

Vapourtec R-Series Controller

R-Series OPC API User Manual

Scope of the Document

This document explains how to communicate with the R-Series SW and control the R-Series equipment through the OPC UA standard.

The explanation uses the free OPC client UAExpert for clarifying some concepts of OPC UA standards and introduces the specific Python-based R-Series OPC UA client.

WARNING: The v2.0 of the API is not compatible with v1.x versions

Revisions

Date	Version	Modifications	Autor
14/09/2021	0.1	Original version.	
15/09/2021	0.2	Revision	
17/09/2021	0.3	Variables writing explained	
8/11/2021	0.4	Manual installation	
26/01/2022	1.0	API Reference	
04/02/2022	1.1	API Reference fixings	
20/09/2022	2.0	OPC Model and API modified	
13/06/2023	2.1	ION addition	

References

Reference	Document Name	Description	Location
[1]	OPCUA Data model	OPCUA Data model	https://app.diagrams.net/#G1rFim7rwJiB0bB1K7u2Ah9ZW6j6yT-2CZ
[2]			

Definitions, acronyms and abbreviations

Term	Definition
TBC	To Be Confirmed
TBD	To Be Defined
OPCUA	Open Platform Communications United Architecture
IDE	Integrated Development Environment

Contenido

1	3
2	4
2.1	4
2.2	7
3	8
4	10
4.1	10
4.2	12
5	14
5.1	14
5.2	14
5.3	15
5.4	15
5.5	17
5.6	20
5.7	20

1 Context

The object of this SW package that can be added to the R Series SW, is to give to a user the possibility to control the R-Series System from third party applications. The API allows a remote computer to automatically load new reaction conditions to the R-Series system, using the R-series software as the control interface.

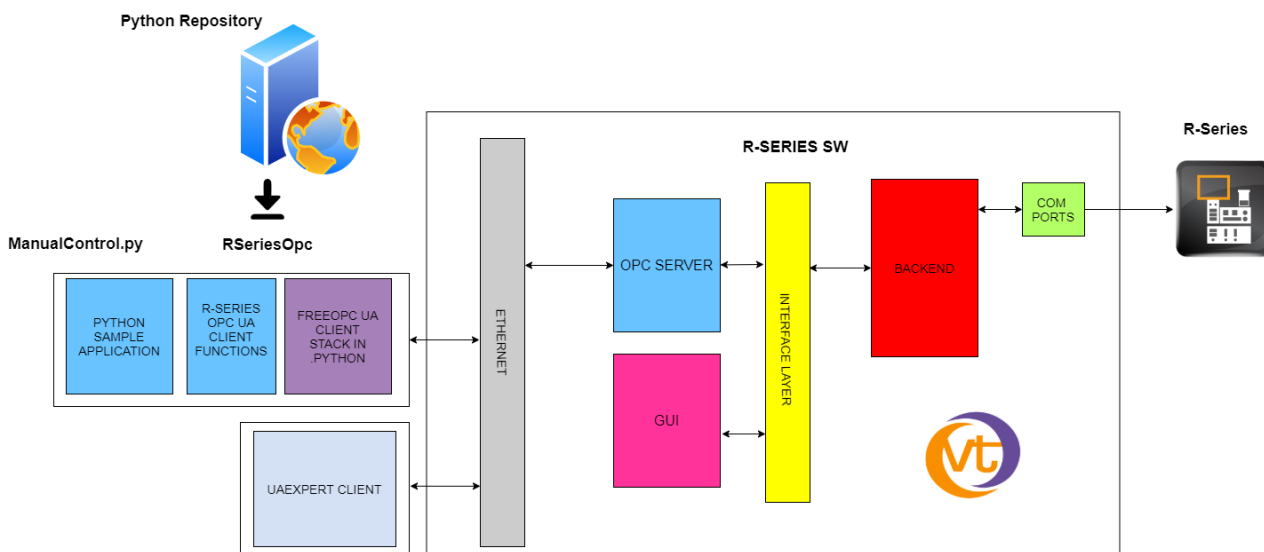


Figure 1-1 Block Diagram of the R-Series SW

2 OPC-UA Server

2.1 OPC UA Server Data Model

The OPC-UA Server Data Model contains all the methods and variables available in the server and the way it is structured. This Information Model provides the ObjectTypes in which the R-Series interact with the clients. A basic structure of the model is shown on Figure 2-1. For check the whole diagram, please refer to "docs/R-Series OPC UA Model.svg"

