eScada

v24.2.0

Lua code

Table of Contents

Lua language

Lua language has been embedded in eScada to offer an efficient way to operate with data.
It is a powerful programming language which extends and completes derived tags objects.
An advanced use of Lua permits to integrate other external libraries written using C, C++ and C# languages.
However this document it is not intended as a complete Lua language manual or whatever regarding Lua; it offers the indications useful to operate with eScada objects using a modern programming language.

For convenience the following content offers a direct point of view regarding how to start writing Lua code.

These are Lua links useful to learn and understand in details what Lua is able to offer

Lua web site:	https://www.lua.org/
Lua about:	https://www.lua.org/about.html
Lua documentation:	https://www.lua.org/manual/5.4/

Before starting using Lua language and its interaction with eScada objects, we recommend of reading the Lua documentation chapters 2, 3 and 6

The use of ChatGPT could represent a valid help in order to get short examples about routines you may need, than it will be easy to replace eScada objects in such an examples.

Values and types

Lua is a dynamically typed language. This means that variables do not have types; only values do. There are no type definitions in the language. All values carry their own type ...

... please continue reading from here

Lexical conventions

Lua is a free-form language. It ignores spaces and comments between lexical elements (tokens), except as delimiters between two tokens. In source code, Lua recognizes as spaces the standard ASCII whitespace characters space, form feed, newline, carriage return, horizontal tab, and vertical tab.

Names (also called identifiers) in Lua can be any string of Latin letters, Arabic-Indic digits, and underscores, not beginning with a digit and not being a reserved word. Identifiers are used to name variables, table fields, and labels ...

... please continue reading from here

Visibility rules

Lua is a lexically scoped language. The scope of a local variable begins at the first statement after its declaration and lasts until the last non-void statement of the innermost block that includes the declaration ...

... please continue reading from here

Common language statements

Here some example of common Lua language statements.

const

In case you need to define constants variables, here you can find the syntax to use.

```
-- Constant definition
local pi<const> = 3.14
```

for

The loop can be interrupted by using the instruction break

```
-- Iterate from 1 to 5
for i = 1, 5 do
    -- do something with loop
end

-- Iterate from 2 to 10 with a step of 2
for i = 2, 10, 2 do
    -- do something with loop
end

-- Countdown from 10 to 1
for i = 10, 1, -1 do
    -- do something with loop
end
```

while

The loop can be interrupted by using the instruction break

```
-- Using a while loop with floating-point values
local i = 1.0
while i <= 5.0 do
    -- do something with loop

    -- increment
    i = i + 0.5
end
```

repeat

The loop can be interrupted by using the instruction break

```
local count = 1
repeat
    -- do something with loop

    -- increment
    count = count + 1
until count > 5
```

if

```
local temperature = 25

if (temperature > 30) then
    -- do something
elseif (temperature > 20) then
    -- do something
else
    -- do something
end
```

tables

A Lua table is like a versatile data structure that shares similarities with C++'s std::map, std::vector, and objects. It can store various types of data and associate them with keys. Similar to a std::map, you can use any type as the key (not just integers), and associate any type of value with that key.

In C++ terms:

- A Lua table is like a dynamic array that doesn't require a predefined size. You can append elements or insert them at specific positions.
- It's similar to std::map, allowing you to associate values with keys, but with even more flexibility in key types.
- Tables can behave like C++ objects, allowing you to store multiple variables and functions together.

Here's an analogy:
Imagine you have a C++ container that can store a variety of data (ints, strings, objects) and you can access them using keys (integers, strings, objects). In Lua, a table is like that container: it can hold different types of data associated with keys that can be almost anything.

In summary, a Lua table is a powerful, dynamic, and all-encompassing data structure that can act as an array, a dictionary, or an object, depending on how you use it. It's a fundamental feature of Lua that enables you to handle a wide range of data storage and manipulation tasks.

Example 1 - (array)

```
-- Creating a table as an array
local fruits = {"apple", "banana", "orange", "grape"}

-- Accessing array elements
local fruit1 = fruits[1] --  apple
local fruit2 = fruits[2] --  banana
local fruit3 = fruits[3] --  orange
local fruit4 = fruits[4] --  grape

-- Append a new element
fruits[#fruits+1] = "strawberry"

-- Inserting an element between 2 and 3
table.insert(fruits, 3, "cherry")
```

```
-- Iterating over the array
for i = 1, #fruits do
    if (fruits[i] == “orange”) then
        break
    end
end
```

Example 2 – (2D array)

```
-- Creating a 2D array with initial values
local rows = 3
local cols = 4
local array2D = {}

--  Initialize
for r = 1, rows do
    -- Create a new row
    array2D[r] = {}

    -- Initialize values for each column in the row
    for c = 1, cols do
        array2D[r][c] = 0
    end
end

-- Accessing values from the 2D array
for r = 1, rows do
    for c = 1, cols do
        -- Check value
        if (array2D[r][c] == 12) then
          -- do something ...
        end
    end
end
```

Example 3 – (3D array)

```
-- Creating a 3D array with initial values
local xSize = 2
local ySize = 3
local zSize = 4
local array3D = {}

for x = 1, xSize do
    -- Create a new 2D "slice"
    array3D[x] = {}

    for y = 1, ySize do
        -- Create a new row within the slice
        array3D[x][y] = {}

        -- Set init values for each cell
        for z = 1, zSize do
```

```
            array3D[x][y][z] = 0
        end
    end
end

-- Accessing values from the 3D array
for x = 1, xSize do
    for y = 1, ySize do
        for z = 1, zSize do
            -- Check value
            if (array3D[x][y][z] == 12) then
                -- do something ...
             end
        end
    end
end
```

Example 4 - (struct/class)

```
-- Creating a Lua table with different kinds of data
local person = {
    name = "John Doe", -- string
    age = 30,               -- integer
    isStudent = false,     -- boolean
    -- This is a table
    contact = {
        email = "john@example.com",
        phone = "123-456-7890"
    },
    -- This is a table
    hobbies = {"reading", "hiking", "photography"}
}

-- Accessing the table's data
local name =  person.name
local age =  person.age
local email =  person.contact.email
local phone =  person.contact.phone

-- is he a student ?
if ( person.isStudent) then
    -- do something
end

-- Iterating over the hobbies
for i = 1, #person.hobbies do
    if (person.hobbies[i] == “hiking”) then
        -- do something
    end
end
```

Example 5 – (array of struct)

```lua
-- Define a function to create a new person
local function createPerson(name, age, isStudent, email, phone)
    -- Here the function returns a table
    return {
        name = name,
        age = age,
        isStudent = isStudent,
        -- This is a table
        contact = {
            email = email,
            phone = phone
        }
    }
end

-- Create an array of persons
local people = {
    createPerson("John Doe", 30, false, "john@example.com", "123-456-7890"),
    createPerson("Jane Smith", 25, true, "jane@example.com", "987-654-3210")
}

-- Accessing by index
local name1 = people[1].name --  name
local name2 = people[2].name --  name

-- Iterating over the array
for i, person in ipairs(people) do
    -- is he a student ?
    if (person.isStudent) then
        -- do something
    end
end
```

text files

Lua provide a standard library to operate with files called io, please refer to its documentation from information about it. [https://www.lua.org/manual/5.4/manual.html#6.8]

Before using files to read and write data, please read carefully the paragraph called “Common recommendations”.

Here some common example useful to understand the basic.

