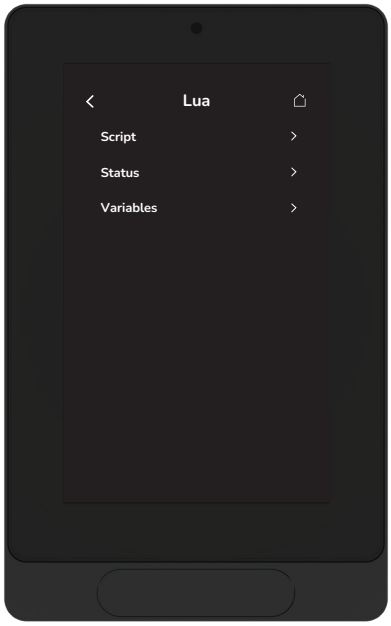
Viconics Room Controllers

Lua Programming Guide

Firmware Revision 2.1



Version 3

Contents

Safety Information

Important Information

Read these instructions carefully and inspect the equipment to become familiar with the device before trying to install, operate, service or maintain it. The following special messages may appear throughout this bulletin or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER
DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING
WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION
CAUTION indicates a hazardous situation which, if not avoided, **could result in** minor or moderate injury.

NOTICE
NOTICE is used to address practices not related to physical injury. The safety alert symbol shall not be used with this signal word.

Please Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Viconics Technologies for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction, installation, and operation of electrical equipment and has received safety training to recognize and avoid the hazards involved.

Lua4RC Programming

Viconics Room Controllers can run custom applications designed to meet specific customer requirements. These scripts, referred to as Lua4RC scripts, can be developed for Integrators, or by qualified Integrators. Lua4RC adds a layer of programming on top of the embedded control logic of a Viconics Room Controller.

The script running on the Room Controller has the ability to override parameters set by the embedded application. With this added flexibility, you can adapt the control logic of the Viconics Room Controllers to meet the specific requirements of your projects.

This section gives an overview of the basic functions of the Lua language. It is not an exhaustive and complete tutorial on the Lua language.

Accessing the Database

When writing a Lua4RC script, the keyword "**ME**" is used to access the objects in the local database. For example, a line that reads "**ME.AV25 = 10**" in a Lua script would set the value of the **AV25** object as 10.

The following object types are available:

1. AI (Analog Input)
2. AO (Analog Output)
3. AV (Analog Value)
4. BI (Binary Input)
5. BO (Binary Output)
6. BV (Binary Value)
7. MSI (Multi-state Input)
8. MV (Multi-state Value)
9. CSV (Character String Value)

Notes:

- A list of each point available, along with the description and possible values can be found in the Viconics Room Controllers BACnet Integration Guide.
- When accessing Binary Objects (BI, BO, BV), the return values are limited to 0 and 1, and not 'true' or 'false'.

Lua4RC Scripts

Lua scripts are stored in the Viconics Room Controller FLASH memory:

- Number of scripts: 1
- Script executed every: 1 second
- Maximum script size = 80 kB
- Script loaded via:
  - USB using the Room Controller Uploader tool (write only), or
  - BACnet File Object 1: "Custom Lua File" (read/write)

Standard Lua Library

The Viconics Room Controller Lua environment uses different versions of LUA. Refer to the following for additional information.

Firmware Revision 1.0

For an introduction to the basics of Lua programming, refer to https://www.lua.org/manual/5.1/manual.html#5, and for more general details on the Lua language, refer to https://www.lua.org/manual/5.1/.

Only basic functions and mathematical functions are implemented. Other libraries (advanced string manipulation, table manipulation, input and output facilities, operating system facilities, and debug libraries) are not available.

To use Lua libraries within a script, the “require” key work is used. Example usage in TRC 1.0:
require “bacnet”
require “math”

Firmware Revision 2.0

For 2.0, we upgraded to Lua 5.4, therefore, refer to https://www.lua.org/manual/5.4/ for more information.