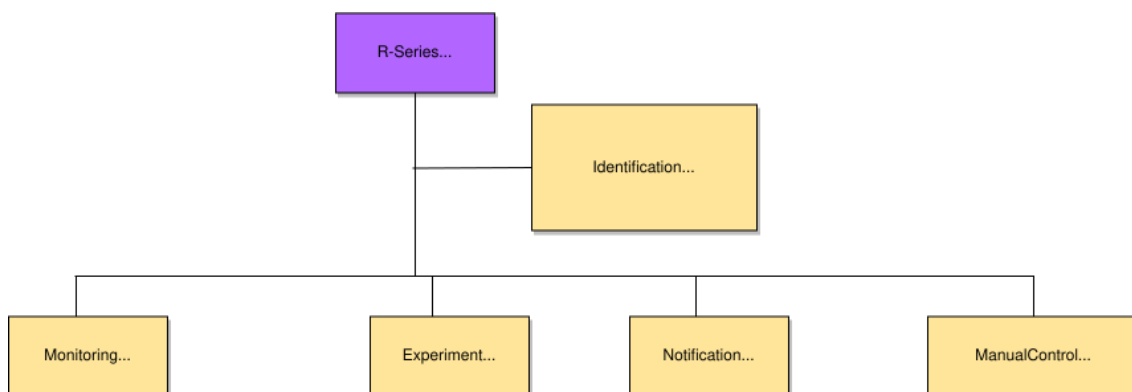


Figure 2—1 Basic Data Model of the RSeries SW

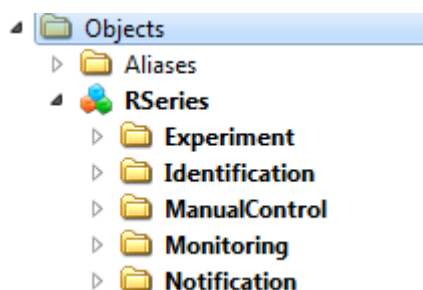


Figure 2-1 Principal ObjectTypes that are used to identify the RSeries SW

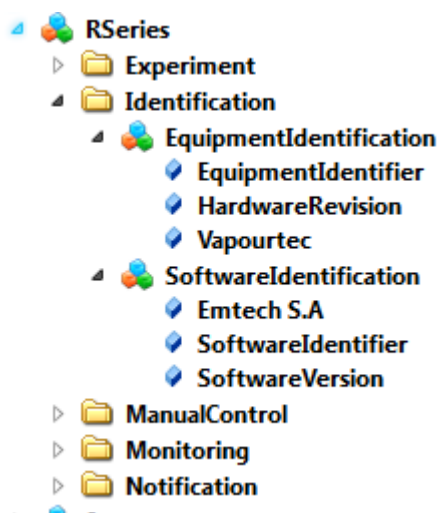


Figure 2-2 Identification ObjectTypes, variables.

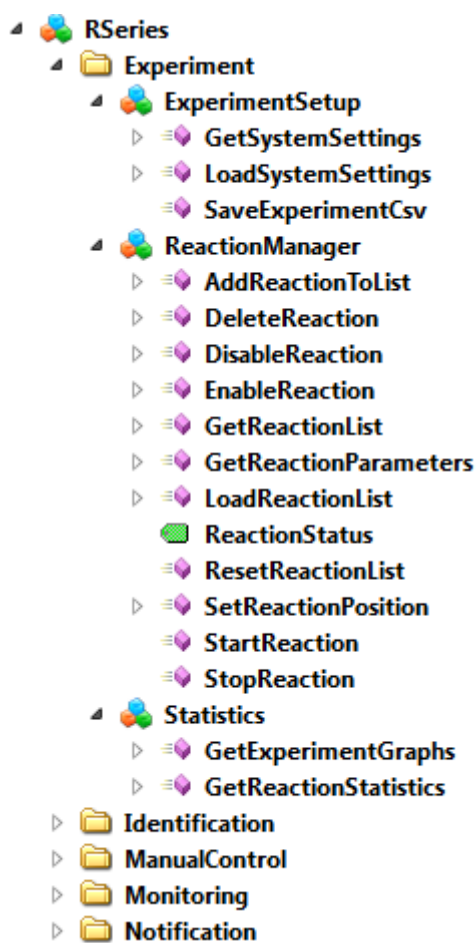


Figure 2-3 Experiment ObjectTypes, variables and methods.

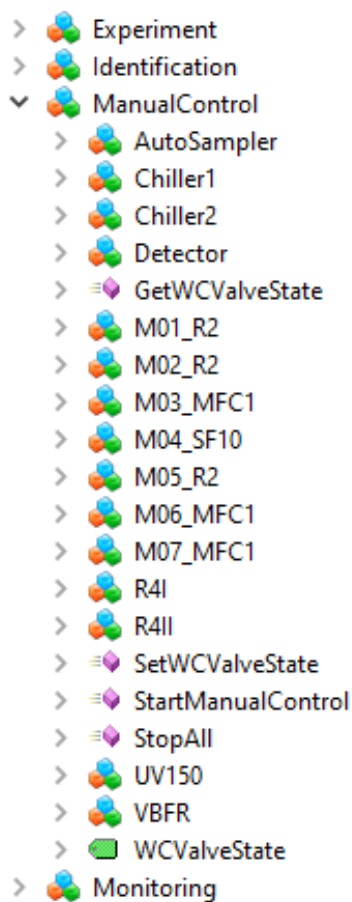
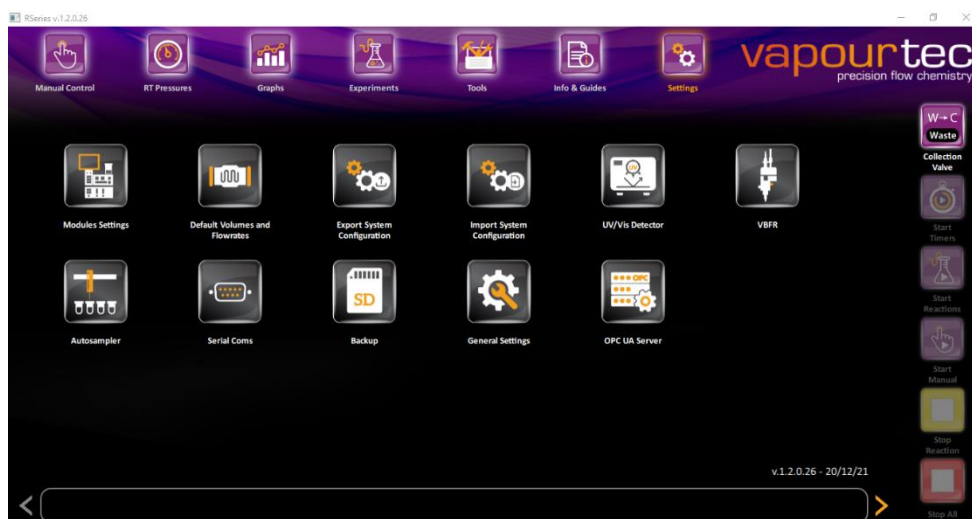
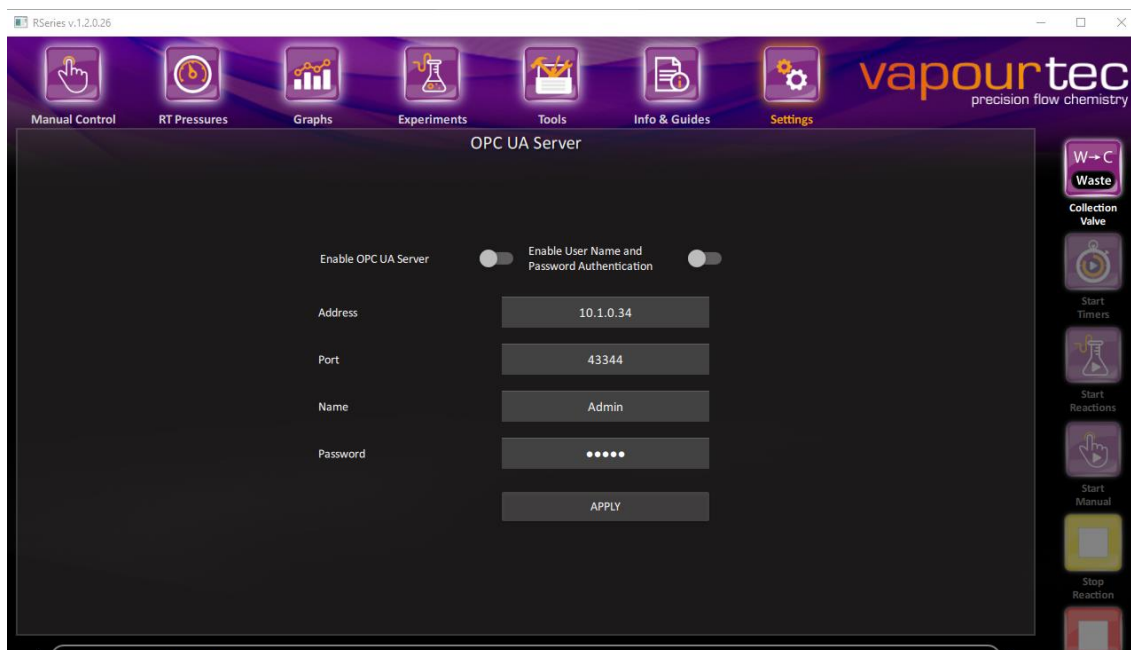


