IoT Edge computing extension for Live Objects

First Step Tutorial

IoT use cases require cloud platforms to collect, store and manage data & devices, and to give a service view to users. But more and more industry, building, energy need treatments close to the devices, to provide short feedback loops, filtering and/or local data containment: what is called Edge treatment.

This tutorial describes how to interact with temperature Modbus end-devices (sensors), with a first level of treatments at the Edge. For this purpose Technilog Dev I/O, software was used. For data management and device management it was decided to use the Cloud platform - Orange IOT platform Live Objects.

Table of Content

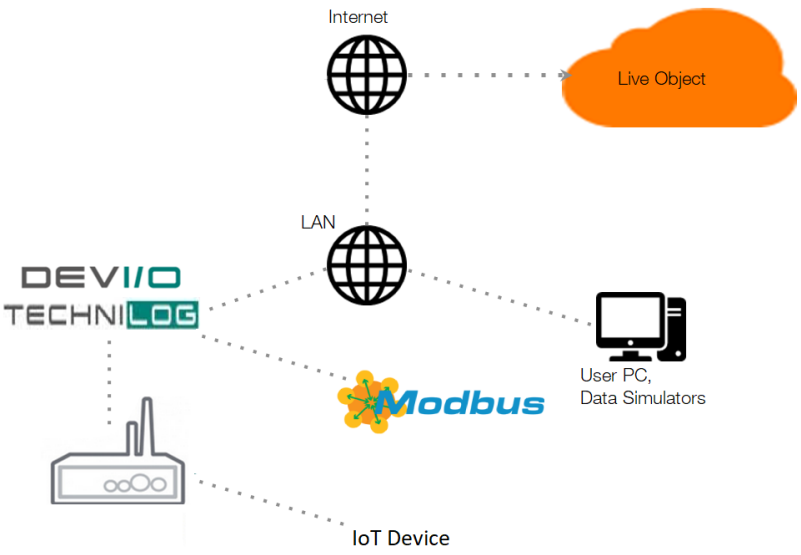
1 Technilog and Orange solutions to address Edge+Cloud needs

Technilog propose a .Net based IoT software middleware solution called Dev I/O that can be based on the Edge with the purpose of management and data aggregation.

- **Dev I/O** is a multi-protocol middleware software which harnesses the different manufacturer communication channels. It allows for unifying data exchange between cloud services (e. g. Live Objects) and IoT devices. Dev I/O uses open, secure, standardized protocols. It is an OPC UA server, using Web services and databases. Dev I/O consists of three separate sub-applications.
    - **Dev I/O Studio** - tool dedicated to manual configuration
    - **Dev I/O server** - handles exchanges between applications (e.g. via OPC) and equipment
    - **Dev I/O XML Configuration module** - tool dedicated to automatic equipment configuration

Dev I/O is integrated with Live Objects solution.

- **Live Objects (LO)** is Orange IoT management cloud platform for data and devices. It provides(through web portal and APIs), functionalities to collect, store, transform and display data. It aggregates mentioned data through API to business applications. Another features enabled through its web portal are APIs provisioning and management of end devices.



Rolled-out architecture, combining Technilog and Orange solutions

2 Resources used

For this tutorial we have installed Dev I/O on both physical and virtual environment using Windows. We have used the Dev I/O with a demonstration license, therefore some components may differ under the production license.

This solution is Windows based thus, the default requirement is:

- Microsoft operating system(32 and 64 bit editions):
    - Windows XP
    - Windows 7,
    - Windows Server 2003,
    - Windows Server 2008,
    - Windows Server 2012.

More globally, installation of Dev I/O additionally requires:

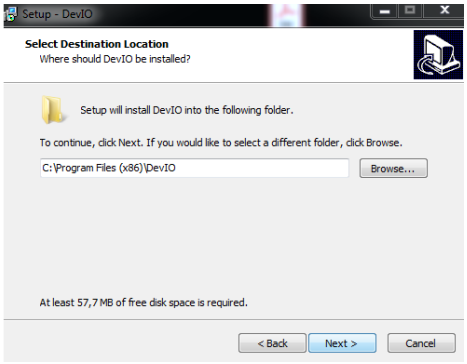
- SQL Server Compact 4.0
- Microsoft .Net Framework 4.5.1
- For Windows 7 and earlier distributions – windows installer 4.5
- Administrator account

In our tutorial we have used additional tools:

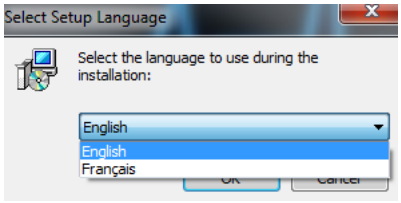
- UA Expert (optional – used for Dev I/O data manipulation)
- ModbusPal Simulator

3 Installation

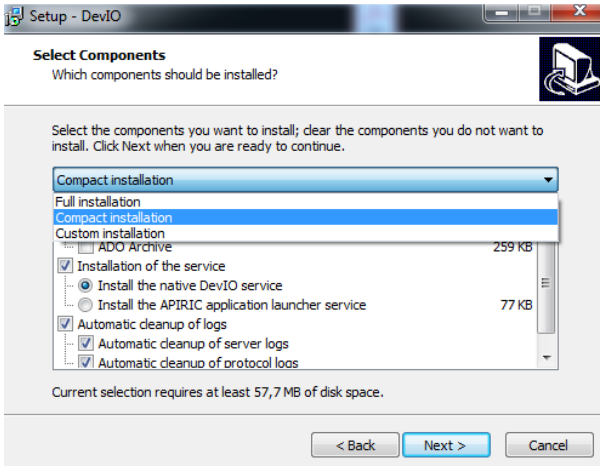
To follow our tutorial be sure that your machine fulfills the requirements defined in section 2. To install all components (Dev I/O Server, Dev I/O Studio, Dev I/O Configuration XML) run the **setup_all.exe** (as an **administrator**).



After running, the language selection window will pop up. User can choose two languages – English or French. However, the English language is OS dependent, therefore to run English distribution the Dev I/O must be installed on English distribution of Windows. If you are running Windows with other language than English the French version of Dev I/O will be installed (default). It was the case for us, therefore French screens appear in the rest of this tutorial.



For the next few windows the typical installation will continue, until the **Select Components** stage. Unless you're sure that the available components are needed for you project choose the **Compact Installation.**



Now the rest of components will install in normal Windows manner.

In our case the project included the Modbus protocol support. This package is installed during the normal installation process. We have as well installed several additional tools to simulate an equipment (ModbusPal simulator described in section 5.1) and to follow current values through another stream than MQTT (OPC UA Expert). These tools are not mandatory in a production customer context.

The last stage of installation is the license activation where user can either enter the activation key and click **apply** or click **quit** to run the demonstration version.

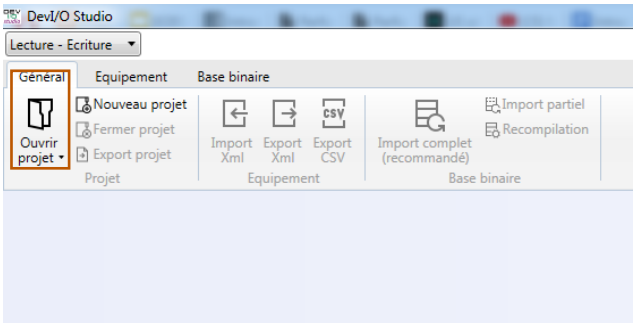
4 Configuration

In this chapter we will discuss how to configure projects in Dev I/O with special consideration of predefined project, as we have mainly worked with them. Technilog can send to client a project, which is tailored support the customer needs e. g. certain protocol. The received project is initially configured, however some properties may need to be changed. More details about our predefined configurations can be found in section 4.2.