To use Lua libraries within a script, the “require” key work is used. Example usage in TRC 2.0:
bacnet = require “bacnet”
math = require “math”

Lua Functions and Tools

These functions offer basic control over common Room Controller applications frequently used in most installations.

tools.switch()

Switch function (on-off with deadband). Simulates the operation of a conventional ON-OFF thermostat. It also provides a deadband function, so an Object does not continuously switch ON and OFF based on a specific value. output (0 or 1) =tools. switch(output, input-expr, off-expr, on-expr).

```
-- tools.switch()
--------------------------------------------------------------------------------
ME.BO5 = tools.switch(ME.BO5, ME.AO21, 0, 15)    --W1(BO5) will activate at 15% PI_
Heat demand, deactivate at 0%.

--The “switch” function is equivalent to:
if ME.AO21 > 15 then
ME.BO5 = 1
end

if ME.AO21 == 0 then
ME.BO5 = 0
end
```

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.

```
-- tools.scale()
--------------------------------------------------------------------------------
ME.AO104 = tools.scale(ME.AO22, 0, 0, 0, 100, 10)      --AO4 = Cool demand (0-100%)
scaled to 0-10 Vdc

ME.AO104 = ME.AO22 / 10        --Simplified version of above scale function

ME.AO104 = tools.scale(ME.AO22, 0, 0, 10, 100, 2)      --AO4 = Cool demand (0-100%)
scaled to 10-2 Vdc (N.O. 2-10 Vdc CWV)
```

os.date()

os.date (format,time). This function returns a string or a table containing date and time, formatted according to the given string format. If the time argument is present, this is the time to be formatted (see the os.time function for a description of this value). Otherwise, date formats the current time.

```
-- os.date()
--------------------------------------------------------------------------------
os = require 'os'

-- Get current time and print in Local and UTC
currentTime = os.time();
print("Local: "..os.date('%Y-%m-%d-%H:%M:%S', currentTime ))
print("UTC: "..os.date('!%Y-%m-%d-%H:%M:%S', currentTime ))

-- Show how to get parts of date/time (left = local, right = UTC)
print("date = ", os.date("%x"), " ", os.date("!%x"))
print("day = ", os.date("%A"), " ", os.date("!%A"))
print("month = ", os.date("%B"), " ", os.date("!%B"))
print("Time = ", os.date("%X"), " ", os.date("!%X"))
print("Hour = ", os.date("%H"), " ", os.date("!%H"))
print("Minute = ", os.date("%M"), " ", os.date("!%M"))
print("Second = ", os.date("%S"), " ", os.date("!%S"))
```

os.time()

os.time (table). This function returns the current time when called without arguments, or a time representing the local date and time specified by the given table. This table must have fields year, month, and day, and may have fields hour (default is 12), min (default is 0), sec (default is 0), and isdst (default is nil). Other fields are ignored. For a description of these fields, see the os.date function.

```
-- os.time()
--------------------------------------------------------------------------------
os = require 'os'

-- Get current time and print in Local and UTC
currentTime = os.time();
print("Local: "..os.date('%Y-%m-%d-%H:%M:%S', currentTime ))
print("UTC: "..os.date('!%Y-%m-%d-%H:%M:%S', currentTime ))
```

os.difftime()

os.date (t2,t1). This function returns the difference, in seconds, from time t1 to time t2 (where the times are values returned by os.time).

```
-- os.difftime()
--------------------------------------------------------------------------------
-- Initialisation
if not init then
    init = true
    os = require 'os'

    tStart = os.time()
end

-- Get current time
curTime = os.time();

-- Print the time from start
print ("Diff = ", os.difftime(curTime , tStart))
```

Scripting Best Practices

This section provides an overview of best practices when writing Lua4RC scripts.