Example 1 – (read)

```
-- Open the file in read mode
local file = io.open("C:/temp/example.txt", "r")

if file then
    -- Read and process each line
    for line in file:lines() do
        -- You can perform any processing on the 'line' variable here
    end

    -- Close the file
    file:close()
else
    -- Error opening the file
end
```

Example 2 – (write)

```
-- Open the file in append mode
local file = io.open("C:/temp/example.txt", "w")

if file then
    -- The data to append
    local content = "This is the file content"

    -- Write the new data to the file
    file:write(content)

    -- Close the file
    file:close()
else
    -- Error opening the file
end
```

Example 3 – (append)

```
-- Open the file in append mode
local file = io.open("C:/temp/example.txt", "a")

if file then
    -- The data to append
    local newline = "This is a new row of data</n>"

    -- Write the new data to the file
    file:write(newline)

    -- Close the file
    file:close()
else
    -- Error opening the file
end
```

functions

Lua functions are blocks of code that can be defined and called to perform specific tasks.
One notable feature of Lua functions is their ability to return multiple values simultaneously.
This is achieved by using a single return statement that includes all the values you want to return, enclosed in parentheses.
This flexibility allows Lua programmers to efficiently bundle and retrieve multiple results from a function call, enhancing the language's expressiveness and versatility.

Example 1 – (return 1 parameter)

```
function addNumbers(a, b)
    local sum = a + b

    -- return value
    return sum
end

-- Function call
local sum =  addNumbers(5, 9)
```

Example 1 – (return 3 parameters)

```
function calculateStats(numbers)
    local average = 0
    local count = 0
    local sum = 0

    -- sum
    for i, num in ipairs(numbers) do
        sum = sum + num
        count = count + 1
    end

    -- average
    if (count > 0) then
        average = sum / count
    end
```

```
    -- return values
    return {sum = sum, average = average, count = count}
end

-- Function call
local numbers = {10, 20, 30, 40, 50}
local stats = calculateStats(numbers)

-- Values
local sum =  stats.sum
local average =  stats. average
local count =  stats. count
```

Comments

Comments can be used everywhere in Lua code

```
 --[[
    This is a multi rows comment section

    my own comment ....
    bla, bla, bla ....
 ]]--

 -- single row comment ....
```

eScada Lua module body

This is only an example as suggested way to write Lua Modules

```
-- Main function, from which to call the rest of the module's functions
function main()
     -- Module functions calls
    MyFunction01()
end

function MyFunction01()
end

 --[[
   Code execution starts here
   The rest of the module's functions should be written above this line
 ]]--
main();
```

Lua & eScada objects

In eScada you can insert Lua modules in two ways:

1) the first one is a new derived object called “LuaModule”, such object behaves like any other derived tags belonging to “Actions” family; it can be executed by using a trigger or evaluated from a script or another Lua module.

2) The second way to execute Lua code is to include, within an eScada project, external Lua files located in a new folder called “lua”, belonging to the eScada’s project folder; modules belonging to this context, can be imported everywhere in other modules.

For user convenience a new context called “Lua” has been added, it is reachable by using a new toolbutton in the application toolbar.
It is the context which should be used as container for large modules written using Lua, here you can find tools to manage the two ways mentioned above.
In this context even other kind of derived tags can be declared, of course they should be tag in relation or used within Lua modules.

Derived objects of type “LuaModule” can be inserted as normal derived tags using the known "Derived" context, we suggest to use this way to execute small Lua code, for large modules please use the new context “Lua” instead.
The new Lua context should be a more comfortable place to manage large Lua modules.

The entire Lua code you can use in a project is able to interact with eScada objects.
Actually eScada exposes five types of object which can be used within Lua code, they are called: esProject, esSystem, esUtility, esString and esTags, you can find more details and examples about them later in this document.

Common recommendations

Folders or names separator

Wherever you need to type paths, even for windows context, it is necessary to insert them using the “slash character /” as folder separator.
The same suggestion must be followed to insert paths useful to express eScada objects names.

Folders: C:/temp/data, //folder1/folder2/folder3, /home/myfolder

Names: mychannel/mydevice/mygroup

How to declare eScada tags objects

eScada tags objects are contained in a container called esTags, before operating with tags it is necessary to declare them using the suggested way below.

- correct tag object declaration

```
local pTag = esTags[“mytagname”]
```

It is highly recommended to use this mode to declare a tag object, because in this way every time you need to rename a tag in device or derived context, the system will be able to replace the new tag name even in Lua modules.

- wrong tag object declaration

```
local tagname = “mytagname”
local pTag = esTags[tagname]
```

Even though this way to use tag objects will work as expected, in case of tag renaming the system won't be able to replace the new tag name in Lua modules.

In certain cases it is not possible to follow such a recommendation, so is up to the user review its code in case of tags renaming.

How to use tags name

Wherever in Lua code it is necessary to use the tag name, please proceed as described below.

```
- correct use of a tag object name
local pTag = esTags[“mytagname”]
local tagname = pTag:Name()
```

Before using tag name it is highly recommended to declare the tag object as shown, after that you can use the method Name() in order to get the tag name.
This suggested way keeps your code safe even in case of tag renaming.

Special characters

In case in a string you need to specify a system character like, **\t** for tab, **\n** for a new line, **\r** for carriage return, we recommend to use the following syntax instead, which will be replaced with the right system character:

- Tab, **</t>** ASCII 9
- New line, **</n>** ASCII 10
- Carriage return, **</r>** ASCII 13

remark:
The syntax above must be respected in Lua module belonging to internal eScada context tags, it can be skipped in case you are editing external Lua modules belonging to the external Lua project folder; in that case you must use the normal Lua syntax for those system characters.

UTF8 strings

Later on this document, there is a section dedicated to functions useful to operate with strings.
All those functions are able to manipulate UTF8 strings without any restrictions.

External modules

Every project has got a folder called “lua”, it is the folder where you can store external Lua modules.
In order to include and use such a files into a Lua module, it is necessary to copy and paste the following code at the beginning of a Lua module.

```
--[
  These lines of code add the project paths for any external lua modules
  Warning that renamed external folders are not updated here, it must be done manually
]--

-- it adds the main Lua folder
package.path = esProject:GetLuaPath() .. "/?.lua;" .. package.path

-- it adds another folder; you have to add as much lines as your folders in Lua folder are.
package.path = esProject:GetLuaPath() .. "/myfolder/?.lua;" .. package.path
```

esProject

This is an object that can help to get some information about the project which eScada server is running.
Among project system tags, it exists a group called $SYS.Server which contains tags called $SYS.Server.SHMI and $SYS.Server.SIHMI with other useful project information, please refer also to them for topics not included in the following methods.

Regarding this object, for some methods where it is necessary to express an object name, it is useful to keep in mind that:
- A channel name is unique among the entire project.
- A device name is unique among the entire channel.
- A device group name is unique among the entire device.
- A derived group name is unique among the entire derived channel.

IsReady()

It is used to check whether eScada server is initialized and ready to operate or not

parameters: none

return value: boolean
false the server is not ready
true the server is ready

example:

```
if (esProject:IsReady()) then
  -- do something with server ready
else
  -- the server is not ready yet
end
```

GetName()

It returns the project name.

parameters: none
return value: string

example:

```
local prjname = esProject:GetName()
```

GetPath()

It returns the project path.

parameters: none
return value: string

example:

```
local prjpath = esProject:GetPath()
```

GetLuaPath()

It returns the Lua path project folder.

parameters: none
return value: string

example:

```
local luapath = esProject:GetLuaPath()
```

Command(x)

It executes a system command within the current project.

parameters:
x[1] integer

return value: boolean
false the given command value is within the valid range
true the given command value is invalid

[1] Admitted command value for the function

0, Logoff all users from their connected clients
1, Acknowledge all active alarms
2, Acknowledge all active user messages
3, Unlock recipes
4, Disconnect all connected clients
5, Disconnect all connected clients except for the first one
6, Shutdown all connected clients computer
7, Shutdown all connected clients computer except for the first one
8, Shutdown the server's computer
9, Save project's tags value
(*All tags with the "Save" option active*)
10, Shows or hides server GUI

example:

```
-- This command will save all tags value
esProject:Command(9)
```

SetOffLine(path, b)

It is useful for changing the offline status for a device object such a channel or device.
When the server starts, all objects belonging to device context used for such a function, must be enabled.

parameters:
path [1] string
b[2] boolean

return value: none

[1] The path must express an existing object, here some examples:
"mychannel" or "mychannel/mydevice" or "mychannel/mydevice/mygroup"
remark: this function rises an error during its runtime execution if the given object doesn't exist or its state is invalid, for instance it is disabled.