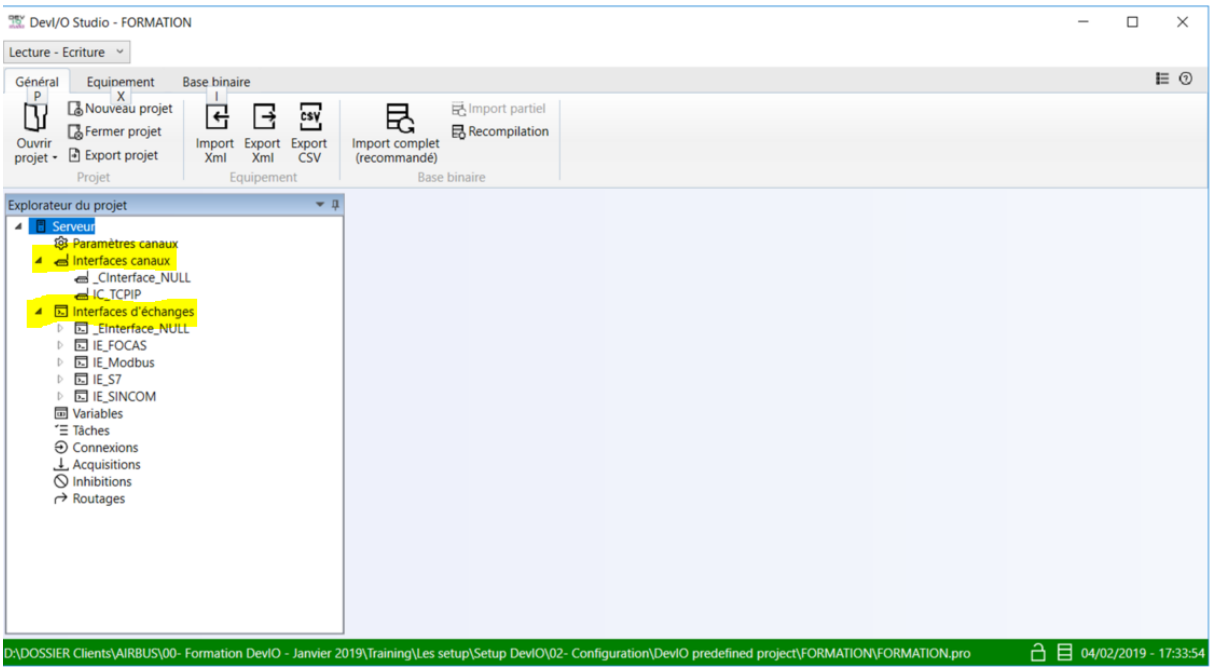
4.1 Open predefined project

Dev I/O allows user to run a predefined project. However, the project has to be in the right path: **Program Files (x86)/DevIO/Data/Bases.** By running the project from a different source (different catalog) possible errors may be encountered.

To open the predefined project, run the Dev I/O. Start program as an administrator **Dev I/O Studio Tray Icon (launch icon)**. Now right click the Dev I/O icon in the right bottom corner of your screen and click **Start Dev I/O Studio.**



Click the **Ouvrir projet** to open the project. And choose the path to the .pro file.



The Dev I/O studio tool is used for manual configuration. However, this requires off-line mode, the Dev I/O server should be turned off.

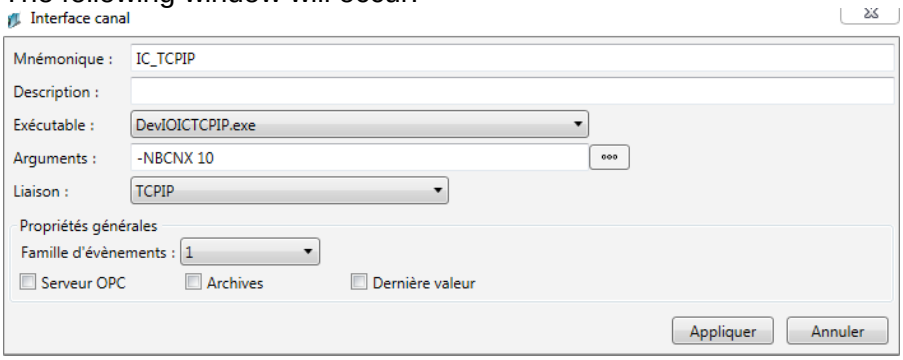
4.2 Our predefined project configuration

In the opened Dev I/O studio, take notice to the server tree on the left and especially to the two components:

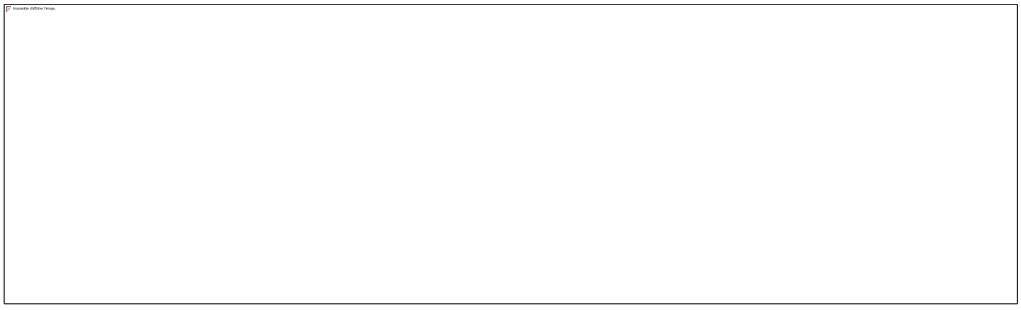
- **Channel Interface** - (“Interfaces canaux”) to configure physical or logical communication channel (COM1 serial port, TCP I/P socket, port associated with the modem.
- **Exchange Interface** – (“Interfaces d’echanges”) to configure end-device. Exchange interfaces are based on model which defines the protocols behavior. **To add a new protocol it is required to contact Technilog support to receive the specific model.**

In our case the default Dev I/O project was extended with two :

- **Channel Interface**: TCP1_Canal utilizing the TCP/IP communication with external devices.
  The following window will occur.



- **Exchange Interface**: Modbus driver enabling the communication with Modbus devices.

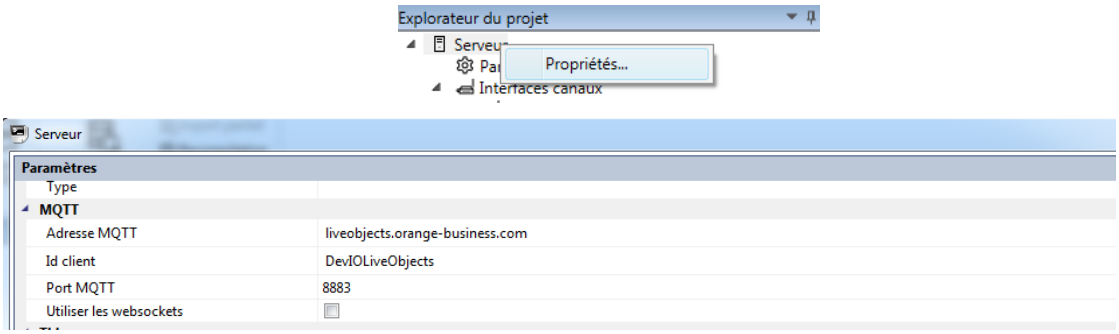


To investigate above components right the element as on image below.

It is advised to don’t change **Channel Interface** component properties, because it may cause errors.

4.3 MQTT Interfaces

Dev I/O allows to configure the MQTT interface. The MQTT parameters can be set under server properties (right click server → properties).



4.4 XML Configuration

Part of Dev I/O configuration needs to be done in a XML file. For example the MQTT interface configuration should be deployed in the same manner as on the image below.

