

Lua4RC Custom Programming

Lua Programming for VT8000 Room Controllers



Table of Contents

Lua4RC Programming for VT8000	3
Accessing VT8000 Series Room Controller	3
Lua4RC Scripts	3
Standard Lua Library	4
Lua Functions and Tools	4
Section 1 - Scripting Best Practices	5
Variable Declaration	5
Memory Management	5
Priority Management	6
Script / BACnet Variables	7
Minimum/Maximum Increment Values	7
Section 2 - User Interface	8
Access Lua Configuration Menu	8
Lua Screen 1/3	8
Lua Screen 2/3	9
Lua Screen 3/3	9
Section 3 - Lua4RC Applied Examples	10
Pre-commission Points and Parameters	10
Custom Button Action	11
Configure Lua Parameter Page Title	13
Window Alarm Time Delay	14
ECM Fan Control	15
Section 4 - Miscellaneous Examples	17
Section 5 - Lua4RC Use Cases	18
Use Case 1 - Print Output	18
Use Case 2 - Debug Output	18
Use Case 3 - Database Read	18
Use Case 4 - Database Write	19
Use Case 5 - Database Write Error	19
Use Case 6 - Logic Operator (with Error)	20
Use Case 7 - DB Write to Specific Priority	21
Use Case 8 - Infinite Loop	21
Technical Support	22

Lua4RC Programming for VT8000

VT8000 series 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 VT8000 series Room Controller.

The scripts running on the Room Controller have the ability to override parameters set by the embedded application. With this added flexibility, you can adapt the control logic of the VT8000 Series Room Controllers to meet 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 VT8000 Series Room Controller 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 identify 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 (Multistate Input)
8. MV (Multistate Value)
9. CSV (Character String Value)

Note: when accessing Binary Objects (BI,BO,BV), the return values are limited to 0 and 1, and not ‘true’ or ‘false’. A list of each point available, along with the description and possible values can be found in the VT8000 Series Room Controllers BACnet Integration Guide.

Lua4RC Scripts

There are 2 distinct locations to store custom Lua scripts in a VT8000 Room Controller, each having different characteristics and limitations:

1. Flash memory
 - Single script
 - Run time 1 second
 - Maximum script size = 16 kB
 - Script loaded via USB using the VT8000 Uploader tool
2. PG objects on BACnet
 - Up to 10 scripts running in single thread
 - Run time 1 second for thread
 - Maximum script size = 420 bytes / script

It is possible to load scripts using both methods on the same Room Controller according to the following:

1. A script loaded to any PG object disables a script loaded to flash
2. A script loaded to flash will only be available if all PG objects are empty

Standard Lua Library

The VT8000 Lua environment uses LUA 5.1. Refer to the following for additional information:

<http://www.lua.org/manual/5.1/manual.html#5> for an introduction to the basics of Lua programming.

<http://www.lua.org/manual/5.1/> provides more general details on the Lua language.

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

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 dead-band 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).

```
ME.BO28 = tools.switch(ME.BO28, ME.AO21, 0, 15)  --W1(BO28) will go (ON at 15% PI_Heat) then (OFF at 0% PI_Heat)
```

The “switch” function is the equivalent of:

```
if ME.AO21 > 15 then
ME.BO28 = 1
end
if ME.AO21 == 1 then
ME.BO28 = 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.

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

Section 1 - Scripting Best Practices

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

Variable Declaration

Variable declaration should always be made at the beginning of a script and contained within an **“init”** statement. This is done to optimise CPU usage, processing time, proper initialisation of values and is a general scripting good practice.

The below shows an example.

