

TeslaSCADA OPC UA Server

User Manual

Version 1.3

Table of Content

About TeslaSCADA OPC UA Server.....	5
Requirements	5
Windows.....	5
Mac OS	5
Linux.....	5
Installation.....	6
Windows.....	6
Mac OS	6
Linux.....	6
Start TeslaSCADA OPC UA Server	7
Project.....	9
Create project.....	9
Save project.....	11
Open project	11
Edit project properties.....	11
Servers	12
Create server.....	12
Modbus RTU server	12
Modbus server	13
Siemens server	13
Allen Bradley server	14
OPC UA server	14
MQTT server	14
Omron server	15
Open server properties.....	16
Copy server	16
Delete server	16
Scripts.....	17
Create script	17
Open script	17
Copy script	17
Delete script	17
Edit script properties.....	17
New script group	17
Add to group.....	17
Export script	17

Import script.....	17
Tags.....	18
Create tag.....	18
Modbus tag settings	18
Siemens tag settings.....	19
AllenBradley tag settings	19
Micrologix tag settings	19
OPC UA tag settings	20
MQTT tag settings	20
Omron tag settings.....	20
Copy tag.....	22
Delete tag	22
Delete all tags.....	22
Set editable	22
Edit tag properties	22
New group tags	22
Add to group.....	22
Reference to.....	22
Export all tags.....	22
Export tags for OPC UA	22
Import tags	23
Export tags to Excel	23
Import tags from Excel	23
Design script.....	24
Create script object	24
Connect script objects	24
Bind script object to the tag	24
Enter value to the value script object	24
Duplicate script object	24
Erase script object.....	24
Erase connection line	24
Script objects of FBD language	25
Input/Output library.....	25
Logical library	25
Bitmap operations library.....	25
Arithmetic library.....	25
Compare library.....	26
Select library.....	26

Arrays library	26
Triggers/Counters library	26
Trigonometric library.....	27
Hex operations library	27
Call screen library.....	27
Strings library	27
Date and time library	27
Servers library	28
Recipes library.....	28
Base64 library	28
Description of ST(Structured text) language.....	29
What is Structured Text Programming?.....	29
Starting with the Syntax of Structured Text	29
Comment Syntax.....	30
Making Statements with Structured Text.....	30
Types in Structured Text.....	31
Operators and Expressions in STL	32
Operators	33
4 Types of Operators, 4 Types of Expressions.....	34
Arithmetic Operators	34
Relational Operators	34
Logical Operators.....	35
Bitwise Operators.....	35
Operators and Statements	36
Assignment Statement and Operator.....	36
Conditional Statements.....	37
IF Statements.....	37
Boolean and Numeric Expressions.....	37
Iteration with Repeating Loops.....	39
FOR Loops.....	39
WHILE Loops.....	39
User-defined functions	41
Using Tags in Structured Text.....	41
Using Object property fields in Structured Text	41
Using Server parameter fields in Structured Text.....	42
Using User parameter fields in Structured Text.....	42
Embedded functions.....	42
Use Telegram Bot.....	46

About TeslaSCADA OPC UA Server

TeslaSCADA OPC UA Server is an environment used for configuring, developing and managing OPC UA Server. In this manual you will find everything you need to create an OPC UA Server. A simple to use interface allows for easy manipulation of the project's configuration and data processing. The project data are stored in a single file (based on xml) for easy backup and restoration.

Requirements

TeslaSCADA OPC UA Server requires Windows, Mac OS or Linux operating systems.

Windows

Processors: Intel Pentium 4, Intel Centrino, Intel Xeon, or Intel Core Duo (or compatible) 1.8 GHz minimum.

Operating systems: Windows 8 (Modern UI (i.e. Metro Mode) is not supported), Windows 7, Windows Vista, Windows XP (not recommended but supported).

Memory: 512MB of RAM (1 GB recommended).

Disc Space: 256MB of free disc space.

Mac OS

Processors: Dual-Core Intel, PowerPC G5

Operating systems: 10.7.3 or greater

Memory: 512MB of RAM (1 GB recommended).

Disc Space: 256MB of free disc space.

Linux

Processors: Intel Pentium 4, Intel Centrino, Intel Xeon, or Intel Core Duo (or compatible) 1.8 GHz minimum.

Operating systems: Ubuntu 10.4 + gtk2 2.18+

Memory: 512MB of RAM (1 GB recommended).

Disc Space: 256MB of free disc space.

Media: You must install the following in order to support AAC audio, MP3 audio, H.264 video, and HTTP Live Streaming:

libavcodec52 and libavformat52 on Ubuntu Linux 10.04, 10.10, 11.04 or equivalent.

libavcodec53 and libavformat53 on Ubuntu Linux 11.10, 12.04 or equivalent.

Installation

Windows

To install TeslaSCADA OPC UA Server download EXE package for your operating system. Run installation file and go through installation procedure.

Mac OS

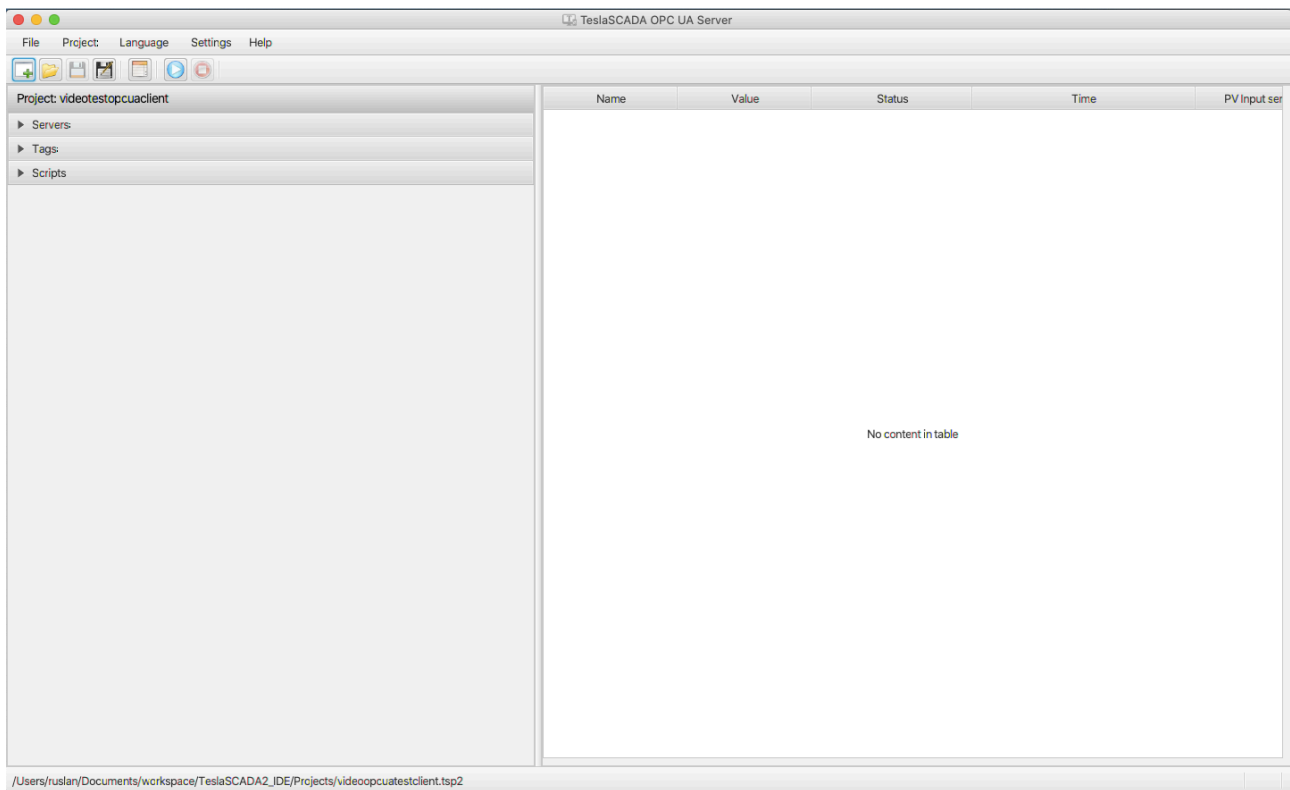
To install TeslaSCADA OPC UA Server download DMG package for your operating system. DMG package provides a simple possibility to install application by double clicking on it.

Linux

To install TeslaSCADA OPC UA Server download RPM package for your operating system. By default RPM package will install the application to /opt, add a shortcut to the application menu. RPM package does not have any UI for installation (normal behavior for Linux)

Start TeslaSCADA OPC UA Server

After opening the application you will see the start screen. Look at the picture below to briefly get to know the TeslaSCADA OPC UA Server interface:



Main menu

File - manipulation with project files.

Project - possibility to create new objects of the project - Server, Tag and Scripts. Also possibility to setup project properties.

Settings - possibility to make some settings in visualisation and activate a license.

Language - possibility to change language of the interface.

Help - opens the help menu

Toolbar

The toolbar consists of the following functions:



New project – creates a new project.



Open project – opens an existing project.



Save – saves your project.



Save as – saves your project with a new name.



Properties – properties of your project.



Run simulation – start simulation of your project.



Stop simulation – stop simulation of your project.

Project window

Project window contains all the information about the project and consists:

Scripts - contains all scripts of the project.

Servers - contains all servers of the project.

Tags - contains all tags of the project.

Status bar

Status bar contains information about path of the current project and information about run or not project.

Canvas

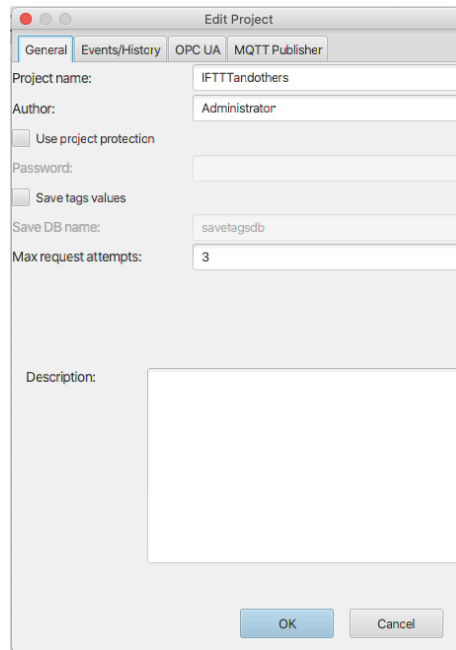
Place for the design FBD script or type ST script.