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Orange mode connector - api-key - -->
<!-- certificat Orange à placer dans le répertoire du .pro -->
<DevIOMqttServer server="DevIOMQTTServer" user ="connector" password="2ab2513fab9849248453c687b0bc83f4"
 certStore="%PATH_BASE%\.." caCertificate="OrangeCA" clientCertificate="" clientPrivateKey="" cleanSession="true"
 minInterval="1" maxInterval="60" ack="false" keepaliveInterval="10" logFlush="false">

  <Model name="registerEqt" publishTopic="DEFINE TOPIC" qos="0" PoD="10">
    <PublishMessage>{PAYLOAD}</PublishMessage>
    <Date format = "ISO" msSeparator =","/>
    <Ignore equipmentName="_MODELE"></Ignore>
  </Model>

  <Model name="unregisterEqt" publishTopic="DEFINE TOPIC" qos="0" PoD="10">
    <PublishMessage>{PAYLOAD}</PublishMessage>
    <Date format = "ISO" msSeparator =","/>
    <Ignore equipmentName="_MODELE"></Ignore>
  </Model>

  <Model name="data" publishTopic="DEFINE TOPIC" qos="0" PoD="10">

    <!-- Valeurs possibles : $Year, $Month, $Day, $Hours, $Minutes, $Seconds, $Millisecs, $Timestamp, $Date, $Valu
                             $State, $Type, $IsHisto, $EquipmentName, $BlocName, $DataName, $ServerName, $Now, $Not
    <!-- $Date : Représente le format de la date dans la balise Date -->
    <PublishMessage>{PAYLOAD}</PublishMessage>

    <Date format = "ISO" msSeparator =","/>
    <Ignore equipmentName="_MODELE"></Ignore>
  </Model>

