R-Series OPC Python API Reference

Release 2.1

Jan 24, 2024

R-SERIES OPC UA CLIENT

CHAPTER

ONE

INTRODUCTION

The R-Series OPC UA Python Client is an API to connect the 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 the process and 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 the freeopcua Python package. For further information about this package, see https://python-opcua.readthedocs.io/en/latest/.

1.1 Requirements

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

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

You can feel free to use your favourite Python IDE and environment manager.

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

CHAPTER

TWO

R-SERIES OPC UA CLIENT

2.1 rseriesopc

2.1.1 client

Created on Wed Jun 23 14:18:43 2021

This module is for the high level opc client configuration. It contains the connection and disconnection methods to the server.

`class RSeriesClient(url)`

The RSeries Client class contructs a OPC client object for RSeries OPC server.

This class inherits opcua.Client class. This allows to use all of the funtions provided by it. Further information about this class and its use cases could be found in https://python-opcua.readthedocs.io/en/latest/

This is a High Level Class that instantiates all the needed components to connect the R-Series Controller through OPC-UA communication

Parameters
`url` (*String*) – It is the RSeries Controller OPC Server address.

`rSeries`

It represents the R-Series machine. This has all the methods to communicate to the controller.

Type
FlowChemistryType

`isConnected`

It is true when the client is connected to the R-Series Controller. Otherwise, this is false.

Type
bool

`connect()`

It connects the client with RSeries OPC Server.

`disconnect()`

It disconnects the client with the R-Series Controller OPC UA Server.

`getRSeries()`

It returns the RSeries machine

`connect()`

It connects the client with RSeries OPC Server.

Returns
It returns true if the connections was successful. Else, it returns false.

Return type
bool

`disconnect()`

It disconnects the client with the R-Series Controller OPC UA Server.

Return type
None.

`getRSeries()`

It returns the RSeries machine

Returns
It is the R-Series machine.

Return type
FlowChemistryType

`loadRSeries()`

This function look up in the server and construct the RSeries object of interest.

Returns
It is True when an RSeries machine was loaded. Else, it is False.

Return type
bool

2.1.2 model package

2.1.2.1 model

2.1.2.1.1 devices package

2.1.2.1.1.1 autosampler

`class AutoSamplerType(`*root*`)`

The AutoSamplerType class instantiates Autosampler objects. This objects allows a high level communication to send orders to the GX271 system, connected to the R-Series Controller.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

This contains the id of the Autosampler in the OPC UA Address Space.

Type
opcua.Node

`collectionValveState`

It is the id of the collection valve variable in the OPC UA Address Space.

Type
opcua.Node

`cleanLoop`(*channel*)

It cleans the channel loop

Parameters
`channel` (*integer*) – This is the channel that will be cleaned.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`cleanNeedle()`

It rinses the needle.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`cleansSampleLoops()`

The probe will inject the solvent from the reservoir and flush the sample loops with the solvent. Depend on the setup, it will activate the number of sample loops as per the experiment setup in R-series controller.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`getCollectionValveState()`

It gives if the valve is in collection or waste direction.

Returns
This is the valve position. If it is true, the valve is in collect position. Else, the valve is in waste position.

Return type
Bool

`getCollectionValveStateNode()`

This method returns the id of the colection valve variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the collection valve variable in the OPC UA Address Space.

Return type
opcua.Node

`goHome()`

It moves the Autosampler arm to home position.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`goToPortA()`

It moves the autosampler arm to port A.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`goToVialForCollection`(*site*, *time*)

It moves the autosampler arm to the especified and take a sample during a given time.

Parameters

- **site** (*integer*) – It is the position where the autosampler collect from.
- **time** (*float*) – It is the time to collect in seconds.

Returns
It is false when the site is bigger than allowed. Otherwise, this is true.

Return type
bool

`loadLoopVolumeFromVial`(*channel*, *position*, *volume*)

It moves the arm to a given position and dispenses product.

Parameters

- **channel** (*integer*) – This is the rack index.
- **position** (*integer*) – It is the position in the rack where the product will be dispensed.
- **volume** (*integer*) – It is the amount of liquid in ml to dispense into the loop.

Returns
It is false when the position is bigger than allowed. Otherwise, it is true.

Return type
bool

`lowerNeedleIntoPortA()`

It moves the needle down into Port A.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`lowerNeedleOverPortA()`

It moves the needle down over Port A.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`lowerSyringePump()`

Use this function when you need to replace the syringe. Refer to section 4.11 Syringe pump installation.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`primeSampleLoop()`

It fills the syringe tube with solvent. The Syringe must be primed before running the first reaction and when the solvent needs to be changed.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`riseSyringePump()`

Use this function when you need to install the syringe. Refer to section 4.11 Syringe pump installation.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setCollectionValveState(`*state*`)`

It sets the autosampler valve in collection or waste position.

Parameters
`state` (*Bool*) – This is the postion of the valve. When it is true, the valve will be set on collect position. Otherwise it will be set on waste position.

Return type
None.

`stopRoutines()`

This method stops any routine execution.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`switchValveToCollect()`

It switches the Waste/Collect to Collect and a few seconds later, the valve comes back to waste in an autmatic way.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.2 azura

`class AzuraPumpType(`*`root`*`)`

An AzuraPumpType object contains all of the methods and object to monitor and control a Azura Pump Module in the R-Series Controller.

Parameters
`root` (*`opcua.Node`*) – It is the parent node in the OPC UA Address Space.

`root`

It is the R2 object id in the OPC UA Address Space.

Type
opcua.Node

`erroState`

It is the ErrorState variable id in the OPC UA Address Space.

Type
opcua.Node

`maxPressureLimit`

It is the MaxPressureLimit variable id in the OPC UA Address Space.

Type
opcua.Node

`pressureLimit`

It is the PressureLimit variable id in the OPC UA Address Space.

Type
opcua.Node

`flowRate`

It is the FlowRate variable id in te OPC UA Address Space.

Type
opcua.Node

`getErrorState()`

It reports if there was an error with the module.

Returns
It is True if there was an error. Otherwise, it si False.

Return type
bool

`getErrorStateNode()`

This method returns the id of the ErrorState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ErrorState variable in the OPC UA Address Space.

Return type
opcua.Node

`getFlowRate()`

This function gives the current pump flow rate.

Returns
It is the pump flow rate in ul/min.

Return type
integer

`getFlowRateNode()`

This method returns the id of the flow rate variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the flowRate variable in the OPC UA Address Space.

Return type
opcua.Node

`getMaxPressureLimit()`

It gives the actual maximum pressure limit.

Returns
It is the actual maximum pressure limit.

Return type
float

`getMaxPressureLimitNode()`

This method returns the id of the SF10 MaxPressureLimit variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 MaxPressureLimit variable in the OPC UA Address Space.

Return type
opcua.Node

`setFlowRate`(*flowRate*)

It sets the flow rate to the pump. When flow rate is setted in 0, the pump is turned off. If the pump is setted in other value than 0, the pump is immediatly turned on. The flow rate must be set in a suitable value for the connected pump in R2. In other case, the flow rate will not be setted in the machine.

Parameters
flowRate (*integer*) – It is the pump flow rate in ul/min.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setMaxPressureLimit`(*newMaxPressureLimit*)

It sets a new maximum pressure limit.