```
if not init then
    ME.MV6 = 2    --Network units = °F
    ME.MV145 = 2 --Set Room_Temp sensor = Local
    init = true
end
-- rest of control script here, if applicable.
```

Memory Management

The amount of memory available to store scripts is not to be confused with the available RAM memory available to perform various functions. Also, the amount of consumed memory will vary depending on which functions are used along with some other variables.

Various steps can be taken to optimize the amount of memory used:

Tip #1: Keep your variable names short. A variable named “RoomTemperatureOverride” will use 23 bytes. Using “RTO” will only use 3 bytes.

Tip #2: Use **if-then-end** instead of multiple **elseif** statements.

Tip #3: Monitor the memory usage of the script by using the tools.memory() function. This function will display the amount of memory consumed in real time.

Usage example:

```
a = tools.memory()
print(a)
```

The output will be visible on the Lua screen of the VT8000, while the Debug Log will show a value in bytes.

Note: the value displayed represents the memory consumption for the entire Lua engine and not only for the script itself. As a result, an empty script would still display a value of around 9kb, and therefore, as a rule of thumb, usage should be kept to under 13kb.

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
```

Warning: priority levels 1, 2 and 3 are written 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 to the Room Controller to reset these values.

Script / BACnet Variables

To interface between the Lua engine of the Room Controller and the BACnet integration, six variables are made available: **AV25** to **AV30**. 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.

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 **Time delay (s)** the following script can be used:

```
ME.AV25_Desc = "Time delay (s)"
```

3/3 Lua	
Time delay (s)	5
Param. B (AV26)	0
Param. C (AV27)	8
Param. D (AV28)	0
Param. E (AV29)	0
Param. F (AV30)	0

Navigation buttons: Back, Forward, Home, Down, Up

Minimum / Maximum and Increment Values

To restrict the values of **AV25-AV30** to a certain range, the minimum and maximum acceptable values can be adjusted by modifying the **_Min** and **_Max** properties of each object.

Note: for Firmware version 1.4.2, **AV30** has a known issue where **_Min**, **_Max** and **_Inc** will not function.

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.

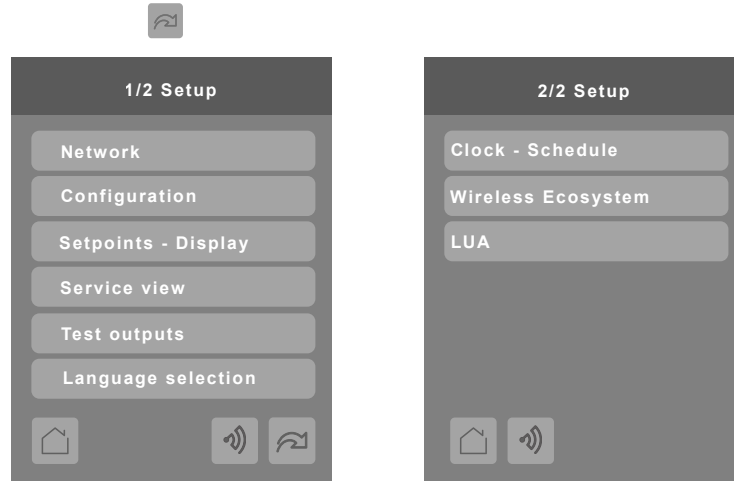
Note: If the increment is set to 0, the parameter is read-only. If the parameter is set to nil, the present value is not displayed.

Section 2 - User Interface

This section describes using the Lua screens on the VT8000 Series Room Controllers.

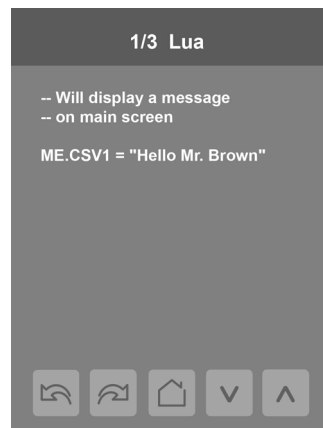
Access Lua Configuration Menu

1. Press and hold top-center area of Room Controller screen for 2 seconds.
2. Navigate to Setup page 2/2 by pressing arrow button and select Lua.