  <SubscribeTopics>
    <Topic type="Commands" name="DEFINE TOPIC" qos="1" >
        <Script>
```

Where some parameters could be modified. Those parameters are related to the specific MQTT broker and the MQTT user properties.

- Username
- Password
- Certificate path
- Publish Topic
- Subscription Topic

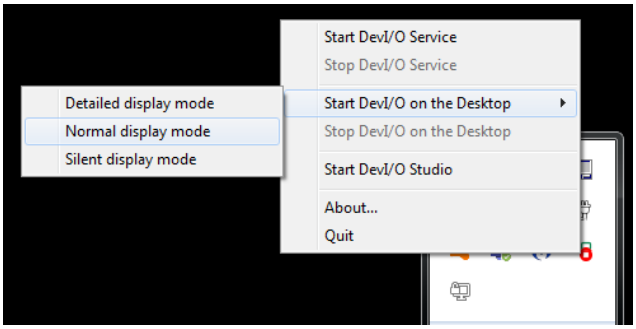
In the XML file the simple logic can be applied to the performed actions. For example, definitions of **publish messages**, or what **variables/data** from the end devices are about to be **sent** to the external cloud.

In Appendix A the whole XML file enabling the bidirectional MQTT communication with the Live Object. It contains definition of former mentioned parameters and a basic script logic defining the behavior of Dev I/O. Which in this case is reaction for receiving Live Object instructions.

The XML file can be placed in the following path:
C:\ProgramFiles(x86)\DevIO\Data\Bases\<Project_Name>

4.5 Run DEV I/O Server

In order to run Dev I/O server, one needs to run the "**Dev IO Tray Icon**" as an administrator. Next be sure that Dev I/O Studio is running in the read only mode.



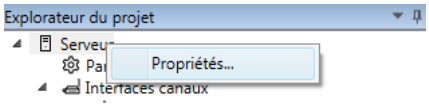
Right click the Dev I/O icon in the low right corner → Start Dev I/O on the Desktop → Normal display mode.

In the scope of our usage, the Dev I/O server was running in console mode (with a demonstration license – Normal display mode); in a production context, the Dev I/O server may be running in different mode type.

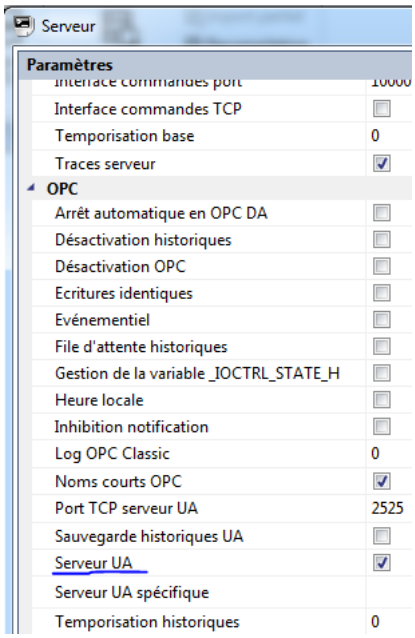
4.6 Enabling of OPC UA Server

In Dev I/O Studio there is **no possibility** to write data on the connected end devices. However, it can be done using an external applications. Such as a OPC UA Expert which connects to the OPC server in Dev I/O. In order to do this, the OPC UA server in Dev I/O must be configured properly.

Right click server → properties.



Find OPC element and enable the **Serveur UA** property.



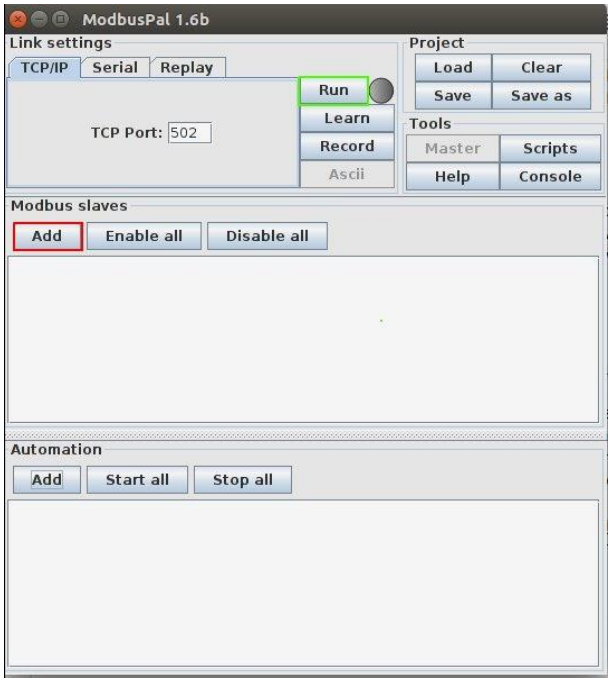
If these actions are performed, an additional window with a link to the DEV I/O OPC UA server will appear, after Dev I/O is started in normal mode.

5 Exchanging data from device to Live Objects

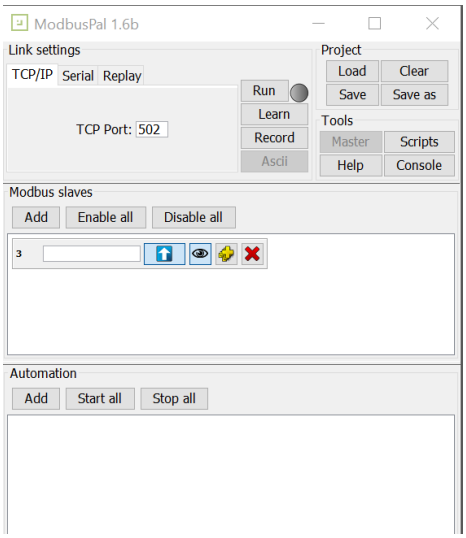
This chapter presents how to establish connections with IoT devices which uses Modbus protocol. Just to inform, usage of Modbus protocol **is not supported in basic** Dev I/O configuration. We were able to use this protocol due to the fact that it was predefined in our project. To add Modbus protocol or any other extension **contact Technilog support** to receive additional models.

5.1 ModbusPal Simulator

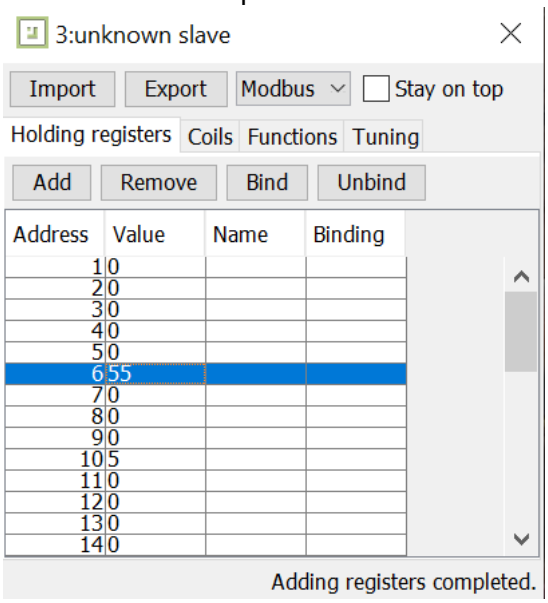
For ease of implementation and test, we use a Modbus Java based device simulator: ModbusPal Simulator. To start with this simulator, be sure to have Java Runtime Environment installed. Than download the ModbusPal and run the **ModbusPal.**jar file. If everything is installed properly following GUI should pop up.



To add a new device, click the **Add** button - highlighted in red. Choose slave id - the same as the unit.id chose in the Modbus driver. After adding a new slave new components should be visible in the Modbus slave section.



Then click the **eye icon** in the new slave and click add. Set the range that contains memory.address parameters set in the components of the channel in the driver.