Figure 2-4 Manual Control ObjectTypes, variables and methods.

2.2 Enable the OPC UA Server

In order to enable the OPC-UA Server in the R-Series SW, the path to follow is Settings->OPC-UA Server.





In the OPC-UA Server configuration window, there are two sliders one to enable the OPC UA Server and the other one to enable the user name and password authentication.

Once the OPC-UA server is enabled the system is ready to be controlled by an OPC-UA client.

3 UaExpert Client

The UaExpert is an OPC UA test client is designed as a general purpose test client supporting OPC UA features like DataAccess, Alarms & Conditions, Historical Access and calling of UA Methods.

The UaExpert main window, which is shown in Figure 3-1, is composed by a group of panes with different purposes. In this document, only three of these panes will be used and explained: the Project View pane, the Address Space View pane and the Data View pane.

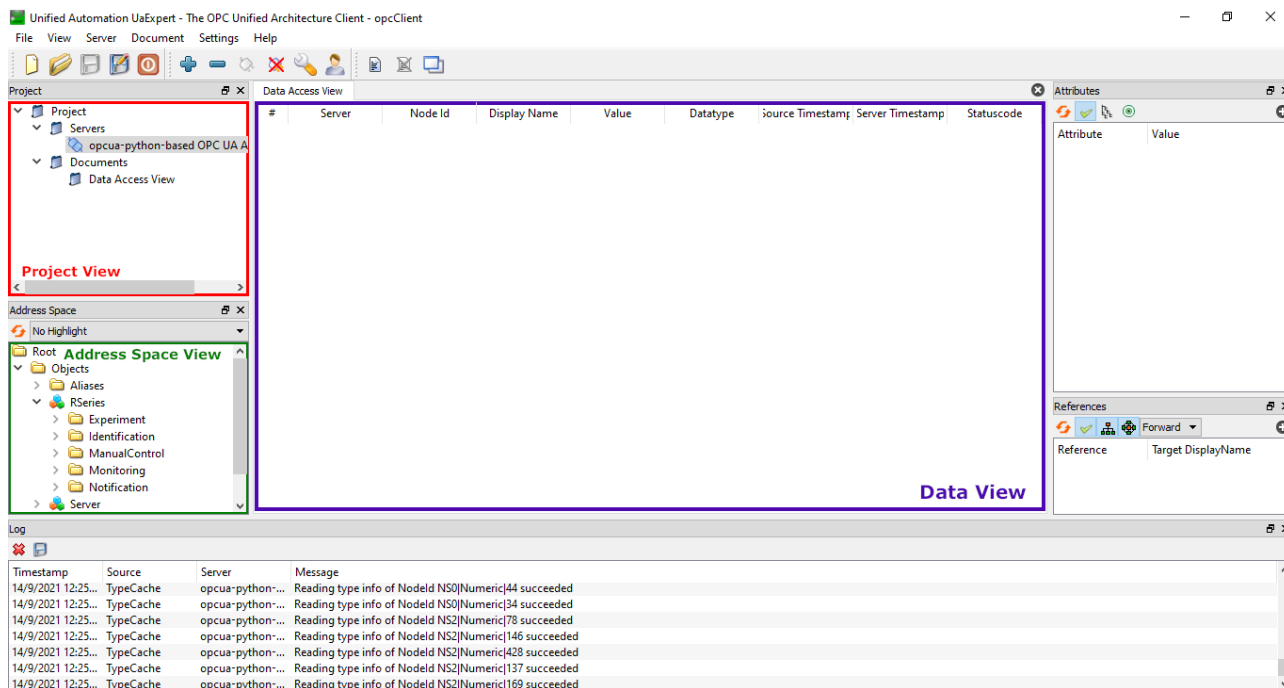


Figure 3-1 The UaExpert main window. The red square is the Project View pane. The green box is the Address Space View pane. The blue one is the Data View pane.