Project

Create project

To create a new project TeslaSCADA OPC UA must be started.

1. Click on the **New** icon in the toolbar or use the command *New* from the main menu *File*. You'll see the following window:



2. On the *General* tab:

2.1. In the **Project name** enter the name of the project.

2.2. In the **Author** write the author of the project if you want.

2.3. Optionally, specify a meaningful **Description** yet.

2.4. If you want to protect your project from opening by non-authorised person check **Use project protection**.

2.5. Enter **Password** for protecting your project.

2.6. Check **Save tags values** if you want to save tag's value when you close application and load them when you open project.

2.7. Enter **Save DB name** where tag's values will be saved.

2.8. Enter **Max requests attempts** for all servers before "Connection lost" message.

3. On the *Events/History* tab:

3.1. Select the time period during which data will be stored in databases in the **Storage DB period** combobox.

3.2. Enter databases names in the **Events DB name** and **History DB name**. If you choose the simple names like *events* or *history* application will create SQLite database in the application directory. If you choose names beginning with **jdbc:mysql:** like ***jdbc:mysql://192.168.0.104:3306/test*** the application will connect to MySQL database and create events or history table. ***Don't create big MySQL databases for connecting from Android devices (MySQL databases need a wide network bandwidth for sending and receiving data).***

3.3. Enter **Username** and **Password** if you use MySQL database.

3.4. Enter **Notifications(Priority<)**. Events with a priority lower than this will be notified about it by using the pop-up window and sound.

3.5. If you check **Show servers events** you'll get information about disconnection, lost or restore servers.

3.6. If you want to use Telegram bot in your project check **Use Telegram Bot**.

For more information about using telegram bot in your project see the chapter below.

3.9. Enter **Bot's name**. You get Telegram Bot's name from BotFather when you creating your bot.

3.10. Enter **Bot's token**. You get Telegram Bot's token from BotFather when you creating your bot.

3.11. Check **Use E-mail client** if you want to use E-mail notifications about Alarms. All event messages that have priority < **Notifications(Priority<)** will be sent by E-mail.

3.12. Enter E-mail **Host**.

3.13. Enter E-mail **Port**.

3.14. Choose **Type** of the connection - TLS or SSL.

3.15. Enter **From** which **E-mail address** the mail will be sent.

3.16. Check **Authentication** if you use Username and Password.

3.17. Enter **Username** of the E-mail.

3.18. Enter **Password** of the E-mail.

3.19. Enter **To** which **E-mail addresses** the mail will be sent. Use commas to separate addresses.

4. If you use OPC UA client certificate to connect to OPC UA servers in your project on the **OPC UA** tab enter **Name** of used/created certificate and **Period(days)** of validation if you create certificate. The certificate stored in the {app}/private directory.

If you want to enable OPC UA server of TeslaSCADA2 check **Use OPC UA server**.

4.3. Enter **TCP port** of your OPC UA server.

4.4. Check **Use Anonymous policy** if you want to use this policy in OPC UA server.

4.5. Check **Use Username/Password policy** if you want to use this policy in OPC UA server.

4.6. Enter **Certificate name** of your OPC UA server.

4.7. Enter validation **Period (days)** of the certificate.

The screenshot shows a window titled "Edit Project" with several tabs: "General", "Events/History", "OPC UA", and "MQTT Publisher". The "OPC UA" tab is selected. It contains a section for "OPC UA client certificate" with the following fields and values: "Name" is "TeslaSCADA2", "Period(days)" is "3650", "Use OPC UA server" is checked, "TCP port" is "8666", "Use Anonymous policy" is checked, "Use Username/Password policy" is checked, "Certificate name" is "opcua.certificatename", and "Period(days)" is "3650". At the bottom right, there are "OK" and "Cancel" buttons.

5. If you want to use **MQTT Publisher** check **Enable MQTT Publisher**.

5.1. Enter **Broker URL** of the MQTT server.

5.2. Enter **Username** and **Password** of the MQTT server.

5.3. Choose **QoS** of MQTT messages.

5.4. Check **Enable TLS/SSL** if you want to use server certificate for encryption messages.

5.5. Enter **Certificate filename**. File should be placed in */private/* folder in the directory where TeslaSCADA2 OPC UA Server execution file.

5.6. Check **Enable Client Certificate** if you want to use client certificate for encryption messages.

5.7. Enter **Client certificate** filename. File also should be placed in */private/* folder.

5.8. Enter **Client Private key** filename. File also should be placed in */private/* folder.

5.9. Enter **Private key password**.

5.10. Check **PEM formatted** if your certificate and key files are PEM formatted.

Publisher's topics are consists of the «name of the project +/Tags/+tagname» for tags and «name of the project+/Events/+tagname» for events.

Save project

To save project:

1. Click on the **Save** icon in the toolbar or select the menu item *File* and *Save*. The first time you save a new project, you will be asked for a location.

2. Now select the location and click the button *Save* (TeslaSCADA OPC UA Server extension .tsp2).

Open project

To open project:

1. Click on the **Open** icon in the toolbar or select the menu item *File* and *Open*.

2. Now select the project and click *Open* (TeslaSCADA project extension .tsp2).

Edit project properties

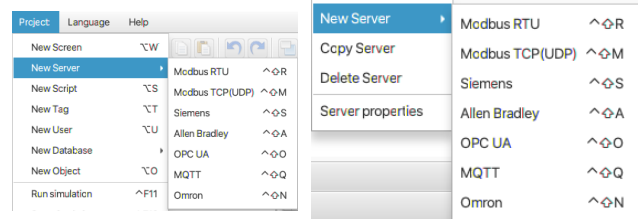
To edit project properties:

1. Click on the **Properties** icon in the toolbar or select the menu item *Project* and *Properties*.

Servers

Create server

To create a new server select the menu item *Project* and *New Server* or choose **Servers** on the **Project Window**, click right button on it and choose *New Server* item. Choose server you want to add to your project.



Modbus RTU server

To create a new Modbus RTU server select the menu item *Modbus RTU*. You'll see the following window:

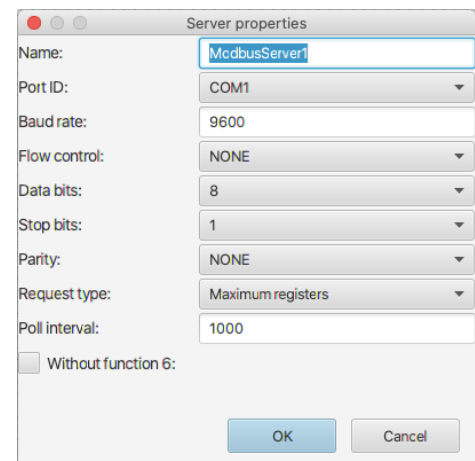
1. In the **Name** enter the name of the Modbus RTU server.
2. Choose **Port ID** (*portid*). If this port won't be able to open in TeslaSCADA2 Runtime other port will be tried to find and open.
3. Enter Modbus RTU **Baud rate** (*baudrate*).
4. Choose **Flow control** of the port. (*flowcontrol*). It can be *NONE*, *RTSCTS* and *XONXOF*.
5. Choose number of **Data bits** (*databits*). It can be 5, 6, 7 and 8.
6. Choose number of **Stop bits** (*stopbits*). It can be 1, 1.5 and 2.
7. Choose **Parity** (*parity*). It can be *NONE*, *EVEN*, *ODD*, *MARK* and *SPACE*.
8. Choose **Request type** (*requesttype*):

- *Maximum registers* - if you choose this type the application during polling will send maximum modbus pointers in 1 polling request.

- *Consecutive registers* - if you choose this type the application during polling will send only consecutive modbus pointers in 1 polling request.

- *1 pointer registers* - if you choose this type the application during polling will send only registers used by 1 pointer in 1 polling request.

9. Check **Without function 6** if your controller doesn't support Modbus writing function 6 (*withoutfun*).



Modbus server

To create a new Modbus server select the menu item *Modbus*.

You'll see the following window:

1. In the **Name** enter the name of the Modbus server.
2. Write IP address or DNS in the **IP or DNS** field (*ipaddress*).
3. Enter Modbus server port in the **Port** (*port*).
4. Define the polling interval of the server in the **Poll interval** field (*interval*).
5. Choose communication protocol in the **Type** (*type*).
6. Choose **Request type** (*requesttype*):

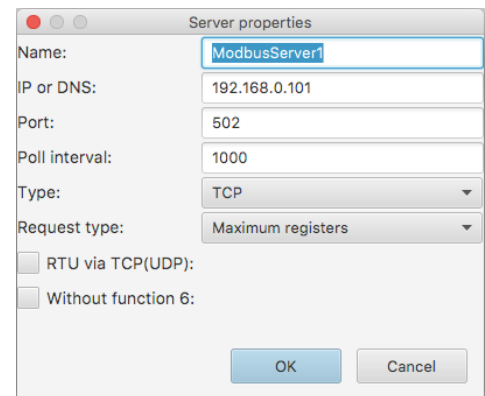
- *Maximum registers* - if you choose this type the application during polling will send maximum modbus pointers in 1 polling request.

- *Consecutive registers* - if you choose this type the application during polling will send only consecutive modbus pointers in 1 polling request.

- *1 pointer registers* - if you choose this type the application during polling will send only registers used by 1 pointer in 1 polling request.

7. Check **RTU via TCP(UDP)** if you use Modbus converter from serial into TCP(UDP) protocol (*rtuviatcp*).

8. Check **Without function 6** if your controller doesn't support Modbus writing function 6 (*withoutfun*).



Siemens server

To create a new Siemens server select the menu item *Siemens*. You'll see the following window:

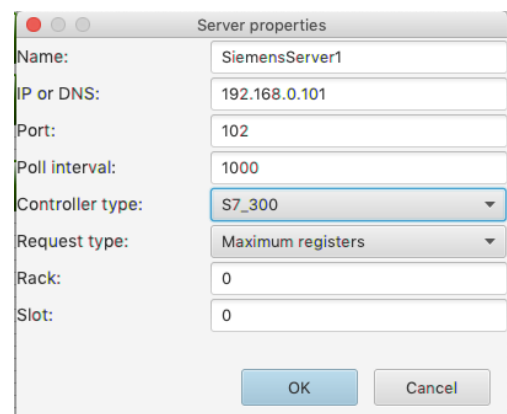
1. In the **Name** enter the name of the Siemens server.
2. Write IP address or DNS in the **IP or DNS** field (*ipaddress*).
3. Enter Siemens server port in the **Port** (*port*).
4. Define the polling interval of the server in the **Poll interval** field (*interval*).
5. Choose type of the Siemens PLC in the **Controller type** (*plctype*).
6. Choose **Request type** (*requesttype*):

- *Maximum registers* - if you choose this type the application during polling will send maximum siemens pointers in 1 polling request.