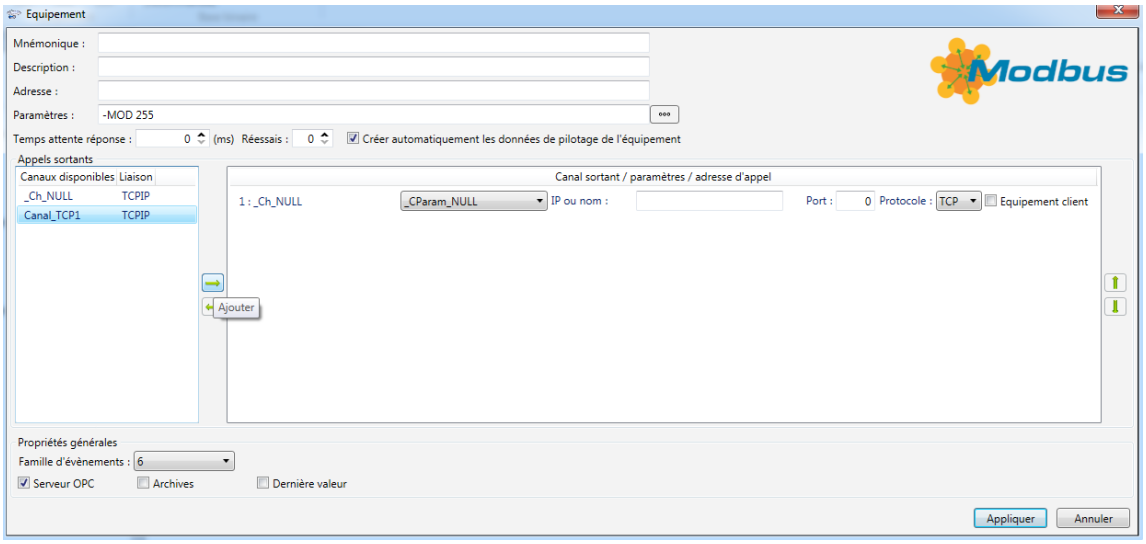
Be sure that you have opened port on which Modbus is working - default - 502. Finally, click the run button available in the main panel and change the value of the desired address as on figure above. In order to check if the connection works properly, in asset click the Data section and observer whether the value of channel has changed.
More information: https://sourceforge.net/p/modbuspal/wiki/Home/

5.2 New end-devices in DEV I/O

There is a numerous ways to configure a new device in Dev I/O. The addition of new device is strictly connected with a model, on which it is basing. For this reason, adding new devices based on other protocols may look different. In this tutorial we will focus on adding Modbus devices.

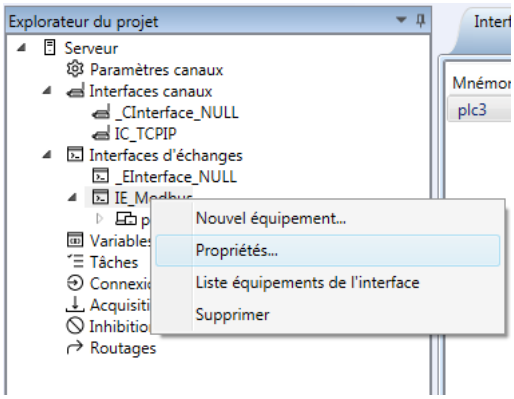
5.2.1 Add new Modbus entity in DEV I/O

To add a new Modbus entity to Exchange Interface inside the Dev I/O studio right click the **IE_Modbus** and choose **Nouvel equipment.** The following window should pop up:

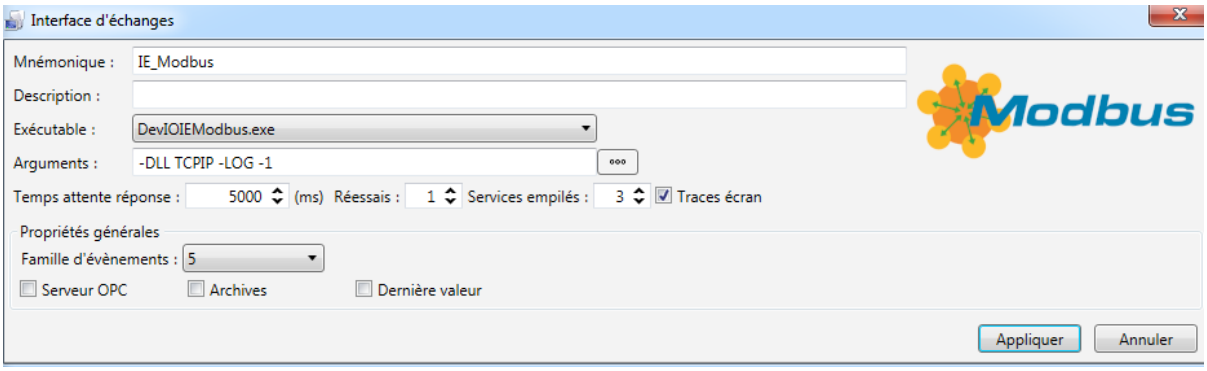


In this panel the name and the address of the new device can be added. You need to change the canal of the interface by clicking the highlighted **Canal_TCP1** and the right arrow.

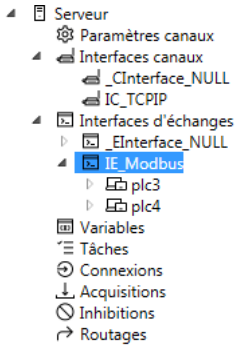
The properties of new added devices can be checked and investigated by checking properties.



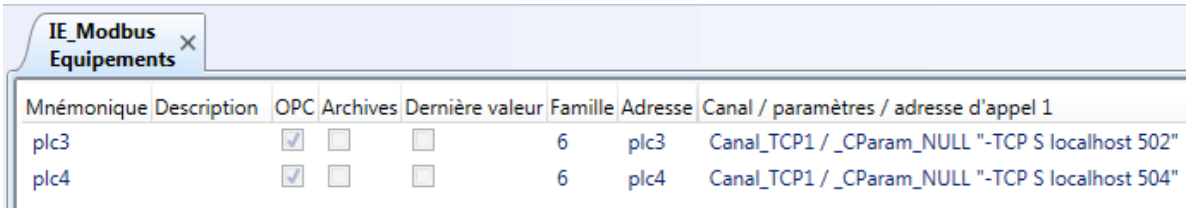
To inspect it, click right mouse button on it a choose **Properties**.



In this panel user can model the Modbus device by changing its parameters. In example in the Arguments section the slave number can be define with **–DLL TCPIP –LOG -<Your slave number>.**



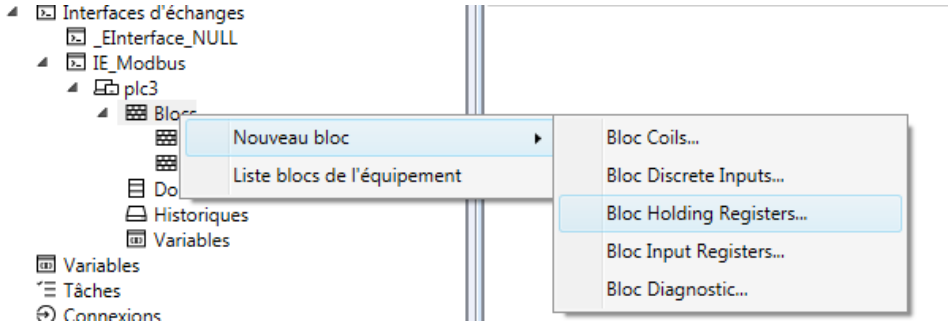
The interface can be as well inspected by double clicking on it.



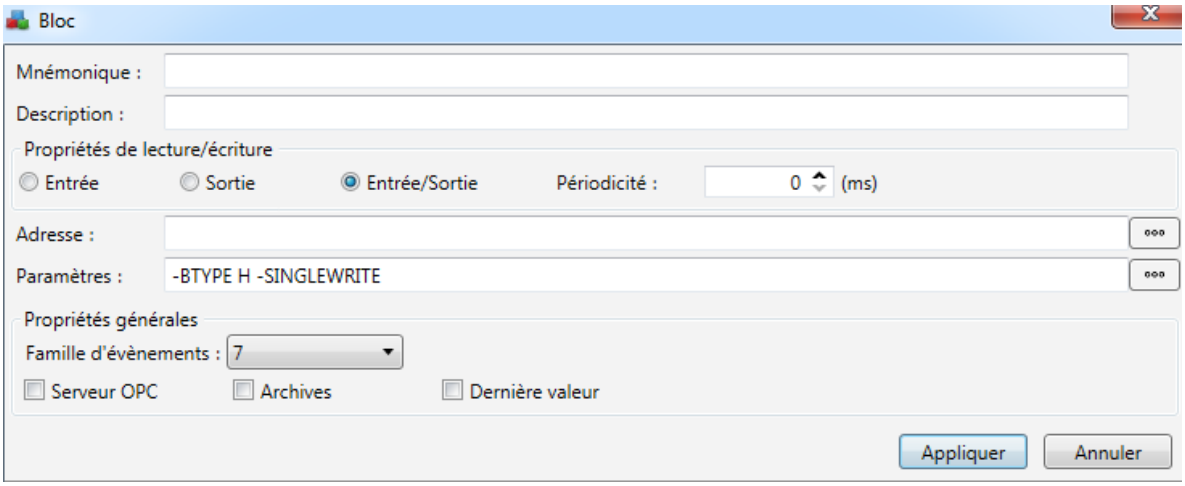
In this panel the **IP Address** as well as **source port** can be changed.

5.2.2 Configure exchange data channel

To add the new Modbus data be sure that you have configured end device in priori defined (5.2.1) exchange interface. For example to add the new holding register, you need to right click the blocs property and choose the **Bloc Holding Registers.**

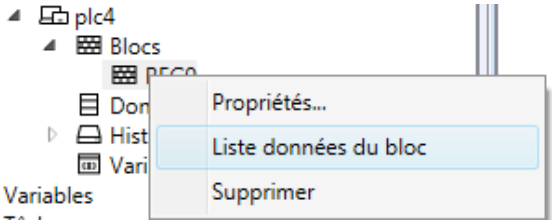


After clicking it, the detailed of the bloc can be configured.

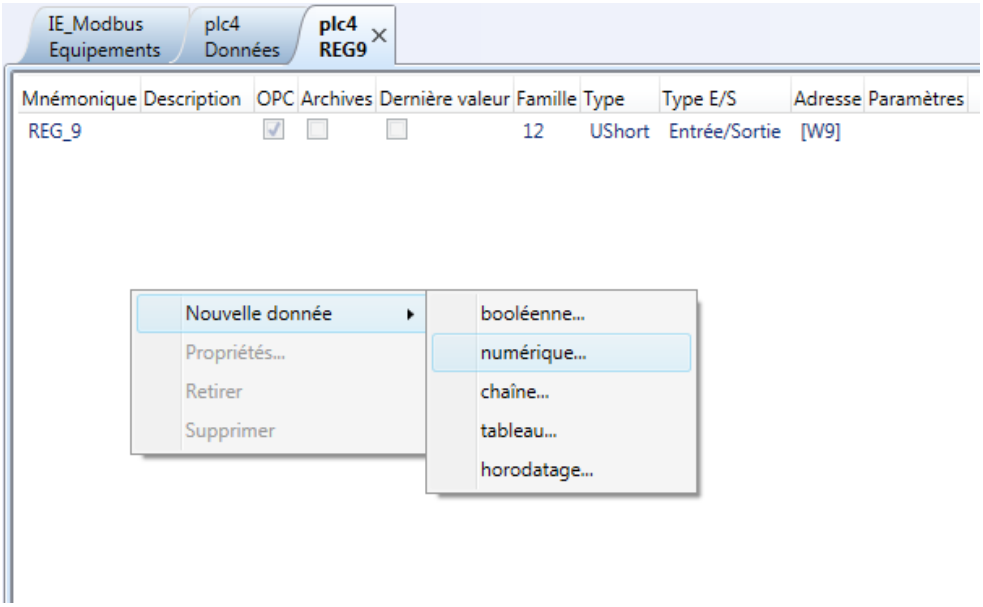


In this panel the parameters of data Bloc can be defined. The most important one is the **Adresse** parameter, where the range of future used address can be defined e.g. to define twenty holding registers from 1 to 20 type **H0/20.**

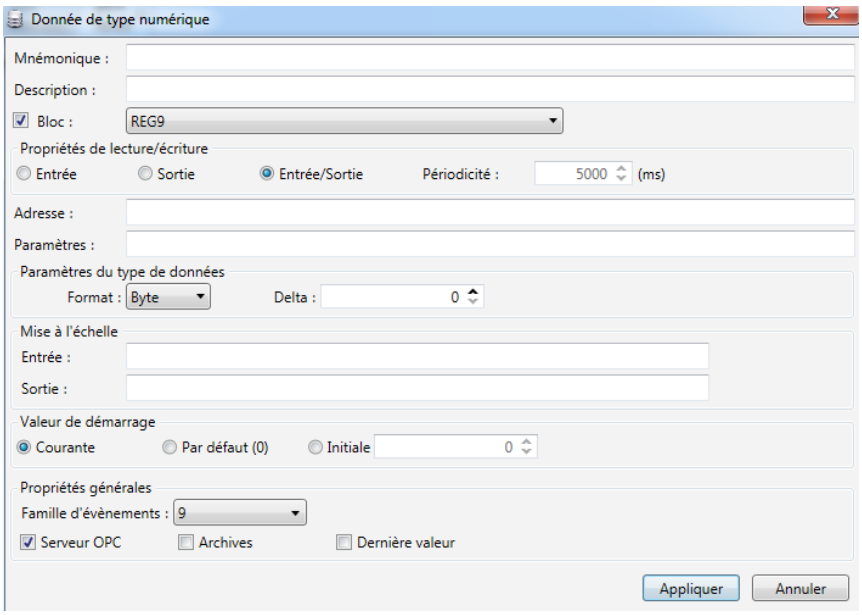
In order to add a new data parameter, list the bloc properties by right clicking the Bloc number



Inside of the new opened panel, right click the empty space and choose the **Nouvelle donnee** parameter.



The following panel will pop up:



Where the new value parameters can be defined. The most important one is the **Adresse ,** here the Modbus channel number can be defined **[W<Number of a channel>]** which is in included in previously defined range**.**

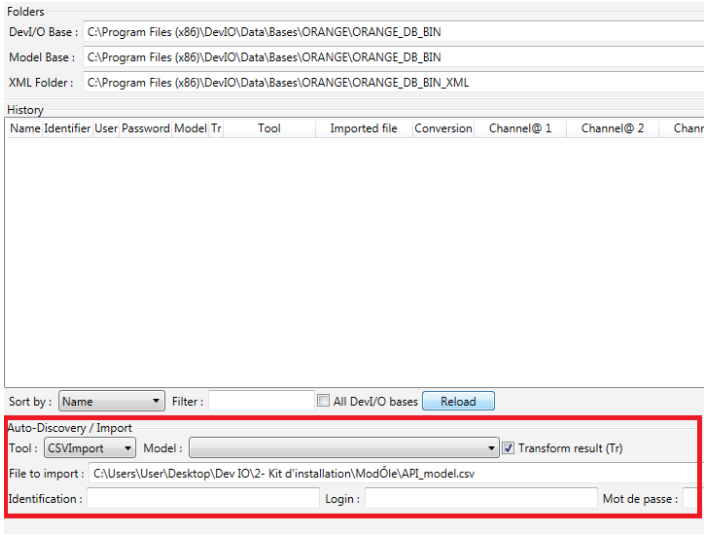
5.2.3 Modbus End device Auto Configuration

XML Configuration Tool is an alternative method basing on importing configuration files. The formats included in Dev I/O are those produced by tools supplied by the manufacturer. With this interface, configuration can be performed during operation.