Lua Screen 1/3

Lua page 1/3 presents the first 10 lines of the first script. The up and down buttons enable browsing through the PG1 to PG10 objects loaded by BACnet, each of which can contain a separate script.

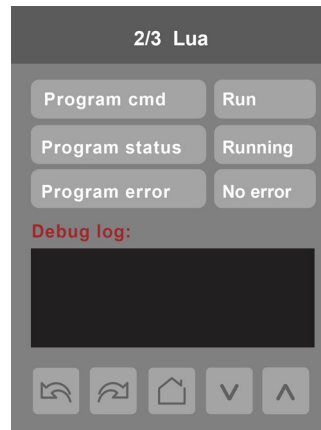


Lua Screen 2/3

This screen gives the option to run or stop the Lua scripts. 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. Note that Scripts loaded using BACnet cannot be controlled individually.

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 is an error or errors, a message explaining the error(s) shows in the Debug log.

The Debug log can also be used for printing values from the script. Print results from multiple scripts are concatenated in the Debug log window. 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.

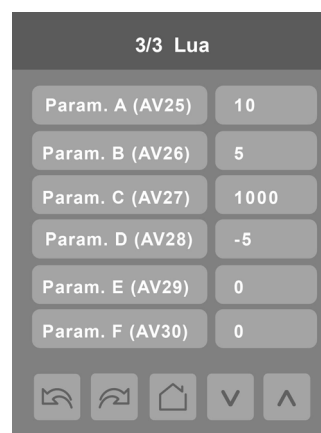


Lua Screen 3/3

The VT8000 Lua environment offers 6 AV objects that can be used in the scripts. 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, the value's text shows red 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 AV25 to AV30 variables, if used by the script, get overridden and can not be changed by the user.



Section 3 - Lua4RC Applied Examples

The section shows some common applied examples using Lua4RC.

Pre-commission Points and Parameters

This allows to quickly set all the necessary configuration points without having to use the HMI of the Room Controller at the time of installation. In this example, the points that are likely to be modified by the installer (or end-user) are set at priority 17 (relinquish default), while others are left at the default priority of 16.

```
if not init then
```

```
ME.MV6 = 2          --Set network units to deg F
ME.MV2_PV[17] = 5   --Dark grey
ME.AV2_PV[17] = 0    --User HMI = 0
ME.MV32_PV[17] = 2   --Standby screen ON
```

```
ME.MV101 = 1  --Disable French
ME.MV103 = 1  --Disable Chinese
ME.MV104 = 1  --Disable Russian
```

```
ME.MV10 = 1    --Local Occupancy
ME.MV16 = 2    --System_Mode Auto
ME.MV17 = 2    --Fan_Mode Auto
ME.MV119 = 2   --HPU
ME.MV117 = 1   --RV_V = 0
ME.AV75 = 1    --Single compressor
```

```
ME.AV63_PV[17] = 4    --Min Deadband
ME.AV39_PV[17] = 69   --Occ_Heat
ME.AV58_PV[17] = 80   --Heat SP limit
ME.AV40_PV[17] = 73   --Occ_Cool
ME.AV59_PV[17] = 60   --Cool SP limit
ME.AV45_PV[17] = 69   --Default Heat SP ... so Cool_SP will be 73
ME.AV67_PV[17] = 0.5  --Standby_Time (hr)
ME.AV68_PV[17] = 0.5  --Unocc_Time (hr)
```

```
ME.AV56_PV[17] = 1954 --Set password to "1954"
ME.AV25_Desc = "Room No."
```

















```
init = true
end
```

Custom Button Action

The section shows how the icon of the 5th button can be changed via MV115. This specific example will change the logo of the 5th button to Lighting and set the function to No Function. It will then toggle physical point Binary Output 8 (BACnet point BO98) when the button is pressed.

Note: The last button press variable (AV92) must be reset to 0 in the script.

The below tables show the various buttons along with descriptions.