Parameters
`newMaxPressureLimit` (*`float`*) – It is the new maximum pressure limit.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.3 chiller

`class ChillerType`(*root*)

A ChillerType Object contains all the variables and methods to know the measurements and information about the Chillers.

Parameters
`root` (*`opcua.Node`*) – It is the id of a Chiller object in the OPC UA Address Space.

`root`

It is the id of the Chiller object in the OPC UA Address Space.

Type
opcua.Node

`errorState`

It is the id of the error state variable for the Chiller in the OPC UA Address Space.

Type
opcua.Node

`setTemperature`

It is the id of the Chiller Temperature Set Point variable in the OPC UA Address Space.

Type
opcua.Node

`temperature`

It is the id of the Chiller Temperature measurement variable in the OPC UA Address Space.

Type
opcua.Node

`chillerType`

It is the id of the type of Chiller variable in the OPC UA Address Space.

Type
opcua.Node

`leaveRunning`

It is the id of the Chiller's leaveRunning variable in the OPC UA Address Space.

Type
opcua.Node

`sensorType`

It is the id of the sensor Type of the Chiller variable in the OPC UA Address Space.

Type
opcua.Node

`getChillerType()`

It gives the chiller type target.

Returns
It is the chiller type. 0 means Julabo Chiller. 1 means Huber Chiller.

Return type
integer

`getChillerTypeNode()`

This method returns the id of the Chiller Type variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the Chiller Type variable in the OPC UA Address Space.

Return type
opcua.Node

`getErrorState()`

It reports if there are an error in the chiller.

Returns
It is True when an error occurs. Otherwise, it is False.

Return type
bool

`getErrorStateNode()`

This method returns the id of the Chiller's error state variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the ErrorState Chiller's variable in the OPC UA Address Space.

Return type
opcua.Node

`getLeaveRunning()`

It gives if the chiller will be left running after reaction list excecution.

Returns
If it is True, the chiller will not be turned off at the end of the reaction lsit excecution.

Return type
bool

`getLeaveRunningNode()`

This method returns the id of the Chiller’s Leave Running variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the LeaveRunning variable in the OPC UA Address Space.

Return type
opcua.Node

`getSensorType()`

It gives the sensor type present on the device.

Returns
It is kind of control for the chiller. 0 means the control is made with the interal sensor. 1 means the control is made with an external sensor.

Return type
integer

`getSensorTypeNode()`

This method returns the id of the Chiller’s Sensor Type variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the Sensor Type variable in the OPC UA Address Space.

Return type
opcua.Node

`getSetTemperature()`

It gives the temperature target.

Returns
It is the temperature target for the chiller.

Return type
float

`getSetTemperatureNode()`

This method returns the id of the Chiller's temperature target variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SetTemperature variable in the OPC UA Address Space.

Return type
opcua.Node

`getTemperature()`

It gives the temperature measurement.

Returns
It is the temperature in the chiller.

Return type
float

`getTemperatureNode()`

This method returns the id of the Chiller's temperature variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the Temperature variable in the OPC UA Address Space.

Return type
opcua.Node

`setChillerType(`*chiller*`)`

It set the chiller controller. Each brand needs an specific controller.

Parameters
chiller (*integer.*) – This is the chiller type. 0 is for Julabo Chiller. 1 is for Huber Chiller.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setLeaveRunning(`*state*`)`

It set the leave running parameter.

When Leave Running is True, the chiller will not be turned off at the end of the reaction list excecution.

Parameters
state (*bool.*) – This is the new leave running state for the chiller.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setSensorType`(*sensor*)

It set the temperature target.

Parameters
`sensor` (*`int.`*) – This is the new sensorType for the chiller. 0 means internal sensor. 1 menas external sensor.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setTemperature`(*temperature*)

It set the temperature target.

Parameters
`temperature` (*`float.`*) – This is the new temperature target for the chiller.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.4 detector

`class DetectorType`(*root*)

A DetectorType object contains the variables with the measurements and information about the UV50D detector

Parameters
`root` (*`opcua.Node`*) – It is the id of this object in the OPC UA Address Space.

`root`

It is the id of the detector in the OPC UA Address Space.

Type
opcua.Node

`lampState`

It is the id of the lamp state variable in the OPC UA Address Space.

Type
opcua.Node

`Wavelengths`

It is the id of the measuring values in the OPC UA Address Space.

Type
opcua.Node

`getLampState`()

It gives the lamp status. It could be:

OFF = 0

HEATING = 1

ON = 2

Returns
It is the lamp status.

Return type
integer

`getLampStateNode()`

This method returns the id of the UV50D Lamp status variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the UV50D Lamp Status variable in the OPC UA Address Space.

Return type
opcua.Node

`getWavelengthAt(`*index*`)`

It gives a measuremnt of the desired wavelength detector.

Parameters
index (*integer*) – It is the index of the detector to measure.

Returns
It is the detector measurement.

Return type
float

`getWavelengthNode()`

This method returns the id of the UV50D measurements variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the UV50D measurements variable in the OPC UA Address Space.

Return type
opcua.Node

`getWavelengths()`

It gives an array with all of detetors measurements.

Returns
These are the measurements.

Return type
list of float

2.1.2.1.1.5 factories

class ModuleTypeFactory

It is a factory that returns the right type of reagent source class when makeReagentSourceType is called. The ModuleType class has any members.

isAReagentSourceModule()

It reports if the selected node is a module that can be created by this object or not.

Parameters
node (*opcua.Node*) – It is the node that wants to be readed from OPC UA address space.

Returns
It is True when the node is a reagent source object. Otherwise, it is False.

Return type
bool

makeReagentSourceType()

It returns an object of one of the reagent source types.

Parameters
node (*opcua.Node*) – It is the node to be readed from OPC UA address space.

Returns
It is the reagent source type object

Return type
R2Type, *SF10Type*, *MFCType*, *AzuraPumpType*, *SyringePumpType* or None

2.1.2.1.1.6 flowChemistry

class FlowChemistryType(*parentNode*)

The FlowChemistry class is the representation of the whole Vapourtec R-Series Flow Chemistry machines.

This OPC UA Client is based on the freeopcua. The Python OPC-UA documentation could be found at python-opcua.readthedocs.io.

Parameters
parentNode (*opcua.Node*) – This is the root object of the R-Series OPC UA Server. From there, all of the relative information about the R-Series Flow Chemistry Machine will be read on connection.

root

It is the address of the R-Series machine in the OPC-UA Address Space.

Type
opcua.Node

identification

It is an object to get the machine identification information.

Type
IdentificationType

monitoring

It is an object that has the methods to get the data to monitor the R-Series status.

Type
MonitoringType

`experiment`

It cointains the experiment settings and the reactions information.

Type
ExperimentType

`notification`

This object has address from variables for subscribe to data changes.

Type
NotificationType

`manualControl`

It contains all the methods for low level controlling of the R-Series subsystems.

Type
ManualControlType

`getExperiment()`

It is a getter to gain easier access to the experiment information.

Returns
It is the node that allows the user managing the experiment setup and data.

Return type
ExperimentType

`getIdentification()`

It is a getter to gain easier access to the identification information.