- *1 pointer registers* - if you choose this type the application during polling will send only registers used by 1 pointer in 1 polling request.

7. Enter rack number in the **Rack** field (*rack*).

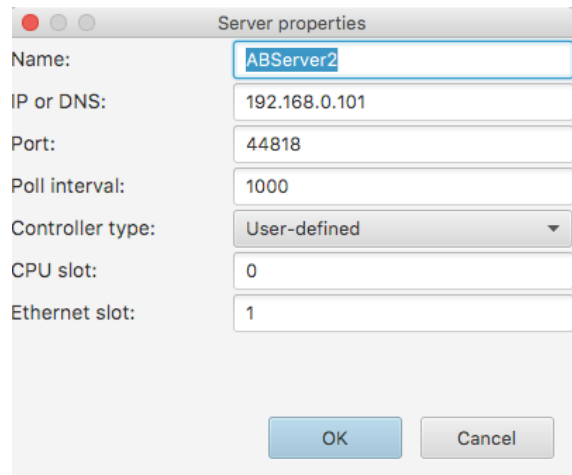
8. Enter slot number in the **Slot** field (*slot*).



Allen Bradley server

To create a new Allen Bradley server select the menu item *Allen Bradley*. You'll see the following window:

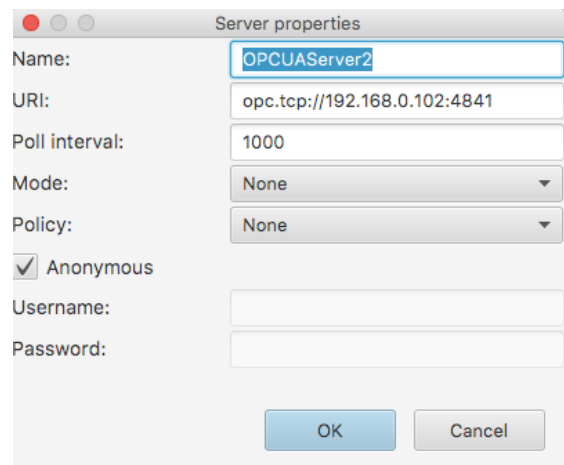
1. In the **Name** enter the name of the Allen Bradley server.
2. Write IP address or DNS in the **IP or DNS** field (*ipaddress*).
3. Enter Allen Bradley server port in the **Port** (*port*).
4. Define the polling interval of the server in the **Poll interval** field (*interval*).
5. Choose type of the Allen Bradley PLC in the **Controller type** (*plctype*).
6. Enter PLC's cpu slot number in the **CPU slot** field (*cpuslot*).
7. Enter PLC's backplane number in the **Backplane** field (*ethernetslot*).



OPC UA server

To create a new OPC UA server select the menu item *OPC UA*. You'll see the following window:

1. In the **Name** enter the name of the OPC UA server.
2. Write OPC UA server address in the **URI** field (*uri*).
3. Define the polling interval of the server in the **Poll interval** field (*interval*).
4. Choose security mode in the **Mode** (*mode*).
5. Choose security policy in the **Policy** (*policy*).
6. Check **Anonymous** if you don't use User token (*anonymous*).
7. Enter **Username** and **Password** into relevant fields if you use User token (*username and password*).



MQTT server

To create a new MQTT server select the menu item *MQTT*. You'll see the following window:

1. In the **Name** enter the name of the MQTT server.
2. Write MQTT server address in the **URI** field (*uri*).
3. Enter **Username** and **Password** into relevant fields (*username and password*).
4. Check **Enable TLS/SSL** if you want to use server certificate for encryption messages (*enablessl*).
5. Enter **Certificate filename**. File should be placed in */private/* folder in the directory where TeslaSCADA2 Runtime execution file (*sslfilename*).
6. Check **Enable Client Certificate** if you want to use client certificate for encryption messages (*enableclientcert*).

7. Enter **Client certificate*** filename. File also should be placed in /private/ folder (*clientcertname*).
8. Enter **Client Private key*** filename. File also should be placed in /private/ folder (*clientprivatekey*).
9. Enter **Private key password*** (*privatekeypassword*).
10. Check **PEM formatted*** if your certificate and key files are PEM formatted (*pem*).

* If you use this project for iOS (iPhone or iPad) you should use .p12 format for the file of the certificate. To create .p12 file you should in openssl utility use this type of command:

```
openssl pkcs12 -export -out [your file name].p12 -in
[your file name].crt -inkey [your file name].key
```

For example:

```
openssl pkcs12 -export -out client.p12 -in client.crt -inkey client.key
```

The name of your .p12 you should place in the **Client certificate** field (client.p12 in our example). **Client Private Key** you can left empty. In the **Private key password** you should enter password of the .p12 file. **PEM formatted** you can left unchecked. All .p12 files are PEM formatted.

Omron server

To create a new Omron server select the menu item *Omron*. You'll see the following window:

1. In the **Name** enter the name of the Omron server.
2. Write IP address or DNS in the **IP or DNS** field (*ipaddress*).
3. Enter Omron server port in the **Port** (*port*).
4. Define the polling interval of the server in the **Poll interval** field (*interval*).
5. Choose communication protocol in the **Type** (*type*).
6. Enter **Network address (DNA)** (*dna*).
7. Enter **Node address (DA1)**. For TCP protocol it will be chosen automatically during communication (*da1*).
8. Enter **Unit number (DA2)** (*da2*).

Open server properties

To open server properties:

1. Double click on the server properties which you want to open.

or

2. Right click on the server properties which you want to open and choose *Server properties* item.

Copy server

To copy server:

1. Right click on the server you want to copy and choose *Copy server* item.

Delete server

To delete server:

1. Right click on the server you want to delete and choose *Delete server* item.
-

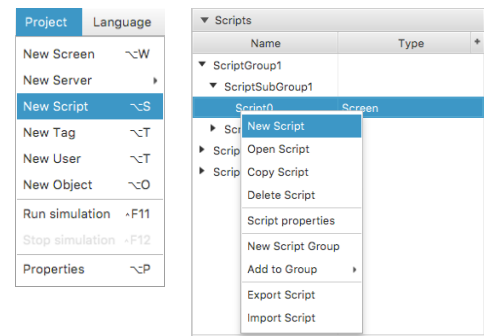
Scripts

Create script

To create a new script select the menu item *Project* and *New Script* or choose **Scripts** on the **Project Window**, click right button on it and choose *New Script* item.

You'll see the following window:

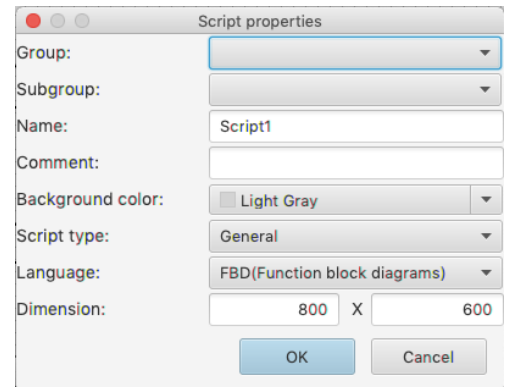
1. In the **Name** enter the name of the screen.
2. Optionally, specify a meaningful **Comment**.
3. Choose **Background** color.
4. Select **Script type**: *General* or *Screen*. General script bind to the whole project. Screen script bind to the Screen.
5. Choose **Language** you use in this script.
6. Enter **Dimension** of the script's design screen.



Open script

To open script:

1. Right click on the script you want to open and choose *Open script* item.
- or
2. Double click on the script you want to open.



Copy script

To copy script:

1. Right click on the script you want to copy and choose *Copy script* item.

Delete script

To delete script:

1. Right click on the script you want to delete and choose *Delete script* item.

Edit script properties

To edit script properties:

1. Right click on the script you want to edit and choose *Script properties* item.

New script group

Create new group of the scripts.

Add to group

Add this script to one of the available group.

Export script

To export script:

1. Right click on the script you want to export and choose *Export script* item.
2. Now select the location and click the button *Save* (TeslaSCADA script extension .tsp2script).

Import script

To import script:

1. Right click on the script window and choose *Import script* item.
2. Now select the script file and click *Open* (TeslaSCADA screen extension .tsp2script).

Tags

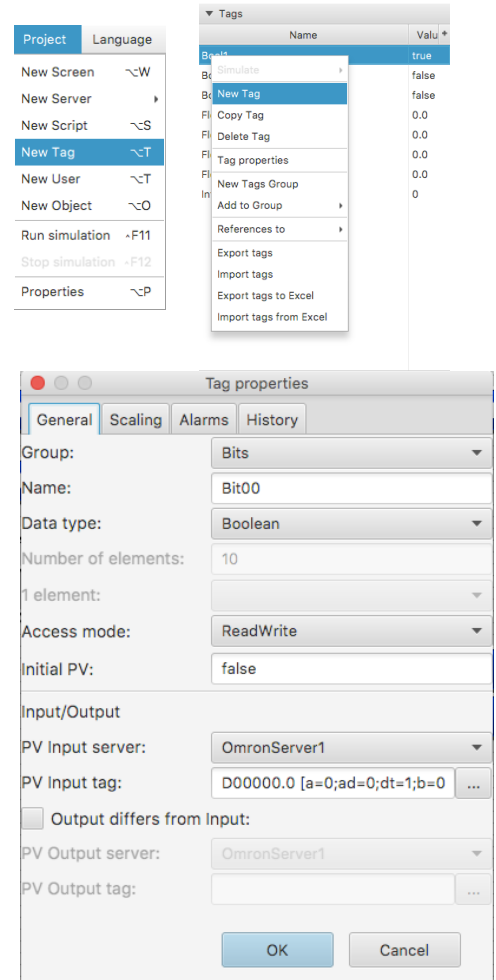
Create tag

To create a new tag select the menu item *Project* and *New Tag* or choose **Tags** on the **Project Window**, click right button on it and choose *New Tag* item.