Image	Icon: MV114	Icon description
	1	Default Button
No Button	2	No Button
	3	System Mode Heat/Cool
	4	System Mode On/Off
	5	Fan Mode
	6	Override Button
	7	Units Button
	8	Help Button
	9	Language Button
	10	Schedule Button
	11	Lighting Button
	12	Blind Button
	13	Lamp Button
	14	Energy Button
	15	Make Room Button
	16	Setting Button
	17	Timer Button

Function: MV115	Button Function
1	Default Function
2	No Function
3	System Mode Function
4	Fan Function
5	Override Function
6	Schedule Function
7	Units Function
8	Help Function
9	Language Function
10	Configuration Function
11	Custom Function
12	Standby Function (displays long message or standby image)

```

if init == nil then
  ME.MV114_PV[17] = 11 -- Custom button icon (11 - Lighting Button)
  ME.MV115_PV[17] = 2 -- Custom button behaviour (2 - No Function)
  init = true
end

if ME.AV92 == 5 then -- Last button pressed is 5th button
  ME.BO98 = 1-ME.BO98 -- Toggle BO 8 Auxiliary Binary Output between 0 and 1
  ME.AV92_PV[17] = 0 -- reset last button pressed
end
init = true
end

```

For reference purposes, the below shows the possible values for the last button pressed value (AV92):

- 1: Button 1 (lower-left)
- 2: Button 2
- 3: Button 3
- 4: Button 4
- 5: Button 5 (lower-right)
- 6: setpoint down-arrow
- 7: up-arrow
- 8: config (upper middle)
- 9: schedule (upper left or right corner)
- 10: other spot on the home screen
- 25 to 30: AV25..30 on custom page
- 35: home button in custom page(s)

Configure Lua Parameter Page Title

This script will change the title of the Lua parameter page to “Lights and Blinds”. The name will appear when the custom button is pressed and requires to be set to “custom”

```
if init == nil then
  ME.MV114 = 11 -- Set icon to light bulb
  ME.MV115 = 11 -- Set icon function to “custom”
  ME.MV114_Desc = “Lights & Blinds” -- rename the page title
init = true
end
```

Lights & Blinds

Param. A (AV25)	5
Param. B (AV26)	0
Param. C (AV27)	8
Param. D (AV28)	0
Param. E (AV29)	0
Param. F (AV30)	0

↶
↷
🏠
✓
^

Window Alarm Time Delay

In normal VT8000 operation, the window alarm (ME.BV35) is triggered as soon as a window opens (window contact status opened, ME.BV3). The Lua4RC script adds a delay before the alarm gets triggered and prints the current status of the window and alarm.

Inputs:

ME.AV25: time delay in minutes. (user defined)

ME.BV3: Window Contact Status

Outputs:

ME.BV35: Window Alarm

Script

Lua4RC_001_WindowAlarmDelay.lua content:

```
-- Lua4RC_001_WindowAlarmDelay.lua R1.0
-- Window Alarm triggered after AV25 minutes
-- Following detection of window opened via
-- Window Contact Status
if ME.BV3 == 1 then
if not t then t = 0 end
t = t + 1
print("Window opened for: " .. t .. " sec.")
if t > ME.AV25 * 60 then
print("Window alarm ON")
ME.BV35 = 1
else
print("Window alarm in " .. (ME.AV25 * 60) - t .. " sec.")
end
else
print("Window closed")
t = nil
ME.BV35 = 0
end
```

ECM Fan Control

This script adjusts UO9 output voltage accordingly to the fan speed output. When fan mode is auto, it sends a command to the fan that is a linear scaling of PI Heat or PI Cool (0-100) between AV25 and AV27 (minimum flow = 2 and maximum flow = 8).

