

Application Note for using node-red-contrib-opcua

Abstract:

This document is about a node “node-red-contrib-opcua”, which allows to fetch OPC UA data from a server. The examples are based on the version v0.2.85. This Application Note is focused on the client-side of this node. The user should have knowledge about Node-RED and “function” nodes and should possess basic skills in JavaScript.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-WL2000-AC	1334950000	FW: 1.13.0
2	UC20-WL2000-IOT	1334990000	FW: 1.13.0
3	IOT-GW30	2682620000	FW: 1.13.0
4	IOT-GW30-4G-EU	2682630000	FW: 1.13.0

Software reference

No.	Software name	Article No.	Software version
1	Node-RED	-	-
2	node-red-contrib-opcua	-	v.0.2.85

File reference

No.	Name	Description	Version
1	AN0056-UC20-WL2000 Use of node-red-contrib-opcua.zip	The file contains a .json file related to this document	-

Contact

Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
www.weidmueller.com

For any further support please contact your
local sales representative:
<https://www.weidmueller.com/countries>

Content

1	Warning and Disclaimer.....	4
2	Node: node-red-contrib-opcua	5
2.1	Overview of the node package	6
2.2	Basic functionality.....	11
2.2.1.	Read.....	11
2.2.2.	Write.....	15
2.2.3.	Subscribe	19
2.2.4.	Info	21
2.2.5.	Read multiple.....	23
2.2.6.	Monitor	25

1 Warning and Disclaimer

Warning

Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer

This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use.

Note

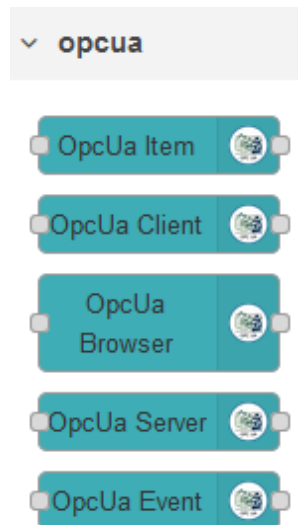
The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

Security notes

In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

2 Node: node-red-contrib-opcua

The npm package node-red-contrib-opcua contains 5 nodes. The two most important nodes for the next sections are **OpcUa Item** and **OpcUa Client**.

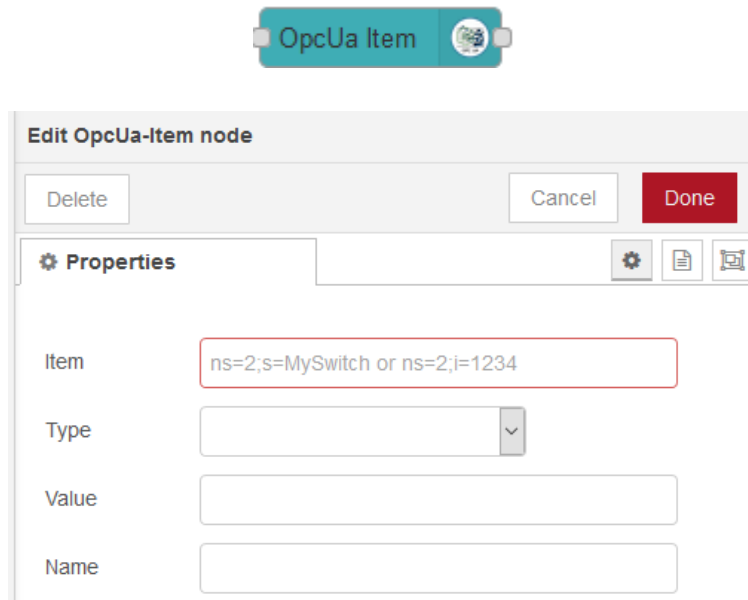


On the u-control web it is **not** recommended to use the browse functionality at all (node-red-contrib-opcua@v0.2.85). This means no usage of **OpcUa Browser** and no usage of the **OpcUa Client** with browse parameters as well.

To get the node ids of interest, third party OPC UA client applications could be used. A free OPC UA client, which is available after the registration, is UaExpert. The application is available on the official [website](#).

2.1 Overview of the node package

A **OpcUA Item** node could be used as comfort block. As input a simple trigger msg without content is needed. The node must be configured according to the figure below. Items need to be connected to an **OpcUA Client** node.



The screenshot shows the 'Edit OpCua-Item node' configuration window. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below these is a 'Properties' section with a gear icon and three sub-icons. The 'Item' field is a text box containing the text 'ns=2;s=MySwitch or ns=2;i=1234'. The 'Type' field is a dropdown menu. The 'Value' and 'Name' fields are empty text boxes.

The configuration for read access needs the properties:

- Item
- Type
- (optionally a name)

The configuration for write access needs the properties:

- Item
- Type
- Value
- (optionally a name)

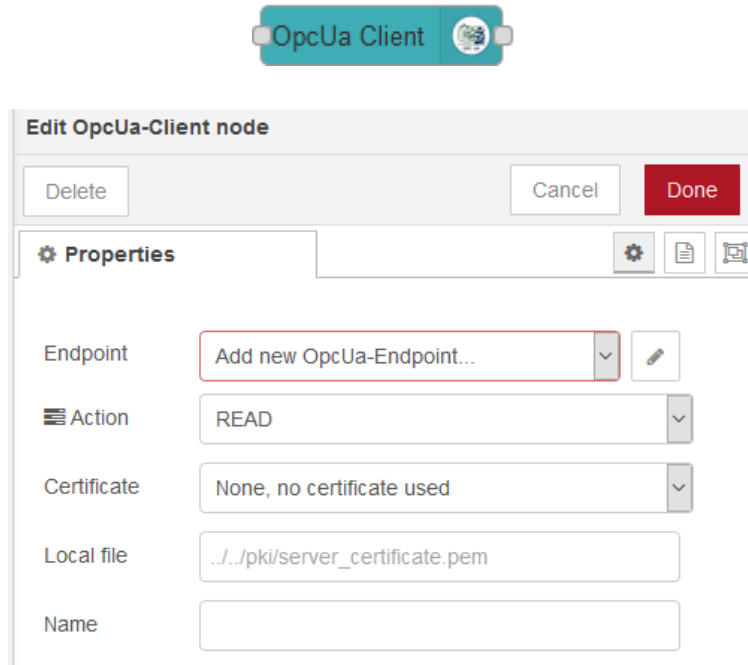
Item is the node id e.g. ns=2; s=MySwitch.

