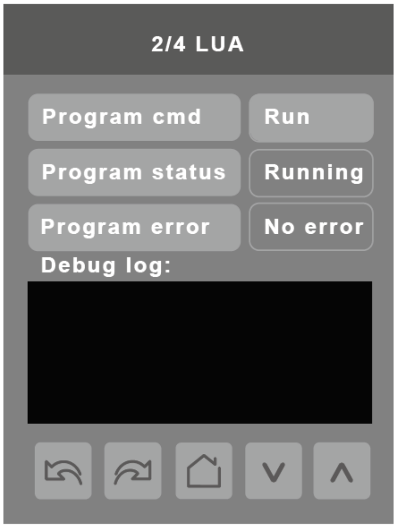
Actual Lua functions: <u>I have not yet tested this very much</u>, and am not sure there’s a use-case for it, but it is possible to use traditional coding functions within the script. The function must be fully defined at the beginning of the code.



```
function increment(num1)
    result = num1 + 1
    return result
end

if not init then
```

Print Statements

The output of the print statement appears in the “description of halt” property for the BACnet Lua script objects (if there are no errors), and on the screen of the device itself in the Debug Log of the Lua Settings Screen.



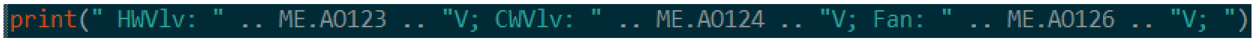
2/4 LUA

Program cmd: Run

Program status: Running

Program error: No error

Debug log:



```
print(" HWVlv: " .. ME.AO123 .. "V; CWVlv: " .. ME.AO124 .. "V; Fan: " .. ME.AO126 .. "V; ")
```

Lua Variables

While you can have as many named variables in the script as you want, variables that can be accessed by EBO (to read or write) are limited to AV25 through AV30, and AV225 through AV230.

It is a good idea to give lua variables a description in the script, using the “Desc” property:

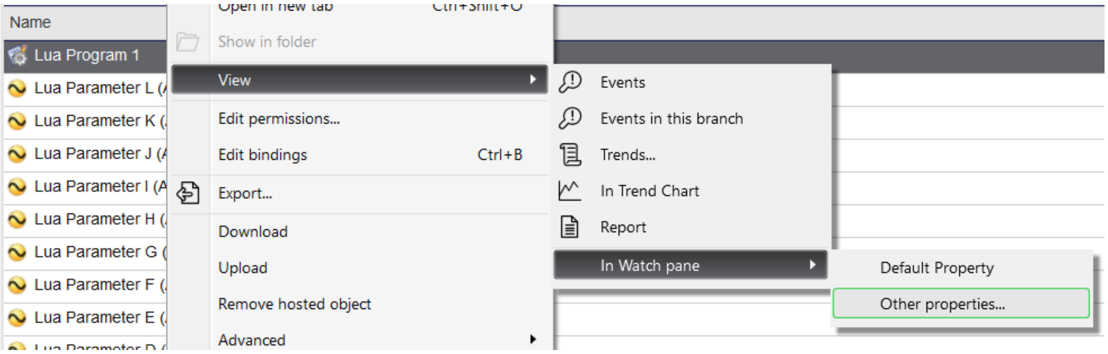
ME.AV25_Desc="MaxHeatSAT”

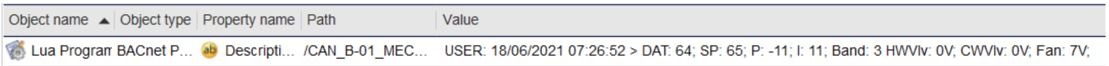
If you receive an error message in the description of halt related to this description, it may be that the String you are using is too long.

Because they are analog values, avoid using TRUE/FALSE values in the script for these EBO-accessible lua variables – I have found they do not always behave as one would expect. Stick to 0/1 values instead.

Debugging

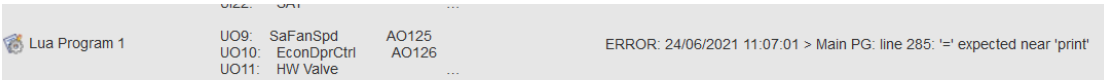
Right-click on the lua script object, hover over “view in watch pane”, then select “other property”. This will allow you to view the “description of halt” property, which will display the print statement from the script. You can also add this as a column in the main window when viewing the contents of a folder.