[2] true, the offline state for the given object is activated
false, the offline state for the given object is deactivated

example:

```
esProject:SetOffLine("MDBTCP", true)                -- Entire channel
esProject:SetOffLine("MDBTCP/PLC", true)            -- Entire device
esProject:SetOffLine("MDBTCP/PLC/GroupName", true)  -- Entire device group
```

IsOffLine(path)

It is useful for checking the offline status for a device object such a channel or device.

parameters:
path [1] string

return value: boolean
false the given object offline status is not active
true the given object offline status is active

[1] The path must express an existing object, here some examples:
"mychannel" or "mychannel/mydevice" or "mychannel/mydevice/mygroup"
remark: this function rises an error during its runtime execution if the given object doesn't exist or its state is invalid, for instance it is disabled.

example:

```
if esProject:IsOffLine("MDBTCP") then               -- Entire channel
...

if esProject:IsOffLine("MDBTCP/PLC") then           -- Entire device
...

if esProject:IsOffLine("MDBTCP/PLC/GroupName") then -- Entire device group
...
```

IsChannelEnabled(name)

It is useful for checking the enable status for a channel.

parameters:
name [1] string

return value: boolean
false the given channel is disabled
true the given channel is enabled

[1] The name must express an existing object, here an example: “mychannel”
remark: this function rises an error during its runtime execution if the given object doesn’t exist.

example:

```
if (not esProject:IsChannelEnabled("mychannel")) then
  -- do something when channel is disabled
end
```

The following system channel names can be used:

$CHNOTIFICATIONS	Notifications context
$CHTRENDS	Trends contexts
$CHRECIPES	Recipes contexts

Other system channels like $CHSYSTEM and $LUACODE are always enabled.

IsDeviceEnabled(path)

It is useful for checking the enable status for a device.

parameters:
name [1] string

return value: boolean
false the given device is disabled
true the given device is enabled

[1] The name must express an existing object, here an example: “mychannel/mydevice”
remark: this function rises an error during its runtime execution if the given object doesn’t exist.

Example:

```
local IsDeviceEnabled = esProject:IsDeviceEnabled("mychannel/mydevice")
if (IsDeviceEnabled) then
  -- do something when device is enabled
else
  -- do something when device is disabled
end
```

IsGroupEnabled(path)

It is useful for checking the enable status for a group of tags belonging to device or derived contexts.

parameters:
name [1] string

return value: boolean
false the given group is disabled
true the given group is enabled

[1] The name must express an existing object, here some examples:
for device context "mychannel/mydevice/groupname"
for derived context "mychannel/groupname"
remark: this function rises an error during its runtime execution if the given object doesn't exist.

Example:

```
local IsGroupEnabled = esProject:IsGroupEnabled("mychannel/mydevice/mygroup")
if (IsGroupEnabled) then
  -- do something when device is enabled
else
  -- do something when device is disabled
end
```

ReadAll(path)

It is used to perform a read action for all tags belonging to a device group.
This is useful in case the tags polling value has been defined to zero.

parameters:
path[1] string

return value: boolean
false the reading action couldn't be executed due to an error
true the reading action has been executed successfully

[1] The path must express an existing object, here an example "mychannel/mydevice/mygroup"
remark: this function rises an error during its runtime execution if the given object doesn't exist or its state is invalid, for instance it is disabled.

example:

```
local mygroup = "mychannel/mydevice/mygroup"

if (esProject:ReadAll(mygroup)) then
  -- reading action has been executed successfully
else
  -- something went wrong
end
```

WriteAll(path)

It is used to perform a write action for all tags belonging to a device group.
This is useful in case the tags polling value has been defined to zero.

parameters:
path [1] string

return value: boolean
false the writing action couldn't be executed due to an error
true the writing action has been executed successfully

[1] The path must express an existing object, here an example "mychannel/mydevice/mygroup"
remark: this function rises an error during its runtime execution if the given object doesn't exist or its state is invalid, for instance it is disabled.

example:

```
local mygroup = "mychannel/mydevice/mygroup"

if (esProject:WriteAll(mygroup)) then
  -- writing action has been executed successfully
else
  -- something went wrong
end
```

UploadValues(path, file, mode)

It is used to save or load tag items value with the attribute 'Upload' active.
The attribute 'Upload' must be active either on tag and its items.

parameters:
path[1] string
file[2] string
mode[3] integer

return value: integer
0, Execution OK
1, Path doesn't exist
2, Can't open file for writing
3, Can't open file for reading
4, The folder is not writeable

[1] Path to folder
[2] File name + extension
[3] 0=Save, 1=Load

Paths and File name parametrisation
Concerning path and file parameters, they can contain these variables in their text:

%d	Actual day
%m	Actual month
%y	Actual year
%H	Actual hour
%M	Actual minute
%S	Actual second
%t	Actual millisecond
{PP}	Project path (**remarks:** please use it only for path)

example:

```
-- Save values
local path = esSystem:GetUserPath() .. “/documents”
local err = 0

-- Execution
err = esProject:UploadValues(path, "%y-%m.txt", 0)

-- Execution evaluation
if (err == 0) then
  -- execution without errors
else
  -- execution with errors
end
```

esSystem

This is an object that can help to get some information about the system where eScada is running. Among project system tags, it exists a group called $SYS.Server which contains system tags with other useful system information, please refer also to them for topics not included in the following methods.

GetTempPath()

It returns the temporary directory for the current user.

parameters: none
return value: string

example:

```
local temppath = esSystem:GetTempPath()
```

GetUserPath()

It returns the user folder which normally contains other folders such a documents, downloads, pictures, videos, etc ...

parameters: none
return value: string

example:

```
local userpath = esSystem:GetUserPath()
local docpath =  userpath .. "/documents"
```

FileExists(s)

It is used for checking whether a file exists or not

parameters:
s[1] string

return value: boolean
false the given file doesn't exist
true the given file exists

[1] File name

example:

```
local myfile = esSystem:GetUserPath() .. "/documents/myfile.txt"

if (esSystem:FileExists(myfile)) then
  -- do something with the existing file
else
  -- the given file doesn't exist
end
```

FolderExists(s)

It is used for checking whether a folder exists or not

parameters:
s[1] string

return value: boolean
false the given folder doesn't exist
true the given folder exists

[1] Folder name

example:

```
local myfolder = esSystem:GetUserPath() .. "/documents"

if (esSystem:FolderExists(myfolder)) then
  -- do something with the existing folder
else
  -- the given folder doesn't exist
end
```

ZipFolder(spath, dpath, fname)

It is used to generate a ZIP file of the given source path.

parameters:
spath[1] string
dpath[2] string
fname[3] string

return value: integer
0, Execution OK
1, Path to compress doesn't exist
2, Destination path doesn't exist
3, Could not possible make a relative path creating zip file
4, Could not possible instance zip file object
5, Could not possible add a new entry in zip file
6, Destination path is not writeable
7, Source folder and destination folder can not be same

[1] Source path (folder to be zipped)
[2] Destination path (where the final zip file will be created)
[3] File name

Paths and File name parametrisation
Concerning paths and file name parameters, they can contain these variables in their text:

%d	Actual day
%m	Actual month
%y	Actual year
%H	Actual hour
%M	Actual minute
%S	Actual second
%t	Actual millisecond
{PP}	Project path (**remarks:** please use it only for paths)

example:

```
-- Zip folder
local sourcepath = esSystem:GetUserPath() .. “/documents/mydata”
local destinationpath = esSystem:GetUserPath() .. “/public”
local err = 0

-- Execution
err = esSystem:ZipFolder(sourcepath, destinationpath, "mydata.zip")

-- Execution evaluation
if (err == 0) then
  -- execution without errors
else
  -- execution with errors
end
```

esUtility

This is a collection of useful functions for general purpose.
Despite Lua owns methods to perform bitwise operations and in general for all which is shown below, here we present eScada methods in a more friendly syntax.