Returns
It is the identification information of the Vapourtec RSeries Flow Chemistry Machine.

Return type
IdentificationType

`getManualControl()`

It is a getter to gain easier access to manual control methods.

Returns
It is the manual control object.

Return type
ManualControlType

`getMonitoring()`

It is a getter to gain easier access to the monitoring information.

Returns
It is the node that contains the machine status information.

Return type
MonitoringType

`getNotification()`

It is a getter to gain easier access to alerts.

Returns
It is the node that emits alerts.

Return type
NotificationType

2.1.2.1.1.7 ion

`class IONType`(*root: Node*)

The IONType class implements the low level functions for controlling the components of the R-Series Machine.

This class allows the user to control the electrical control functions for the Vapourtec ION Electrochemial Reactor.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

This is the parent node in the OPC UA Address Space.

Type
opcua.Node

`voltage`

This is the id of the voltage variable in the OPC UA Address Space.

Type
opcua.Node

`current`

It is the id of the current variable in the OPC UA Address Space.

Type
opcua.Node

`getCurrent()`

It gives a measuremnt of the ION Reactor's current.

Returns
It is the reactor's current measurement.

Return type
float

`getCurrentNode()`

This method returns the id of the ION Reactor's current variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the reactor's current variable in the OPC UA Address Space.

Return type
opcua.Node

`getVoltage()`

It gives a measuremnt of the ION Reactor's voltage.

Returns
It is the reactor's voltage measurement.

Return type
float

`getVoltageNode()`

This method returns the id of the ION Reactor's voltage variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the reactor's voltage variable in the OPC UA Address Space.

Return type
opcua.Node

`setCurrent`(*newCurrentInA: float*)

It sets a new target for the ION Reactor's current.

Parameters
`index` (*`float`*) – It is the new current target in Amperes.

Returns
It returns a boolean. It it is true, means the value was set. Otherwise, it is false

Return type
bool

`setVoltage`(*newVoltageInV*)

It sets a new target for the ION Reactor's voltage.

Parameters
`index` (*`float`*) – It is the new voltage target in Volt.

Returns
It returns a boolean. It it is true, means the new value was set. Otherwise, it is false

Return type
bool

`start()`

It starts the electrochemical control functions of the reactor.

`stop()`

It stops the electrochemical control functions of the reactor.

2.1.2.1.1.8 manualControl

class ManualControlType(*root: Node*)

The ManualControlType class implements the low level functions for controlling the components of the R-Series Machine.

This class allows the user to write and run scripts for custom automation routines.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

root

This is the parent node in the OPC UA Address Space.

Type
opcua.Node

module

It is a collection of ModuleType objects. Those objects contains the reagent source modules and the methods related to control the pumps.

Type
list of a reagent source type

r4

It is a collection of R4Type objects. Those objects contains the R4 modules and the methods related to set the power and temperatures of the reactors.

Type
dict of *R4Type*

uv150

This is an object to control UV150 reactors.

Type
UV150Type

autoSampler

It is an object that implements the methods to control the AutoSampler module

Type
AutoSamplerType

wcValveState

It is an the WCValveState variable in the OPC UA address space. It can control the Waste/Collect valve.

Type
opcua.Node

chiller

It is a collection of ChillerType bbjects. Those objects contains the chiller modules and the methods related to control and monitoring them.

Type
dict of *ChillerType*

getAutoSampler()

If there is an AutoSampler module, this gives that node.

Returns
If the AutoSmapler is in RSeries, it is returned.

Return type
AutoSamplerType or None

`getChiller1()`

If there is a Chiller, this gives that node.

Returns
If Chiller is in RSeries, it is returned. Else, None is returned

Return type
ChillerType or None

`getChiller2()`

If there is a Chiller, this gives that node.

Returns
If Chiller is in RSeries, it is returned. Else, None is returned

Return type
ChillerType or None

`getChillers()`

It returns a dictionary with all Chillers modules.

Returns
These are all of the Chillers modules.

Return type
dict

`getDetector()`

It returns an external detector object.

Returns
It is the detector.

Return type
DetectorType

`getR4()`

It returns a dictionary with all R4 modules.

Returns
These are all of the R4 modules.

Return type
dict

`getR4I()`

If there is a R4 Module, this gives that node.

Returns
If R4Module is in RSeries, it is returned. Else, None is returned

Return type
R4Type or None

`getR4II()`

If there is a second R4 Module, this gives that node.

Returns
If a second R4Module is in RSeries, it is returned. Else, None is returned

Return type
R4Type or None

`getUV150()`

If there is a UV150 module, this gives that node.

Returns
If a UV150module is in RSeries, it is returned.

Return type
UV150Type or None

`getVBFR()`

It gives the Variable Bed Flow Reactor object.

Returns
It is the VBFR.

Return type
VBFRType

`getWCValveState()`

This gives the state of the valve

Returns
True means liquid from Collect to Waste Else, the liquid moves in the another way.

Return type
bool

`getWCValveStateNode()`

This method returns the id of the valve state variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ValveState variable in the OPC UA Address Space.

Return type
opcua.Node

`setWCValveState(`*state*`)`

It sets the state of the valve.

Parameters
`state` (*bool*) – True means fluid from Collect to Waste. Else, the fluid moves in the another way.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`startManualControl()`

This function allows the user to reach the control of the machine from this Python Client. It starts the manual control when the machine gives back a confirmation.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`stopAll()`

This function stops the manual control of the Vapourtec R-Series Flow Chemistry Machine.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.9 mfc

`class MFCType(`*root: Node*`)`

A MFCType object contains all of the methods and object to monitor and control a MFC Module in the R-Series Controller.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

It is the R2 object id in the OPC UA Address Space.

Type
opcua.Node

`erroState`

It is the ErrorState variable id in the OPC UA Address Space.

Type
opcua.Node

`gasType`

It is the GasType variable id in the OPC UA Address Space.

Type
opcua.Node

`flowRate`

It is the FlowRate variable id in the OPC UA Address Space.

Type
opcua.Node

`getErrorState()`

It reports if there was an error with the module.

Returns
It is True if there was an error. Otherwise, it si False.

Return type
bool

`getErrorStateNode()`

This method returns the id of the ErrorState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ErrorState variable in the OPC UA Address Space.

Return type
opcua.Node

`getFlowRate()`

This function gives the current pump flow rate.

Returns
It is the pump flow rate in ul/min.

Return type
integer

`getFlowRateNode()`

This method returns the id of the flow rate variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the flowRate variable in the OPC UA Address Space.

Return type
opcua.Node

`getGasType()`

It gives the gas type MFC has set.

Returns
This is the code for the gas type.

Return type
integer

`getGasTypeNode()`

This method returns the id of the ErrorState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ErrorState variable in the OPC UA Address Space.

Return type
opcua.Node

`setFlowRate`(*flowRate*)

It sets the flow rate to the pump. When flow rate is setted in 0, the pump is turned off. If the pump is setted in other value than 0, the pump is immediatly turned on. The flow rate must be set in a suitable value for the connected pump in MFC. In other case, the flow rate will not be setted in the machine.

Parameters
`flowRate` (*integer*) – It is the pump flow rate in ul/min.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.10 module

`class ModuleType`(*root*)

This is a base class for class of objects that delivers some type of reagent and/or solvent.