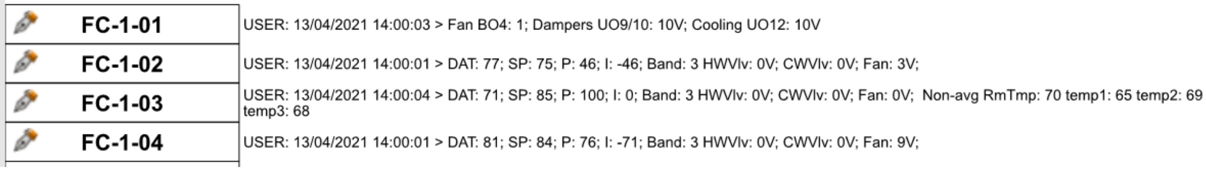
Object name	Object type	Property name	Path	Value
Lua Prograr	BACnet P…	Descripti…	/CAN_B-01_MEC…	USER: 18/06/2021 07:26:52 > DAT: 64; SP: 65; P: -11; I: 11; Band: 3 HWVlv: 0V; CWVlv: 0V; Fan: 7V;

If there is an error in the script it will be displayed by the “description of halt” property instead. It will indicate what line in the code contains the error.



```
Lua Program 1     UO9:   SaFanSpd      AO125
                  UO10:  EconDprCtrl   AO126
                  UO11:  HW Valve      ...
ERROR: 24/06/2021 11:07:01 > Main PG: line 285: '=' expected near 'print'
```

The “description of halt” property of the Lua script object is a String that can be bound to a text field within a graphic. This can be useful for debugging, or as a summary of information you want the script to present to someone viewing the graphic.



FC-1-01	USER: 13/04/2021 14:00:03 > Fan BO4: 1; Dampers UO9/10: 10V; Cooling UO12: 10V
FC-1-02	USER: 13/04/2021 14:00:01 > DAT: 77; SP: 75; P: 46; I: -46; Band: 3 HWVlv: 0V; CWVlv: 0V; Fan: 3V;
FC-1-03	USER: 13/04/2021 14:00:04 > DAT: 71; SP: 85; P: 100; I: 0; Band: 3 HWVlv: 0V; CWVlv: 0V; Fan: 0V; Non-avg RmTmp: 70 temp1: 65 temp2: 69 temp3: 68
FC-1-04	USER: 13/04/2021 14:00:01 > DAT: 81; SP: 84; P: 76; I: -71; Band: 3 HWVlv: 0V; CWVlv: 0V; Fan: 9V;

Depending on how elaborate you want to make it, you can use the split() method of the String object in Javascript to break down that text into pieces and use only certain parts, or concatenate it with other text.

Screensaver

A custom screen can be displayed on the device as a screen saver. End-users tend to enjoy seeing their logo on the display. The difficulty is in getting the image to fit the requirements of the SE8K: a BMP file with a width of 240, height of 320, and bit depth of either 24 or 32. For this I recommend using Gimp, a free and open-source image editor (think of it as a free version of Photoshop). Last time I did this the image would appear off-center on the screen, so it took a few tries adjusting the logo’s location in the image to get it right (don’t assume the image will work properly on the first try).

Useful Code Scraps

Sending a value over BACnet.