You'll see the following window:

On the *General* tab:

1. Choose **Group** of the tag.
2. In the **Name** enter the name of the tag. The name should be unique for the project.
3. Choose **Data type**.
4. If you select *String* or *Array* data types enter **Number of elements** (letters).
5. If you select *String* or *Array* data types choose data type of **1 element** (letter).
6. Choose **Access mode** to the tag: *Read*, *Write* or *ReadWrite*.
7. Enter default tag's value into **Initial PV**.
8. In the **Input/Output** section bind tag to the server's tag. In the **PV Input server** choose server you want to bind. Then click «...» button to set up server's tag settings or enter it into the **PV Input tag**.
9. If the output server's tag differs from the input server's tag check **Output differs from input** and select **PV Output server** and enter **PV Output tag**.



Depending on the type of **PV Input server** or **PV Output server** you'll see different server's tag (pointer) settings window:

Modbus tag settings

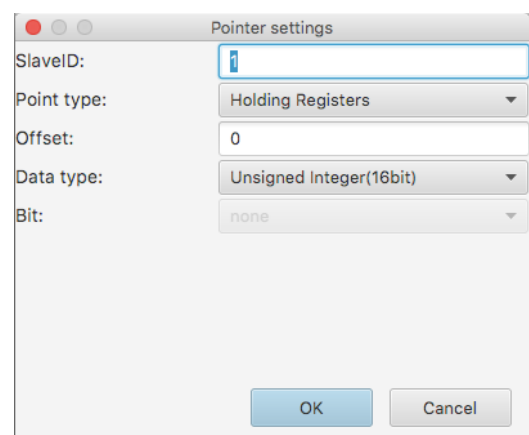
You'll see the following window:

1. Enter **SlaveID** of the modbus device.
2. Choose **Point type** of the register.
3. Write offset of the register into **Offset**.
4. Choose **Data type** of the modbus tag.
5. Choose number of **Bit** if the point type is boolean.

After clicking OK you'll get pointer settings in **PV Input tag** encoded in String like:

s=1;pt=3;o=0;dt=2;

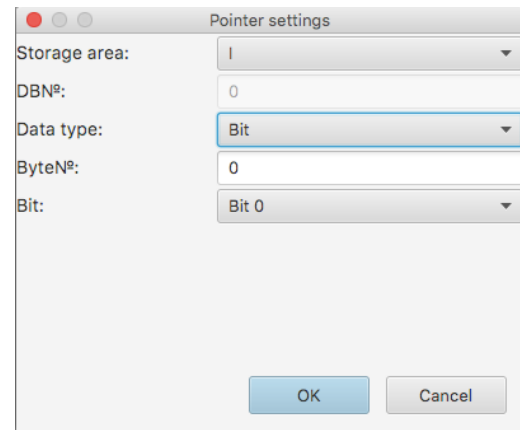
Where: s - SlaveID, pt - Point type, o - Offset, dt - Data type



Siemens tag settings

You'll see the following window:

1. Choose **Storage area** of the siemens tag: *I, Q, M* or *DB*.
2. Write DB number in the **DB№** field if you choose DB storage area.
3. Choose **Data type** of the siemens tag.
4. Enter byte number of the area into **Byte№** field.
5. Choose number of **Bit** if the data type is *Bit*.



After clicking OK you'll get pointer settings in **PV Input tag** encoded in String like:

I0.0 [a=0;db=0;dt=0;bn=0;b=0;]

Where: a - Storage area, db - DB№, dt - Data type, bn - Byte№, b-Bit.

(I0.0 - its just for Siemens users and it's not used in encoding)

AllenBradley tag settings

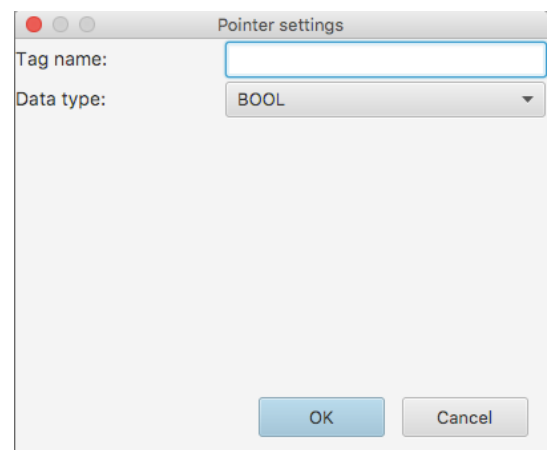
You'll see the following window:

1. Enter **Tag name**.
2. Choose **Data type** of the allen bradley tag.

After clicking OK you'll get pointer settings in **PV Input tag** encoded in String like:

type=0;name=Tag

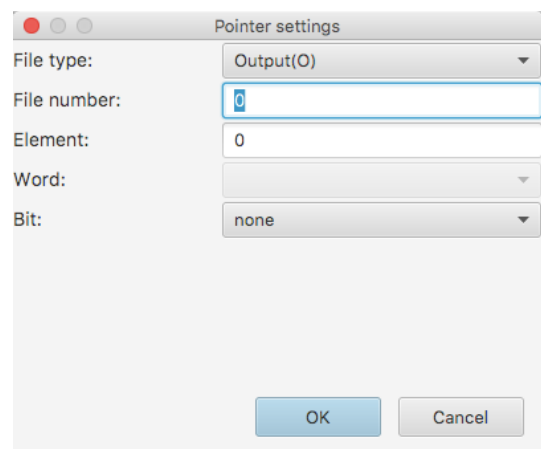
Where: type - Data type, name - Tag name



Micrologix tag settings

If you choose Micrologix or SLC500 controller type in the Allen Bradley server settings you'll see the following window:

1. Choose **File type** of the server's tag.
2. Write **File number** in the field.
3. Enter **Element** of the servers tag.
4. Choose **Word** for some file types.
5. Choose number of **Bit**.



After clicking OK you'll get pointer settings in **PV Input tag** encoded in String like:

O0:0

Where: O - File type, 0 - File number, 0-Element

OPC UA tag settings

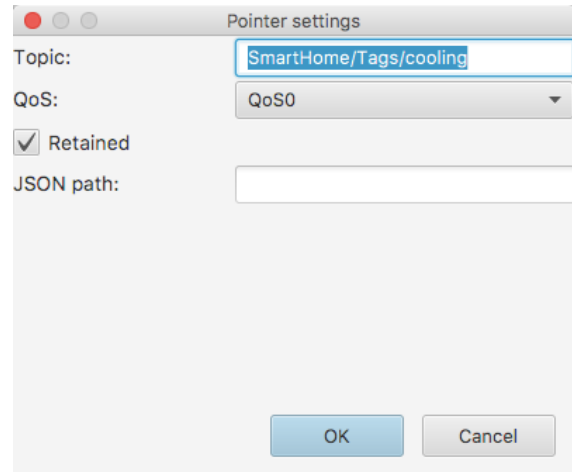
After clicking «...» button when you choose OPC UA server you'll get into the Address Space window. Browse through the address space by double clicking on the nodes and choose the tag(node) you need by clicking right button on it and choosing *Select* menu item on the popup window.

You'll get NodeID in **PV Input Tag**.

MQTT tag settings

You'll see the following window:

1. Enter **Topic**.
2. Choose **QoS** of the MQTT tag.
3. Check **Retained** if you want to use this property.
4. If MQTT response contains JSON array enter **JSON path** to parse the value. For example if response is: «{foo: bar, lat: 0.23443, long: 12.3453245}» to get long value enter «long» in the field. If response is not JSON format left field empty. If response contains multi dimension JSON format, separate keys by commas without blank spaces.



After clicking OK you'll get pointer settings in **PV Input tag** encoded in String like:

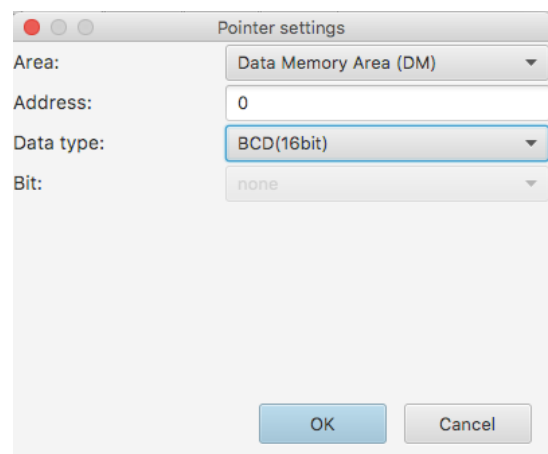
t=SmartHome/tags/cooling;qos=0;r=1;json=

Where: t - Topic, qos - QoS, r-Retained, json - JSON path

Omron tag settings

You'll see the following window:

1. Choose address **Area**.
2. Enter **Address** of the tag.
3. Choose **Data type**.
4. Choose **Bit** for Binary data type.



After clicking OK you'll get pointer settings in **PV Input tag** encoded in String like:

D00000 [a=0;ad=0;dt=16;]

Where: a - Area, ad - Address, dt - Data type, b-Bit.

(D0000 - its just for Omron users and it's not used in encoding)

On the *Scaling* tab of the *Tag properties* window:

1. Check **Enable I/O scaling** if you want to scale a value get from the server.

2. Enter minimum server tag's value into **Raw value minimum** field.
3. Enter maximum server tag's value into **Raw value maximum** field.
4. Enter minimum tag's value in engineer units into **EU value minimum** field.
5. Enter maximum tag's value in engineer units into **EU value maximum** field.
6. Write tag's value offset into **EU value offset**.