`getPumps`()

It gives the pumps dictionary.

Returns
It is the list of pumps.

Return type
dict of *PumpType*

2.1.2.1.1.11 pump

`class PumpType`(*root*)

A PumpType object contains all of the methods and variables to control and monitor a pump inside an R2Module.

This object can set and get the values of their attributes members but cannot subscribe them. In order to subscribe, the variable id must to be getted from inside the member object.

In example, to subscribe the flow rate, the variable id has to be getted from :

PumpType.flowRate.getFlowRateNode().

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

It is the id of the pump object in the OPC UA Address Space.

Type
opcua.Node

`flowRate`

It is the id of flow rate variable in the OPC UA Address Space.

Type
opcua.Node

`srValveState`

It is the id of the state of the valve in the OPC UA Address Space.

Type
opcua.Node

`ilValveState`

It is the id of the state of the valve in the OPC UA Address Space.

Type
opcua.Node

`getFlowRate()`

This function gives the current pump flow rate.

Returns
It is the pump flow rate in ul/min.

Return type
integer

`getFlowRateNode()`

This method returns the id of the flow rate variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the flowRate variable in the OPC UA Address Space.

Return type
opcua.Node

`getILValveState()`

This gives the state of the inject or load selection valve.

Returns
Depends on which is teh valve, true means liquid from Inject to Load. Else, the liquid moves in the another way.

Return type
bool

`getILValveStateNode()`

This method returns the id of the ILValveState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ILValveState variable in the OPC UA Address Space.

Return type
opcua.Node

`getSRValveState()`

This gives the state of the Solvent or Reagent selection valve

Returns
True means liquid from Reagent Else, the liquid moves in the another way.

Return type
bool

`getSRValveStateNode()`

This method returns the id of the SRValveState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the SRValveState variable in the OPC UA Address Space.

Return type
opcua.Node

`setFlowRate`(*flowRate*)

It sets the flow rate to the pump. When flow rate is setted in 0, the pump is turned off. If the pump is setted in other value than 0, the pump is immediatly turned on. The flow rate must be set in a suitable value for the connected pump in R2. In other case, the flow rate will not be setted in the machine.

Parameters
flowRate (*integer*) – It is the pump flow rate in ul/min.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setILValveState`(*state*)

It sets the state of the inject or load selection valve.

Parameters
state (*bool*) – Depends on which is the valve, true means fluid from Inject to Load. Else, the fluid moves in the another way.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setSRValveState`(*state*)

It sets the state of the Solvent or Reagent selection valve.

Parameters
state (*bool*) – Depends on which is the valve, true means fluid from Reagent to Solvent Else, the fluid moves in the another way.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.12 r2

`class R2Type(`*root*`)`

A R2Type object contains all of the methods and object to monitor and control a R2 Module in the R-Series Controller.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

It is the R2 object id in the OPC UA Address Space.

Type
opcua.Node

`maxPressureLimit`

It is the id of the maxPressureLimit variable in the OPC UA Address Space. This contains information about the maximum pressure limit allowed for the pump.

Type
opcua.Node

`pressureLimit`

It is the pressureLimit variable id in the OPC UA Address Space. This is the current pressure limit set point in system.

Type
opcua.Node

`pumpsPressure`

It is the measured pumpsPressure variable id in the OPC UA Address Space.

Type
opcua.Node

`pumpsPerformance`

It is the id of PumpsPerformance variable in the OPC UA Address Space.

Type
opcua.Node

`pump`

It is a dictionary of pumps. Each dictionary entry contains the methods, object and variable necesary to control and monitor a pump. The pumps are alphabetic labeled, so the key of the dict are [‘A’, ‘B’].

Type
dict of *PumpType*

`getMaxPressureLimit()`

It gives the maximum pressure limit setted in the module.

Returns
It is the pressute limit in mbar.

Return type
float

`getMaxPressureLimitNode()`

This method returns the id of the max pressure limit variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the maxPressureLimit variable in the OPC UA Address Space.

Return type
opcua.Node

`getPressureLimit()`

It gives the pressure limit value.

Returns
It is the actual pressure limit.

Return type
float

`getPressureLimitNode()`

This method returns the id of the pressure limit variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the pressureLimit variable in the OPC UA Address Space.

Return type
opcua.Node

`getPumpA()`

It returns the pump A object.

Returns
It is the the object that contains the methods, objects and variables to control and monitor the Pump A.

Return type
PumpType

`getPumpB()`

It gives the pump B object.

Returns
It is the object that contains the methods, objects and variables to control and monitor the Pump B.

Return type
PumpType

`getPumpsPerformance()`

It returns the performance of all of the pumps.

Returns
It is a list with the performance of each pump. [0] is the pumpA performance [1] is the pumpB performance

Return type
list

`getPumpsPerformanceNode()`

This method returns the id of the pumps performance variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of PumpsPerformance variable in the OPC UA Address Space.

Return type
opcua.Node

`getPumpsPressures()`

It returns the pressures of the pumps and the R2 Module.

Returns
It is a list with the pressure each pump and the system. list()[0] is the module Pressure list()[1] is the pumpA pressure list()[2] is the pumpB pressure

Return type
list

`getPumpsPressuresNode()`

This method returns the id of the pumps pressures variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the PumpsPressures variable in the OPC UA Address Space.

Return type
opcua.Node

`setMaxPressureLimit(`*pressure*`)`

Sets the maximum pressure admited in the pumps by the system

Parameters
pressure (*float*) – It is the new maximum pressure limit for all of the pumps.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setPressureLimit`(*pressure*)

It sets the pressure limit value.

Parameters
`pressure` (*float*) – It is the new pressure to set.

Returns
It is true if the value set was successfully aplied.

Return type
bool

2.1.2.1.1.13 r4

`class R4Type`(*root*)

The R4Type is an object that contains all the methods, variables and objects that contains the information to control and monitor the R4Modules.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

It is the id of the R4 object in the OPC UA Address Space.

Type
opcua.Node

`reactor`

It is a dictionary that contains all of the reactors. Each R4 has 4 reactors. The key of each reactor is a number from 1 to 4.

Type
dict of *ReactorType*

`getReactors`()

It return a dictionary with all the reactors in R4 module.

Returns
This has all the reactors.

Return type
dict

2.1.2.1.1.14 reactor

`class ReactorType`(*root*)

This ReactorType class represent each reactor that has present in the R-Series Controller. This contains all the methods and variables with information about one reactor.

Parameters
`root` (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

It is the id of the Reactor in the OPC UA Address Space.

Type
opcua.Node

`temperature`

This is the id of the temperature measurement in the OPC UA Address Space.

Type
opcua.Node

`power`

This is the id of the power variable in the OPC UA Address Space.

Type
opcua.Node

`reactorType`

It is the id of the reactor type variable in the OPC UA Address Space.

Type
opcua.Node

`class ReactorTypeEnum`

The posible types of reactor founded HEATED_NOR COOLED_TUBE COOLED_COLUMN MIDRANGE_TUBE UV150 NIK_BATCH CHIP_REACTOR EC_REACTOR

`getPower()`

It returns the power of the Reactor.

Returns
The reactor power.

Return type
float

`getPowerNode()`

This method returns the id of the reactor's power. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the power variable in the OPC UA Address Space.