Type is a dropdown list which helps to select the correct supported data type¹.

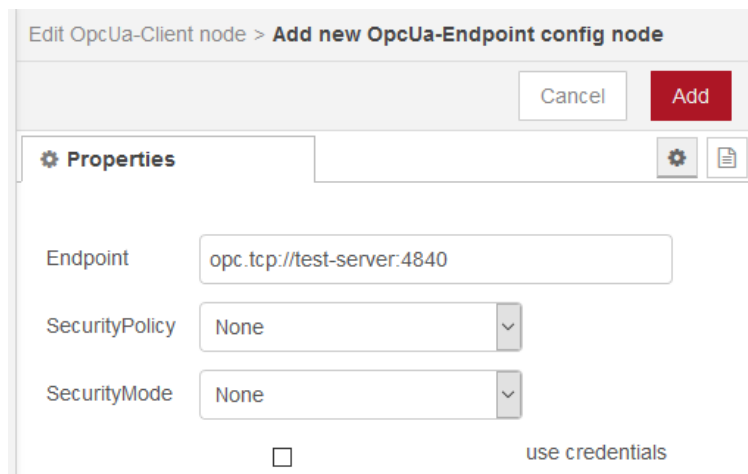
Value is the value to be written to the data point.

¹ Supported types are: Int8, Int16, Int32, UInt8, UInt16, UInt32, Byte, SByte, Float, Double, Boolean, String, LocalizedText, DateTime, Int8 Array, Int16 Array, Int32 Array, UInt8 Array, UInt16 Array, UInt32 Array, Byte Array, SByte Array, Float Array, Double Array.

The **OpcUa Client** node delivers the full client functionality for reading, subscribing or monitoring. It is not mandatory to use the configuration UI by double clicking on the node. This node can also be configured using msg-properties. An exception is the endpoint. The endpoint must be configured once within the configuration UI.

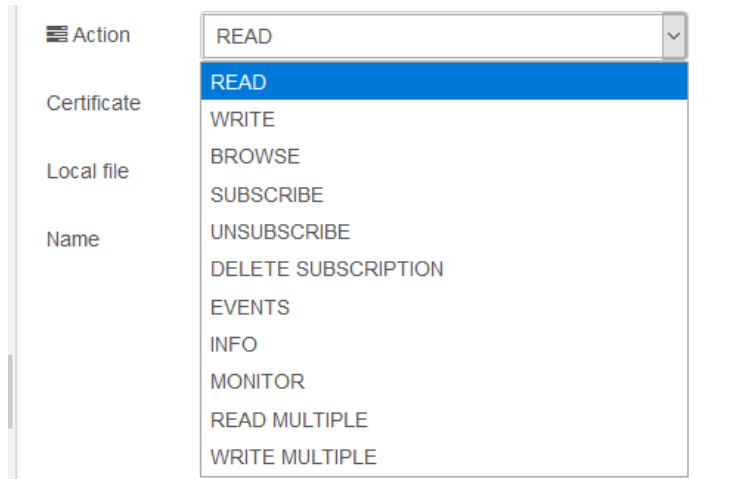


First the endpoint **must** be configured once.



Within this next step you could define the access to the OPC UA server.

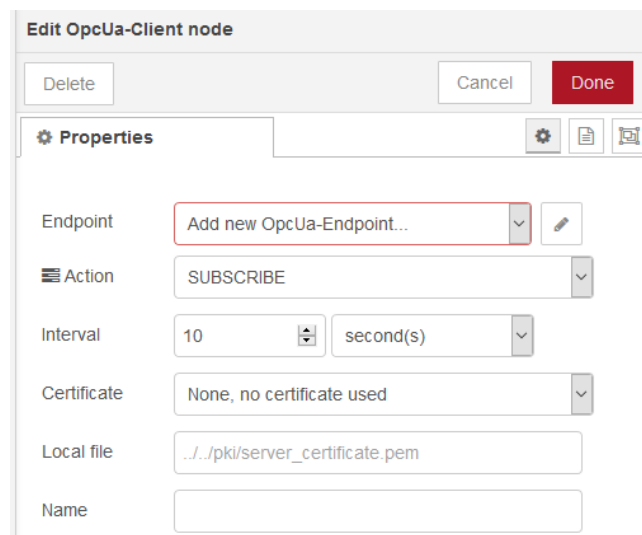
Alternatively you could use the `msg.action` property.



A screenshot of a dropdown menu for the 'Action' property. The menu is open, showing a list of actions: READ (highlighted in blue), WRITE, BROWSE, SUBSCRIBE, UNSUBSCRIBE, DELETE SUBSCRIPTION, EVENTS, INFO, MONITOR, READ MULTIPLE, and WRITE MULTIPLE. The 'Action' label is visible on the left side of the dropdown.

Depending on the type of action the UI might change, and additional parameters are requested.
Example **SUBSCRIBE** or **EVENTS** and **MONITOR**.

SUBSCRIBE and **EVENTS** need an interval, also addressable via `msg.interval` or `msg.payload`.



A screenshot of the 'Edit OpcUa-Client node' dialog box. The dialog has a 'Delete' button, a 'Cancel' button, and a 'Done' button. Below these buttons is a 'Properties' section with a gear icon, a document icon, and a refresh icon. The 'Endpoint' field is a dropdown menu with the text 'Add new OpcUa-Endpoint...' and a pencil icon. The 'Action' field is a dropdown menu with the text 'SUBSCRIBE'. The 'Interval' field has a numeric input '10' and a unit dropdown 'second(s)'. The 'Certificate' field is a dropdown menu with the text 'None, no certificate used'. The 'Local file' field is a text input with the text '.../pki/server_certificate.pem'. The 'Name' field is an empty text input.