Variable Declaration

Variable declarations should always be made at the beginning of a script and contained within an **"init"** statement. This is done to optimize CPU usage, processing time, proper initialization of values and is a general scripting good practice. The below shows an example.

```
--Variable declaration
-------------------------------------------------------------------------------
if not init then
ME.MV6 = 1              --Network units SI (C)              [forced]
ME.MV58 = 2             --Setpoints function = Attached     [forced]
ME.MV16_PV[17] = 2      --System Mode = Auto                [set at every boot-up,
released to be user-adjustable]
                        --(add other configurations here if needed)
dehum = 0               --Virtual variable to be used in the active part of the Lua
init = true
end
```

Priority Management

Lua4RC accesses various points of the Room Controller using BACnet naming convention and priorities. The default priority of the Room Controller's internal control sequence is 17. When writing a Lua4RC script, the default write priority is priority 16. As a result, the internal control is overridden by LUA commands such as "**ME.AV25 = 10**".

Apply caution when using priorities other than the default value as this could result in some values being permanently overridden. To write to another priority, an array is used.

The example below would write a value of 20 to AV25, using priority 8:

```
ME.AV25_PV[8] = 20
```

To release this specific priority, you can set it to nil:

```
ME.AV25_PV[8] = nil
```

To access the relinquish default (the Room Controller's internal logic application priority), priority 17 can be used.

```
ME.AV25_PV[17] = 30
```

NOTICE

PRIORITY LEVELS

Priority levels 1, 2, 3 and 17 (Relinquish Default) are stored in the non-volatile (EEPROM) memory of the Room Controller.

- This means the stored values will remain in the memory after a power cycle. It is necessary to perform a factory reset of the Room Controller to reset these values.
- Each location in the EEPROM memory is limited to 1 million (1,000,000) writes, so care must be taken to minimize changes written to priorities stored in EEPROM. Failure to do so will damage the device.
  - Do not write values that change regularly to priority levels stored in EEPROM. Use other priority levels that are stored in RAM only and support infinite read/write cycles.
  - Do not write to EEPROM locations on every cycle of the script (every 1 second). If you must write data in priority levels stored in EEPROM, do so less frequently (e.g. every 15 minutes).
  - Do not "NIL" a value before writing to it. This is unnecessary and causes two writes to the EEPROM location. For example, avoid the following:
    - ME.MV16_PV[17] = nil
    - ME.MV16_PV[17] = 2

Failure to follow these instructions can result in equipment damage.

Script / BACnet Variables

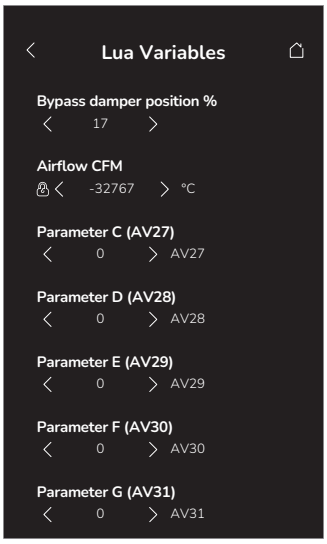
To interface between the Lua engine of the Room Controller and the BACnet integration, 18 variables are made available: **AV25** to **AV36** and **AV332** to **AV337**. These can be read and written from the Lua engine and BACnet. Since these variables are also visible and configurable from the Room Controller's HMI, they can be exposed to the User for quick customization or parametrization of points.

There are also 18 "scratchpad" variables that are available from the Lua engine and BACnet, but they are not visible from the Room Controller's HMI: **AV338** to **AV355**, named "Lua Scratchpad 1" to "Lua Scratchpad 18". These scratchpad variables are similar to the other Lua Variables, but they do not have the configurable name or limits. The scratchpad variables are editable via BACnet only. Use the scratchpad variables if your script has some configuration that you wish to expose to BACnet or store in non-volatile memory, but not to users in the Room Controller's HMI.

To change the name of one or more of these variables, the **_Desc** property of each point can be modified from a Lua script. This will be visible both in the HMI and BACnet.