Return type
opcua.Node

`getReactorType()`

It returns the Reactor Type in an integer. This is defined by self.ReactorTypeEnum

Returns
It is the reactor type.

Return type
ReactorTypeEnum

`getReactorTypeNode()`

This method returns the id of the Reactor Type variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the reactor type variable in the OPC UA Address Space.

Return type
opcua.Node

`getTemperature()`

It gets the last temperature measuremt of the reactor.

Returns
This is the measured temperature. A -1000 value means the reactor is turned off.

Return type
float

`getTemperatureLimits()`

It returns the temperature limits for the set point.

Returns
It is a list that contains the low and high limits.

Return type
list

`getTemperatureLimitsNode()`

This method returns the id of the tempearture limits. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the Temperature Limits variable in the OPC UA Address Space.

Return type
opcua.Node

`getTemperatureNode()`

This method returns the id of the Temperature variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the Temperature variable in the OPC UA Address Space.

Return type
opcua.Node

`setTemperature`(*temp*)

It sets the temperature set point. This temperature will be reached when the StartManualControl method is run.

The temperature will be set into the limits. If a lower or higher target temperature is send, the closer limit will be set

Parameters
`temp` (*`integer`*) – This is the new temperature set point.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.15 sf10

`class SF10Type`(*root*)

The SF10Type class implements the low level functions for controlling the components of the R-Series Machine.

This class allows the user to write and run scripts for custom automation routines.

Parameters
`root` (*`opcua.Node`*) – It is the parent node in the OPC UA Address Space.

`root`

This is the parent node in the OPC UA Address Space.

Type
opcua.Node

`maxPressureLimit`

This is the MaxPressureLimit node in the OPC UA Address Space.

Type
opcua.common.node.Node

`pressureLimit`

This is the PressureLimit node in the OPC UA Address Space.

Type
opcua.common.node.Node

`bprRegulationPressure`

This is the BPRRegulationPression node in the OPC UA Address Space.

`pumpPressure`

This is the PumpPressure node in the OPC UA Address Space.

Type
opcua.common.node.Node

`gasFlowRate`

This is the GasFlowRate node in the OPC UA Address Space.

Type
opcua.common.node.Node

`flowrate`

This is the FlowRate node in the OPC UA Address Space.

Type
opcua.common.node.Node

`srValveState`

This is the SRValveState node in the OPC UA Address Space.

Type
opcua.common.node.Node

`getBPRRegulationPressionNode()`

This method returns the id of the SF10 BPRRegulationPression variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 BPRRegulationPression variable in the OPC UA Address Space.

Return type
opcua.Node

`getBPRRegulationPressure()`

It gives the current pressure set for Back Pressure Regulation mode.

Returns
It is the current pressure set.

Return type
float

`getFlowRate()`

It gives the current flow rate for constant flow rate operation mode.

Returns
It is the current flow rate.

Return type
float

`getFlowRateNode()`

This method returns the id of the FlowRate variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 FlowRate variable in the OPC UA Address Space.

Return type
opcua.Node

`getGasFlowRate()`

It gives the current flow rate for gas operation mode

Returns
It is the current flow rate.

Return type
float

`getGasFlowRateNode()`

This method returns the id of the SF10 GasFlowRate variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 GasFlowRate variable in the OPC UA Address Space.

Return type
opcua.Node

`getMaxPressureLimit()`

It gives the actual maximum pressure limit.

Returns
It is the actual maximum pressure limit.

Return type
float

`getMaxPressureLimitNode()`

This method returns the id of the SF10 MaxPressureLimit variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 MaxPressureLimit variable in the OPC UA Address Space.

Return type
opcua.Node

`getOperationMode()`

It gives the current operation mode set on SF10 pump.

Returns
It is the current operation mode 0 means Constant flow rate mode 1 means Pressure Regulation mode 2 means volume dose regulation mode 3 means ramp flowrate mode 4 means gas mode 5 means Oscillation Mode

Return type
integer

`getOscillation()`

It gives the parameters set for the SF10 Oscillation mode.

Returns
It returns a list with the speedul and displacement parameters.

Return type
list

`getPressureLimit()`

It gives the current pressure limit set on the device.

Returns
It is the current pressure limit

Return type
float

`getPressureLimitNode()`

This method returns the id of the SF10 PressureLimit variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 PressureLimit variable in the OPC UA Address Space.

Return type
opcua.Node

`getPumpPressure()`

It gives the current pressure on the pump

Returns
It is the pump pressure

Return type
float

`getPumpPressureNode()`

This method returns the id of the SF10 Pumppressure variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 PumpPressure variable in the OPC UA Address Space.

Return type
opcua.Node

getSRValveState()

It gives the current state for the solvent-reagent selection valve.

Returns
This is the current valve state. True means the pump delivers reagent. False means the pump delivers solvent.

Return type
bool

getSRValveStateNode()

This method returns the id of the SRValveState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the SF10 SRValveState variable in the OPC UA Address Space.

Return type
opcua.Node

setBPRRegulationPressure(*newPressure*)

It sets the pressure for the Back Pressure Regulation mode.

Parameters
newPressure (*float*) – It is the new pressure for the regulator.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

setFlowRate(*newFlowRate*)

It sets the flow rate for constant flow rate mode.

Parameters
newFlowRate (*float*) – It is the new flow rate.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

setGasFlowRate(*newFlowRate*)

It sets the flow rate for gas mode operation

Parameters
newFlowRate (*float*) – It is the new flow rate

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setMaxPressureLimit`(*newMaxPressureLimit*)

It sets a new maximum pressure limit.

Parameters
`newMaxPressureLimit` (*`float`*) – It is the new maximum pressure limit.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setOperationMode`(*newMode*)

It sets the Pump Mode to Pressure Regulation. You can switch between Flow, Pressure Regulation (REG), DOSE, RAMP, GAS.

Parameters
`newMode` (*`integer`*) – It is the new operation mode. 0 means Constant flow rate mode 1 means Pressure Regulation mode 2 means volume dose regulation mode 3 means ramp flowrate mode 4 means gas mode 5 means Oscillation Mode

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setOscillation`(*speedul*, *displacement*)

It configures the oscillation parameters for SF10 scillation mode operation.

Parameters

- **`speedul`** (*`integer`*) – It is the speed at which the pump oscillates
- **`displacement`** (*`integer`*) – It is the amount of volume that is oscillated back and forth

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setPressureLimit`(*newPressureLimit*)

It sets the pressure limit for SF10 operation.

Parameters
`newPressureLimit` (*`float`*) – It is the new pressure limit

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setSRValveState`(*newState*)

It sets the state of the solvent or reagent selection valve.

Parameters
`newState` (*`bool`*) – This is the new valve state. True means the pump will deliver reagent. False means the pump will deliver solvent.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.16 syringe

`class SyringePumpType(`*root*`)`

SyringePumpType class is an object to monitor and control external syringe pumps.

Parameters
`root` (*opcua.Node*) – It is the id of this node in the OPC UA Address Space.

`root`

It is the OPC UA Address Space information about this object.

Type
opcua.Node

`valveState`

It is the id of the ValveState variable in the OPC UA Address Space.