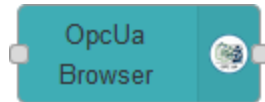
MONITOR needs additionally a deadband type and a deadband value, also addressable via `msg.deadbandType` and `msg.deadbandValue`

A full overview for all properties can be found at the [GitHub webpage](#). The following table was extracted from there.

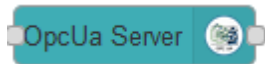
Property	Function/Value	Notes
payload	Set interval for subscription or monitorItem	-
interval	Subscription interval	-
queueSize	Subscription queue size	-
deadbandType	"a" abs. or "p" percent	Action monitor
deadbandValue	Integer for deadband	Action monitor
topic	NodeId and DataType in format ns=3;s=Counter;datatype=Int32	
action	Subscribe	nodeId / variable
	Unsubscribe	nodeId / variable
	Deletesubscription	subscription
	Browse	nodeId / folder
	Info	nodeId
	Read	nodeId
	Write	nodeId & value
	Monitor	deadbandtype abs/pro
	Events	nodeId
	Readmultiple	[nodeId + datatype]

Property	Function/Value	Notes
	Writemultiple	[nodeId + datatype + value]

The **OpcUa Browser** node is for browsing and discovering the connected server. If working properly, it might be useful to get dynamically the node ids by using the browse function but as mentioned above it is **not recommended to use this functionality** since it might cause problems on the u-control web / IoT gateway side.



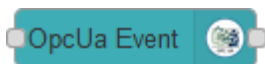
The **OpcUa Server** node is for creating and defining an own OPC UA server.



The server node is still under development, but some commands are already implemented. According to the documentation, the following actions can be performed:

- Create endpoint
- Restart server
- Add folder
- Set folder
- Add variable
- Delete by nodeId

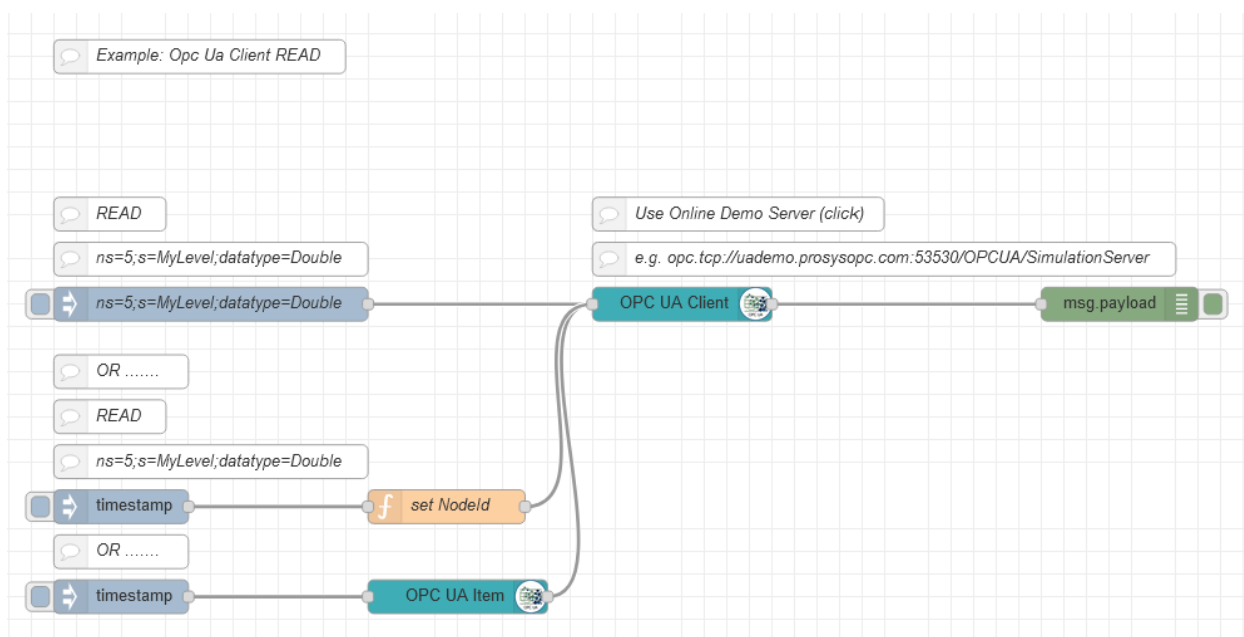
With the **OpcUa Event** node the user can fetch events from the server.



2.2 Basic functionality

2.2.1. Read

A basic read action requires minimum three node instances as shown on the figures below: an **Inject** node, an **OPC UA Client** node and a **Debug** node. As you can see, there are different ways to read out a variable. The following is an overview of some of them.



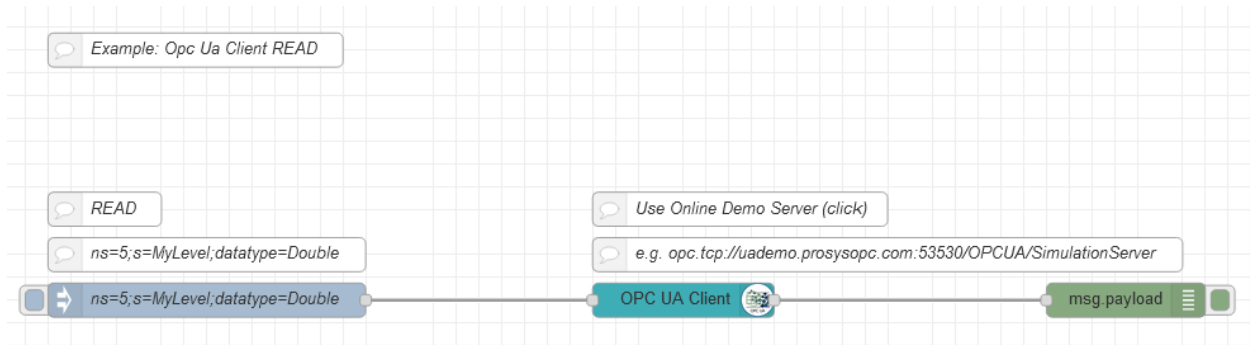
Here **OPC UA Client** node is configured as follows.