The Project View pane shows the server connection state and it allows the reset the server properties and configurations.

The most important pane for the purpose of this document is the Address Space view. At this pane, the user is allowed to browse all of the components and information exposed by the OPC server and could be acquired through the OPC UA Communication by the client.

The Address Space View shows a drop-down-styled menu, which contains the information elements accessible from the server. Figure 3.2 shows detailed Address Space elements. There are two highlighted elements, the one in the red square which are two methods and the other one in the blue square, which is variable.

A method is an element that makes one or more actions inside the server when it is called. For instance, when the method DeleteReaction is called, the R-Series Controller will make all of the needed steps to remove a reaction from the equipment list of reactions.

A variable, however, is a piece of information that can be monitored when the user subscribes to it. The subscription is done by dragging and dropping the variable on the Data View pane. When a variable is subscribed, the server will send to the client the value of the variable periodically.

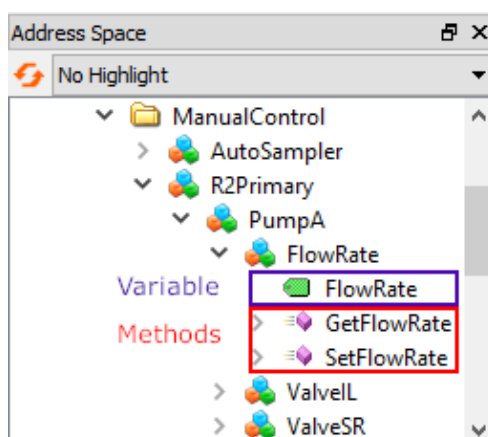



Figure 3-2 Detail of Address Space View pane. The label into the blue square is a Variable. The labels in the red square are Methods.

The Data View pane shows the variables that were subscribed and their value. The values will be refreshing when the variable value changes.

4 Working with the UaExpert Client

4.1 Setting up

Launch UaExpert from the desktop icon , or from StartMenu -> Unified Automation -> UaExpert.

The main window of UaExpert will be opened. From here, set up the connection to the server. For this, a server must be added, clicking on the plus icon. This will open a dialogue box which is showed in Figure 4.1.

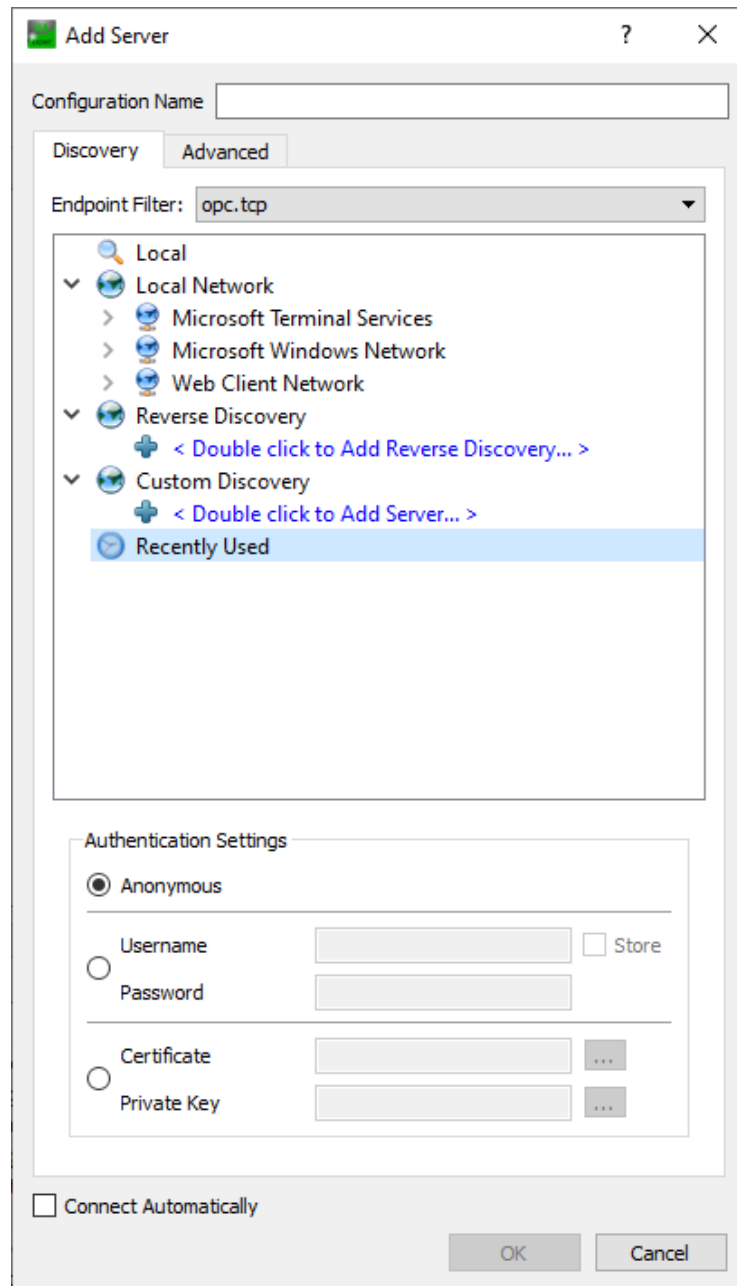


Figure 4-1 OPC Server Configuration dialog box.

In order to set up, the server, click on the Advanced tab. Then complete the followings fields:

- Endpoint Url : `opc.tcp://localhost:43344`
- Configuration Name: write a configuration name. This name will be displayed in Project View later.