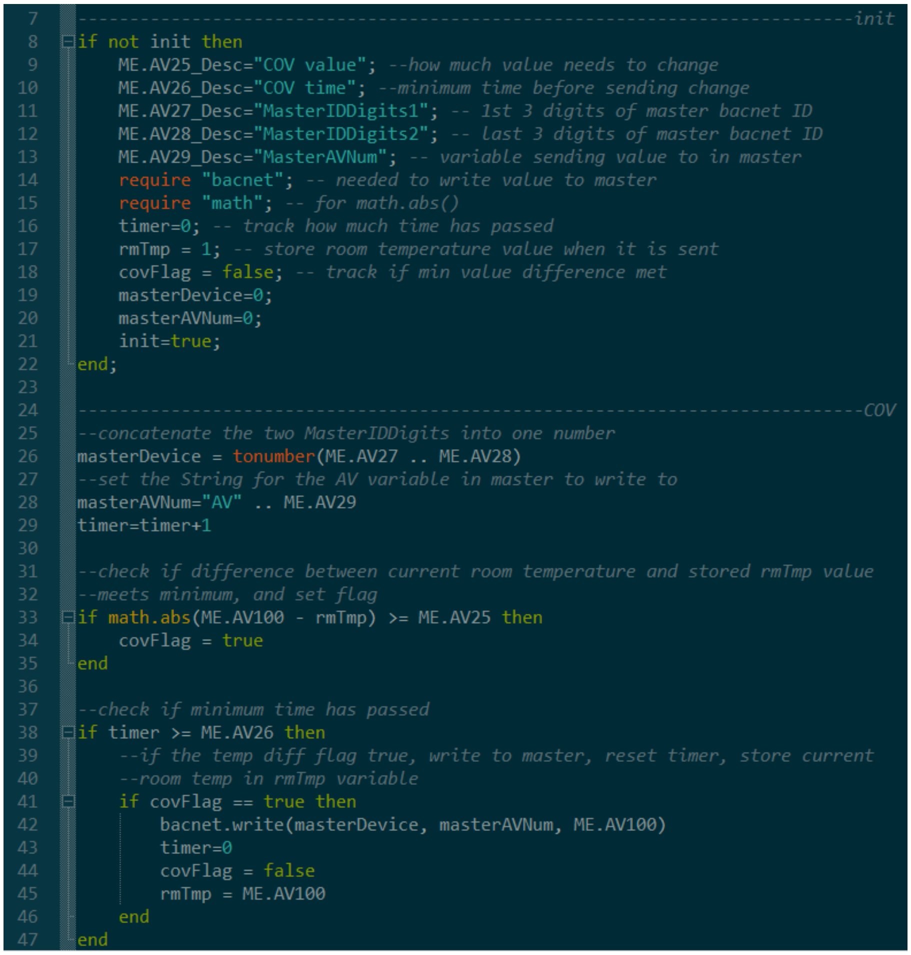
If one device needs to receive a value directly from another, binding the values to each other within EBO seems to not have them follow standard COV rules (as tested with a wireshark capture using EBO 3.1). If you are trying to optimize network traffic, in the “sending” device have a script with time/COV rules for transferring values (when tracking time remember that the script runs once per second), and use the function

bacnet.write(masterDevice, masterAVNum, valueToSend)

where masterDevice is the BACnet ID of the recipient of the value, masterAVNum is the bacnet object receiving the value (ex: AV100), and value to send is the value or variable to transfer. The init section of the script will need the line

require “bacnet”

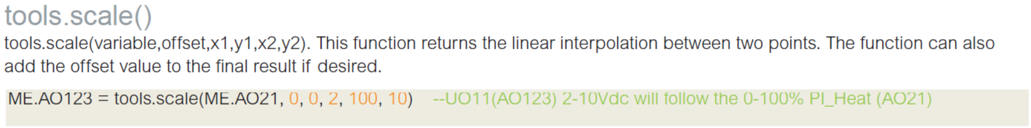
The Lua variables used for configuration (ID, time, value) may need to be hard-coded or bound to values within EBO, to prevent the values from being lost in case of a loss of power.



```
---------------------------------------------------------------------------init
if not init then
    ME.AV25_Desc="COV value"; --how much value needs to change
    ME.AV26_Desc="COV time"; --minimum time before sending change
    ME.AV27_Desc="MasterIDDigits1"; -- 1st 3 digits of master bacnet ID
    ME.AV28_Desc="MasterIDDigits2"; -- last 3 digits of master bacnet ID
    ME.AV29_Desc="MasterAVNum"; -- variable sending value to in master
    require "bacnet"; -- needed to write value to master
    require "math"; -- for math.abs()
    timer=0; -- track how much time has passed
    rmTmp = 1; -- store room temperature value when it is sent
    covFlag = false; -- track if min value difference met
    masterDevice=0;
    masterAVNum=0;
    init=true;
end;

-----------------------------------------------------------------------------COV
--concatenate the two MasterIDDigits into one number
masterDevice = tonumber(ME.AV27 .. ME.AV28)
--set the String for the AV variable in master to write to
masterAVNum="AV" .. ME.AV29
timer=timer+1

--check if difference between current room temperature and stored rmTmp value
--meets minimum, and set flag
if math.abs(ME.AV100 - rmTmp) >= ME.AV25 then
    covFlag = true
end

--check if minimum time has passed
if timer >= ME.AV26 then
    --if the temp diff flag true, write to master, reset timer, store current
    --room temp in rmTmp variable
    if covFlag == true then
        bacnet.write(masterDevice, masterAVNum, ME.AV100)
        timer=0
        covFlag = false
        rmTmp = ME.AV100
    end
end
```

Scaling a value.