The screenshot shows the configuration window for the OPC UA Client node. The window has a title bar "Edit OpcUa-Client node" and buttons for "Delete", "Cancel", and "Done". The "Properties" tab is selected, showing the following fields:

- Endpoint:** `opc.tcp://uademo.prosysopc.com:5353`
- Action:** `READ`
- Certificate:** `None, no certificate used`
- Local file:** `../pki/server_certificate.pem`
- Name:** (empty field)

Application Note for using node-red-contrib-opcua

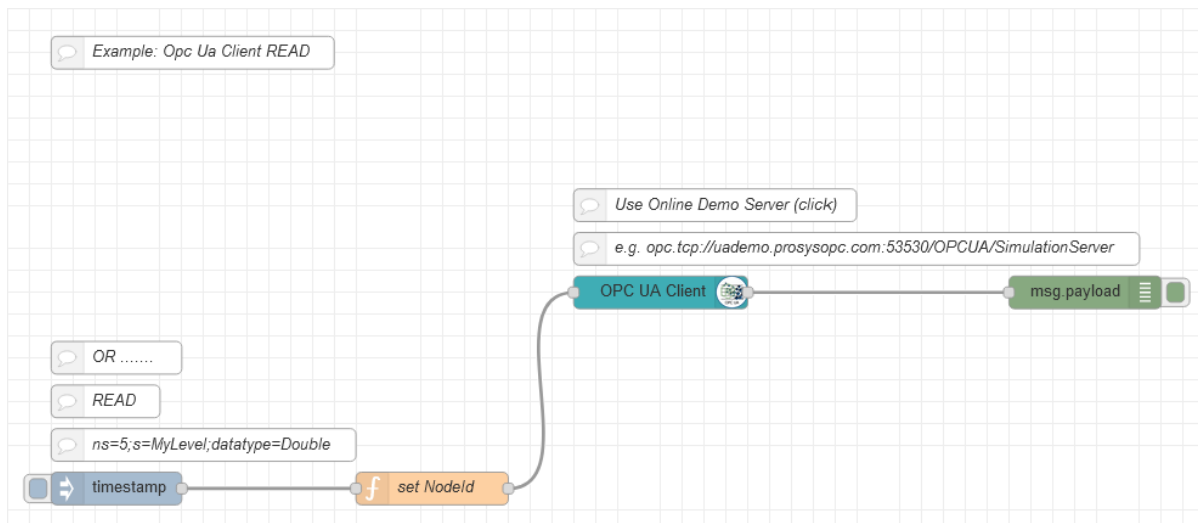
First way is a single configured **Inject** node.



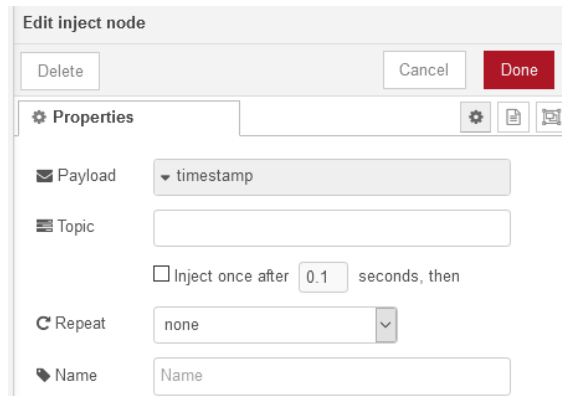
As `msg.topic` the complete node id "ns=5; s=MyLevel" along with "datatype=Double" was set.

The screenshot shows the "Edit inject node" dialog box. It has a "Delete" button, a "Cancel" button, and a "Done" button. Under the "Properties" tab, the "Payload" is set to "timestamp". The "Topic" is set to "ns=5;s=MyLevel;datatype=Double". There is a checkbox for "Inject once after 0.1 seconds, then" which is unchecked. The "Repeat" dropdown is set to "none". The "Name" is set to "ns=5;s=MyLevel;datatype=Double".

The second option is a single **inject** node as timestamp injection combined with a **function** node.



The **Inject** node is configured the following way:



Edit inject node

Delete Cancel Done

Properties

Payload timestamp

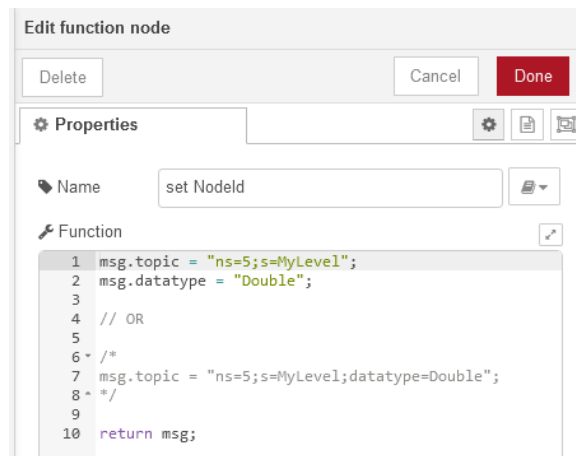
Topic

☐ Inject once after 0.1 seconds, then

Repeat none

Name Name

The code inside the **function** node is the following. `msg.topic` was set to “ns=5; s=MyLevel” and `msg.datatype` was set to “Double”. This works as well.