When you get some value from the server application use this formula:

$$\text{value} = (\text{value} - \text{rawmin}) * (\text{eumax} - \text{eumin}) / (\text{rawmax} - \text{rawmin}) + \text{eumin} + \text{offset}$$

On the **Alarms** tab of the **Tag properties** window:

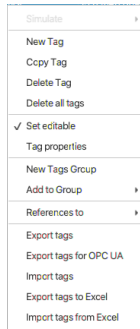
1. Check **Enable alarms** if you want to use alarms for this tag.
2. Check **HiHi**, **Hi**, **Lo**, **LoLo** or **Normal** if you want to use the correspondent alarm(event).
3. Write **Limit** for the correspondent alarm(event). If the value of the tag plus **Deadband** will be more than **HiHi** or **Hi** limit the correspondent alarm will be called and be written into Event database. If the value of the tag minus **Deadband** will be less than **LoLo** or **Lo** limit the correspondent alarm will be raised and be written into Event database.
4. Enter **Priority** for the correspondent alarm(event). If the priority of the alarm(event) is less than value of **Notifications(Priority<)** you set in the project properties the notification dialog will be called.
5. Enter **Message** for the correspondent alarm(event).
6. Check **Enable OPC UA event** if you bind this tag to the OPC UA server tag(node) and you want to use EventNotifier of this tag(node).

On the **History** tab of the **Tag properties** window:

1. Check **Enable history** if you want to storage values of this tag.
2. Enter **Storage period(ms)**.
3. Check **Store in DB** if you want to store data in history database.

On the **Script** tab of the **Tag properties** window:

1. Check **Enable script** if you want to use script bind to this tag's value.
2. Choose **Script** you want to bind to this tag's value.
3. Enter **Value** you want to compare with current value.
4. Choose **Type** of the compare operation.
5. Enter **Deadband** for the value.



Copy tag

To copy tag:

1. Right click on the tag you want to copy and choose *Copy tag* item.

Delete tag

To delete tag:

1. Right click on the tag you want to delete and choose *Delete tag* item.

Delete all tags

To delete all tags:

1. Right click on any tag and choose *Delete all tags* item.

Set editable

To edit tag in the table:

1. Right click on any tag and choose *Set editable* item.

Edit tag properties

To edit tag properties:

1. Right click on the script you want to edit and choose *Tag properties* item.

or

2. Double click on the tag you want to edit.

New group tags

Create new group of the tags.

Add to group

Add this tag to one of the available group.

Reference to

To find objects and scripts that use this tag:

1. Right click on any tag and choose *Reference to* item. And choose where you want to find this tag in objects or in the scripts.

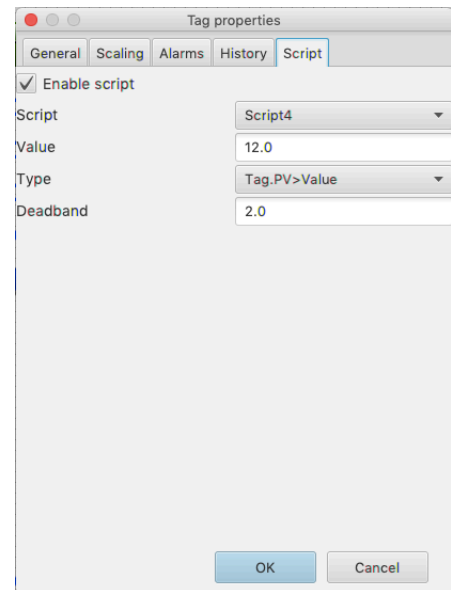
Export all tags

To export all tags:

1. Right click on the tags window and choose *Export all tags* item.
2. Now select the location and click the button *Save* (TeslaSCADA tags extension .tsp2tags).

Export tags for OPC UA

If you want to use this project like OPC UA server you can export tags for OPC UA client. The item will be created with OPC UA nodes in PV input source of the tag.



Import tags

To import tags:

1. Right click on the tags window and choose *Import tags* item.
2. Now select the tags file and click *Open* (TeslaSCADA screen extension .tsp2tags).

Export tags to Excel

To export tags to Excel:

1. Right click on the tags window and choose *Export tags to Excel* item.
2. Now select the location and click the button *Save* (Tags will be saved in Excel file).

Import tags from Excel

To import tags from Excel:

1. Right click on the tags window and choose *Import tags from Excel* item.
2. Now select the Excel file with tags and click *Open*.

Design script

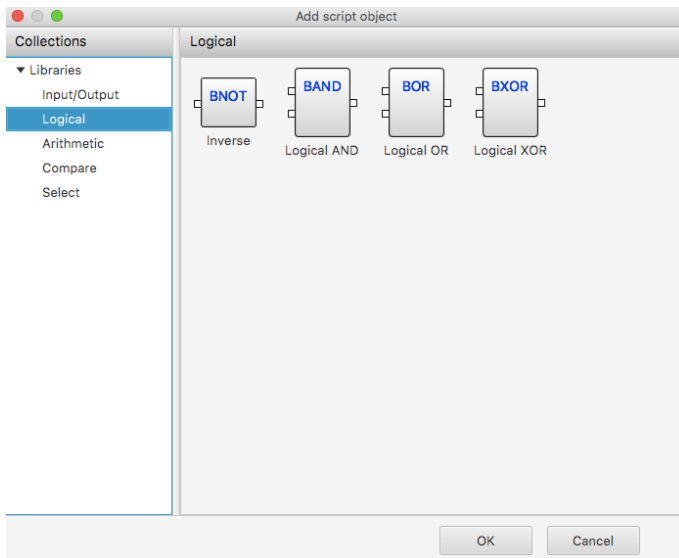
To start designing the script you want, you should double click on it or click right button on the **Project window->Scripts** and choose *Open script*. For creating scripts you should use FBD objects.

Create script object

Add new object on the screen you can in this way: click right button on the **Canvas** and choose *New object* item

New Object
Duplicate
Erase

You'll see the **Add script object** window:



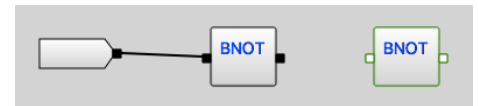
Select library which object you want to use in your project (all libraries and their objects described below). Select object you can in several ways:

1. By double clicking on the object.
2. By clicking on the object (select rectangle will appear) and then clicking OK button.
3. By clicking right button and choosing *Select* item.

Add script object window will disappear and you can select the location on the screen where you want to place an object.

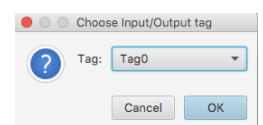
Connect script objects

To connect two objects, click the end of the first (the end to paint over) and click start the second. This will bring up a line connection.



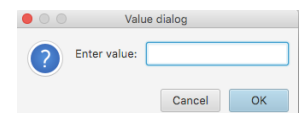
Bind script object to the tag

You can bind Input/Output script objects to the tag. To do this click on Input/Output script object, dialog will appear. Select tag you want to bind.



Enter value to the value script object

You can enter value to value script objects. To do this click on value script object, dialog will appear. Enter value you want to use with this object.



Duplicate script object

You can duplicate script object. Right click on the object you want to duplicate and select *Duplicate* menu item.

Erase script object

You can erase script object. Right click on the object you want to erase and select *Erase* menu item.

Erase connection line

You can erase connection line. Right click on the line you want to erase and select *Erase* menu item.

Script objects of FBD language

Below description of script libraries and object.

Input/Output library

Input tag - this script object used to bind input tag to the script.

Output tag - this script object used to bind output tag to the script.

Value - this script object used to bind input constant value to the script.

Logical library

Inverse - this script object used to inverse input boolean value (Output = ! Input).

Logical AND - this script object used to logical operation AND for input boolean values (Output = Input & Input2).

Logical OR - this script object used to logical operation OR for input boolean values (Output = Input || Input2).

Logical XOR - this script object used to logical operation XOR for input boolean values (Output = Input XOR Input2).

Bitmap operations library

Inverse - this script object used to inverse input integer value (Output = ~ Input).

Bitmap AND - this script object used to logical operation AND for input integer values (Output = Input & Input2).

Bitmap OR - this script object used to logical operation OR for input integer values (Output = Input || Input2).

Bitmap XOR - this script object used to logical operation XOR for input integer values (Output = Input XOR Input2).

Left Shift - this script object used to left shift bits of input value (Output = Input << № of bits).

Right Shift - this script object used to right shift bits of input value (Output = Input >> № of bits).

Bytes to Short - this script object used to pack 2 bytes in the short (Output = Input<<8+Input2).

Short to Bytes - this script object used to unpack short value in 2 bytes (Output = Input[Input2]).

Shorts to Int - this script object used to pack 2 shorts in the int (Output = Input<<16+Input2).

Int to Shorts - this script object used to unpack int value in 2 shorts (Output = Input[Input2]).

Read bit - this script object used to read bit of the input value (Output = Input[Input2]).

Set bit - this script object used to set bit of the input value (Output = Input | 1<<Input2).

Reset bit - this script object used to reset bit of the input value (Output = Input & ~(1<<Input2)).

Arithmetic library

Addition - this script object used to arithmetic operation addition for input values (Output = Input + Input2).

Subtraction - this script object used to arithmetic operation subtraction for input values (Output = Input - Input2).

Multiplication - this script object used to arithmetic operation multiplication for input values (Output = Input * Input2).

Division - this script object used to arithmetic operation division for input values (Output = Input / Input2).

Modulo - this script object used to arithmetic operation modulo for input values (Output = Input % Input2).

Power - this script object used to arithmetic operation power for input values (Output = $\text{Input}^{\text{Input2}}$).

ABS - this script object used to arithmetic operation absolute for input value (Output = $|\text{Input}|$).

Sign - this script object used to arithmetic operation sign for input value (Output = $-\text{Input}$).

Int - this script object used to arithmetic operation for getting integer part of the input value (Output = $\text{int}(\text{Input})$).

Sqrt - this script object used to arithmetic operation sqrt of the input value (Output = $\text{Math.Sqrt}(\text{Input})$).

Ln - this script object used to arithmetic operation ln (natural logarithm) of the input value (Output = $\text{Ln}(\text{Input})$).

Log - this script object used to arithmetic operation log (logarithm) of the input value (Output = $\text{Log}_{\text{Input2}}\text{Input}$).

Compare library

Equal - this script object used to comparison operation equal for input values (Output = $\text{Input} == \text{Input2}$).

Not Equal - this script object used to comparison operation not equal for input values (Output = $\text{Input} != \text{Input2}$).

Greater - this script object used to compare operation greater for input values (Output = $\text{Input} > \text{Input2}$).

Less - this script object used to compare operation less for input values (Output = $\text{Input} < \text{Input2}$).

Equal or Greater - this script object used to compare operation equal or greater for input values (Output = $\text{Input} \geq \text{Input2}$).

Equal or Less - this script object used to compare operation equal or less for input values (Output = $\text{Input} \leq \text{Input2}$).