To change the name of **AV25** (for example) to **Bypass damper position %** the following script can be used:

```
ME.AV25_Desc = "Bypass damper position %"
```



To hide a variable on the Room Controller's HMI, add an exclamation point "!" to the first character of the variable name:

```
ME.AV27_Desc = "!Parameter C (AV27)"
```

It is recommended to hide the variables that are not used in the Lua script.

Minimum / Maximum and Increment Values

To restrict the values of **AV25**-**AV36** and **AV332**-**AV337** to a certain range, the minimum and maximum acceptable values can be adjusted by modifying the **_Min** and **_Max** properties of each object:

```
ME.AV27_Min = "-10"
ME.AV27_Max = "10"
```

It is also possible to modify the increment property of a point. This will be used when adjusting the point on the Room Controller. An increment of 10 means the Room Controller will cycle between 0,10,20, etc. when using the up/down arrows, while an increment of 5 will cycle between 0,5,10,15, etc.

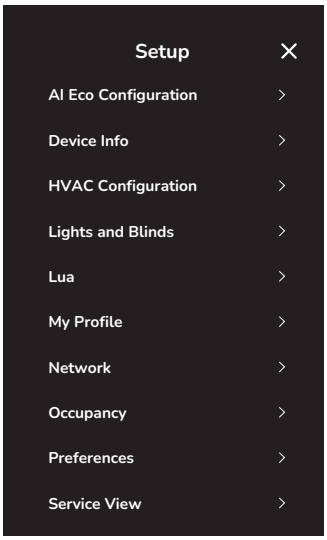
Note: If the increment is set to 0, the parameter is read-only. If the parameter is set to nil, the present value is not displayed.

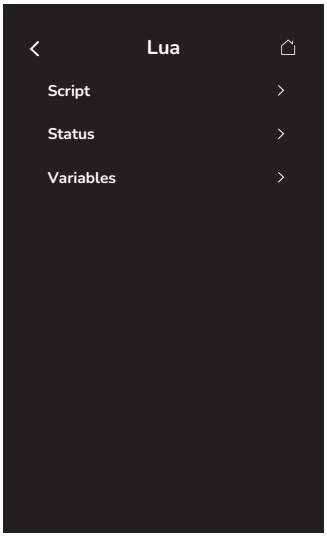
User Interface

This section describes using the Lua screens on the Viconics Room Controllers.

Access Lua Configuration Menu

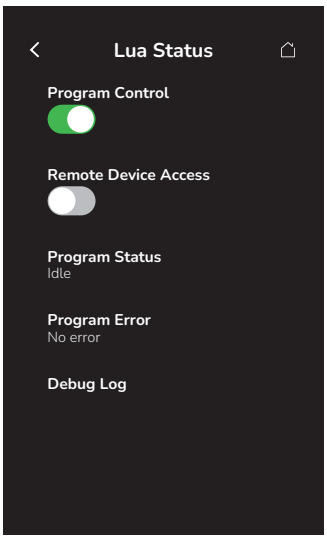
1. Tap the lower left corner of the Room Controller home screen.
2. Tap the arrow button to select Lua.