Edit function node

Delete Cancel Done

Properties

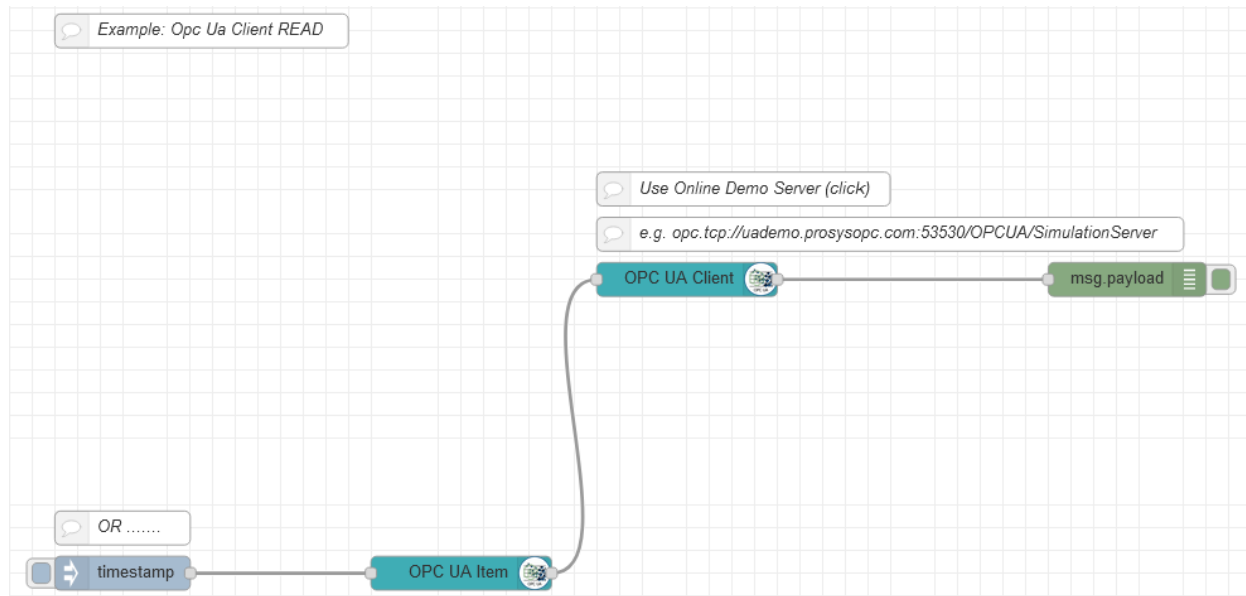
Name set NodeId

Function

```
1 msg.topic = "ns=5;s=MyLevel";
2 msg.datatype = "Double";
3
4 // OR
5
6 /*
7 msg.topic = "ns=5;s=MyLevel;datatype=Double";
8 */
9
10 return msg;
```

Application Note for using node-red-contrib-opcua

A third way of reading out a variable is using the **OPC UA Item** node.



The **Inject** node is configured as follows:

The "Edit inject node" dialog box is shown. It has a title bar "Edit inject node" and buttons "Delete", "Cancel", and "Done". Below the title bar is a "Properties" section with a gear icon. The "Payload" field is set to "timestamp". The "Topic" field is empty. The "Inject once after" checkbox is unchecked, with a value of "0.1" seconds. The "Repeat" dropdown is set to "none". The "Name" field is set to "Name".

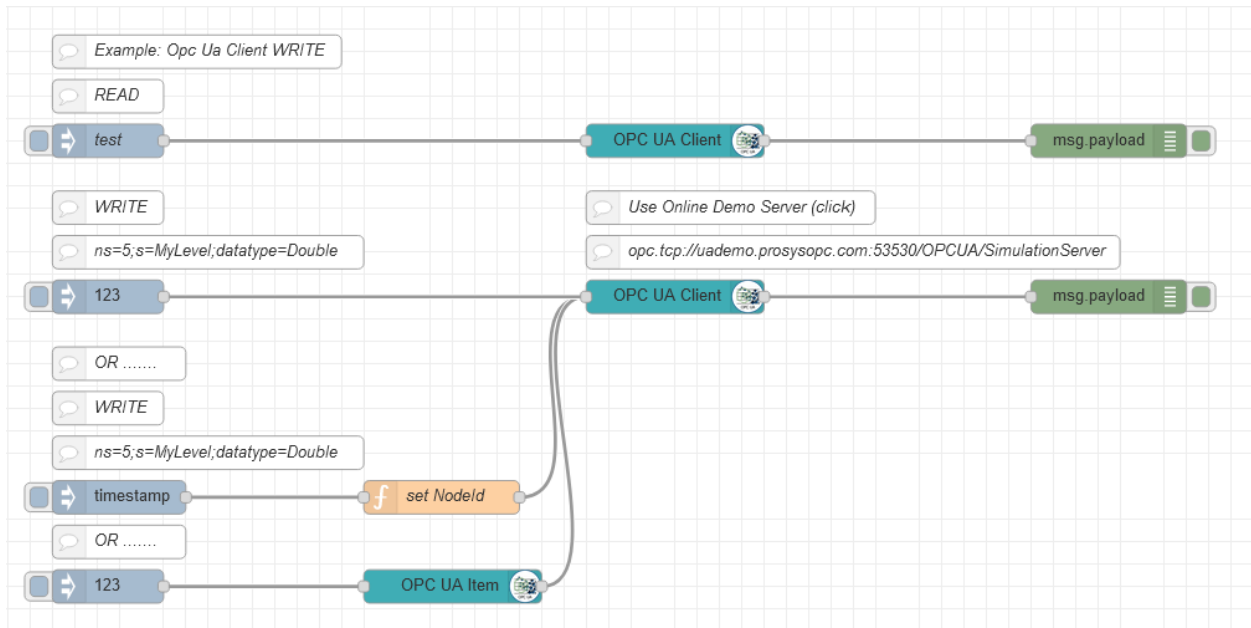
The way to configure **OPC UA Item** node is shown below.

The "Edit OpcUa-Item node" dialog box is shown. It has a title bar "Edit OpcUa-Item node" and buttons "Delete", "Cancel", and "Done". Below the title bar is a "Properties" section with a gear icon. The "Item" field is set to "ns=5;s=MyLevel". The "Type" dropdown is set to "Double". The "Value" field is empty. The "Name" field is empty.