Finally, click on the Ok button. Once the server is set up, this could be saved on a file, by clicking on the diskette icon.

4.2 Working with the R-Series Server

After the setup is done, connect to the server by clicking on the plug icon. When the server is connected a plugged icon will be showed in the Project View panel. In addition, the tree of the R-Series model will be available to browse the variables in de Address Space View.

To execute a method, right-click on it and select "Call...". A dialogue box will be displayed with its input and/or output fields available.

To monitor a Variable, it could be dragged and dropped in the Data View panel.

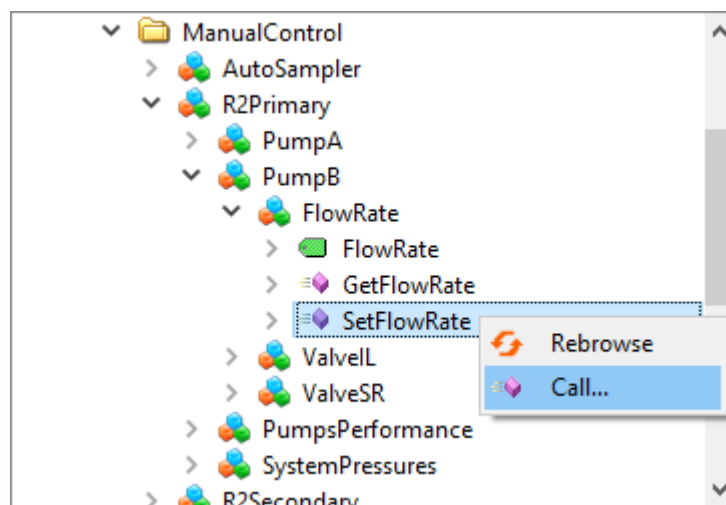


Figure 4-2 Right-Click options for methods in the Address View Pane.

Example 1: Calling methods.

Setting the Flow Rate of the pump B into the R2Primary to 2.8 ml/min:

- Browse in the Address Space View pane :
RSeries -> ManualControl -> R2Primary -> PumpB -> FlowRate
- Right-click on the SetFlowRate method and select the "Call..." option.
- Into the input field of the dialog box displayed, write the desired Flow Rate in ul/min. It means you write 2800.
- Click on the "Call" button.

This method, that is shown in Figure 4-2, will set the setpoint to 2.8 ml/min. To turn on the pump, the StartManualControl method needs to be sent:

- Browse in the Address Space View pane:
RSeries -> ManualControl
- Right-click on StartManualControl and select the "Call..." option.
- Click on the "Call" button in the dialog box displayed.

At this moment, the R-Series controller will be starting the pump.

To check the pump flow rate, the method to call is GetFlowRate:

- Browse again in the Address Space View pane:
RSeries -> ManualControl -> R2Primary -> PumpB -> FlowRate
- Right-click on GetFlowRate and select the "Call..." option.

- Click on the "Call" button in the dialog box.
- The value will be displayed in the output FlowRate field.

To stop the system, the method StopAll needs to be sent:

- Browse in the Address Space View pane:
RSeries -> ManualControl
- Right-click on StopAll and select the "Call.." option.
- Click on the "Call" button.

Example2: Working with Variables

The power and the temperature of Reactor 1 in the R4 Primary will be monitored and changed from its variable.

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	opcua-python-...	NS3 Numeric 1...	Temperature	25	Float	17:11:28.948	17:11:28.948	Good
2	opcua-python-...	NS3 Numeric 1...	Power	0	Float	17:12:28.151	17:12:28.151	Good

Figure 4-3 Temperature and Power Variables subscribed in Data View pane.

In the first place, both variables will be subscribed by putting them on the Data View pane.

- Browse in the Address Space View pane:
RSeries -> ManualControl -> R4I -> Reactor1 -> Power
- Drag the variable Power and drop it into the Data View pane
- Browse in the Address Space View pane:
RSeries -> ManualControl -> R4I -> Reactor1 -> Temperature
- Drag the variable Temperature and drop it into Data View pane

At this moment, both variables will be visible in the Data View pane. In the field value of the table showed by Data View, the value of each variable could be monitored in real-time. This is shown in Figure 4-3.

Some of the variables are modifiable. In this example, changing the temperature variable, change the temperature setpoint of the reactor. In the case of power, it is not modifiable. In other words, the power could be only be read but a change in its value will not be any means in the R-Series Controller. At the refresh time, this variable will be set in its original value by the controller.

To modify a variable value, the variable must be subscribed. After that, double-clicking the value field, in the Data View pane, allows the user to modify that value. When the variable is modifiable, the value will be persistent. In other case, when the server refreshes the variable, the value will be set to its original one. Figure 4-3 shows the blue-highlight value field for Reactor 1's Temperature variable, after it was double-clicked.

Example 3: Calling high levels methods.

High-levels methods mean commands or methods to communicate the server actions that will be done in automatic mode. In opposition, low-level methods are the functions that control the specific behavior of a pump, reactor, or an individual component in the system.

For instance, adding reactions to the Reaction List, enable or disable them, even start to run a bunch of reactions in a row are high-level methods. These methods are under the Reaction Manager component of the Vapourtec R-Series Controller OPC model.

To enable a Reaction and run it, a bigger setup is needed. For run reactions in an automatic way, in the first place, the R-Series Controller needs to be loaded or created an experiment in the schematic editor and there have to be some reactions in the Reaction List. After the R-Series Controller is configured, the Reaction Manager can enable, disables or remove reactions in the list.

Once the reaction list has information, its reactions could be enabled. A reaction could be enabled with the followings steps:

- In the Address Space View pane, browse the AddReactionToList method:
RSeries -> Experiment -> ReactionManager -> EnableReaction
- Right-click on EnableReaction method and select the "Call..." option.
- Write the position of the reaction in the list which will be enabled. The position is 0-based index, so for enable the 1st reaction, the sent position will be 0, for the 2nd reaction, send 1 and go on.
- Click on the "Call" button.