Inputs

ME.AV25: minimum voltage

ME.AV26: medium voltage

ME.AV27: maximum voltage

Outputs

ME.AO125: UO9 output voltage Script

```
--Lua4RC_004_EcmFan.lua
--This script is required to add variable speed to fans in fan coil applications.
--Setup:
-- AV25 = min voltage
-- AV26 = med voltage
-- AV27 = high voltage
if not init then
--config UO9 to analog
ME.MV96 = 1
--Set Fan sequence to L-M-H-A
ME.MV57 = 3
init = true
else
--Safety check on AV's
if ME.AV25 < 0 then
ME.AV25_PV[17] = 2
end
if ME.AV26 < 0 then
ME.AV26_PV[17] = 5
end
if ME.AV27 < 0 then
ME.AV27_PV[17] = 8
end
--AV25 must be smaller than AV27
if ME.AV25 > ME.AV27 then
ME.AV25_PV[17] = 2
ME.AV27_PV[17] = 8
1
11
--Fan is on in l-m-h-a mode
if ME.BO95 == 1 or ME.BO96 == 1 or ME.BO97 == 1 then
--energize BO8 aux out
ME.BO98 = 1
--fan mode = low
if ME.MV17 == 1 then
print("FanMode = Low")
--ME.AV25 = 2
ME.AO125 = ME.AV25
end
--fan mode = med
if ME.MV17 == 2 then
print("FanMode = Med")
--ME.AV26 = 5
ME.AO125 = ME.AV26
end
--fan mode = high
if ME.MV17 == 3 then
print("FanMode = High")
--ME.AV27 = 8
ME.AO125 = ME.AV27
end
```

```
--Fan mode = auto
if ME.MV17 == 4 then
--if cooling
if ME.AO22 > 1 then
print("FanMode = Auto_Cooling")
--linear scaling of UO9 with PI_cool
ME.AO125 = tools.scale(ME.AO22, 0, 0, ME.AV25, 100, ME.AV27)
--if heating
elseif ME.AO21 > 1 then
print("FanMode = Auto_Heating")
--linear scaling of UO9 with PI_heat
ME.AO125 = tools.scale(ME.AO21, 0, 0, ME.AV25, 100, ME.AV27)
--No demand UO9= min voltage in AV25
else
print("FanMode = Auto_IDLE")
ME.AO125 = ME.AV25
end
end
--Fan is off
else
-- de-energize BO8 aux out
ME.BO98 = 0
print("FanMode = OFF")
end
end
```


Section 4 - Miscellaneous Examples

Reverse 0-100% input to 10-0 Vdc output

```
ME.AO123 = 10 - (ME.AO21/10) -- Modulating based on Heating Demand
```

Reverse 0-100% input to 10-0 Vdc output

```
ME.AO123 = 10 - (ME.AO21/10) -- Modulating based on Heating Demand
```

Convert a 0-100% input to 10-2 Vdc output

```
ME.AO123 = 10 - (0.8 * (ME.AO21/10)) -- Modulating based on Heating Demand
```

Reverse Y1 output

```
ME.BO26_PV[16] = (1 - ME.BO26_PV[17])
```

Average Room_Temp and Remote (use UI20 (RS) for remote sensor)

```
if init == nil then
  ME.MV145 = 2 --Set Room_Temp sensor = Local (this line goes in the "init" section)
  init = true
end
ME.AV100_PV[16] = (ME.AV100_PV[17] + ME.AV105) / 2
```

Use Raw Values for 0-10 Vdc inputs, below AI7 (UI23 set as Voltage (MV143 = 3))

```
ME.AV29 = tools.scale(ME.AI7,0,0,0,3469,100)
```

Section 5 - Lua4RC Use Cases

The following use cases show the behavior (program results, debug log print out) of the Lua4RC engine in specific use cases.

Use Case 1 - Print Output

Description: Correct use of `print()` function.

Action:

Program: `print("Hello World")`

Result:

Program status: Running

Program error : No error

Debug log:

USER: xx/xx/xxxx xx:xx:xx >

Hello World

Use Case 2 - Debug Output

Description: Calling a non-existing function (print instead of print)

Action:

Program: `print("Hello World")`

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log::

ERROR: xx/xx/xxxx xx:xx:xx >

Line 1: attempt to call global

'print' (a nil value)

Use Case 3 - Database Read

Description: Correct printing of the concatenation of a string and the value of a database object.