Select library

Selectable enable - this script object used to select value form Input2 if Input1 is true (IF $\text{Input} == \text{true}$ THEN $\text{Output} = \text{Input2}$).

Selectable negate - this script object used to select value form Input2 if Input1 is false (IF $\text{Input} == \text{false}$ THEN $\text{Output} = \text{Input2}$).

Minimum - this script object used to select minimum value of Input2 and Input1 (Output = $\text{Min}(\text{Input}, \text{Input2})$).

Maximum - this script object used to select maximum value of Input2 and Input1 (Output = $\text{Max}(\text{Input}, \text{Input2})$).

Arrays library

Index read - this script object used to select array's element. Input1 is an array. Input2 is index of element (Output = $\text{Input1}[\text{Input2}]$).

Index write - this script object used to change array's element. Input1 is an element. Input2 is index of element (Output[Input2] = Input1).

Triggers/Counters library

Rising edge trigger - this script object used to generate rising impulse duration PV ms when Input1 get TRUE from FALSE.

Falling edge trigger - this script object used to generate rising impulse duration PV ms when Input1 get FALSE from TRUE.

RS trigger- this script object used to imitate RS trigger.

Timer ON- this script object used for delay timer for the duration PV when Input1 get TRUE from FALSE.

Timer OFF- this script object used for delay timer for the duration PV when Input1 get FALSE from TRUE.

Counter- this script object used to count impulses of boolean value in Input1. Counter resets when Output become equal PV.

Counter Down- this script object used to count impulses of boolean value in Input1. Counter starts from value PV. Counter resets when Output become equal 0.

Multivibrator - this script imitates impulse generator with PV period. It starts when IN1 changed from *false* to *true*.

Trigonometric library

Degrees to radians - this script object used to convert degrees to radians.

Radians to degrees - this script object used to convert radians to degrees.

Sine - this script object used to calculate sin of Input value. (Output = sin(Input)).

Cosine - this script object used to calculate cos of Input value. (Output = cos(Input)).

Tangent - this script object used to calculate tag of Input value. (Output = tag(Input)).

Arc Sine - this script object used to calculate arc sin of Input value. (Output = arc sin(Input)).

Arc Cosine - this script object used to calculate arc cos of Input value. (Output = arc cos(Input)).

Arc Tangent - this script object used to calculate arc tag of Input value. (Output = arc tag(Input)).

Hex operations library

Hex to Integer - this script object converts hex value into integer.

Integer to Hex - this script object converts integer value into hex.

Call screen library

Call screen - this script object used to call screen when Input's value turns from *false* to *true*.

Call popup - this script object used to call popup screen when Input's value turns from *false* to *true*.

Strings library

Equal Strings - this script object compare two strings in Inputs and if their are equal it sets true into Output value.

String to Double - this script object converts Input's string value into Output's double value.

Double to String - this script object converts Input's double value into Output's string value.

Strings concat - this script object concatenate Input's strings values into Output's string value. (Output = Input1+Input2).

String cut end - this script object cuts end of Input's string value by the № of characters and place result into Output's string value.

String cut begin - this script object cuts begin of Input's string value by the № of characters and place result into Output's string value.

Date and time library

Current date and time - this script object used to get date and time components depending on Input value:

0 - get seconds.

1 - get minutes.

2 - get hour of the day considering AM/PM.

- 3 - get hour of the day.
- 4 - get day of the week (1-Sunday, 2-Monday...).
- 5 - get day of month.
- 6 - get month (0 - January, 1 - February...).
- 7 - get year.
- 8 - get minutes of the day (hour*60 + minutes).

Servers library

IP or URI address - this script object used to change server's IP or URI address when Input's value changed.

Reconnect - this script object used to reconnect server when Input's value turns from *false* to *true*.

Recipes library

Select recipe - this script object used to choose recipe row. Input2 is an input that contains name of the recipe. Input1 is number of the row (starting from 1). Output = true if recipe row is chosen.

Base64 library

Decode Base64 to Array - this script object used to decode Base64 string to byte array. Input contains base64 encoded string. In Output will be decoded byte array.

Encode Array to Base64 - this script object used to encode byte array to Base64 string. Input contains byte array. In Output will be encoded Base64 string.

Description of ST(Structured text) language

When you choose ST(Structured text) language in script properties and open this script you'll see two windows like in the picture. Top window is a **Code area** and below window is a **Debug(or log) area**. You can enter your script program in the top window and compile this code by clicking **Run** button on the *Tool bar*. All debug and log information you can see in the below window. Later in this chapter we will describe the rules of the ST language.



What is Structured Text Programming?

Structured Text for TeslaSCADA2 is different from PLC programming language defined by PLCOpen in IEC 61131-3. The programming language is text-based, compared to the graphics-based Function Block Diagram.

If you are already familiar with high-level programming languages like Java, PHP, Python and C, Structured Text will seem familiar to you. The syntax of Structured Text is developed to look like the syntax of a high-level programming language with loops, variables, conditions and operators.

Before you read this tutorial I recommend that you take a brief look at this TeslaSCADA2 program written in Structured Text. Try to see if you can understand the function of this program. Does Structured Text look familiar to you?

```
1 int a = 5;
2 int b = 7;
3 int c;
4 if (a>b) {
5     c=a+b;
6     print(c);
7 }
8 else{
9     c=a-b;
10    print(c);
11 }
```

Starting with the Syntax of Structured Text

The syntax of a programming language is the definition of how it is written. To be more precise, what symbols is used to give the language its form and meaning. As you can see in the example, Structured Text is full of colons, semicolons and other symbols. All these symbols has a meaning and is used to represent something. Some of them are operators, some are functions, statements or variables. All the details of the syntax will be explained as you move through this tutorial. But there are some general rules for the syntax of Structured Text you should know about. You don't have to memorize all the syntax rules for now, as you will when you get your hands into the programming:

All statements are divided by semicolons

Structured Text consists of statements and semicolons to separate them.

The language is case-sensitive

It is good practice to use upper- and lowercase for readability.

Spaces have no function

But they should be used for readability.

What's really important to understand here is that, when you write a TeslaSCADA2 program in IDE in Structured Text, your computer will translate that to a language the TeslaSCADA2 Runtime can understand. Before you use project that contains the Structured Text TeslaSCADA2 program to your TeslaSCADA2 Runtime, the IDE will compile your program. This means that it will translate the code to a sort of machine code which can be executed by the TeslaSCADA2 Runtime.

The compiler uses the syntax of the programming language to understand your program.

For example: Each time the compiler sees a semicolon, it will know that the end of the current statement is reached. The compiler will read everything until it reaches a semicolon, and then execute that statement.

Comment Syntax

In textual programming languages you have the ability to write text that doesn't get executed. This feature is used to make comments in your code. Comments are good, and as a beginner you should always comment your code. It makes it easier to understand your code later. In Structured Text you can make either one line comments or multiple line comments.

Single line comment:

```
// comment
```

Multiple line comment:

```
/* start comment
```

```
...
```

```
end comment */
```

Making Statements with Structured Text

So, Structured Text consists of statements. But what is statements? A statement is you telling the TeslaSCADA2 what to do. Let's take the first statement as an example:

```
bool x;
```

The compiler will read this as one statement, because when it reaches the semicolon, it knows that this is the end of that statement. Remember, statements are separated by semicolons. That's the main syntax rule of this language. In this statement you are telling the TeslaSCADA2 to create a variable called X and that variable should be a BOOL type. By default value of the variable is *false*.

Types in Structured Text

Data types of Structured Text are similar to data types of TeslaSCADA2:

Data Type	Format	Range
bool	Boolean	False/True
byte	Byte	-128 ... 127
short	Short	-32768 ... 32767
int	Integer	$-2^{31} \dots 2^{31}-1$
long	Long Integer	$-2^{63} \dots 2^{63}-1$
float	Float	$\pm 3.40282347\text{E}+38\text{F}$
double	Double	$\pm 1.79769313\text{E}+308$
string	Character string	“My string”
array	Array	byte[], short[], int[], float[]

Examples of variable initialisation:

```
bool x=false;
```

```
byte b = 2;
```

```
short s = 45;
```

```
int i = -4546;
```

```
long l = 394394832;
```

```
float f = 1.23;
```

```
double d = -545.64;
```

```
string str = “Hello”;
```

```
byte bytes[10] = [1,2,3,4,5,6,7,8,9,10];
```

Operators and Expressions in STL

The next thing you should know about is operators. Operators are used to manipulate data and is a part of almost any programming language. This leads us to the second thing you should know about – expressions. Just like operators, expressions are a crucial part of programming languages.

An expression is a construct that, when evaluated, yields a value. This means that when the compiler compiles an expression, it will evaluate the expression and replace the statement with the result.

Take this example with the two variables A and B. A contains the value 10 and B contains 8.

A+B

The result of this expression is 18. So instead of A+B, the compiler will put in the value 18.

An expression are composed of operators and operands. So what are operators and operands?

Since, you just saw an example of an expression, you just saw both an operator and two operands. A and B are both operands and the + is an operator. Remember that operators are used to manipulate data. That is exactly what the + is doing. It is taking the value of the variable A and adding it to the value in B. The + is also called the addition operator because the operation is addition.

Operators

There are several operators available in Structured Text language:

Operation	Symbol	Precedence
Parenthesization	(expression)	Highest
Negation Complement	– !	
Multiply Divide Modulo	* / %	
Add Subtract	+ –	
Left Shift Right Shift	<< >>	
Comparison	<, >, <=, >=, ==, !=	
Boolean AND Boolean OR Boolean XOR	& ^	Lowest

All the operators in the table above are sorted after precedence. This is also called order of operations, and you may know about it from mathematics. The order of operations is the order in which the operations are executed or calculated. Just take a look at this expression:

A + B * C

How will this expression be evaluated by the compiler?

There are two operations left: multiply and addition. But since multiply has a higher precedence, that will be the first to be evaluated. B * C comes first and then the result is added to A.

Every time an expression is evaluated, the evaluation follows the order of precedence as in the table above.

4 Types of Operators, 4 Types of Expressions

The operators used for expressions in Structured Text can be divided into four groups. Each group of operators will have its specific function and will yield a specific data type.

1. Arithmetic Operators

2. Relational Operators

3. Logical Operators

4. Bitwise Operators

Arithmetic Operators

All the **arithmetic operators** are often just called mathematical operators because they represent math. The result will always be the mathematical result of the expression.