Type
opcua.Node

`position`

It is the id of the Position variable in the OPC UA Address Space.

Type
opcua.Node

`flowRate`

It is the id of the FlowRate variable in the OPC UA Address Space.

Type
opcua.Node

`getErrorState()`

It reports if there was an error with the module.

Returns
It is True if there was an error. Otherwise, it is False.

Return type
bool

`getErrorStateNode()`

This method returns the id of the ErrorState variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ErrorState variable in the OPC UA Address Space.

Return type
opcua.Node

`getFlowRate()`

This function gives the current pump flow rate.

Returns
It is the pump flow rate in ul/min.

Return type
integer

`getFlowRateNode()`

This method returns the id of the flow rate variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the flowRate variable in the OPC UA Address Space.

Return type
opcua.Node

`getPosition()`

It gives the syringe position

Returns
It is the syringe position

Return type
integer

`getPositionNode()`

This method returns the id of the Position variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the Position variable in the OPC UA Address Space.

Return type
opcua.Node

`getValveState()`

This gives the state of the valve

Returns
Depends on which is teh valve, true means liquid from Reagent to Solvent, Collect to Waste or Inject to Load. Else, the liquid moves in the another way.

Return type
bool

`getValveStateNode()`

This method returns the id of the valve state variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the ValveState variable in the OPC UA Address Space.

Return type
opcua.Node

`setFlowRate`(*flowRate*)

It sets the flow rate to the pump. When flow rate is setted in 0, the pump is turned off. If the pump is setted in other value than 0, the pump is immediatly turned on. The flow rate must be set in a suitable value for the connected pump in R2. In other case, the flow rate will not be setted in the machine.

Parameters
flowRate (*integer*) – It is the pump flow rate in ul/min.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`setPosition`(*newPosition*)

It sets a new position for the syringe

Parameters
newPosition (*integer*) – It is hte new position to set.

`setValveState`(*state*)

It sets the state of the valve.

Parameters
state (*bool*) – Depends on which is the valve, true means fluid from Reagent to Solvent, Collect to Waste or Inject to Load. Else, the fluid moves in the another way.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.17 uv150

class `UV150Type`(*root*)

The UV150Type class instantiates UV150 objects. This objects allows the user to turn the UV lamp on and modify its power.

Parameters
root (*opcua.Node*) – It is the parent node in the OPC UA Address Space.

`root`

This is the id of the UV150 in the OPC UA Address Space.

Type
opcua.Node

`lampPower`

This is the id of the lamp power in the OPC UA Address Space.

Type
opcua.Node

`getLampPower()`

It gets the UV150 lamp power

Returns
This returns an integer that is lamp power. The lamp power is a percentage number.

Return type
integer

`getLampPowerNode()`

This method returns the id of the Lamp Power. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the Lamp Power in the OPC UA Address Space. This id allows to subscribe the variable to monitor it.

Return type
opcua.Node

`setLampPower(`*power*`)`

It sets the UV150 lamp power

Parameters
power (*integer.*) – This is the new lamp power. The lamp power is a percentage number, but only can be between 50 and 100. To turn the lamp off, power needs to be 0.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.1.18 vbfr

`class VBFRType(`*root*`)`

A VBFRType Object contains all the variables and methods to know the measurements and information about the Variable Bed Flow Reactor

Parameters
`root` (*opcua.Node*) – It is the id of the VBFR object in the OPC UA Address Space.

`root`

It is the id of the VBFR object in the OPC UA Address Space.

Type
opcua.Node

`position`

It is the id of the VBFR Piston Position variable in the OPC UA Address Space.

Type
opcua.Node

`dPress`

It is the id of the VBFR Differential Pressure variable in the OPC UA Address Space.

Type
opcua.Node

`volume`

It is the id of the VBFR Volume variable in the OPC UA Address Space.

Type
opcua.Node

`getDifferentialPressure()`

It gives the value of the VBFR differential pressure in mBar.

Returns
This is the differential pressure.

Return type
integer

`getDifferentialPressureNode()`

This method returns the id of the VBFR Diffetential Pressur variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the VBFR Diferential Pressure variable in the OPC UA Address Space.

Return type
opcua.Node

`getPosition()`

It gives the Piston Position in mm.

Returns
It is the position of the piston.

Return type
float

`getPositionNode()`

This method returns the id of the VBFR Piston Position variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the VBFR Piston Position variable in the OPC UA Address Space.

Return type
opcua.Node

`getVolume()`

It gives the inyected or loaded volume from the VBFR in mm^3

Returns
It is the VBFR volume.

Return type
float

`getVolumeNode()`

This method returns the id of the VBFR Volume variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the VBFR Volume variable in the OPC UA Address Space.

Return type
opcua.Node

2.1.2.1.2 experiment package

2.1.2.1.2.1 experiment

`class ExperimentType(`*root*`)`

ExperimentType class is an object that contains the high level commands to work with the R-Series machine.

It has the experiment settings information and the reaction manager methods to command the list of reactions that are running in the system.

Also, it gives statistics about the experiments.

Parameters
`root` (*opcua.Node*) – It is the id of this node in the OPC UA Address Space.

`root`

It is the id of the experiment object in the OPC UA Address Space.

Type
opcua.Node

`experimentSetup`

It is the id of the experiment setup object in the OPC UA Address Space.

Type
ExperimentSetupType

`reactionManager`

It is the id of the reaction manager object in the OPC UA Address Space.

Type
ReactionManagerType

`statistics`

It is the id of the experiment statistics in the OPC UA Address Space.

Type
ExperimentStatisticType

`getExperimentSetup()`

It returns the experiment setup object.

Returns
It is the setup object.

Return type
ExperimentSetupType

`getReactionManager()`

It gives the Reaction Manager object.

Returns
It is the Reaction Manager object.

Return type
ReactionManagerType

`getStatistics()`

It gives the Experiment Statistics object

Returns
It is the Statistics object.

Return type
ExperimentStatisticsType

2.1.2.1.2.2 experimentSetup

`class ExperimentSetupType`(*root*)

The ExperimentSetupType class contain the methods that allow the user to save and restore the System settings.

Parameters
`root` (*opcua.Node*) – It is the experiment setup object id in the OPC UA Address Space.

`root`

It is the id of the experimentSetup object in the OPC UA Address Space.

Type
opcua.Node

`getExperiment`(*pathToSave=None*)

It gives the experiment data. The data returned could be saved in a file and then it could be send by loadExperiment.

Returns
This is the experiment data.

Return type
JSON

`getSystemSettings()`

It returns the settings of all the components presetns in the loaded experiment on the R-Series Controller.

Returns
These are the system settings.

Return type
JSON

`loadExperiment`(*experiment*)

It loads an experiment into the system. The experiment data to send is getted by getExperiment method.

Parameters
`experiment` (*JSON*) – It is the experiment to load.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`loadSystemSettings`(*settings*)

It loads all the system settings in the current experiment on the R-Series Controller.

Parameters
`settings` (*JSON*) – It is the system settings.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

2.1.2.1.2.3 experimentStatistics

class ExperimentStatisticType(*root*)

An ExperimentStatisticsType object contains the methods to obtain the statistics information about the loaded experiment.