2.2.2. Write

Writing operations work very similar to the reading operation. The difference is the configuration of **OPC UA Client** node or the property `msg.action`. The flow below shows a read and a write operation. The read operation is to check whether writing operation took effect or not.

Below three ways of writing a variable are described.



Upper **OPC UA Client** node (left figure), lower **OPC UA Client** node (right figure).

Edit OpcUa-Client node

Delete Cancel Done

Properties

Endpoint:

Action:

Certificate:

Local file:

Name:

Edit OpcUa-Client node

Delete Cancel Done

Properties

Endpoint:

Action:

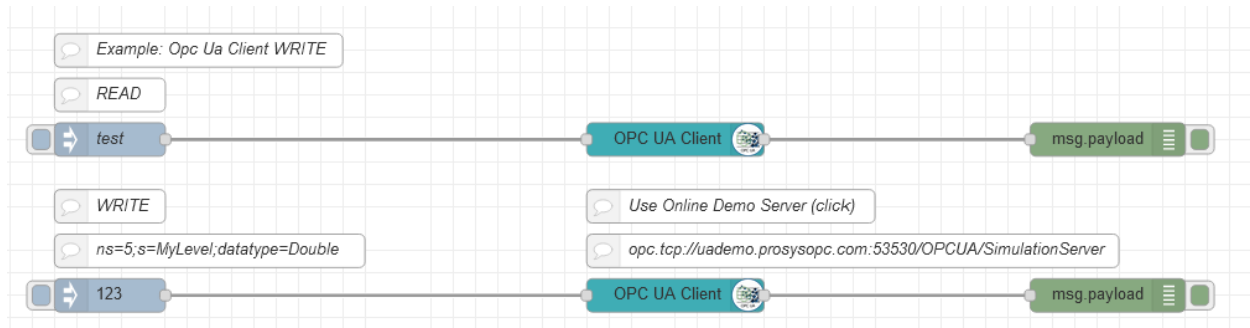
Certificate:

Local file:

Name:

Application Note for using node-red-contrib-opcua

First approach to writing a variable is by writing the nodeId into the `msg.topic` property of an **Inject** node like shown below. The value to be written must be placed in the `msg.payload` property.

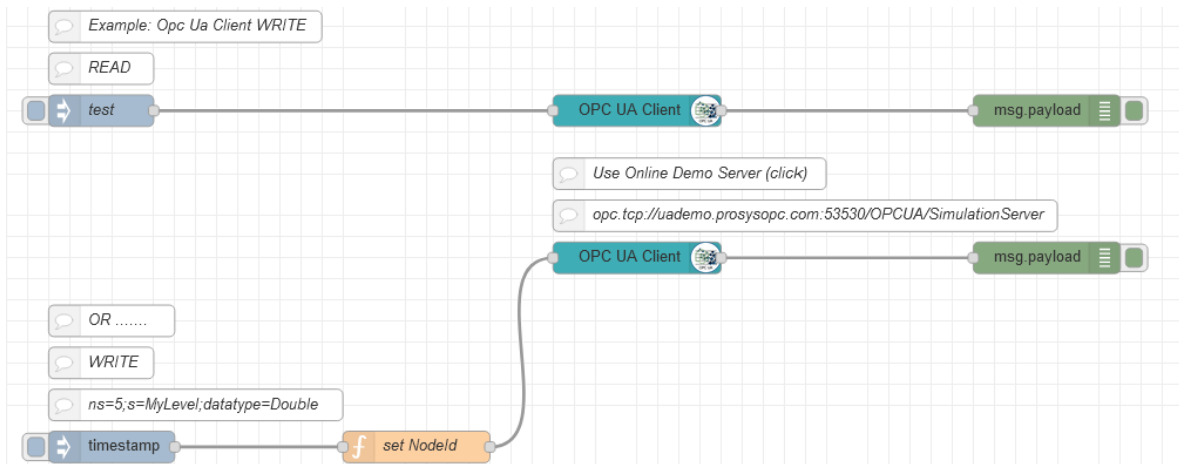


Configuration of the **inject** node.

The screenshot shows the 'Edit inject node' configuration dialog. It has a 'Delete' button, a 'Cancel' button, and a 'Done' button. The 'Properties' section is expanded, showing the following fields: 'Payload' with a dropdown menu showing '123', 'Topic' with a text field containing 'ns=5;s=MyLevel;datatype=Double', 'Inject once after' with a checkbox and a value of '0.1' seconds, 'Repeat' with a dropdown menu showing 'none', and 'Name' with a text field containing 'Name'.

Application Note for using node-red-contrib-opcua

The second approach to write a variable more independently is by using a **Function** node.



The **Inject** node triggers a message with a timestamp as `msg.payload`.