- + (add)
- – (subtract/negate)
- * (multiply)
- / (divide)
- % (modulo divide)

Example:

15 % 4

Result:

3

Relational Operators

To compare or find a relation between two values you can use one of the relational operators. They are used for comparison and the result will be a boolean value (BOOL type), either TRUE or FALSE.

- == (equal)
- < (less than)
- <= (less than or equal)
- > (greater than)
- >= (greater than or equal)
- != (not equal)

Example:

TEMPERATURE = 93.9;

TEMPERATURE >= 100.0

Result:

false

Logical Operators

If you want to compare boolean values (BOOL) and make some logic out of it, you have to use logical operators. These operators also yields a boolean value of TRUE or FALSE as a result of the expression.

- &&
- ||
- ^
- !

Example:

```
limitswitch1 = true;
```

```
limitswitch2 = false;
```

```
limitswitch1 || limitswitch2
```

Result:

```
true
```

Bitwise Operators

The last group of operators are called bitwise operators because the operations are performed bitwise. It simply means that a logic operation is performed for each bit of two numbers. The result is a new number – the total result of the bitwise operations.

- &
- |
- ^
- <<
- >>

Example:

```
15 & 8
```

Result:

```
8
```

Since this operation is bitwise the calculation will be per bit. So to understand what's going on here, you have to convert the numbers to binary values:

```
15 = 1111      8 = 1000
```

Now each bit in the number 1111 (15) can be used in a logical operation with the other number 1000 (8): 1111 AND 1000

Bit number	1111 (15)	1000 (8)	Result
0	1	0	0
1	1	0	0
2	1	0	0
3	1	1	1

Operators and Statements

So, in the previous section you learned that **expressions evaluate**. Meaning that all expressions will yield the result and the compiler will replace the expression with the result. But what if you want the TeslaSCADA2 (compiler) not to evaluate something, but to DO something? Statements are the answer. Let's take a look at the actions or statements that you can make in Structured Text.

Assignment Statement and Operator

There are several statements available in Structured Text. All of them represent an action or a condition. Beginning with actions, the most fundamental statement in Structured Text is the assignment statement. Here's how an assignment statement looks like:

A = B;

What does this statement tell the compiler to do? To take the value of the variable B and put it in the variable A. The TeslaSCADA2 is assigning a value to a variable. Here's an even simpler example:

A = 10;

This statement will take the value 10 and put it into the variable A. Or said in another way – the variable A will be assigned the value 10. Since the value of A is now 10, we can make another statement, but this time with an expression:

B = A + 2;

When this line of code is compiled, the expression $A + 2$ will be evaluated to 12. The compiler will replace the expression with the result 12. The statement will now look like this to the compiler:

B = 12;

What will happen now, is that the compiler will assign the value 12 to the variable B.

The last thing is that the = symbol is called the assignment operator.

You can have all sorts of expressions in your assignment statements, from simple values like numbers to variables and functions. Because all expressions will be evaluated first, and then, the result of that evaluation will be used in the assignment statement.

Conditional Statements

A TeslaSCADA2 program is a piece of logic and therefore has to make some decisions.

So in your TeslaSCADA2 program you need a way to make decisions. This brings us to conditional statements. Conditional statements are used for exactly that: To make decisions.

There are one way of doing conditional statements in Structured Text: IF statement.

IF Statements

IF statements are decisions with conditions. There's a special syntax for IF statements. This means, that you have to write it in a certain way for the compiler to understand it. Because just like semicolons are used to end statements, there are special keywords to make an IF statement.

Here's how the syntax for IF statements looks like in STL for TeslaSCADA@:

```
if (boolean expression) {  
    <statement>;  
}  
else if (boolean expression){  
    <statement>;  
}  
else {  
    <statement>;  
}
```

Statement starts with keyword IF. Then parentheses. Between those two brackets are the condition, which is an expression. But not just any expression. A boolean expression.

Boolean and Numeric Expressions

You can divide expressions into two groups depending on what they yield.

Boolean expressions evaluates to a BOOL type value, TRUE or FALSE.

Here's an example of a boolean expression:

```
1 == 1
```

This expression will evaluate to or yield TRUE. A boolean expression could also look like this:

```
1 > 2
```

But this time the boolean expression will evaluate to FALSE, since 1 is not larger than 2.

Numeric expressions evaluates to an integer or a floating point number.

A numeric expression could look as simple as this one:

```
13.2 + 19.8
```

This expression will evaluate to the floating point number 33.0, and therefore is a numeric expression.

Boolean expressions are used in IF statements as conditions.

IF the boolean expression evaluates to TRUE, then the following statements will be executed.

The TeslaSCADA2 will only execute the statements after the open bracket {, if the expression evaluates to TRUE. This is illustrated by the following example:

```
A = 0;  
IF (A == 0) {  
    B = 0;  
}
```

Line number 3 will only be executed if A is equal to 0. In this case it will. A 0 is assigned to the variable A in a statement right before the IF statement.

For now, you've seen a simple IF statement, where statements are only executed if an expression is TRUE. If that expression evaluates to FALSE the statements will simply not be executed. What to do if you want to use multiple conditions? Just like most other programming languages you can use the ELSE IF and ELSE keywords for multiple conditions in the same IF statement.

Both ELSE IF and ELSE are optional in IF statements, but this is how the syntax looks like:

```
if (boolean expression) {  
    <statement>;  
}  
else if (boolean expression) {  
    <statement>;  
}  
else{  
    <statement>;  
}
```

If the boolean expression on line 1 is FALSE, the statements below will simply not be executed. Instead the compiler will check the boolean expression after the ELSE IF keyword. Here it works just like with the IF keyword: If the boolean expression after the keyword is true, the following statements will be executed. At last is the ELSE keyword. It works as a default option for your IF statement. If all the IF and ELSE IF boolean expressions are evaluated to FALSE, the statements after the ELSE keyword will be executed.

Combining Operators for Advanced Conditions

Beside making multiple conditions you can also expand your conditions to include multiple variables. You can combine multiple expressions, typically done with a logical operator, to get a larger expression.

What if you want not just 1 but 2 inputs to be TRUE before an output is set. The expression would look like this:

```
if (INPUT1 & INPUT2) {  
    OUTPUT1 = TRUE;  
}
```

Now the expression will evaluate to TRUE, only if INPUT1 and INPUT2 is TRUE.

Iteration with Repeating Loops

Probably one of the most powerful features in Structured Text is the ability to make loops that repeat lines of code. In relation to TeslaSCADA2 programming loops can be used for many different purposes. You might have a function or a set of statements that you want to execute a certain amount of times or until something stops the loop.

In Structured Text for TeslaSCADA2 you will find 2 different types of repeating loops:

- FOR
- WHILE

Common for all the types of loops is that they have a condition for either repeating or stopping the loop. The condition in FOR and WHILE loops decides whether the loop should repeat or not.

FOR Loops

The first loop is the FOR loop and is used to repeat a specific number of times. This is the syntax of FOR loops in Structured Text for TeslaSCADA2:

```
for (count = initial_value; condition; increment){  
    <statement>;  
}
```

Keyword that starts the FOR loop statement.

count = initial_value

This assignment operation is where you set the initial value you want to count from. Count is the variable name and initial_value is the value you want to start counting from.

;

Semicolon before condition statement.

condition of the loop's continuation.

;

Semicolon before incremental statement.

increment statement. Usually used to increment initial value - count in this case.

Then you place statements between {} that will execute during loops.

WHILE Loops

The while loop is a little different from the FOR loop, because it is used to repeat the loop as long as some conditions are TRUE. A WHILE loop will repeat as long as a boolean expression evaluates to TRUE. Here's the syntax of WHILE loops:

```
WHILE (boolean expression){  
    <statement>;  
}
```

Between the parentheses are the boolean expression. If that boolean expression evaluates to TRUE, all the statements between braces {} will be executed. When } is reached, the boolean expression will be evaluated again. This will happen over and over again until the expression doesn't evaluate to TRUE. But to make the loop stop at one point, you have to change a value in the boolean expression. Only in that way can the boolean expression go from TRUE to FALSE.

Here's an example of a WHILE loop in Structured Text:

```
counter = 0;
while (counter < 10){
    counter = counter + 1;
    machine_status = counter * 10;
}
```

If you look at the third line you will see how the loop will eventually stop repeating. The boolean expression uses the counter variable and checks if its value is less than 10. But since the value of counter is set to 0 right before the WHILE loop, the boolean expression will be TRUE unless counter is changed. That is what's happening in line 3. This is the first statement in the WHILE loop, and with the other statements, are executed each time the loop repeats. In the third line the value of the counter variable is increased by 1. You can say that the incremental value is 1.

In the example above, the loop will repeat 10 times. When the value of count reaches 10, the boolean expression will be evaluated to FALSE (because 10 is not less than 10) and the loop will stop.

You can also use the BREAK keyword in the WHILE loop to stop repeating the loop before the boolean expression is FALSE. The syntax is an IF statement with the BREAK keyword in. Place it anywhere between braces {}.

```
if (boolean expression) {
    break;
}
```


User-defined functions

Also you can use user-defined functions in Structured Text language for TeslaSCADA2. You can find example below:

```
function fun(a,b){  
int c;  
if (a>b){  
    c=a+b;  
}  
else{  
    c=b-a;  
}  
return c;  
}  
int d = fun(13,17);  
print(d);
```

In this example user function starts with key word **function**. Then name of the function. Then in parentheses arguments are listed. Inside braces {} statements of the function. User-defined function must be announced before main program. In this example program text of function **fun** is in the beginning. And only after statements of **fun** function, text of the main program.

Results of this script will be **4** in the log window.

Using Tags in Structured Text

Of course for our purposes we need to use Tags in our scripts written in Structured Text language. How to do that? You can include Tags in your project's scripts by using keyword **Tags**. Then type dot (.) and name of your Tag. For possibility to compile this code the name of the tag should contain only English letters without whitespaces and any signs.

Example:

```
int var = 10;  
Tags.Tag1 = var;
```

In this example value of the variable **var** will be assigned to Tag with name Tag1.

Other Example:

```
float f = Tags.Float1;
```

In this example value of the Tag with name Float1 will be assigned to variable **f**.