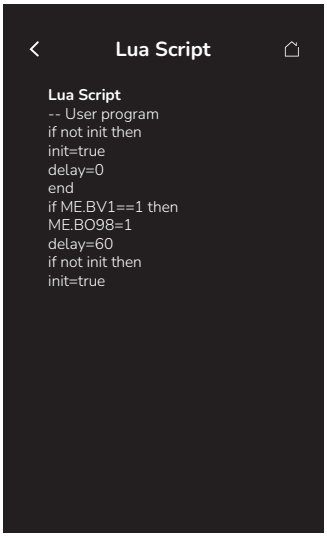
Lua Status Screen

This screen gives the option to run or stop the Lua script. When the Room Controller is started up and a script is loaded, the script is automatically in Running status, shown in the Program Status field.
If the script is not in Running status, the Program Error field shows if there is an error in the script. If the field shows anything other than No error, there are errors in the loaded script and it does not run. If there are errors, a message explaining the error(s) is shown in the Debug Log.
The most frequent error is, for example: “Unable to write to AV39_Present_Value[16]” and most of the time it is because the value is out of range. Writing a temperature in °F like Heating Setpoint = 73 when the Network units are set to default degree Celsius results in 73°C being out of range. To resolve this issue, set Network Units to °F (MV6=2) in the INIT section before writing to any temperature points.
The Debug Log can also be used for printing values from the script. The Debug Log window can hold a maximum of 78 characters on 3 lines. The Debug Log is refreshed every time the script loops back on itself.
Remote Device Access is only editable by an Administrator user. When enabled, Lua scripts running on the TRC are allowed to send BACnet read and write commands to other devices on the BACnet network. This feature should be kept disabled unless Lua access to remote devices is required.



Lua Script Screen

The Lua Status screen presents the first 10 lines of the script to help identify the script loaded.



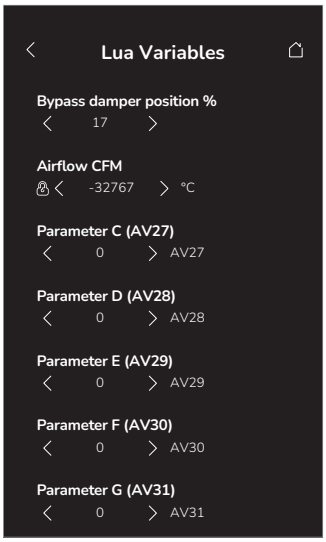
Lua Variables Screen

The Viconics Room Controller Lua environment offers 18 AV objects that can be used in the scripts: **AV25** to **AV36** and **AV332** to **AV337**. These objects are regular BACnet AV objects and are accessible directly from the screen. The values can be set in the user interface, and the user interface sets the value at the lowest level of priority (relinquish default level 17).

Any value gets overridden by a value set in a script or via BACnet. When the value is overridden, a padlock icon appears next to the value's text and it is not possible to change it in the interface. To release the override, the script or the BACnet client must set the value priority to nil.

IMPORTANT: Even if the script is not currently running, the variables used by the script get overridden and cannot be changed by the user.

The Lua Variable page is customizable as each variable can can be renamed, have its range limited, or can be completely hidden. Refer to "Script / BACnet Variables" on page 8 for more information.



Lua4RC Applied Examples

The section shows some common applied examples using Lua4RC.

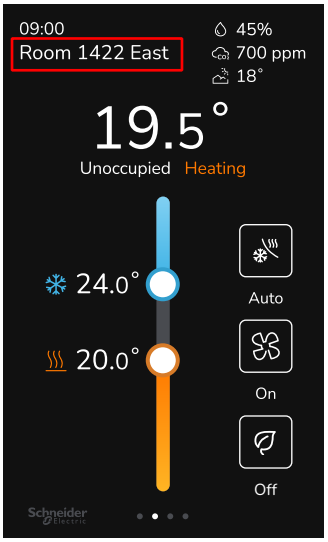
Custom Messages

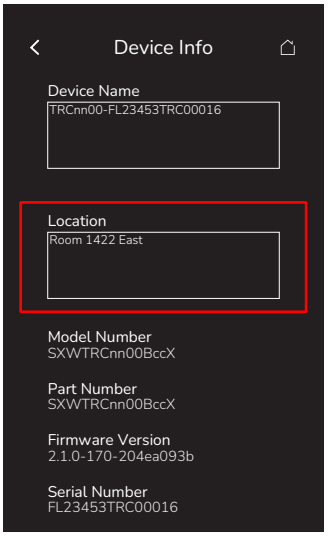
The Viconics Room Controller supports various methods of displaying information on screen from Lua or BACnet.