This is already found in the "Lua Custom Programing Guide" mentioned earlier, but I include it here as well because I've found that I use it a lot.

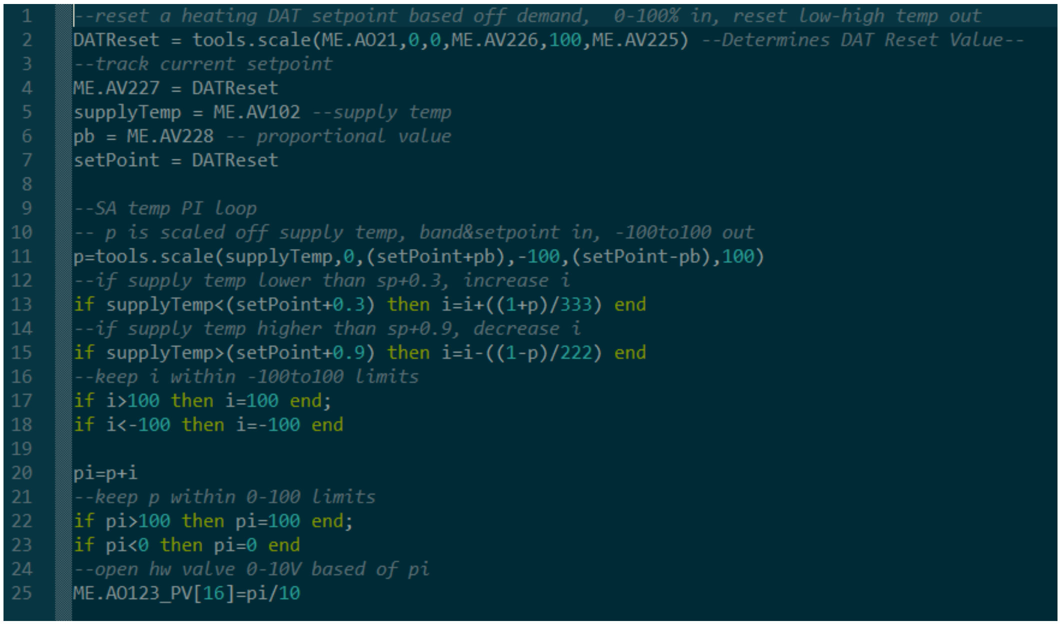


tools.scale()

tools.scale(variable,offset,x1,y1,x2,y2). This function returns the linear interpolation between two points. The function can also add the offset value to the final result if desired.

```
ME.AO123 = tools.scale(ME.AO21, 0, 0, 2, 100, 10)    --UO11(AO123) 2-10Vdc will follow the 0-100% PI_Heat (AO21)
```

PID Loop



```
1   --reset a heating DAT setpoint based off demand,  0-100% in, reset low-high temp out
2   DATReset = tools.scale(ME.AO21,0,0,ME.AV226,100,ME.AV225) --Determines DAT Reset Value--
3   --track current setpoint
4   ME.AV227 = DATReset
5   supplyTemp = ME.AV102 --supply temp
6   pb = ME.AV228 -- proportional value
7   setPoint = DATReset
8   
9   --SA temp PI loop
10  -- p is scaled off supply temp, band&setpoint in, -100to100 out
11  p=tools.scale(supplyTemp,0,(setPoint+pb),-100,(setPoint-pb),100)
12  --if supply temp lower than sp+0.3, increase i
13  if supplyTemp<(setPoint+0.3) then i=i+((1+p)/333) end
14  --if supply temp higher than sp+0.9, decrease i
15  if supplyTemp>(setPoint+0.9) then i=i-((1-p)/222) end
16  --keep i within -100to100 limits
17  if i>100 then i=100 end;
18  if i<-100 then i=-100 end
19  
20  pi=p+i
21  --keep p within 0-100 limits
22  if pi>100 then pi=100 end;
23  if pi<0 then pi=0 end
24  --open hw valve 0-10V based of pi
25  ME.AO123_PV[16]=pi/10
```

Other than changing the "p" variable, changing the upper and lower limits of "i" on lines 17 and 18 is another way to affect the behavior of the loop. Recently the need for this bit of code has been more common with an increased focus on meeting regulatory requirements within the sequence of operation (California Title 24 specifically).