After the execution of the method, the reaction will be enabled in the reaction list. It could be checked in the R-Series Controller, under Experiments -> Edit Reaction.

5 Python Client

5.1 Introduction

The R-Series OPC UA Python client is an API to connect Vapourtec R-Series flow chemistry controller into an OPC UA network. This allows the user to write and run Python scripts.

The Python script could be a way to automate process and to control the R-Series flow chemistry controller in a remote way through the OPC UA network.

The R-Series OPC UA Python client is based on freeopcua Python package. For further information about this package, see <https://python-opcua.readthedocs.io/en/latest/>.

5.2 Requirements

To run and install the R-Series OPC UA Python Client, the system must have installed a Python 3.x version.

Because the Python's popularity, there is a lot of environments for python scripts developing and execution. For this explanations, we use the Anaconda environment under Windows OS, and the Spyder IDE with is provided in installing Anaconda.

You can fell free to use your favorite Python IDE and environment manager.

For downloading and installing Anaconda, check its website: www.anaconda.com

5.3 API Installation

Open the Anaconda Prompt and navigate to the directory where the OPC UA Python Client is store, in example "cd Desktop\rseriesopc":

In that directory, run the installation script as follow:

```
python setup.py install
```

Once the script finishes to run, the R-Series OPC UA Python Client is ready to use in Python's scripts, just importing the *rseriesopc* package.

5.4 R-Series OPC UA Python Client structure

The R-Series OPC UA Python repository had the following structure:

- **rseriesopc/**
 - **docs/**
 - **html**
 - R-Series-API_User_Manual_300.pdf
 - R-Series OPCUA API Reference.pdf
 - R-Series OPC UA Model.svg
 - **examples/**
 - addingReactionExample.py
 - getAndLoadExperimentExample.py
 - importCsvExample.py
 - makingReactionsExample.py
 - manualControlExample.py
 - modifyingReactionExample.py
 - reactionManagerExample.py
 - readingDataExample.py
 - manualControlDemo.py
 - reactionManagerDemo.py
 - **src/**
 - **rseriesopc/**
 - client.py
 - utils.py
 - **model/**
 - reaction.py
 - base.py
 - factories.py
 - **devices/**
 - autosampler.py
 - azura.py
 - chiller.py
 - detector.py
 - factories.py
 - flowChemistry.py

- ion.py
 - manualControl.py
 - mfc.py
 - module.py
 - pump.py
 - r2.py
 - r4.py
 - reactor.py
 - sf.py
 - syringe.py
 - uv150.py
 - vbfr.py
- **experiment/**
 - experiment.py
 - experimentSetup.py
 - experimentStatistics.py
 - reactionManager.py
- **identification/**
 - equipmentIdentification.py
 - identification.py
 - softwareIdentification.py
- **monitoring/**
 - monitoring.py
 - status.py
- **notification/**
 - notification.py
- **files/**
 - test.csv
 - **example4/**
 - example4.rs
 - example4.rscfg
 - example4.rsrl
 - **example5/**
 - example5.rs
 - example5.rscfg
 - example5.rsrl
 - **example6/**
 - example6.rs
 - example6.rscfg
 - example6.rsrl
- setup.py

All the relative information about this Python module is in the rseriesopc folder. This is the root directory of the module. In there, the more important files are:

- src: This is the more important folder in the R-Series OPC UA Python Client. This folder contains the script that runs the API.
- docs: The information relative to the module is located there.
- examples: There are a few case-of-use scripts that are aimed to help developers to understand the API.
- files: This folder contains basic experiment files. They are needed for running some examples.
- setup.py: is the script to install the package.

5.5 Example codes

For running the examples codes, first of all, open the script found in on the Python IDE, and then click on Run Script option

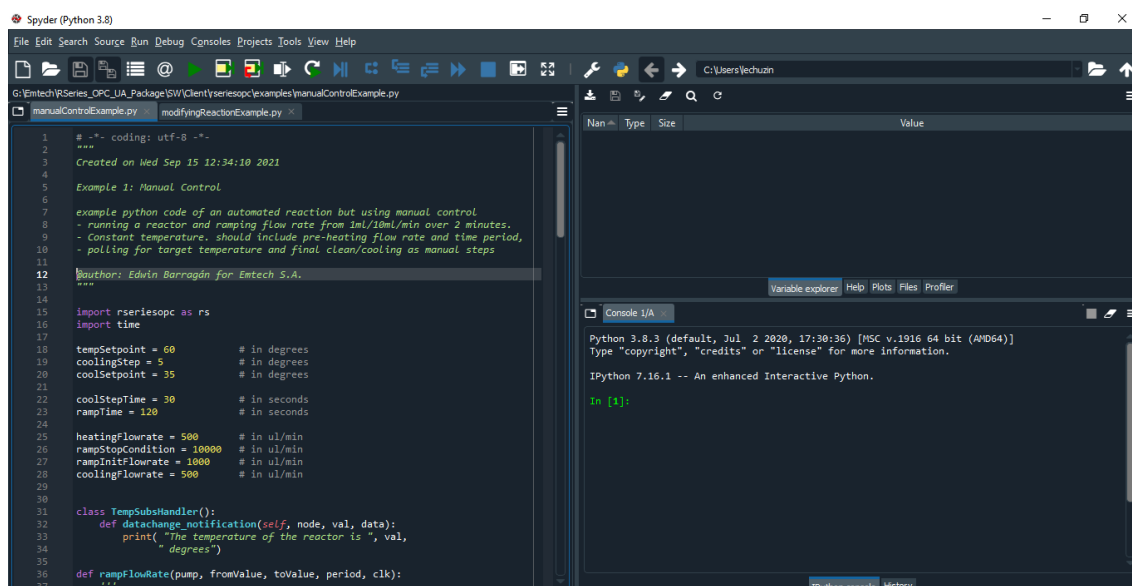



Figure 5-1 Spyder IDE Main View with the example 1 on-screen

In Figure 5-1 the Spyder IDE is shown. To run a script, just open it and click on the green triangle icon . The first time the script is run, Spyder ask to configure the Python console. The displayed dialog is shown in Figure 5-2. The recommended configuration is:

- Console: Select the "Execute in a dedicated console" option.
- General settings: Set the "Remove all variables before execution" option.
- Working directory settings: Select the "The directory of the file being executed" option.