Parameters
root (*opcua.Node*) – It is the id of the ExperimentStatistics in the OPC UA Address Space.

root

It is the id of the ExperimentStatistics in the OPC UA Address Space.

Type
opcua.Node

getExperimentGraphs(*filename=None*)

It gives the graphs from the R-Series Controller. This graph could be stored in a file or could be returned as a list.

Parameters
filename (*string, optional*) – It is the path where the file will be stored. The default is None.

Returns

rspList – It is the list with all graphs.

If a filename is given, the response of this method is None and the graphs will be stored in a file.

Return type
list of list of string or None.

getReactionStatistics(*reaction*)

It gives the temporal statistic of a given reaction.

Parameters
reaction (*integer*) – It is the index of the reacction in ReactionList

Returns
These are the statistics of the selected reaction. If the reaction is not in the RactionList, it returns None.

Return type
JSON or None.

2.1.2.1.2.4 reactionManager

class ReactionManagerType(*root*)

The Reaction Manager class allows the user manage the Reaction List.

Reactions can be added, deleted or desabled from the Reaction List. Then, this list could be started or stoped with the methods provides from an object of this class.

Parameters
root (*opcua.Node*) – It is the Reaction Manager's id in the OPC UA Address Space.

`root`

It is the Reaction Manager's id in the OPC UA Address Space.

Type
opcua.Node

`reactionStatus`

It is the ReactionStatus variable id in the OPC UA Address Space.

Type
opcua.Node

`addReactionToList`(*reaction*)

This method add one reaction to the current Reaction List.

Parameters
`reaction` (*String*) – This is the reaction to add. The reaction must be the followings field, in a CSV format.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`deleteReaction`(*reaction*)

It deletes a given reaction.

Parameters
`reaction` (*integer*) – This is the index of the reaction to be removed of the Reaction List.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`disableReaction`(*reaction*)

This method disables a given reaction. When a reaction is disabled, it will not be runned when startReaction is called.

Parameters
`reaction` (*integer*) – This is the index of the reaction to be disabled in the Reaction List.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`enableReaction`(*reaction*)

This method enables a given reaction. When a reaction is enabled, it will be runned when startReaction is called.

Parameters
`reaction` (*integer*) – This is the index of the reaction to be enabled in the Reaction List.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`getCurrentReactionIndex()`

It returns the position of the current reaction in the reaction list.

Returns
It is the index of the reaction.

Return type
integer

`getCurrentReactionStatus()`

It gives the status of the current reaction. It could be one of the following states:

IDLE = 0

NOT_RUNNING = 1

WAIT_RACK_CHANGE = 2

SET_PREHEATING = 3

HEATING = 4

WAIT4PRE_WASH = 5

PUMP_DEAD_VOLUME = 6

WAIT4DEAD_VOLUME = 7

WAIT4SAMPLE_LOOP_LOADS = 8

START_REACTION = 9

RUNNING_REACTION = 10

WAIT4COLLECTION = 11

POST_WASHING, = 12

WAIT4POST_WASH, = 13

FINAL_COLLECTION, = 14

WAIT4FINAL_COLLECTION, = 15

CHECK4REACTIONS, = 16

CLEANING, = 17

WAIT4CLEANING, = 18

COOLING, = 19

WAIT4COOLING, = 20

STOP_REACTION = 21

Returns
It is the current reaction status.

Return type
integer

`getEnableReactionStatus(`*index*`)`

It reports if a given reaction is enabled to be automatically excecuted or not.

Parameters
`index` (*`integer`*) – It is the 0-based position of the desired reaction in the reaction list

Returns
It is True when the reaction is enabled.

Return type
bool

`getReactionList()`

It gives the reaction list of the actual experiment file loaded. It needs to be an experiment set and loaded a reaction list or created before to be meaningful. The reaction list is in CSV Format. It is received the string separator and the decimal point.

Returns
It is the reaction list in CSV format. The first line is the table's header. The following lines are the fields. The lines are delimited by 'n'.

Return type
String

`getReactionParameters(`*position=0*`)`

It returns all the parameteres of a given reaction in the Reaction List.

Parameters
position (*integer*) – It is the index of the reaction in the reaciton list

Returns
These are the reaction list parameters, including CSV string separators, CSV deciman point, the information about all of the pumps and reactors of each reaction in the Reaction List and the amount of reactions. It could be direct loaded to a JSON object.

Return type
String

`getReactionStatusNode()`

This method returns the id of the reaction status variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
This is the id of the reaction status variable in the OPC UA Address Space.

Return type
opcua.Node

`loadReactionList(`*reactionList*`)`

loadReactionList replaces the actual reaction list with a new one. This method needs an experiment set before its execution.

Parameters
reactionList (*String*) – It is a CSV format string with the reaction list to load. It has to be a valid Vapourtec RSeries reaction table.

Return type
None.

`resetReactionList()`

This method deletes all of the reactions in the current Reaction List.

Return type
None.

`setReactionPosition`(*curPos*, *newPos*)

setReactionPosition changes the position of a Reaction in the Reaction List.

Parameters

- **`curPos`** (*`integer`*) – This is the current position of the reaction.
- **`newPos`** (*`integer`*) – This is the position where the reaction will be placed.

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`startReaction`()

startReaction sends a command to run all of the enabled reactions in the current Reaction List

Returns
It is true when the method was applied successfully to the system. Otherwise, it is false.

Return type
bool

`stopReaction`()

stopReaction sends a command to stop the current reaction execution.

Return type
None.

2.1.2.1.3 identification package

2.1.2.1.3.1 identification

`class IdentificationType`(*root*)

IdentificationType is an object that gives the machine information. It has a opc Node that contains its opc address. Also has two components that gives the software and hardware information. The data could be accesed by itself, by calling this class methods or by calling each component methods.

Parameters
`root` (*`opcua.Node`*) – It is the id of this node in the OPC UA Address Space.

`root`

This has the OPC-UA information.

Type
opcua.Node

`equipmentIdentification`

It contains the hardware information.

Type
EquipmentIdentificationType

`softwareIdentification`

It has the software data.

Type

SoftwareIdentificationType

`getEquipmentIdentifier()`

This function gives the name of the machine.

Returns

This machine is an RSeries Flow Chemistry.

Return type

String

`getEquipmentManufacturer()`

This function returns Vapourtec. This is the buider company of the machine.

Returns

Vapourtec design and made the RSeries machine.

Return type

String

`getHardwareRevision()`

This funtion gives the hardware version.

Returns

This is the hardware version. The format of the harware revision is vXX.YY.ZZ, where :

- v is an arbitrary character
- XX is a major change
- YY is a minor change
- ZZ is a patch

Return type

String

`getSoftwareDeveloper()`

This function gives the name of the software developer company.

Returns

Developer name.

Return type

String

`getSoftwareIdentifier()`

This function gives the human readable name of the software at the controller.

Returns

This is the software name identifier.

Return type

String

`getSoftwareVersion()`

This function gives the version of the software at the controller.

Returns

This is the software version. The format of the software version is vXX.YY.ZZ, where:

- v is an arbitrary identifier
- XX is a major change.
- YY is a minor change.
- ZZ is a patch.

Return type
String

2.1.2.1.3.2 equipmentIdentification

`class EquipmentIdentificationType(`*root*`)`

An EquipmentIdentificationType object has the hardware identification information.