Sleep(tms)

Sleeps the code execution for the specified number of milliseconds.

parameters:
tms[1] integer

return value: none

[1] milliseconds

example:

```
-- Sleep for 1s
esUtility:Sleep(1000)
```

PackBits(...)

parameters:
...[1] boolean

return value: integer
It is the value obtained converting the given boolean values.

[1] This function accepts a dynamic amount of parameters in the range from 1 to 64
The first parameter is the value for the bit number 0, the second for the bit number 1, and so on ...

example:

```
local bit0 = true
local bit1 = false
local bit2 = true
local value = 0

--[
  In this example the function converts a bit string with this representation “100101”
  The converted value is: 37
--]
value = esUtility:PackBits(bit0, bit1, bit2, false, false, true)
```

ValueToBits(value, bits, reverse)

This function converts the given value in a bit string.

parameters:
value[1] integer, floating point
bits[2] integer
reverse[3] boolean

return value: string

[1] Value number to convert in a bit string
[2] This is the string width you want to obtain, the string will be padded from right with the char 0
Only 8, 16, 32 or 64 will be accepted as valid parameter values.
[3] true the bit string will be reversed

example:

```
local bitstring1 = “”
local bitstring2 = “”

-- The return value will be a string like: 00100101
bitstring1 = esUtility:ValueToBits(37, 8, false)

-- The return value will be a string like: 10100100
bitstring2 = esUtility:ValueToBits(37, 8, true)
```

SetBit(value, bit)

This function sets the given bit number to value 1

parameters:
value[1] integer
bit[2] integer

return value: integer

[1] Value where the bitwise operation will be performed
[2] Bit number from 0 to 63

example:

```
local ivalue = 10

-- Before the bitwise operation ivalue is 1010 (10 dec), after is 1110 (14 dec)
ivalue = esUtility:SetBit(ivalue, 2)
```

ToggleBit(value, bit)

This function toggles the given bit number

parameters:
value[1] integer
bit[2] integer

return value: integer

example:

```
local ivalue = 10

-- Before the bitwise operation ivalue is 1010 (10 dec), after is 1110 (14 dec)
ivalue = esUtility:ToggleBit(ivalue, 2)

-- Before the bitwise operation ivalue is 1110 (14 dec), after is 1010 (10 dec)
ivalue = esUtility:ToggleBit(ivalue, 2)
```

ResetBit(value, bit)

This function sets the given bit number to value 0

parameters:

value[1]	integer
bit[2]	integer

return value: integer

[1] Value where the bitwise operation will be performed
[2] Bit number from 0 to 63

example:

```
local ivalue = 10

-- Before the bitwise operation ivalue is 1010 (10 dec), after is 1000 (8 dec)
ivalue = esUtility:ResetBit(ivalue, 1)
```

TestBit(value, bit)

This function return the value of the given bit

parameters:

value[1]	integer
bit[2]	integer

return value: boolean

[1] Value where the bitwise operation will be performed
[2] Bit number from 0 to 63

example:

```
local ivalue = 10

-- Here ivalue bit string is 1010
if (esUtility:TestBit(ivalue, 3)) then
  -- The fourth bit value is TRUE
else
  -- The fourth bit value is FALSE
end
```

Base64Encode(s)

This function encodes the given data using base64 and returns the output as a string.
Here you can find more information about Base64 algorithm:
[https://en.wikipedia.org/wiki/Base64]

parameters:
s[1] string

return value: string
There is no error return.

[1] Text to encode

example:

```
local mystring = “my string”
local encstring = “”

--[[
  Encode the given string
  The variable encstring will contain the string “bXkgc3RyaW5n”
]]--
encstring = esString:Base64Encode(mystring)
```

Base64Decode(s)

This function decodes a Base64-encoded string and returns the original string.
Here you can find more information about Base64 algorithm:
[https://en.wikipedia.org/wiki/Base64]

parameters:
s[1] string

return value: string
There is no error return.

[1] Text to decode

example:

```
local encstring = “bXkgc3RyaW5n”
local mystring = “”

--[[
  Decode the given string
  The variable mystring will contain the string “my string”
]]--
mystring = esString:Base64Decode(encstring)
```

esString

This object offers a collection of useful functions for manipulating strings.
Despite Lua owns its methods and procedures to operate with strings, here we offer some useful eScada functions able to manipulate strings in a more friendly way.

remarks

- The following functions are able to manipulate strings with UTF8 format.
- Wherever you need to specify a starting or ending point along the string, please observe that the first char of a string has got the index number 1.
- Wherever you need to specify an amount of chars, they are literally chars and not bytes.

Translate(s, id)

It returns the given text translated using the given language id.

parameters:
s[1] string
id[2] integer

return value: string
If the translation for the parameter **s** doesn't exist or it hasn't been declared, the function will return the parameter itself.

[1] Text to translate
It must be declared using the "string" sub context into "resources" context.
[2] Language ID
It is a number from 1 to 16

example:

```
local itstring = "casa"
local enstring = ""

--[[
  Translate the string from it to en
  enstring will contain "house" if it has been declared as project's string
]]--
enstring = esString:Translate(itstring, 2);
```

IsNumber(s)

It is used for checking whether a string contains a number or not.
The number can be either an integer or a floating point one.

parameters:
s[1] string

return value: boolean
false the given string is not a number
true the given string is a number

[1] String to evaluate

example:
```
if (esString:IsNumber("123.34")) then
  -- The given string is a number
else
  -- The given string is not a number
end
```

IsEmpty(s)

It is used for checking whether a string is empty or not.

parameters:
s[1] string

return value: boolean
false the given string is not empty
true the given string is empty

[1] String to evaluate

example:
```
local mystring = ""  -- an empty string

if (esString:IsEmpty(mystring)) then
  -- The given string is empty
else
  -- The given string is not empty
end
```

IsAscii(s)

It is used for checking whether a string contains ASCII text only or not.

parameters:
s[1] string

return value: boolean
false the given string is not an ASCII string
true the given string is an ASCII string

[1] String to evaluate

example:
```
local mystring = "abcd 家 12,34"  -- a not ASCII string

if (esString:IsAscii(mystring)) then
  -- The given string is ASCII
else
  -- The given string is not ASCII
end
```

Matches(s, mask)

Returns true if the string contents matches a mask containing '*' and '?'

parameters:
s[1] string
mask[2] string

return value: boolean
false the given string does not match the mask
true the given string matches the mask

[1] String to evaluate
[2] Mask to be verified

example:

```
local mystring = “abcd 家 12,34”
local mask = “*家*”

if (esString:Matches(mystring, mask)) then
  -- The given string does match the mask
else
  -- Do something if the string does not match the mask
end
```

Validate(s, chars, mode)

This function is useful to validate the given string using a second string which contains the allowed chars.

parameters:
s[1] string
chars[2] string
mode[3] integer

return value: boolean
It depends on the given mode:
false the given string does not match the mode criteria.
true the given string matches the mode criteria.