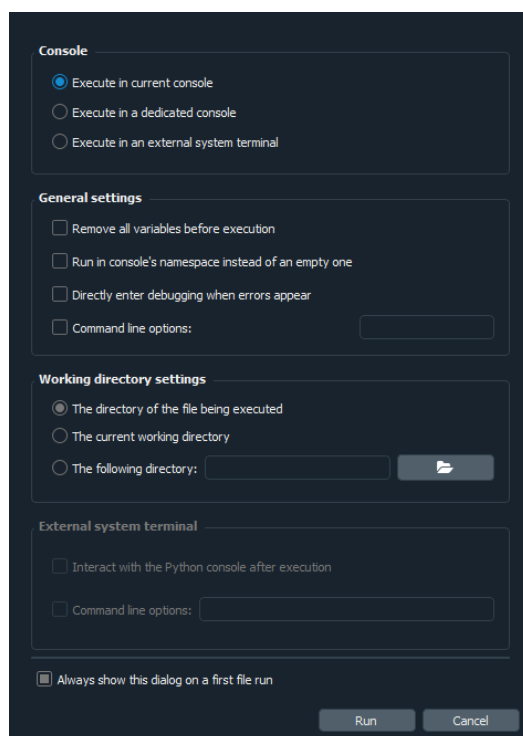


Figure 5-2: Script configuration dialog.

After the configuration, the script will run and show the outputs in the IPython console tab. Also, the variables will be shown in the Variable explorer.

Each Python script for the R-Series controller has a first configuration. In this, an `RSeriesClient` object has to be instantiated with the OPC URL of the server. Then, the objects could be accessed in the same way that has shown in the R-Series OPC Server Model architecture.

Example 1: Manual Control

The manual control example implements some of the functionalities of the low-level functions, modelling an automated reaction controlled by this example. This reaction consists of a pre-heat, a run reaction and a cooling step.

Before running this script, the R-Series controller needs to be connected to an R2 Primary Module. The script works with Reactor 3 and Pump A of the R-Series machine.

When the script is running, in the pre-heat stage, the pump starts to flow and the reactor temperature is set to a predefined temperature. Later, the script will be polling the actual temperature.

The run reaction stage starts once the temperature reaches the set point. In this stage, the pump flow rate will be increased following a ramp function.

In the cooling stage, the pump flow rate is fixed and the temperature of the reactor is decreased in a step way.

Example 2: Reading data

There are two different ways to read data from the R-Series OPC UA server. The first one is about getting the values through getters. This could be polling anywhere in the script.

The second one is by subscribing variables. When a variable is subscribed, the server informs the client of every change from the variable in an automatic way. To perform a subscription, a handler class has to be implemented.

This example shows both methods of reading different values.

Example 3: Getting reaction parameters

This example shows how to retrieve reaction parameters with the JSON file. This example script displays the CSV delimiters and enumerates the reactions present in the ReactionList loaded in the R-Series Controller.

Example 4: Modifying a reaction

This example shows how some low-level commands work. This is composed of two different scripts. In the first one, example 4a, a reaction list is read from the R-Series Controller, the first reaction is modified by setting the first reactor temperature to 100 and then, this reaction is appended to the list in the controller.

In example 4b, the reaction list is read from R-Series Controller, a reaction is modified and then the old reaction is replaced by the new one in the reaction list.

Before running this script, experiment example4 must be loaded in R-Series Controller and the OPC server must be enabled. The example will change the reaction list so making a backup of the example folder is recommended.

When the script is run, the reaction list is read from the R-Series controller, then this is decoded and loaded in the script form modifying the reactor parameter. Then, the new reaction is sent to the R-Series controller and will be shown in the last element of the reaction list that is loaded in the controller.

Example 5: Creating and loading a reaction list

This example reads a reaction in the reaction list. Then, 5 reactions are created, based on that reaction, increasing the temperature of a given reactor by 10 C. Finally, the reaction list is replaced by the new reaction list, composed of the created reactions.

For running the example, the experiment example5 must be loaded in R-Series Controller and the OPC server must be enabled. The example will change the reaction list so making a backup of the example folder is recommended.

Example 6: Saving and restoring experiments

This example reads the current experiment on the R-Series and stores it in a desired location on the PC. Then take a stored experiment on a given location in the PC and loads it on the R-Series.

For running the example, any experiment must be loaded in R-Series Controller and the OPC server must be enabled. The example will store the reaction loaded so changing the default destination should be recommended.

5.6 R-Series OPC UA Client API Reference

Introduction

The implementation of this R-Series OPC UA Client Python API has been done in a similar way to the R-Series OPC UA Server Data Model. The main difference is the main object is an `RSeriesClient` object that allows the user to manage the connection and disconnection events and the security credentials.

The `RSeriesClient` object also has a `FlowChemistryType` object instantiation called `RSeries`. The `FlowChemistryType` has five objects (`IdentificationType`, `MonitoringType`, `ExperimentType`, `NotificationType` and `ManualControlType`) in the same way as the R-Series OPC UA Server Data Model (Figure 2—1).

In addition, there is a `Reaction` object to add and edit some interesting parameters and manage the reaction list easily.