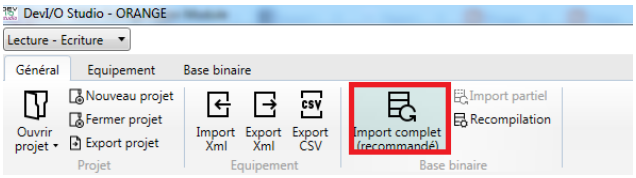
Before using the **XML Configuration Tool** be sure to run the Dev I/O server, so the changes will be applied. The procedure of adding new device may vary for different protocols. We mainly focused on the Modbus protocol. Therefore we will show how the configuration can be performed using it. In order to add new Modbus end device the user needs to have the Modbus **.csv** model file. In this file the data blocs and bloc addresses can be defined as well as the format of the data and the repeat period time.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	BlocName	BlocAddre	BlocAcces	Name	Address	Format	Element	Size	Access	Paramete	Descriptic	Historical	HistoPara	HistoPeric	Periodicit	BlocParameters	
2	TOR1_3	C0/3	RW	TOR1	[0/0]	bool			RW			0			2000		
3	TOR1_3	C0/3	RW	TOR2	[0/1]	bool			RW			0			2000		
4	TOR1_3	C0/3	RW	TOR3	[0/2]	bool			RW			0			2000		
5	REG1_3	H0/6	RW	REG1	[W0]	ushort			RW			0			2000		
6	REG1_3	H0/6	RW	REG2	[W1]	ushort			RW			0			2000		
7	REG1_3	H0/6	RW	REG3	[W2]	ushort			RW			0			2000		

For adding new device we advise to use the **Dev I/O – XML Configuration Tool**. On the image below in the red section, in the **Tool** section the CSVImport must be chosen. For the **Model** variable choose Modbus parameter. Than choose the csv file, in the file to import. Now apply the IP number of your Modbus device and the port number on which it is working. Finally set the name of the new equipment in the low left corner and click the import button in the low right corner.



Now in Dev I/O Studio to observe the changes user needs to click the **import complete** button. If import was unsuccessful, check if the Dev I/O server is running.



If the device is added properly you should be able to observe the new interface.

5.2.4 Manage end-device

The device data can be managed either directly form the Dev I/O or with help of the OPC UA (described in 5.4.2).

5.3 Edge Feature

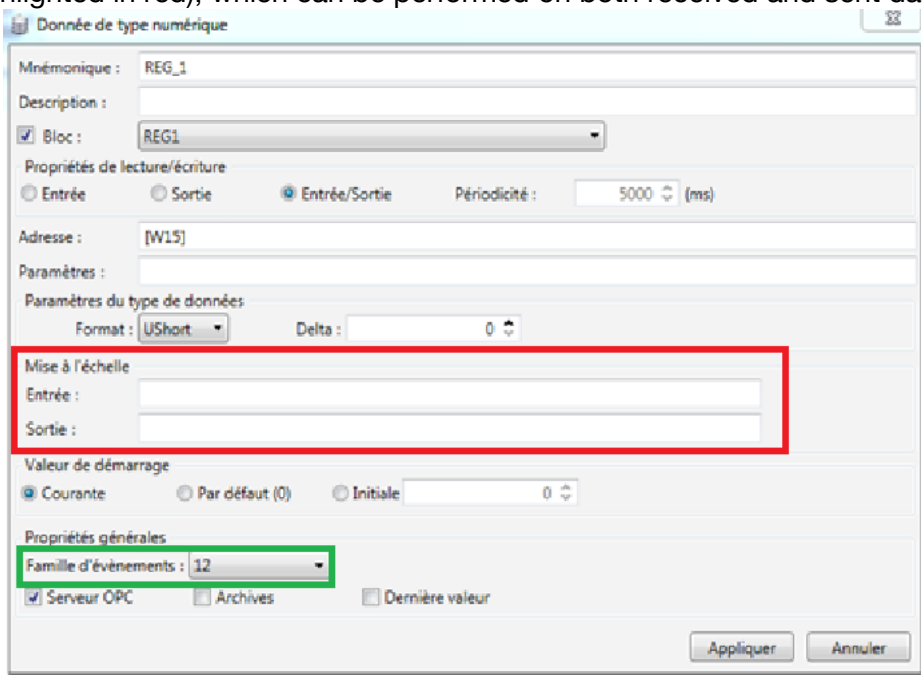
In this chapter the edge feature of Dev I/O are presented.

5.3.1 Data Aggregation

In this category we can distinguish:

- **Mathematical Operations**

While receiving or sending data, in the entity we can define the simple equations (in the section highlighted in red), which can be performed on both received and sent data.



- **Data Families**

Dev IO entities can be assigned to an event family (green section above). The event family can have a value from 0 to 255.

- 0: associated with "never", the value associated with the variable is never transmitted to the application (OPC UA client, etc.),

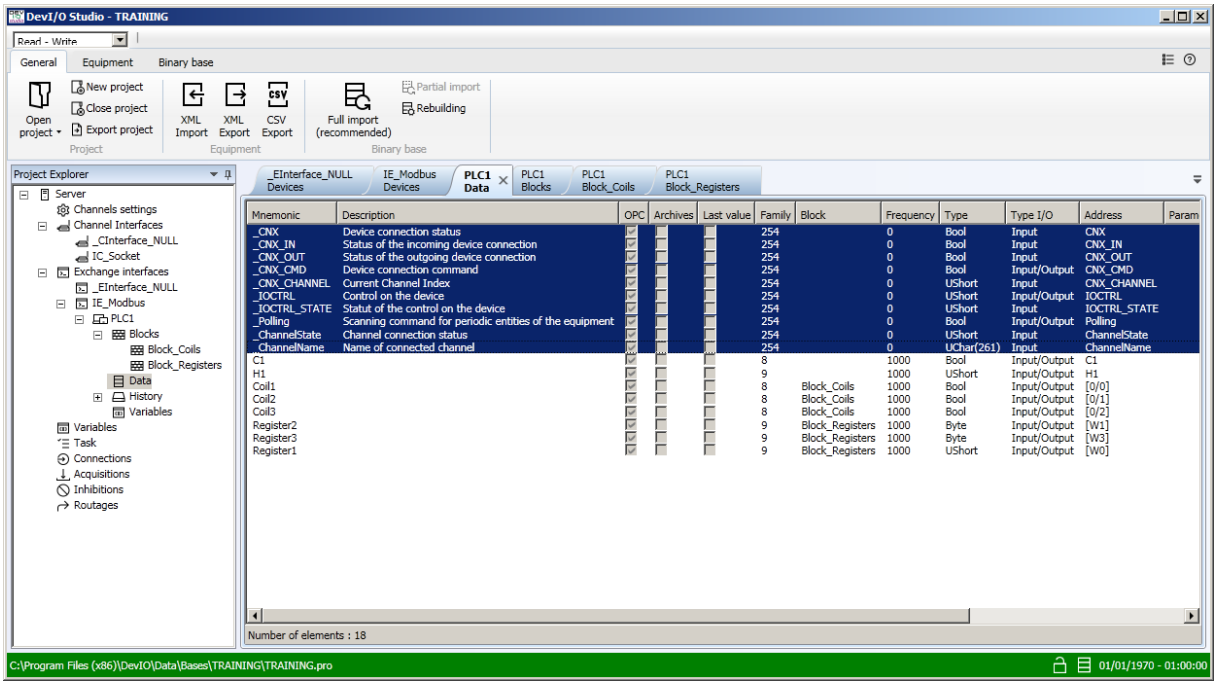
- 255: associated with "always", the value associated with the variable is transmitted to the application concerned (even if the value has not changed),
- 1 to 254 (253 is associated with “alarms”) : the value associated with a variable is transmitted to the application only on the change of values or states.

5.3.2 Storage

To store Dev I/O data an external DBMS (Database Management System) needs to be installed.

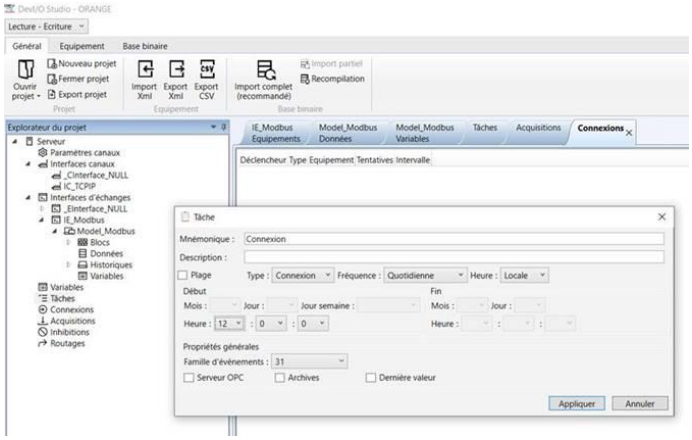
5.3.3 Variables

Each device has a predefined internal variable that allows to perform the certain actions on the device e. g. turning on and turning off. Variables are described on the figure below.