[1] String to evaluate
[2] Chars to be verified
[3] Evaluation mode
0 (The string must be composed only with the given chars)
1 (The string must not contains the given chars)

example:

```
if (esString:Validate(“1.234”, “0123456789.,-”, 0)) then
-- Valid string
end

if (esString:Validate(“Lua is a powerful language”, “0123456789”, 1)) then
-- Valid string
end
```

Contains(s1, s2)

It is used for checking whether a string contains another one.

parameters:
s1[1] string
s2[2] string

return value: boolean
false the string **s1** does not contain the string **s2**
true the string **s1** contains the string **s2**

[1] String to evaluate
[2] Checking string

example:

```
local mystring = "This is your 家, does it?"

if (esString:Contains(mystring, "your  家")) then
  -- The given string contains the second one
end
```

StartsWith(s1, s2)

It is used for checking whether a string starts with another one.

parameters:
s1[1] string
s2[2] string

return value: boolean
false the string **s1** does not start with the string **s2**
true the string **s1** starts with the string **s2**

[1] String to evaluate
[2] Checking string

example:

```
local mystring = "This is your 家, does it?"

if (esString:StartsWith(mystring, "This is your 家")) then
  -- The given string starts with the second one
end
```

EndsWith(s1, s2)

It is used for checking whether a string ends with another one.

parameters:
s1[1] string
s2[2] string

return value: boolean
false the string **s1** does not end with the string **s2**

true the string **s1** ends with the string **s2**

[1] String to evaluate
[2] Checking string

example:

```
local mystring = "This is your 家, does it?"

if (esString:EndsWith(mystring, "家, does it?")) then
  -- The given string ends with the second one
end
```

AfterFirst(s, char)

Gets all the characters after the first occurrence of **char**.
Returns the whole string **s** if **char** is not found.

parameters:
s[1] string
char[2] string

return value: string

[1] String to evaluate
[2] It must be a string with a single char

example:

```
local mystring = "eScada|is|a|smart|HMI|software"
local svalue = esString:AfterFirst(mystring, "|") -- svalue = is|a|smart|HMI|software
```

AfterLast(s, char)

Gets all the characters after the last occurrence **char**.
Returns the whole string **s** if **char** is not found.

parameters:
s[1] string
char[2] string

return value: string

[1] String to evaluate
[2] It must be a string with a single char

example:

```
local mystring = "eScada|is|a|smart|HMI|software"
local svalue = esString:AfterLast(mystring, "|") -- svalue = software
```

BeforeFirst(s, char)

Gets all characters before the first occurrence **char**.
Returns the whole string **s** if **char** is not found.

parameters:
s[1] string
char[2] string

return value: string

[1] String to evaluate
[2] It must be a string with a single char

example:

```
local mystring = "eScada|is|a|smart|HMI|software"
local svalue = esString:BeforeFirst(mystring, "|") -- svalue = eScada
```

BeforeLast(s, char)

Gets all characters before the last occurrence **char**.
Returns the whole string **s** if **char** is not found.

parameters:
s[1] string
char[2] string

return value: string

[1] String to evaluate
[2] It must be a string with a single char

example:

```
local mystring = "eScada|is|a|smart|HMI|software"
local svalue = esString:BeforeLast(mystring, "|") -- svalue =  eScada|is|a|smart|HMI
```

IsSameAs(s1, s2, case)

Test whether the string **s1** is equal to string **s2**.
The test is case-sensitive if **case** is true or not if it is false.

parameters:
s1[1] string
s2[2] string
case[3] boolean

return value: boolean
false string **s1** is not equal to string **s2**
true string **s1** is equal to string **s2**

[1] First string
[2] Secon string
[3] Case-sensitive flag

example:

```
if (not esString:IsSameAs("House", "house", true)) then
  -- The two strings are not equal
end
```

Compare(s1, s2, case)

Returns a positive value if the string **s1** is greater than **s2**, zero if it is equal to it or a negative value if it is less than **s2**.
The test is case-sensitive if **case** is true or not if it is false.

parameters:
s1[1] string
s2[2] string
case[3] boolean

return value: integer
- positive value if **s1** is greater than **s2**
- zero if **s1** is equal to **s2**
- negative value if **s1** is less than **s2**

[1] First string
[2] Secon string
[3] Case-sensitive flag

example:

```
if (esString:Compare("String2", "String1", false) > 0) then
  -- The first string is greater then second
end
```

Left(s, chars)

Returns the first characters of the string.

parameters:
s[1] string
chars[2] integer

return value: string

[1] Source string
[2] Amount of characters needed

example:

```
local sourcestring = "Lua is a powerful, efficient, lightweight, embeddable scripting language"
local svalue = esString:Left(sourcestring, 17)        -- svalue = Lua is a powerful
```

Right(s, chars)

Returns the last characters of the string.

parameters:
s[1] string
chars[2] integer

return value: string

[1] Source string
[2] Amount of characters needed

example:

```
local sourcestring = “Lua is a powerful, efficient, lightweight, embeddable scripting language”
local svalue = esString:Right(sourcestring, 18)      -- svalue = scripting language
```

Mid(s, first, chars)

Returns a substring starting at **first**, with length **chars**.

parameters:
s[1] string
first[2] integer
chars[3] integer

return value: string

[1] Source string
[1] Starting character
[3] Amount of characters needed

example:

```
local sourcestring = “Lua is a powerful, efficient, lightweight, embeddable scripting language”
local svalue = esString:Mid(sourcestring, 5, 13)     -- svalue = is a powerful
```

SubString(s, from, to)

Returns the part of the string between the indices **from** and **to** inclusive.

parameters:
s[1] string
from[2] integer
to[3] integer

return value: string

[1] Source string
[2] From character
[3] To character

example:

```
local sourcestring = “Lua is a powerful, efficient, lightweight, embeddable scripting language”
local svalue = esString:SubString(sourcestring, 10, 17)   -- svalue = powerful
```

Insert(s1, pos, s2)

Insert the string **s2** in string **s1** starting at **pos**.

parameters:
s1[1] string
pos[2] integer
s2[3] string

return value: string

[1] Source string
[2] Starting character
[3] String to insert

example:

```
local sourcestring = “Lua powerful”
local svalue = esString:Insert(sourcestring, 4, “is a ”) -- svalue = Lua is a powerful
```

Prepend(s1, s2)

Prepends the string **s2** to string **s1**.

parameters:
s1[1] string
s2[2] string

return value: string

[1] Source string
[2] String to prepend

example:

```
local sourcestring = “powerful”
local svalue = esString:Prepend(sourcestring, “Lua is a ”) -- svalue = Lua is a powerful
```

Append(s1, s2)

Appends the string **s2** to string **s1**.

parameters:
s1[1] string
s2[2] string

return value: string

[1] Source string
[2] String to append

example:

```
local sourcestring = “Lua is a ”
local svalue = esString:Append(sourcestring, “powerful”) -- svalue = Lua is a powerful
```

Trim(s, mode)

Removes white-space (space, tabs, form feed, newline and carriage return) from the left or from the right end of the string **s** depending on **mode**.

parameters:
s[1] string
mode[2] integer

return value: string

[1] Source string
[2] Trimming mode
0, From left
1, From right
2, Both sides

example:

```
local sourcestring = "   Lua is a powerful   "
local svalue1 = esString:Trim(sourcestring, 0) -- svalue1 = "Lua is a powerful   "
local svalue2 = esString:Trim(sourcestring, 1) -- svalue2 = "   Lua is a powerful"
local svalue3 = esString:Trim(sourcestring, 2) -- svalue3 = "Lua is a powerful"
```

Pad(s, width, char, side)

Adds (**with** – **s** length) copies of **char** at the beginning or at the end of **s** depending on **side**.

parameters:
s[1] string
width[2] integer
char[3] string
side[4] integer

return value: string

[1] Source string
[2] Final string length
[3] It must be a string with a single char
[4] Padding mode
0, Left
1, Right

example:

```
local sourcestring = "mystring"
local svalue1 = esString:Pad(sourcestring, 10, "_", 0) -- svalue1 = "__mystring"
local svalue2 = esString:Pad(sourcestring, 10, "_", 1) -- svalue2 = "mystring__"
```