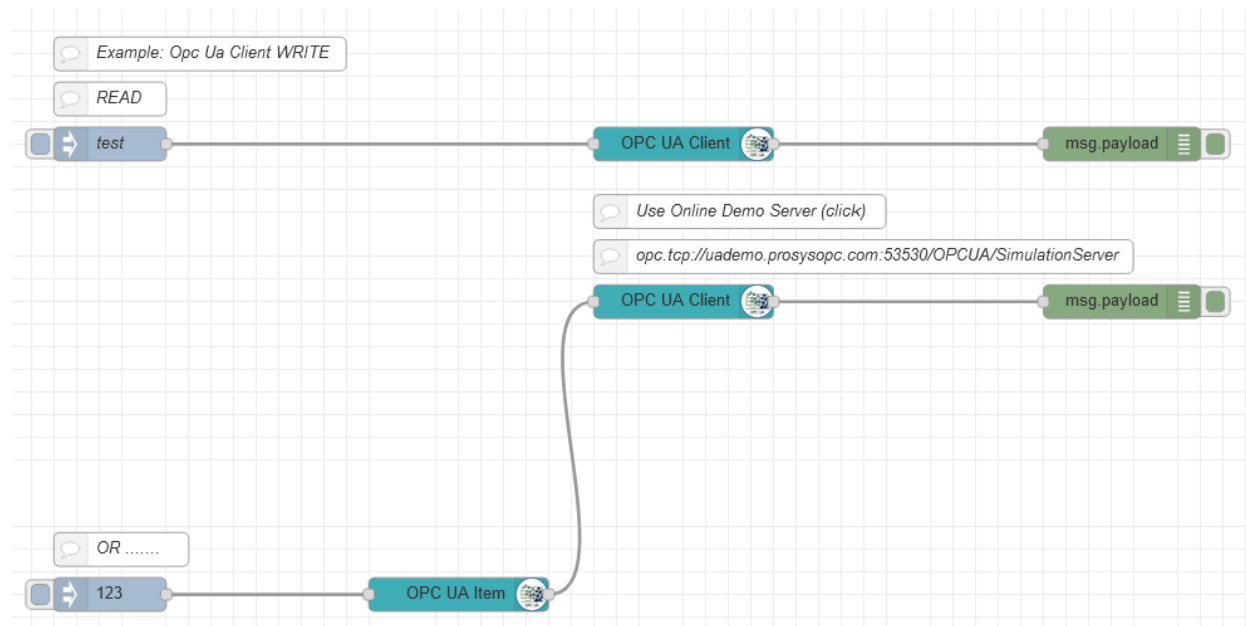
The 'Edit inject node' window shows the configuration for the inject node. The 'Payload' is set to 'timestamp'. The 'Topic' is empty. The 'Inject once after' checkbox is unchecked, with a value of '0.1' seconds. The 'Repeat' dropdown is set to 'none'. The 'Name' field is empty.

The **Function** node modifies `msg.topic`, `msg.datatype` and `msg.payload` according to the target node id. Here: `msg.topic = "ns=5; s=MyLevel"`; `msg.datatype = "Double"`; `msg.payload = 123`;

The 'Edit function node' window shows the configuration for the function node. The 'Name' is set to 'set NodeId'. The 'Function' code is as follows:

```
1 msg.topic = "ns=5;s=MyLevel";
2 msg.datatype = "Double";
3 msg.payload = 123;
4 // OR
5
6 /*
7 msg.topic = "ns=5;s=MyLevel;datatype=Double";
8 msg.payload = 123;
9 */
10
11 return msg;
```

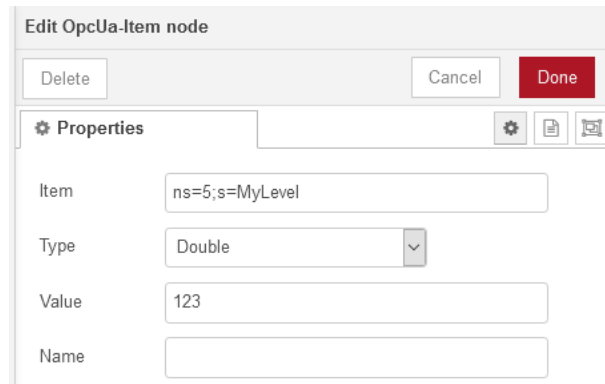
The third way of writing a variable is by using a **OPC UA Item** node.



For this purpose, an **Inject** node is configured as shown below. The value must be entered in msg.payload.

The 'Edit inject node' configuration window is shown. It includes a 'Delete' button, 'Cancel' and 'Done' buttons, and a 'Properties' section. The 'Payload' field is set to '123'. The 'Topic' field is empty. The 'Inject once after' checkbox is unchecked, with a value of '0.1' seconds. The 'Repeat' dropdown is set to 'none'. The 'Name' field is empty.

The **OPC UA Item** node has this configuration. Item describes the node id. Type describes the OPC UA variable type and Value is the text input for a value which should be written.

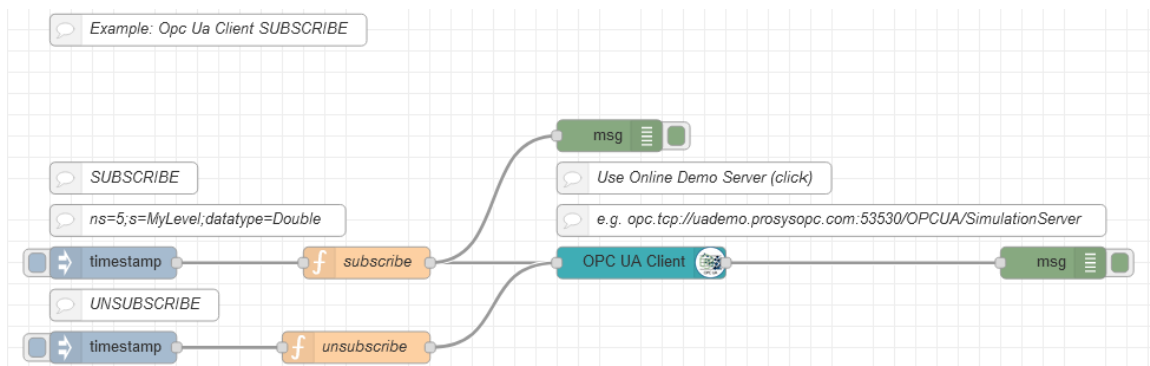


The 'Edit OpcUa-Item node' window shows the following configuration:

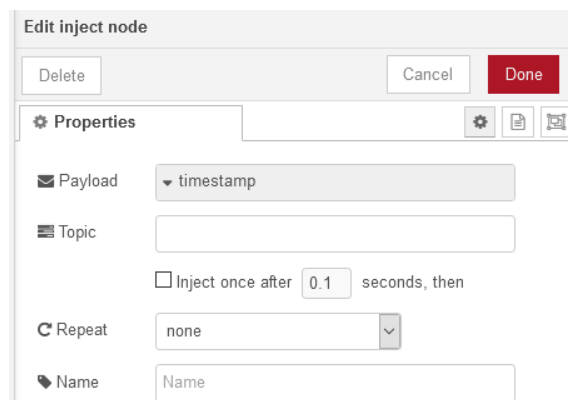
- Item:** ns=5;s=MyLevel
- Type:** Double
- Value:** 123
- Name:** (empty)

2.2.3. Subscribe

Subscribing to OPC UA variables is useful if you have a list of variables which should be read out cyclically after a certain amount of time. With a subscription you register a variable to the **OPC UA Client** node connected with a time interval. The subscription must be done only once after Node-RED starts up and as first operation after the run of the flows. Multiple subscription may lead to performance issues.