Parameters
`root` (*opcua.Node*) – It is the id of this node in the OPC UA Address Space.

`root`

It is the OPC-UA Address Space of this object

Type
opcua.Node

`harwareRevision`

It is a OPC-UA variable. It has the hardware revision data in a String value.

Type
opcua.Node

`equipmentIdentifier`

It is a OPC-UA variable. It has the equipment identifier data in a String value.

Type
opcua.Node

`equipmentManufacturer`

It is a OPC-UA variable. It contains the equpiment manufacturer data in a String value.

Type
opcua.Node

`equipmentIdentifier()`

This gives the name of the machine.

Returns
This machine is an RSeries Flow Chemistry.

Return type
String

`equipmentManufacturer()`

This function returns Vapourtec. This is the buider company of the machine.

Returns
Vapourtec design and made the RSeries machine.

Return type
String

`hardwareRevision()`

This gives the hardware version.

Returns

This is the hardware version. The format of the harware revision is vXX.YY.ZZ, where :

- v is an arbitrary character
- XX is a major change
- YY is a minor change
- ZZ is a patch

Return type
String

2.1.2.1.3.3 softwareIdentification

`class SoftwareIdentificationType(`*root*`)`

The SoftwareIdentificationType class contains information about the software development.

Parameters
`root` (*opcua.Node*) – It is the id of this node in the OPC UA Address Space.

`root`

It is the OPC-UA Address Space information

Type
opcua.Node

`softwareRevision`

It is a OPC-UA variable. This contains the software revision version in a String value.

Type
opcua.Node

`softwareIdentifier`

It is a OPC-UA varialbe. This variable contains the software identifier in a String value.

Type
opcua.Node

`softwareDeveloper`

It is a OPC-UA variable. It contains the software developer in a String value.

Type
opcua.Node

`softwareDeveloper()`

This gives the name of the software developer company.

Returns
Developer name

Return type
String

`softwareIdentifier()`

This gives a human readable name of the software at the controller.

Returns
This is the software name identifier.

Return type
String

`softwareVersion()`

This gives the version of the software at the controller.

Returns
This is the software version. The format of the software version is vXX.YY.ZZ, where:

- v is an arbitrary identifier
- XX is a major change.
- YY is a minor change.
- ZZ is a patch.

Return type
String

2.1.2.1.4 monitoring package

2.1.2.1.4.1 monitoring

`class MonitoringType(`*root*`)`

MonitoringType class is an object to get information about the system status.

Parameters
`root` (*`opcua.Node`*) – It is the id of this node in the OPC UA Address Space.

`root`

It is the OPC-UA Address Space information about this object.

Type
opcua.Node

`rSeriesStaus`

It is an object to get the status of the R-Series machine.

Type
StatusType

`getRSeriesStatus()`

It gives the RSeries status object in this machine.

Returns
It is the status node. This node will give access to call methods in StatusType.

Return type
StatusType

2.1.2.1.5 notification package

2.1.2.1.5.1 notification

class NotificationType(*root*)

A NotificationType Object contains variables and methods related to equipment alerts.

Parameters
: **root** (*opcua.Node*) – It is the NotificationType's id in the OPC UA Address Space.

root

It is the id of this object in the OPC UA Address Space.

Type
: opcua.Node

getEquipmentAlerts()

It gives the equipment status.

The status could be one of the followings:

SYS_OK = 0

SYS_OVERPRESSURE_TRIP = 1

PUMPA_OVERPRESSURE_TRIP = 2

PUMPB_OVERPRESSURE_TRIP = 3

UNDERPRESSURE_TRIP = 4

PUMPA_UNDERPRESSURE_TRIP = 5

PUMPB_UNDERPRESSURE_TRIP = 6

PUMPA_LOGIC_ERROR = 7

PUMPB_LOGIC_ERROR = 8

PUMPC_OVERPRESSURE_TRIP = 9

PUMPD_OVERPRESSURE_TRIP = 10

PUMPC_UNDERPRESSURE_TRIP = 11

PUMPD_UNDERPRESSURE_TRIP = 12

PUMPC_LOGIC_ERROR = 13

PUMPD_LOGIC_ERROR = 14

Returns
: It is the sttatus of the equipment

Return type
: integer

getEquipmentAlertsNode()

This method returns the id of the equipment alerts variable. This id allows the user to subscribe the variable to a handler object.

To subscribe a variable, the handler class has to be defined and implement a datachange_notification(self, node, value, data) function. This give the user the capability of monitoring the variable and makes a custom control of it.

Example of this will be found in readingDataExample.py

Returns
It is the id of the flowRate variable in the OPC UA Address Space.

Return type
opcua.Node

2.1.2.1.6 base

`class BaseNode`

The BaseNode class has the common methods with the tasks that every device has to perform in order to get the right nodes on the server side.

`class NodeInfo`(*variableName*, *TagName*, *prototypeClass*)

This is an auxiliar class that allows groups variables on a standard python container.

2.1.2.1.7 factories

`class NodeFactory`

The NodeFactory is a class of object that is provides the client the right type of member of a device class

`getNode`(*node: Node*, *prototype: object = None*) → object | Node

The getNode method gives an object of proto instantiated with the node argument.

If proto is None, the node is returned.

Parameters

- **node** (*Node*) – This is the parameter of the prototype given.
- **prototype** (*object, optional*) – This is the prototype to instantiate a new object, by default None

Returns
This is the instantiated object of prototype. If prototype is None, the node will be returned.

Return type
object | Node

2.1.2.1.8 reaction

`class Reaction`

This class represents a Reaction in a Reaction List. Allows the user to load a Reaction and make copies with different configurations.

`class Collection`

A collection is the way how reaction stores the final produt. It has an state, a diverterTime, a collectionTime, a number of vials, a total volume and a volume for each vial. A Reaction has information of one Collection.

`class Pump`

A Pump has Flowrate, Volume Ratio, Concentration Ratio, Quantity, Advance Retard and Regent Concentration. It depends of the kind of Pump the experiment has, the valid changes of Volume or Conentration Ratio. The time which the pump is working is a function of flowrate and quantity selected. A Reactor has 8 Pumps, stored in a list numbered from 1 to 8.

class Reactor

A reactor has a tempreature and a residence time. Each Reaction has 8 reactors, numbered from 1 to 8.

class UVReactor

An UVReactor is a kind of Reactor which has an extra UV power.

class Wash

A wash is a time where a Pump is cleaning its system. A wash has a flowrate and a volume, which defines the wash time. A Reaction has two different times for wash before and after the Reaction is run.

2.1.3 utils

get_interest_child(*node*, *target*)

Return the node's child that matchs with target string

Parameters

- **node** (*opcua.Node()*) – Node that will be filtered
- **target** (*string*) – string to match filtering the node

Returns

Node which browse name match with the string

Return type

opcua.Node()

infoAllChildren(*node*)

It prints all the browse name of the children of a given node

Parameters

node (*opcua.Node*) – It is the parent node whose children will be printed.

Return type

None.

toLetter(*num*)

Convert an integer to equivalent capitalized alphabet letter.

Parameters

num (*integer*) – It is the number to convert.

Returns

It is the capitalized alphabet letter

Return type

char

PYTHON MODULE INDEX

INDEX