Action:

Program: `print("ME.AV40 value is: " .. ME.AV40)`

Result:

Program status: Running

Program error : No error

Debug log:

USER: xx/xx/xxxx xx:xx:xx >

ME.AV40 value is: 75

Use Case 2.1 - Database Read error

Description: Error caused by the concatenation of a string and a nil value (AV0 does not exist)

Action:

Program: `print("ME.AV0 value is: " .. ME.AV0)`

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log:

ERROR: xx/xx/xxxx xx:xx:xx >

Line 1: attempt to concatenate field

'AV0' (a nil value)

Use Case 4 - Database Write

Description: Correct writing of a value to a database object.

Action:

Program: ME.AV41 = 50; error("ME.AV41 value is: " .. ME.AV41)

Result:

Program status: Running

Program error : No error

Debug log:

USER: xx/xx/xxxx xx:xx:

Use Case 5 - Database Write Error

Description: Error caused by an attempt to write to a nonexistent database object

Action:

Program: ME.AV1 = 50; error("ME.AV1 value is: " .. ME.AV1)

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log:

ERROR: xx/xx/xxxx xx:xx:xx >

Unable to find

AV1_PV

Description: Error caused by an attempt to write an invalid value to a database object

Action:

Program: ME.AV41 = 25; error("ME.AV41 value is: " .. ME.AV41)

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log:

ERROR: xx/xx/xxxx xx:xx:xx >

Unable to write to

AV41_PV

Use Case 6 - Logic Operator (with Error)

Description: Correct usage of `if/then/else` statement.

Action:

Program: `if not init then print("init done");init= true end`

Result:

Program status: Running

Program `error` : No `error`

Debug `log`:

USER: `xx/xx/xxxx xx:xx:xx >`

`init done`

Use Case `4.1` - Logic Operator `error`

Description: Incorrect use of Lua keyword (`en` instead of `end`)

Action:

Program: `if not init then print("init done");init=true;en;`

Result:

Program status: Idle

Program `error` : Syntax

Debug `log`:

ERROR: `xx/xx/xxxx xx:xx:xx >`

Line `1`: `'='` expected near `';`

Use Case 7 - DB Write to Specific Priority (with Error)

Description: Correct way of writing to a specific priority of a database object.

Action:

Program: ME.AV41_PV[5] = 50; print("ME.AV41 at priority 5 value is: " .. ME.AV41_PV[5])

Result:

Program status: Running

Program error : No error

Debug log:

USER: xx/xx/xxxx xx:xx:xx >

ME.AV41 at priority 5 value is: 50

Description: Incorrect way to write to the specific object of a database object.

Action:

Program: ME.AV41[5] = 50; print("ME.AV41 at priority 5 value is: ",ME.AV41[5])

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log:

ERROR: xx/xx/xxxx xx:xx:xx >

Line 1: attempt to index field

'AV41' (a number value)

Description: Correct way to read the value of a specific priority of a database object.

Action:

Program: print("ME.AV41 at priority 5 value is: " .. ME.AV41_PV[5])

Result:

Program status: Running

Program error : No error

Debug log:

USER: xx/xx/xxxx xx:xx:xx >

ME.AV41 at priority 5 value is: 50

Description: Incorrect way to read the value of a specific priority of a database object.

Action:

Program: print("ME.AV41 at priority 5 value is: ",ME.AV41[5])

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log:

ERROR: xx/xx/xxxx xx:xx:xx >

Line 1: attempt to index field

'AV41' (a number value)

Use Case 8 - Infinite Loop

Description: Error caused by an infinite loop.

Action:

Program: i=1; j= 1; while (i== 1) do j=j+ 1; end;

Result:

Program status: Running/Waiting each 10s

Program error : Runtime

Debug log:

ERROR: xx/xx/xxxx xx:xx:xx >

Line 1: instruction limit reached