5.3.4 Data scheduling

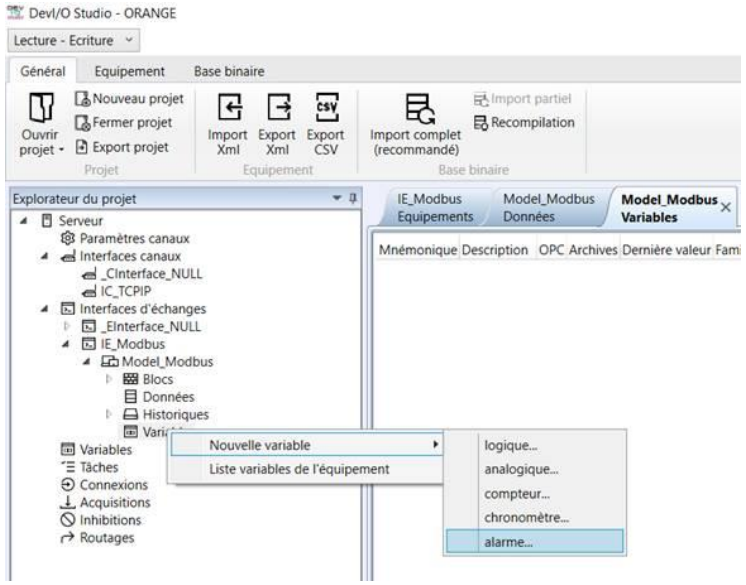
Dev I/O allows to schedule the connection to one or a set of equipment’s or to schedule data change. It can be done in the **Taches** section of Dev I/O studio.



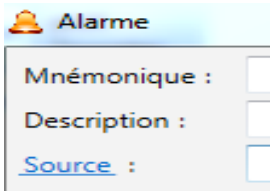
But, in a general way it is advised to use the Dev I/O internal variables (command variables: _CNX_CMD, _IOCTRL, _Polling and status variables: _CNX, _IOCTRL_State) and implement these functions at the applicative level.

5.3.5 Alarms

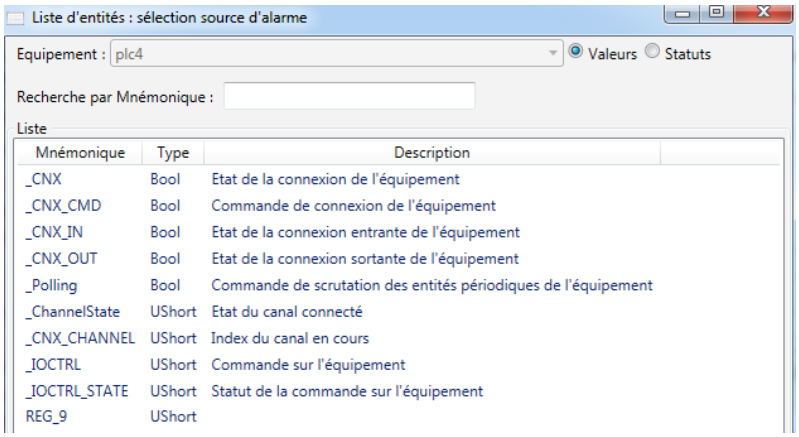
Dev I/O gives ability to create alarms in case the certain value will be obtained or any abnormal will occur. Dev I/O doesn't manage notifications such as : email, SMS. To set an alarm, follow the image below.



In the alarm panel it is possible to define the desired parameters.



To choose, which value should be observed click the **source** button in blue:



5.4 Management Features

In this chapter the management feature of Dev I/O are presented

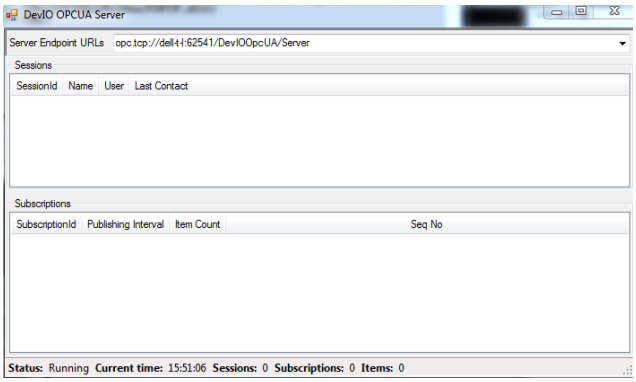
5.4.1 Remote access – RDP

For the remote access we have used RDP due to the fact that the that Dev I/O doesn't offer remote configuration capabilities. When it comes to devices, they can be managed using the OPC UA client or MQTT.

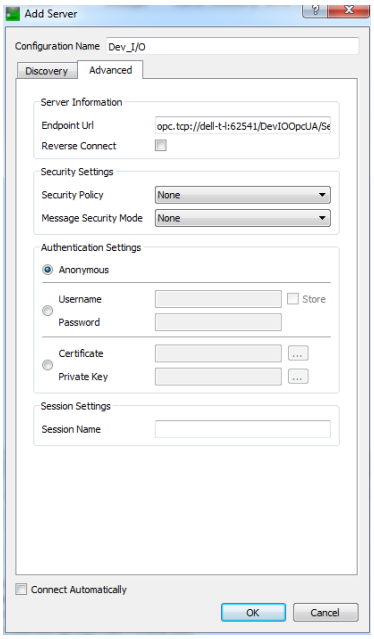
5.4.2 OPC UA – Modify values

When Dev I/O is working and it is connected with devices the value of devices can be manipulated. First enable the OPC UA server properties in according to 4.5.

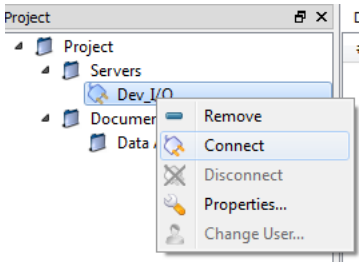
Than set the READ ONLY option in Dev I/O studio and start the Dev I/O server. If the OPCUA Server is working properly the following window should be seen. Now copy the **Server Endpoint URL.**



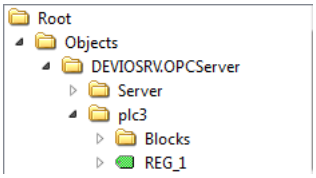
Open the UA Expert https://www.unified-automation.com/downloads/opc-ua-clients.html . In the panel in the left top corner right click the Server section and click Add. Now in window that have just opened click the advanced button and paste the previously copied URL in the **Endpoint URL** section.



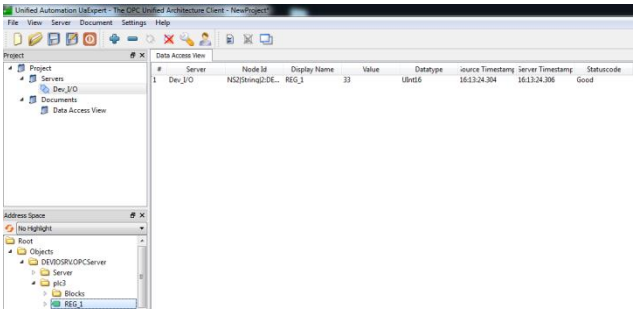
After adding new connect to it by right clicking the name of a server that you want to connect and click connect.



Now in the Addresses Space on the left side of the screen you should see the parameters of your configured Dev I/O interfaces,



In order to modify them drag and drop the components to the middle of the application and click the value parameter of the channel. The desired value can be simply typed in.



5.5 Integration with Live Object (LO)

The bidirectional integration can be achieved with help of the MQTT forwarder. In can be configured in the:

C:\ProgramFiles(x86)\DevIO\Data\Bases\<Project_Name>\DevIOMqttServer.xml

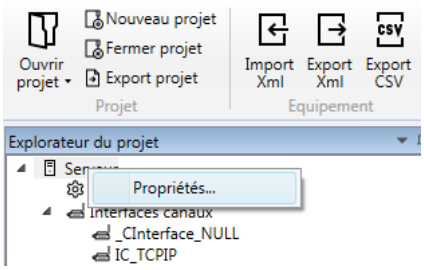
In this file described in Appendix A, the integration was prepared by the Technilog according to the LO MQTT interface. The communication with Live Object is achieved thanks to appropriate configuration of the data structure which is send and receive by usage of LO connector mode.

From the user perspective, the only parameter that we have to changed is the API Key of the Live Object account. API key should have External connector or Customized profile.

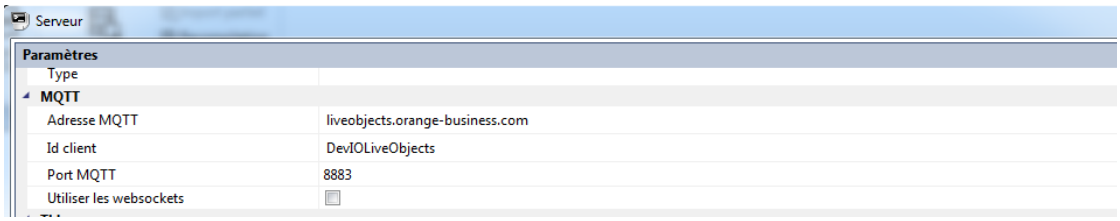
```
<?xml version="1.0" encoding="utf-8"?>