Location

The location text can be set from Lua and will be displayed on the home screen and on the Device Info setup screen. This can be used to describe the location of the device, or present any other short message to the user.

```
--Location:
-------------------------------------------------------------------------------
if not init then
ME.CSV35 = "Room 1422 East"
init = true
end
```





Short Screen Message Text (Scrolling Banner)

Notifications can be set from BACnet and Lua to appear at the top of the home screen, providing a highly visible method of giving information to the user. For example, a Lua script could read the state of an input that is connected to a custom sensor, then notify the user when the sensor activates.

Notifications consist of:

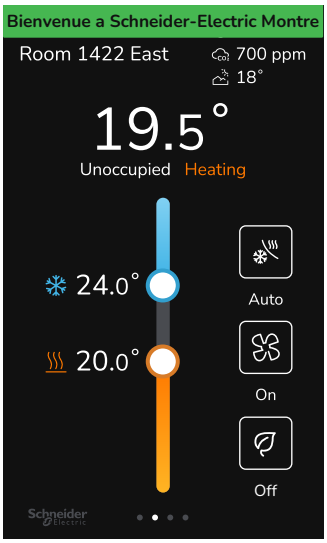
- A text message, which will scroll if it is longer than the width of the screen
- A type/color:
  - 1=Disabled
  - 2=Critical (Red)
  - 3=Warning (Yellow)
  - 4=Ok (Green)
  - 5=Informative (Blue)

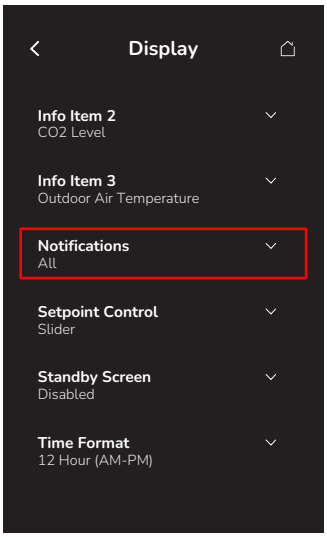
Note: For custom notifications to be displayed, "Notification Display Type" (MV187) must be set to 2 (Custom Only) or 3 (All).

```
--Short Screen Message Text (Scrolling banner):
-------------------------------------------------------------------------------
if not init then
ME.MV186=4          --Notification Type= Green
ME.MV187=3          --Notification Display Type= All

ME.CSV1 = "Bienvenue a Schneider-Electric Montreal  -  Welcome to Schneider-Electric
Montreal"

init = true
end
```





Standby Screen

The Room Controller supports the display of a standby screen with a full screen image supplied by the user, which can be loaded via: USB or BACnet.

The Standby Screen is enabled when a custom image is selected via the Preferences/Display menu, or on BACnet:

- Use Standby Screen:
  - BACnet ID = MV32
  - 1=Disabled (Default)
  - 2=Custom Image

Size and format:

- Resolution: 480 x 800 pixels
- File format: 24-bit bitmap (.bmp)

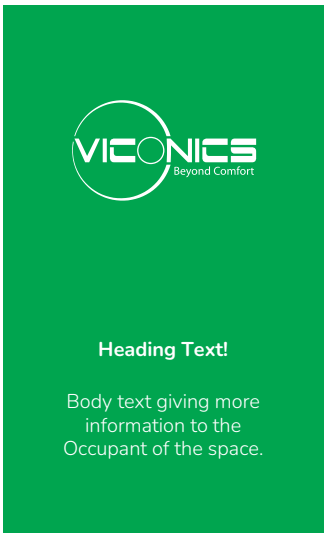
The text overlay has 3 properties:

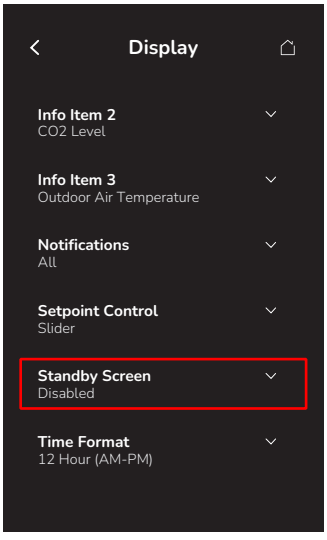
- Custom Standby Heading Text:
  - BACnet ID = CSV41
  - Maximum length: 64 characters
  - Value: Input Characters: ISO-8859-1 (Western Europe) character set
  - Displayed if string is not empty
- Custom Standby Body Text:
  - BACnet ID = CSV42
  - Maximum length: 160 characters
  - Value: Input Characters: ISO-8859-1 (Western Europe) character set
  - Displayed if string is not empty
- Custom Standby Text Color:
  - BACnet ID = MV190
  - 1=White (Default)
  - 2=Black

```
--Standby Screen:
--------------------------------------------------------------------------------
if not init then
ME.MV32=2          --Standby Screen= Custom Image
ME.MV190=1         --Custom Standby Text Color= White

ME.CSV41 = "Heading Text!"
ME.CSV42 = "Body text giving more information to the Occupant of the space."

init = true
end
```





Remote Wired Humidity Sensor

All Viconics Room Controllers have an internal Relative Humidity sensor. They can also use a remote wireless Zigbee sensor instead, but they do not have a dedicated input for a wired remote Relative Humidity sensor.

This LUA will configure UI8 Universal Input as 0-10 Vdc. Monitor the input to see if there is at least 0.5 Vdc (5% RH) on UI8. If that is the case, the LUA will automatically use this reading instead of the internal sensor (the LUA will update the value only if it is different from the previous value (COV)). If the reading is less than 0.2 Vdc (2% RH), the LUA will release the reading to the internal value as if the remote sensor is absent or defective.

```
--TRC Remote Humidity sensor on UI8.lua
--Remote humidity transmitter (0-10 Vdc = 0-100% RH) on UI8

-- Init
--------------------------------------------------------------------------------
if not init then

ME.MV148=3              --Set UI8 to Voltage
                        --(add other configurations here if needed)
init=true
end

-- Active script
--------------------------------------------------------------------------------
if ME.AV118>0.5 then    --If voltage on UI8 is greater than 0.5 Vdc (5% RH)
ME.AV103=ME.AV118*10    --Overwrite Room Humidity value with voltage x10

print "TRC uses remote RH sensor"   --Message displayed in "Lua/Lua Status/Debug Log"
TRC page
end

if ME.AV118<0.2 then    --If voltage on UI8 is lower than 0.2 Vdc (2% RH)
ME.AV103=nil            --Use internal Humidity sensor reading

print "TRC uses internal RH sensor" --Message displayed in "Lua/Lua Status/Debug Log"
TRC page
end
```

TR6500 Active Dehumidification

This LUA will do “active” dehumidification on a call for dehumidification, The Fan will be forced On as well as the Y1 and Y2 Cooling stages and W1 Heating stage. Dehumidification is also conditional to Occupancy, System Mode and Room Temperature.

```
--TRC65 Active Dehum Y1-Y2-W1.lua

-- Init
--------------------------------------------------------------------------------
if not init then

ME.MV6=2                --Network units Imperial (F)
dehum=0                 --virtual variable used as logic flag
                        --(add other configurations here if needed)
init=true
end

-- Dehum ON
--------------------------------------------------------------------------------
if ME.BV38==1 and ME.AV100>(ME.AV39-1) and ME.AV100<(ME.AV40+1) then
ME.BO4=1                --G forced On
ME.BO3=1                --Y1 forced On
ME.BO2=1                --Y2 forced On
dehum=1
ME.CSV35= “Dehum. ON”   --Display status message on TRC main page
end

if dehum==1 then
if ME.AV100<(ME.AV40-1.5) then ME.BO5=1 end          --W1 ON
if ME.AV100>(ME.AV40+0.5) then ME.BO5=0 end          --W1 Off
end

-- Dehum OFF
--------------------------------------------------------------------------------
if ME.BV38==0 or ME.AV100<(ME.AV39-3) or ME.AV100>(ME.AV40+3) then
ME.BO2=nil              --Y2 released to normal operation
ME.BO3=nil              --Y1 released to normal operation
ME.BO5=nil              --W1 released to normal operation
ME.BO4=nil              --G released to normal operation
dehum=0
ME.CSV35= “ ”           --Clear status message from TRC main page
end
```