Using Object property fields in Structured Text

You can include Object property fields in your project's scripts by using keyword **Objects**. Then type dot (.), name of your Object, again type dot (.) and name of property field. For possibility to compile this code the name of the object and object property fields should contain only English letters without whitespaces and any signs.

Example:

```
int width = 100;  
Objects.Rectangle.width = var;
```

In this example value of the variable **width** will be assigned to Object with name **Rectangle** and field property name **width**. Name of the property fields you can find out in parentheses of object and property descriptions above.

Using Server parameter fields in Structured Text

You can include Server parameter fields in your project's scripts by using keyword **Servers**. Then type dot (.), name of your Server, again type dot (.) and name of parameter field. For possibility to compile this code the name of the server and server parameter fields should contain only English letters without whitespaces and any signs.

Example:

```
Servers.ModbusServer.ipaddress = "192.168.0.102";
```

In this example value "192.168.0.102" will be assigned to the server with name **ModbusServer** and field property name **ipaddress**. Name of the property fields you can find out in parentheses of server and parameter descriptions above. Also for parameters are written in descriptions you can use: *lostconnection*, *connect* and *connected*.

Using User parameter fields in Structured Text

You can include User parameter fields in your project's scripts by using keyword **Users**. Then type dot (.), name of your User, again type dot (.) and name of parameter field. For possibility to compile this code the name of the user and user parameter fields should contain only English letters without whitespaces and any signs.

Example:

```
Users.Operator.controlfunctions = true;
```

In this example value **true** will be assigned to the user with name **Operator** and field property name **controlfunctions**. Name of the property fields you can find out in parentheses of user and parameter descriptions above.

Embedded functions

In the Structured Text language for TeslaSCADA2 there are number of embedded functions:

print(Input) - print input in the log.

sqrt(Input) - arithmetic operation square root of the input value.

pow(Input1, Input2) - arithmetic operation power for input values. output = Input1^Input2.

log(Input1, Input2) - arithmetic operation logarithm of the input value (Output = Log_{Input2}Input).

ln(Input1) - arithmetic operation ln (natural logarithm) of the input value (Output = Ln(Input)).

bytestoshort(Input1, Input2) - used to pack 2 bytes in the short (Output = Input<<8+Input2).

shorttobyte(Input1, Input2)- used to unpack short value in 2 bytes (Output = Input[Input2]).

shortstoint(Input1, Input2) - used to pack 2 shorts in the int (Output = Input<<16+Input2).

inttoshort(Input1, Input2) - used to unpack int value in 2 shorts (Output = Input[Input2]).

readbit(Input1, Input2) - used to read bit of the input value (Output = Input[Input2]).

setbit(Input1, Input2)- used to set bit of the input value (Output = Input | 1<<Input2).

resetbit(Input1, Input2) - used to reset bit of the input value (Output = Input & ~(1<<Input2)).

min(Input1, Input2) - used to select minimum value of Input2 and Input1 (Output=Min(Input, Input2)).

max(Input1, Input2) - used to select maximum value of Input2 and Input1 (Output=Max(Input, Input2)).

abs(Input) - used to arithmetic operation absolute for input value (Output = |Input|).

sign(Input) - used to arithmetic operation sign for input value (Output = -Input).

int(Input) - used to arithmetic operation for getting integer part of the input value (Output = int(Input)).

toradians(Input) - used to convert degrees to radians.

todegrees(Input) - used to convert radians to degrees.

sin(Input) - used to calculate sin of Input value. (Output = sin(Input)).

cos(Input) - used to calculate cos of Input value. (Output = cos(Input)).

tan(Input)- used to calculate tag of Input value. (Output = tag(Input)).

asin(Input) - used to calculate arc sin of Input value. (Output = arc sin(Input)).

acos(Input) - used to calculate arc cos of Input value. (Output = arc cos(Input)).

atan(Input)- used to calculate arc tag of Input value. (Output = arc tag(Input)).

hextoint(Input) - converts hex value into integer.

inttohex(Input) - converts integer value into hex.

stringequals(Input1, Input) - compare two strings in Inputs and if there are equals it returns true.

stringtodouble(Input) - converts Input's string value into double value.

doubletostring(Input) -converts Input's double value into string value.

stringtoint(Input) - converts Input's string value into integer value.

inttostring(Input) - converts Input's integer value into string value.

substring(Input1, Input2, Input3) - used to cut begin and end of Input1's string value by the № of characters defined in Input2 and Input3.

base64decode(Input) - used to decode Base64 string to byte array. Input contains base64 encoded string. In Output will be decoded byte array.

base64encode(Input) - used to encode byte array to Base64 string. Input contains byte array. In Output will be encoded Base64 string.

datetime(Input) - used to get date and time components depending on Input value:

- 0 - get seconds.
- 1 - get minutes.
- 2 - get hour of the day considering AM/PM.
- 3 - get hour of the day.
- 4 - get day of the week (1-Sunday, 2-Monday...).
- 5 - get day of month.
- 6 - get month (0 - January, 1 - February...).
- 7 - get year.
- 8 - get minutes of the day (hour*60 + minutes).

reconnect(Input1,Input2) - used to reconnect to server with name from Input1 to IP address from Input2.

selrecipe(Input1, Input2) - used to choose recipe row. Input2 is an input that contains name of the recipe. Input1 is number of the row (starting from 1). Output = true if recipe row is chosen.

sendemail(Input1, Input2) - send email (if it setup in Project properties) with subject from Input1 and message from Input2.

createdbsqliteconnection(Input1) - used to create create connection to SQLite database with name in Input1.

Example: **createdbsqliteconnection**("filename");

createdbconnection(Input1, Input2, Input3) - used to create connection to database with name in Input1, with username in Input2 and password in Input3.

Example: **createdbconnection**("jdbc:mysql://192.168.0.76:3306/test", "username", "password");
in this example MySQL database is created. ("jdbc:mysql" in the beginning means that MySQL connection is created).

closedbconnection(Input1) - used to close database connection with name in Input1.

Example: **closedbconnection**("filename");

createdbtable(Input1, Input2, Input3) - used to create table in database with name of database in Input1, table name in Input2 and columns in Input3 (columns should be separated by commas, every table has auto incremented column "_id")

Example: **createdbtable**("databasename", "tablename", "title, parameter1, parameter2");

insertvaluesintodb(Input1, Input2, Input3) - used to insert row into database with name of database in Input1, table name in Input2 and values in Input3 (values should be separated by commas)

Example: **insertvaluesintodb**("databasename", "tablename", "Title, 10, 20");

readvaluefromdb(Input1, Input2, Input3, Input4) - used to read value from database with name of database in Input1, table name in Input2, name of the read column in Input3 and condition of read row in Input4 (if several rows fit to condition first row is read)

Example: **readvaluefromdb**("databasename", "tablename", "parameter1", "_id=1");

readvaluefromdbinpos(Input1, Input2, Input3, Input4, Input5) - used to read value from database with name of database in Input1, table name in Input2, name of the read column in Input3, condition of read row in Input4 and position of the row in Input5.

Example: **readvaluefromdbinpos**("databasename", "tablename", "parameter1", "title = Title", 1);

updatevalueindb(Input1, Input2, Input3, Input4, Input5) - used to update value in database with name of database in Input1, table name in Input2, name of the updated column in Input3, condition of the updated row in Input4 and updated value in Input5 (if several rows fit to condition all rows values are changed)

Example: **updatevalueindb**("databasename", "tablename", "parameter1", "title = Title", "10");

deleterowindb(Input1, Input2, Input3) - used to delete row(s) in database with name of database in Input1, table name in Input2 and condition that should fit the row(s) in Input3.

Example: **deleterowindb**("databasename", "tablename", "_id=1");

ifttttrigger(Input1, Input2, Input3, Input4, Input5) - used to send trigger event ifttt.com service.

Input1 contains key; Input2 contains event trigger name; Input3, Input4, Input5 contain value1, value2 and value3 for ifttt.com service.

Example: **ifttttrigger**("yourkey", "tag_trigger", "Tag is become true", Tags.Tag_2, "current value");

httppostcreate(Input1, Input2) - used to create HTTP post request. Input1 contains name of the request; Input2 contains url address.

Example: **httppostcreate**("namehttppost", "https://hooks.zapier.com/hooks/catch/zapkey/otherzap/");

httppostaddvalue(Input1, Input2, Input3) - used to add value into HTTP post request. Input1 contains name of the request; Input2 contains name of the value; Input3 contains value.

Example: **httppostaddvalue**("namehttppost", "valuenam", "value");

httppostexecute(Input1) - used to execute HTTP post request. Input1 contains name of the request. Function returns HTTP post response.

Example: **httppostexecute**("namehttppost");

httppostgetvalue(Input1, Input2) - used to get value from the HTTP post response. Input1 contains response string; Input2 contains name of response value. Function returns value from the HTTP post response.

Example: **httppostgetvalue**("{valuenam: value}", "valuenam");

currentdatetime(Input1) - used to get current date and time in string format. Input1 contains format of the date and time. Function returns formatted current date and time.

Example: **currentdatetime**("YYYY-MM-dd HH:mm:ss");

Use Telegram Bot

If you want to get events notification from your project in TeslaSCADA2 OPC UA server you can use Telegram messenger for this purpose. To do this you should create Telegram Bot:

1. You should have Telegram messenger installed on your device and have an account.
2. Open in browser <https://telegram.me/botfather>
3. Click button «Send message» or «Open in Telegram Web» (you should have login in web telegram client).
4. Open your Telegram client and choose BotFather.
5. Click button Start or type /start.
6. Enter /newbot.
7. Enter your bot's name. The name should be unique. This name you should enter in **Bot's name** field of project properties.
8. Then you should choose username for your bot.
9. After entering username you'll get the telegram bot's token. Enter it in **Bot's token** field of project properties.

Now you can use telegram bot in getting notification messages from TeslaSCADA2 OPC UA server. To do this you should find your created bot in your telegram messenger client and click button Start or enter **/start**. To stop getting notification messages enter **/stop**. Also you can get some information from your project:

1. Enter **/tags** to get current values of tags. You'll get information only from currently monitored tags (tags that enable history, events and tags of objects that displayed on currently opened screen).
2. Enter name of the tag used in your project. You'll get information about value of this tag and if tags supports history you'll get trend for last hour period. You can choose other period by clicking proper button.

Warning don't use underline in the name of the tags. Telegram have problems with working with this kind of names.