<!-- Orange mode connector - api-key - -->
<!-- certificat Orange à placer dans le répertoire du .pro -->
<DevIOMqttServer server="DevIOMQTTServer" user ="connector" password="2ab2513fab9849248453c687b0bc83f4"
```

By changing the **password** parameter Live Object account can be connected to the Dev I/O instance. To complete the Live Object integration, you need to open the server properties as follows.

Then find the MQTT properties right click the Server and open properties.



Then put values as on the image below.



The bi-directional integration is complete and it is now possible to send data from devices connected with Dev I/O to LO.

6 Appendix A

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Orange mode connector - api-key - -->
<!-- certificat Orange à placer dans le répertoire du .pro -->
<DevIOMqttServer server="DevIOMQTTServer" user ="connector"
password="2ab2513fab9849248453c687b0bc83f4" certStore="%PATH_BASE%\.."
caCertificate="OrangeCA" clientCertificate="" clientPrivateKey=""
cleanSession="true" minInterval="1" maxInterval="60" ack="false"
keepaliveInterval="10" logFlush="false">

  <Model name="registerEqt"
publishTopic="connector/v1/nodes/$EquipmentName/status" qos="0" PoD="10">

<PublishMessage>{"status":"ONLINE","capabilities":{"command":{"available":t
rue}}}</PublishMessage>
    <Date format = "ISO" msSeparator ="."/>
      <Ignore equipmentName="_MODELE"></Ignore>
  </Model>

  <Model name="unregisterEqt"
publishTopic="connector/v1/nodes/$EquipmentName/status" qos="0" PoD="10">

<PublishMessage>{"status":"OFFLINE","capabilities":{"command":{"available":
true}}}</PublishMessage>
    <Date format = "ISO" msSeparator ="."/>
      <Ignore equipmentName="_MODELE"></Ignore>
  </Model>

  <Model name="data" publishTopic="connector/v1/nodes/$EquipmentName/data"
qos="0" PoD="10">
    <!-- Valeurs possibles : $Year, $Month, $Day, $Hours, $Minutes,
$Seconds, $Millisecs, $Timestamp, $Date, $Value, $ObjectType,
                          $State, $Type, $IsHisto, $EquipmentName,
$BlocName, $DataName, $ServerName, $Now, $NowISO -->
    <!-- $Date : Représente le format de la date dans la balise Date -->

<PublishMessage>{"timestamp":"$Date","value":{"$DataName":"$Value"}}</Publi
shMessage>
    <Date format = "ISO" msSeparator ="."/>
      <Ignore equipmentName="_MODELE"></Ignore>
  </Model>

  <SubscribeTopics>
      <Topic type="Commands" name="connector/v1/requests/command" qos="1" >
            <Script>
                  function tprint (tbl, indent)
                    if not indent then indent = 0 end
                    for k, v in pairs(tbl) do
                       formatting = string.rep("  ", indent) .. k .. ": "
                       if type(v) == "table" then
                         print(formatting)
                         tprint(v, indent+1)
                       elseif type(v) == 'boolean' then
                         print(formatting .. tostring(v))
                       else
```

```
                    print(formatting .. v)
                end
            end
        end

        tprint(incoming_message,2)

        devio = DevIO.new()
        local msgId = incoming_message.id;
        local equipmentName = incoming_message.nodeId;
        local request = incoming_message.value.req;
        local ackMode = incoming_message.ackMode;
        local dataArray = {}
        local count = 0
        for k, v in pairs(incoming_message.value.arg) do
            local element = {dataName=k, dataValue=v}
            table.insert(dataArray, element)
            count = count + 1
        end
        devio:writeData(equipmentName, dataArray, count > 1)

        if ackMode == "APPLICATIVE" then
            out_msg = {id = msgId, nodeId = equipmentName}
            -- tprint(out_msg,2)
            out_topic = "connector/v1/responses/command"
            return out_topic, out_msg
        end
    </Script>
  </Topic>
</SubscribeTopics>

<Ignore name = "_CNX" />
<Ignore name = "_CNX_IN" />
<Ignore name = "_CNX_OUT" />
<Ignore name = "_CNX_CMD" />
<Ignore name = "_CNX_CHANNEL" />
<Ignore name = "_IOCTRL" />
<Ignore name = "_IOCTRL_STATE" />
<Ignore name = "_State" />
<Ignore name = "_DATA_UPDATED" />
<Ignore name = "_HISTO_PENDING" />
<Ignore name = "_SERVER_WATCHDOG" />

</DevIOMqttServer>
```