TR6500 VFD Follow Heat-Cool Demand

This LUA will configure and use the AO4 output to control a fan equipped with a variable frequency drive (VFD). The fan is enabled/disabled by “G” (D04) output, the speed will follow the heating and cooling demand, scaled between the minimum and maximum speed, adjusted by the installer, in the TRC LUA Variables page.

```
--TRC65 VFD follow Heat-Cool demand.lua
--BO4 Fan authorisation (24 Vac)
--AO4 Output to VFD (0-10 Vdc)

-- Init
------------------------------------------------------------------------------
if not init then

ME.MV6=2                --Network units in deg. F
ME.MV99=1               --Set AO4 Output to Voltage

ME.AV25_Desc= "VFD Lowest speed (%)"      --VFD Lowest speed, Ex: 25% (2.5 Vdc), in
Fan_Mode Auto and Smart
ME.AV25_Min=10
ME.AV25_Max=100

ME.AV26_Desc= "VFD Highest speed (%)"     --VFD Highest speed, Ex: 65% (6.5 Vdc), in
Fan_Mode Auto and Smart
ME.AV26_Min=10
ME.AV26_Max=100

ME.AV27_Desc= "VFD speed in Fan_Mode ON (%)"    --VFD speed when Fan_Mode is set to
ON Ex: 45% (4.5 Vdc)
ME.AV27_Min=10
ME.AV27_Max=100

ME.AV28_Desc= " "       --Erase the field description
ME.AV29_Desc= " "       --Erase the field description

init=true
end

-- VFD speeds presets
------------------------------------------------------------------------------
if ME.AV25==0 then ME.AV25_PV[17]=25 end
if ME.AV26==0 then ME.AV26_PV[17]=65 end
if ME.AV27==0 then ME.AV27_PV[17]=45 end

-- VFD analog output scale based on Heat or Cool demand
------------------------------------------------------------------------------
if ME.BO4==1 then
ME.AO104=tools.scale((ME.AO21+ME.AO22),0,0,(ME.AV25/10),100,(ME.AV26/10))
else
ME.AO104=0
end
```

Miscellaneous Examples

Timer to Toggle a Binary Output (BO) Every 10 Seconds

```
-- Timer to toggle a Binary Output (BO), every 10 seconds:
-------------------------------------------------------------------------------
if not init then
timer = 0
init = true
end

if timer < 10 then
timer = timer + 1
else
ME.BO1 = (1 - ME.BO1)
timer = 0
end
```

Reverse Outputs

```
-- Reverse outputs:
-------------------------------------------------------------------------------
ME.BO1 = (1 - ME.BO1_PV[17])         --Reverse Binary Output

ME.AO101 = (10 - ME.AO101_PV[17])    --Reverse Analog Output
```

Validates Remote Wired Temperature Sensor and Average with TRC Internal Temperature Sensor

```
-- Validates remote wired temperature sensor and average with TRC internal
temperature sensor:
-------------------------------------------------------------------------------
if ME.AV121 ~= -40 then                    --If UI1 remote sensor is wired and good
ME.AV100 = (ME.AV100_PV[17] + ME.AV121) / 2     --average with internal sensor
else
ME.AV100 = nil    --If remote sensor is not wired or bad, use internal sensor only
end
```