Transforms(s, mode)

Transforms the source string **s** depending on **mode**.

parameters:
s[1] string
mode[2] integer

return value: string

[1] Source string
[2] Mode
0, Convert to lower case
1, Convert to upper case
2, Convert the first character to the upper case and the rest to the lower one

example:

```
local sourcestring = “MYsTrINg”
local svalue = esString:Transforms(sourcestring, 0) -- svalue = “mystring”

local sourcestring = “mystring”
local svalue = esString:Transforms(sourcestring, 1) -- svalue = “MYSTRING”

local sourcestring = “mysTriNg”
local svalue = esString:Transforms(sourcestring, 2) -- svalue = “Mystring”
```

Length(s)

Returns the length of the string **s**.

parameters:
s[1] string

return value: string

[1] Source string

example:

```
local sourcestring = “mystring”

if (esString:Length(sourcestring) == 8) then
-- Valid string length
end
```

GetChar(s, charid)

Returns the character value at position **charid**.
The value is the ASCII or UNICODE value for the character needed.

parameters:
s[1] string
charid[2] integer

return value: integer
If **s** is an empty string, it returns **-1**

[1] Source string
[2] Character index

example:

```
local sourcestring = “mystring”
local charvalue = esString:GetChar(sourcestring, 2)  -- charvalue = 121
```

LastChar(s)

Returns the last character value of **s**.
The value is the ASCII or UNICODE value for the character needed.

parameters:
s[1] string

return value: integer
If **s** is an empty string, it returns **-1**

[1] Source string

example:

```
local sourcestring = “mystring”
local charvalue = esString:LastChar(sourcestring)  -- charvalue = 103
```

Replace(s, old, new)

Replace all occurrences of **old** in **s** with the **new** one.

parameters:
s[1] string
old[2] string
new[3] string

return value: string

[1] Source string
[2] Old string to be replaced
[3] New string

example:

```
local sourcestring = “I like bear”
local svalue = esString:Replace(sourcestring, “bear”, “beer”)  -- svalue = I like beer
```

Split(s, chars)

Helps you to break a string up into a number of items.

parameters:
s[1] string
chars[2] string

return value: table
- The returned table can be accessed by using the following syntax:

```
local item1 = mytable[1] -- First item
local item2 = mytable[2] -- Second item
local item3 = mytable[3] -- Third item
...
```

- The number of table items can be retrieved by using the following syntax:

```
local count = #mytable
```

[1] Source string
[2] Field delimiters
If needed, delimiters can be more than one: e.g. “;|”

example:

```
-- Declare tag objects
local pString = esTags["lua.strings"]

-- Get the table of items (remark: the item #4 is empty)
local tItems = esString:Split("one|two|three|four||five", "|")

-- Get the number of items in the table
local count = #tItems

-- Set items into pString items
for item = 1, count do
  if (item < pString:Items()) then
    pString:SetValue(item, tItems[item])
  end
end
```

esTags

This is the container which contains all project's tags.
They can be accessed using the recommendations written in "Common recommendations"

- correct tag object declaration
```
local pTag = esTags["mytagname"]
```

- correct tag object usage
```
local value = esTags["mytagname"]:GetValue(10)
```

remarks:
- The tag you want to use must be enabled, in case it is not enabled a runtime error will be raised.

- The following functions does not apply to all tag types. In case the function does not make sense for the tag which is being used, a runtime error will be raised.
For instance calling Bit() function for a tag which contains strings will result in a runtime error, is up to the user take care of calling the right function.

- Later in this paragraph several described functions need the tag item's index as parameter.
Such a parameter is indicated as **[id]** and it is optional, if it is not expressed the index **0** will be used; this means that for a single tag you don't need to specify **0** as item's index.

GetValue([id])

It returns the value for the given item id.

parameters:
id[1] integer [optional]

return value: any (it depends on the tag data type)
For instance, if the tag data type Is boolean, the return value will be boolean too, and so on for all data types.

[1] Item's index

example:
```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get value for the item #10
local itemvalue = pTag:GetValue(10)
```

GetValueHex([id])

It returns the value for the given item id using hexadecimal notation.
This function make sense for numerics data type.

parameters:
id[1] integer [optional]

return value: string

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Get value for the item #10
local hexvalue = pTag:GetValueHex(10)
```

SetValue([id], val)

It permits to set the value for the given item id.

remark
Please keep in mind that, in case you need to modify several items, you have to consider to use the way shown with BeginUpdate() function.

parameters:
id[1] integer [optional]
val[2] any (it depends on the tag data type)

return value: boolean
true, action successfully executed.
false, something went wrong.

[1] Item’s index
[2] Value to write

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Set value for the item #10
if (pTag:SetValue(10, “my string”)) then
  -- Value successfully written
end
```

SetValueHex([id], val)

It permits to set the value for the given item id using the hexadecimal notation for value.

remark
Please keep in mind that, in case you need to modify several items, you have to consider to use the way shown with BeginUpdate() function.

parameters:
id[1] integer [optional]
val[2] string

return value: boolean
true, action successfully executed.
false, something went wrong.

[1] Item’s index
[2] Value to write (hexadecimal)

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Set value for the item #10
if (pTag:SetValueHex(10, “10AB4F”)) then
  -- Value successfully written
end
```

BeginUpdate()

It permits to start an update session for the tag, the session must be closed using the EndUpdate() function.
This kind of session, is useful to update several items belonging to a tag with many items; in particular we recommend to use this function every time it is necessary to update a device tag items.
In fact, for a device tag, every single value modification a frame protocol must be sent to the device.
Using the following described way, all items can be written to device using one protocol frame only.

parameters: none

return value: boolean
true, update session can start
false, update session can not start

example:

```
-- Tag object declaration
local pBoolean = esTags["lua.Boolean"]

-- Get value from item #10
local value = esTags[“mytag”]:GetValue(10)

-- Start session update
if (pBoolean:BeginUpdate()) then
  for item = 1, 10 do
    -- Set item value
    pBoolean:SetValue(
      item - 1,
      esUtility:TestBit(value, item - 1)
    )
  end

  -- Last bit
  pBoolean:SetValue(19, esUtility:TestBit(value, 63))

  -- End session update
  pBoolean:EndUpdate()
end
```

EndUpdate()

It permits to end an update session started using the BeginUpdate() function.
It is mandatory to close an update session using this function.

parameters: none
return value: none

example:

Please see the example for BeginUpdate() function

IsUpdating()

It permits to check whether a tag is updating its items values or not.
This function returns true if somewhere else a BeginUpdate() function has been called for the same tag.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Tag evaluation
if (not pTag:IsUpdating()) then
  -- do something
end
```

SkipEvaluation(b)

It permits to enable or disable tag's evaluation consent.

parameters:
b[1] boolean

return value: none

[1] Evaluation status
true, skips tag evaluation
false, the tag can be evaluated

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Skips tag evaluation
pTag:SkipEvaluation(true)
```

Evaluate(b)

It permits to evaluate the tag from Lua code without using its trigger event.
Tags belonging to this list[2] are not evaluable using this function.

parameters:
b[1] boolean

return value: boolean
- if **b** is true (synchronous)
true, action successfully executed.
false, something went wrong.

- if **b** is false (asynchronous)
Always false

[1] Evaluation mode
true, synchronous evaluation
false, asynchronous evaluation

[2] Not evaluable tag types. [See TagType() function]

0, Device tag	49, Private	97, DeviceTagInfo
2, Counter	53, Item	98, DerivedTagInfo
3, Time counter	57, Pulse	109, OpcUaTcp (server)
4, Integrator	64, Stack	
5, ON Delay timer	66, Derivative	
6, OFF Delay timer	70, Series	
7, Public	81, Interpolation	
10, Stats	87, WS (server)	
11, Notification	88, ModbusTcp (server)	
48, Pointer	92, RgbColour	

example:

```
-- Tag object declaration
local pTag = esTags[“myderivedtag”]

-- Skips tag evaluation
If (pTag:Evaluate(true)) then
  -- Synchronous evaluation OK
end
```

Read()

It permits do perform a read action from device.
This is useful in case the tag polling value has been defined to zero.

parameters: none

return value: boolean
true, action successfully executed.
false, something went wrong.

example:

```
-- Tag object declaration
```

```
local pTag = esTags["mytagname"]

-- Read tag items
if (pTag:Read()) then
  -- do something with values ...
end
```

Write()

It permits do perform a write action to device.
This is useful in case the tag polling value has been defined to zero.

parameters: none

return value: boolean
true, action successfully executed.
false, something went wrong.

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Read tag items
if (pTag:Write()) then
  -- write action ok
end
```

GetPlcMin([id])

It returns the PLC minimum admitted value for the tag's item at the given index id.
Normally this value is defined for tags which the value scaling is defined.

parameters:
id[1] integer [optional]

return value: float

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get property value for the item #5
local plcmin =  pTag:GetPlcMin(5)
```

GetPlcMax([id])

It returns the PLC maximum admitted value for the tag's item at the given index id.
Normally this value is defined for tags which the value scaling is defined.

parameters:
id[1] integer [optional]

return value: float

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get property value for the item #5
local plcmin =  pTag:GetPlcMax(5)
```

GetHmiMin([id])

It returns the HMI minimum admitted value for the tag's item at the given index id.
This is referred to the range the operator can insert, it could be present even without a range defined for the PLC.
It is used to validate values during their modifications.

parameters:
id[1] integer [optional]

return value: float

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get property value for the item #5
local plcmin =  pTag:GetHmiMin(5)
```

GetHmiMax([id])

It returns the HMI maximum admitted value for the tag's item at the given index id.
This is referred to the range the operator can insert, it could be present even without a range defined for the PLC.
It is used to validate values during their modifications.

parameters:
id[1] integer [optional]

return value: float

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Get property value for the item #5
local plcmin =  pTag:GetHmiMax(5)
```

PlcLimitsOn([id])

It returns true in case the PLC limits are defined.
PLC Limit values can be retrieved by using GetPlcMin() and GetPlcMax() functions.

parameters:
id[1] integer [optional]

return value: boolean

[1] Item’s index

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]
local plcmin = 0
local plcmax = 0

-- Get property values for the item #0
if (pTag:PlcLimitsOn()) then
   plcmin = pTag:GetPlcMin()
   plcmax = pTag:GetPlcMax()
end
```

HmiLimitsOn([id])

It returns true in case the HMI limits are defined.
HMI Limit values can be retrieved by using GetHmiMin() and GetHmiMax() functions.

parameters:
id[1] integer [optional]

return value: boolean

[1] Item’s index

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]
local hmimin = 0
local hmimax = 0

-- Get property values for the item #0
if (pTag:HmiLimitsOn()) then
   hmimin = pTag:GetHmiMin()
   hmimax = pTag:GetHmiMax()
end
```

ScalingOn([id])

It returns true in case both PLC limits and HMI limits are defined.
When both limits are defined, the item value is scaled.

parameters:
id[1] integer [optional]

return value: boolean

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get property values for the item #0
if (pTag:ScalingOn()) then
   -- do something
end
```

Name()

It returns the tag name

parameters: none
return value: string

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]
local name = pTag:Name()
```

Description()

It returns the tag description

parameters: none
return value: string

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]
local description = pTag:Description()
```

Items()

It returns the items count

parameters: none
return value: integer

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]
local items = pTag:Items()

-- Loop on tag items
-- Keep in mind that the first item has got index 0
for item = 0, items - 1 do
  if (pTag:GetValue(item) == 100) then
    -- do something
  end
end
```

Bits()

It returns the number of bits for this variable.

parameters: none
return value: integer
8, byte
16, word
32, D word
64, Q word

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Bits evaluation
if (pTag:Bits() == 16) then
  -- do something
end
```

DataType()

It returns tag's data type

parameters: none
return value: integer
0, Single bit
1, Unsigned 8 bit
2, Signed 8 bit
3, Unsigned integer 16 bit
4, Signed integer 16 bit
5, Unsigned integer 32 bit
6, Signed integer 32 bit
7, Single precision 32 bit
8, Unsigned integer 64 bit
9, Signed integer 64 bit
10, Double precision 64 bit
11, Array of bytes
12, Array of bytes. (Siemens S7 style)
13, Array of bytes. (Allen Bradley style)

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]
local bValue = false
local nValue = 0
local sValue = “”

-- Data type evaluation
if (pTag:DataType() == 0) then
  -- Get boolean value
  bValue =  pTag:GetValue()
elseif (pTag:DataType() < 11) then
  -- Get numeric value
  nValue = pTag:GetValue()
else
  -- Get string value
  sValue = pTag:GetValue()
end
```

StringLength()

It returns the number of maximum bytes a string can contain.

parameters: none
return value: integer

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]
local stringsize = pTag:StringLength()
```

TagType()

It returns tag's type

parameters: none
return value: integer

0, Device tag	28, Length	56, Bitwise	84, BitsMask
1, Function	29, Transform	57, Pulse	85, SendEmail
2, Counter	30, Trim	58, Shift	86, TrendLog
3, Time counter	31, Left	59, Mask	87, WS (server)
4, Integrator	32, Right	60, Complement	88, ModbusTcp (server)
5, ON Delay timer	33, Mid	61, WriteFile	89, HexToValue
6, OFF Delay timer	34, SubString	62, LoadFile	90, ZipFolder
7, Public	35, Split	63, DataColumn	91, LinearScale
8, System	36, ToHex	64, Stack	92, RgbColour
9, Thresholds	37, ToBinary	65, SqlCommand	93, SwapBytes
10, Stats	38, ToSInt8	66, Derivative	94, ArrayChanged
11, Notification	39, ToSInt16	67, Execute	95, ValueChanged
12, Reserved	40, ToSInt32	68, PackBits	96, UploadValues
13, Reserved	41, ToSInt64	69, BitsShift	97, DeviceTagInfo
14, Reserved	42, ToUInt8	70, Series	98, DerivedTagInfo
15, Reserved	43, ToUInt16	71, Contains	99, ConvertTo
16, Copy	44, ToUInt32	72, StartsWith	100, EditArray
17, Pad	45, ToUInt64	73, EndsWith	101, Script
18, Append	46, ToDouble	74, AfterFirst	102, Folder
19, Prepend	47, ToFloat	75, AfterLast	103, File
20, Find	48, Pointer	76, BeforeFirst	104, ValueToHex
21, Matches	49, Private	77, BeforeLast	105, ValidateString
22, IsSameAs	50, SplitBits	78, AssignItem	106, ParametrizedText
23, Replace	51, Insert	79, AssignPointer	107, HttpCommand
24, GetChar	52, Format	80, DeviceTag	108, Size
25, LastChar	53, Item	81, Interpolation	109, OpcUaTcp (server)
26, IsNumber	54, SetItem	82, Sequencer	110, Translate
27, IsEmpty	55, CRC	83, Compose	111, LuaModule

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Tag type evaluation
if (pTag:TagType() == 90) then
  -- Do something
end
```

CommOk()

It returns the communication flag status.

parameters: none

return value: boolean
true, communication ok
false, communication error

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Communication evaluation
if (not pTag:CommOk()) then
  -- do something with bad communication status
end
```

IsEnabled()

It returns true if the tag is enabled.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsEnabled()) then
  -- do something
end
```