API Reference

For the complete R-Series OPC UA Client API Reference, please refer to `docs/R-Series OPCUA API Reference.pdf` or `docs/html/index.html`. There, the API's attributes structure and methods will be explained in detail.

5.7 Reactions specifications

A reaction contains all of the information about the pumps and reactor set points, the collection valve states and the times. **TBD**

Table 5-1. Reaction Parameters

Position	Parameter	Note
1	REACTION_NAME	The name will be displayed in reaction list
2	RESIDENCE_TIME	Residence time
3	TEMPERATURE_R1	Reactor 1 Temperature
4	RESIDENCE_TIME_R1	Reactor 1 Residence time calculation (True or False)
5	TEMPERATURE_R2	Reactor 2 Temperature
6	RESIDENCE_TIME_R2	Reactor 2 Residence time calculation (True or False)
7	TEMPERATURE_R3	Reactor 3 Temperature

8	RESIDENCE_TIME_R3	Reactor 3 Residence time calculation (True or False)
9	UV_POWER_R3	Reactor 3 UV Power
10	TEMPERATURE_R4	Reactor 4 Temperature
11	RESIDENCE_TIME_R4	Reactor 4 Residence time calculation (True or False)
12	TEMPERATURE_R5	Reactor 5 Temperature
13	RESIDENCE_TIME_R5	Reactor 5 Residence time calculation (True or False)
14	TEMPERATURE_R6	Reactor 6 Temperature
15	RESIDENCE_TIME_R6	Reactor 6 Residence time calculation (True or False)
16	UV_POWER_R6	Reactor 6 UV Power
17	TEMPERATURE_R7	Reactor 7 Temperature
18	RESIDENCE_TIME_R7	Reactor 7 Residence time calculation (True or False)
19	TEMPERATURE_R8	Reactor 8 Temperature
20	RESIDENCE_TIME_R8	Reactor 8 Residence time calculation (True or False)
21	FLOWRATE_CHEM_A	Pump A flow rate
22	VOL_RATIO_CHEM_A	Pump A volumetric ratio
23	CONC_RATIO_CHEM_A	Pump A stoichiometric ratio
24	QUANTITY_CHEM_A	Pump A quantity (reagent volume)
25	AD_RET_CHEM_A	Pump A advance retard
26	REAG_CONC_CHEM_A	Pump A reagent concentration
27	FLOWRATE_CHEM_B	Pump A flow rate
28	VOL_RATIO_CHEM_B	Pump A volumetric ratio
29	CONC_RATIO_CHEM_B	Pump A stoichiometric ratio
30	QUANTITY_CHEM_B	Pump A quantity (reagent volume)
31	AD_RET_CHEM_B	Pump A advance retard
32	REAG_CONC_CHEM_B	Pump A reagent concentration
33	FLOWRATE_CHEM_C	Pump A flow rate
34	VOL_RATIO_CHEM_C	Pump A volumetric ratio
35	CONC_RATIO_CHEM_C	Pump A stoichiometric ratio
36	QUANTITY_CHEM_C	Pump A quantity (reagent volume)
37	AD_RET_CHEM_C	Pump A advance retard
38	REAG_CONC_CHEM_C	Pump A reagent concentration
39	FLOWRATE_CHEM_D	Pump A flow rate
40	VOL_RATIO_CHEM_D	Pump A volumetric ratio
41	CONC_RATIO_CHEM_D	Pump A stoichiometric ratio
42	QUANTITY_CHEM_D	Pump A quantity (reagent volume)
43	AD_RET_CHEM_D	Pump A advance retard
44	REAG_CONC_CHEM_D	Pump A reagent concentration
45	FLOWRATE_CHEM_E	Pump A flow rate
46	VOL_RATIO_CHEM_E	Pump A volumetric ratio
47	CONC_RATIO_CHEM_E	Pump A stoichiometric ratio
48	QUANTITY_CHEM_E	Pump A quantity (reagent volume)
49	AD_RET_CHEM_E	Pump A advance retard

50	REAG_CONC_CHEM_E	Pump A reagent concentration
51	FLOWRATE_CHEM_F	Pump A flow rate
52	VOL_RATIO_CHEM_F	Pump A volumetric ratio
53	CONC_RATIO_CHEM_F	Pump A stoichiometric ratio
54	QUANTITY_CHEM_F	Pump A quantity (reagent volume)
55	AD_RET_CHEM_F	Pump A advance retard
56	REAG_CONC_CHEM_F	Pump A reagent concentration
57	FLOWRATE_CHEM_G	Pump A flow rate
58	VOL_RATIO_CHEM_G	Pump A volumetric ratio
59	CONC_RATIO_CHEM_G	Pump A stoichiometric ratio
60	QUANTITY_CHEM_G	Pump A quantity (reagent volume)
61	AD_RET_CHEM_G	Pump A advance retard
62	REAG_CONC_CHEM_G	Pump A reagent concentration
63	FLOWRATE_CHEM_H	Pump A flow rate
64	VOL_RATIO_CHEM_H	Pump A volumetric ratio
65	CONC_RATIO_CHEM_H	Pump A stoichiometric ratio
66	QUANTITY_CHEM_H	Pump A quantity (reagent volume)
67	AD_RET_CHEM_H	Pump A advance retard
68	REAG_CONC_CHEM_H	Pump A reagent concentration
69	PRE_WASH_VOLUME	Pre-Wash volume
70	PRE_WASH_FR	Pre-Wash flow rate
71	POST_WASH_VOLUME	Pre-Wash volume
72	POST_WASH_FR	Pre-Wash flow rate
73	COLLECTION	Collection state. It can be ALL, STEADY STATE or MANUAL
74	DIVERT_TO_WASTE	Time for divert to waste
75	COLLECT_FOR	Time to collect for
76	COLLECTION_VOLUME	Total collection volume
77	VOL_PER_VIAL	Volume per vial
78	NR_OF_VIALS	Number of vials