IsEvaluating()

It returns true if the tag is being evaluate.
For instance, in this case the function Evaluate() shouldn't be called.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (not pTag:IsEvaluating()) then
  -- Evaluate tag
  pTag:Evaluate()
end
```

IsReadOnly()

It returns true if the tag is a read-only one.
For instance, in this case you can assign value to its items.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (not pTag:IsReadOnly()) then
  -- Set tag items
  pTag:SetValue(0, 100)
  pTag:SetValue(5, 105)
end
```

IsString()

It returns true if the tag data type is string.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsString()) then
  -- do something
end
```

IsNumeric()

It returns true if the tag data type is numeric.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsNumeric()) then
  -- do something
end
```

IsFloat()

It returns true if the tag data type is floating point 32 bits.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsFloat()) then
  -- do something
end
```

IsDouble()

It returns true if the tag data type is floating point 64 bits.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsDouble()) then
  -- do something
end
```

IsFloatingPoint()

It returns true if the tag data type is either floating point 32 bits or 64 bits.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsFloatingPoint()) then
  -- do something
end
```

IsInteger()

It returns true if the tag data type is numeric with integer format.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsInteger()) then
  -- do something
end
```

IsSigned()

It returns true if the tag data type is signed.
Floating point numbers are considered always signed.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsSigned()) then
  -- do something
end
```

IsUnsigned()

It returns true if the tag data type is unsigned; integers only.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsUnsigned()) then
  -- do something
end
```

IsBoolean()

It returns true if the tag data type is boolean.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsBoolean()) then
  -- do something
end
```

IsByte()

It returns true if the tag data type is signed or unsigned byte, 8 bit.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsByte()) then
  -- do something
end
```

IsWord()

It returns true if the tag data type is signed or unsigned word, 16 bit.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsWord()) then
  -- do something
end
```

IsDWord()

It returns true if the tag data type is signed or unsigned double word, 32 bit.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsDWord()) then
  -- do something
end
```

IsQWord()

It returns true if the tag data type is signed or unsigned quadruple word, 64 bit.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsQWord()) then
  -- do something
end
```

IsPointer()

It returns true if the tag is a pointer to tag.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsPointer()) then
  -- do something
end
```

IsItem()

It returns true if the tag is a pointer to an item.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsItem()) then
  -- do something
end
```

IsNotification()

It returns true if the tag is a notification.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (pTag:IsNotification()) then
  -- do something
end
```

IsOffLine()

It returns true if offline status for tag is active.
Offline status is referred to device tags, if they are offline their items value are not updated.

parameters: none
return value: boolean

example:

```
-- Tag object declaration
local pTag = esTags[“mytagname”]

-- Tag evaluation
if (not pTag:IsOffline()) then
  -- do something
end
```

SetOffLine(b)

It is useful for changing the tag's offline status.

parameters:
b[1] boolean

return value: none

[1] Offline status
true, the offline state is activated
false, the offline state is deactivated

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Activates offline status
pTag:SetOffLine(true)

-- Deactivates offline status
pTag:SetOffLine(false)
```

GetItemDescription([id])

It returns the description for the given item id.

parameters:
id[1] integer [optional]

return value: string

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get property value for the item #10
local itemdesc = pTag:GetItemDescription(10)
```

SetItemDescription([id], s)

It permits to set the description for the given item id.

parameters:
id[1] integer [optional]
s[2] string

return value: none

[1]Item's index
[2]New description

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Set property value for the item #10
pTag:SetItemDescription(10, "New item description")
```

GetItemUnit([id])

It returns the unit for the given item id.

parameters:
id[1] integer [optional]

return value: string

[1] Item's index

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get property value for the item #10
local itemunit = pTag:GetItemUnit(10)
```

SetItemUnit([id], s)

It permits to set the unit for the given item id.

parameters:
id[1] integer [optional]
s[2] string

return value: none

[1]Item's index
[2]New unit

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Set property value for the item #10
pTag:SetItemUnit(10, "mm")
```

AssignPointer(s)

It permits to change the reference to a tag object.
This function works with pointers only.

parameters:
s[1] string

return value: boolean
true, action successfully executed.
false, something went wrong. (not existing tag name)

[1] New tag name

example:

```
-- Tag object declaration
local pPointer = esTags[“mypointer”]

--[
  Please keep in mind that the right way to insert tag names, whenever you need, is here
  exemplified.
]--
local pTag1 = esTags[“mytag1”]
local pTag2 = esTags[“mytag2”]

--  Valid pointer to tag
if (pPointer:IsPointer() then
  --  Change the pointed tag #1
  if (pPointer:AssignPointer(pTag1:Name())) then
    -- Set tag’s item values by using a pointer to tag
    pPointer:SetValue(0, 1.23)
    pPointer:SetValue(1, 2.3)
    pPointer:SetValue(2, -1.23)
  end

  --  Change the pointed tag #2
  if (pPointer:AssignPointer(pTag2:Name())) then
    -- Set tag’s item values by using a pointer to tag
    pPointer:SetValue(0, 8.3)
    pPointer:SetValue(1, -0.31)
    pPointer:SetValue(2, 1.23)
  end
end
```

AssignItem(s, id)

It permits to change the reference to an item object.
This function works with pointer to items only.

parameters:
s[1] string
id[2] integer

return value: boolean
true, action successfully executed.
false, something went wrong. (not existing tag name or item id out of bounds)

[1] New tag name
[2] New item index

example:

```
-- Tag object declaration
local pItem = esTags[“myitem”]

--[
  Please keep in mind that the right way to insert tag names, whenever you need, is here
  exemplified.
]--
local pTag1 = esTags[“mytag1”]
local pTag2 = esTags[“mytag2”]

--  Valid pointer to item
if (pItem:IsItem() then
  --  Change the pointed item #1
  if (pItem:AssignItem(pTag1:Name(), 10)) then
    -- Set item’s value by using a pointer to item
    pItem:SetValue(“new value”)
  end

  --  Change the pointed tag #2
  if (pPointer:AssignItem(pTag2:Name(), 0)) then
    -- Set item’s value by using a pointer to item
    pItem:SetValue(55)
  end
end
```

GetPolling()

It returns the polling time for the device tag.

parameters: none
return value: integer

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get tag's poling time [ms]
local polling = pTag:GetPolling()
```

SetPolling(tms)

It permits to set the polling time

parameters:
tms[1] integer [ms]

return value: none

[1] New polling time

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Set polling time
pTag:SetPolling(1000)
```

SetItemLog([id], b)

It permits to set the log status for the given item id.

parameters:
id[1] integer [optional]
b[2] boolean

return value: none

[1] Item's index
[2] New status
true, logging value changes enabled
false, logging value changes disabled

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Disable logging for the item #10
pTag:SetItemLog(10, false)
```

SetItemPublish([id], b)

It permits to set the publish status for the given item id.

parameters:
id[1] integer [optional]
b[2] boolean

return value: none

[1] Item's index
[2] New status
true, item's value sent to clients
false, item's value not sent to clients

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Disable sending for the item #10
pTag:SetItemPublish(10, false)
```

Format([id], format)

Formats the numeric value using the given **format**: [sign]integers[.][decimals]
Instead of this function you could try Lua function called string.format
[https://www.lua.org/manual/5.4/manual.html#6.4]

parameters:
id[1] integer [optional]
format[2] string

return value: string

[1] Item's index
[2] Format
examples:

format	value	Return value
0000	23	0023
s0	23	+23
0	-23	-23
0	23	23
s000.000	12.4	+012.400
0.00	231.129	231.13

example:

```
-- Tag object declaration
local pTag = esTags["mytagname"]

-- Get formatted value for the item #10
local svalue = pTag:Format(10, "0.000")
```