Within this example the **Inject** node is configured as manually triggered but should be configured as **Inject once after x.x seconds, then → Repeat = none**.

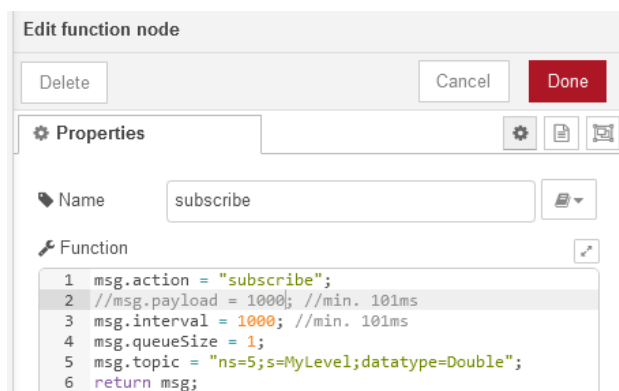


The 'Edit inject node' window shows the following configuration:

- Payload:** timestamp
- Topic:** (empty)
- Inject once after:** 0.1 seconds, then
- Repeat:** none
- Name:** Name

Application Note for using node-red-contrib-opcua

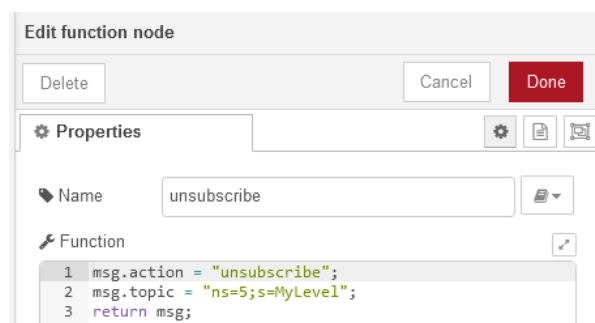
To subscribe a variable the Function node **subscribe** should look as follows. Interval or payload define the cycle time.



The screenshot shows the 'Edit function node' dialog for a function named 'subscribe'. The 'Function' tab is active, displaying the following JavaScript code:

```
1 msg.action = "subscribe";
2 //msg.payload = 1000; //min. 101ms
3 msg.interval = 1000; //min. 101ms
4 msg.queueSize = 1;
5 msg.topic = "ns=5;s=MyLevel;datatype=Double";
6 return msg;
```

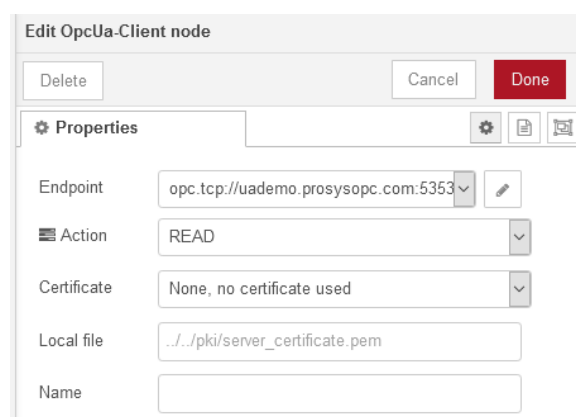
To unsubscribe a variable the Function node **unsubscribe** should look as shown below.



The screenshot shows the 'Edit function node' dialog for a function named 'unsubscribe'. The 'Function' tab is active, displaying the following JavaScript code:

```
1 msg.action = "unsubscribe";
2 msg.topic = "ns=5;s=MyLevel";
3 return msg;
```

This is how the **OPC UA Client** is configured. Please note that the Action is overwritten by the `msg.action` property. This means that `msg.action` has higher priority than the entry within the Action text input below.

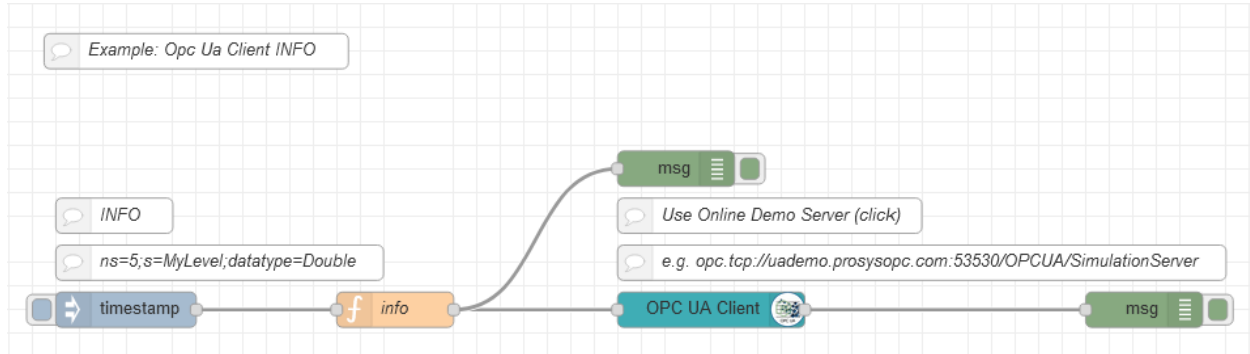


The screenshot shows the 'Edit OpcUa-Client node' dialog. The 'Properties' tab is active, showing the following configuration:

- Endpoint: `opc.tcp://uademo.prosysopc.com:5353`
- Action: `READ`
- Certificate: `None, no certificate used`
- Local file: `../pki/server_certificate.pem`
- Name: (empty)

2.2.4. Info

The **Info** action outputs information about browse name, user access level and the node id to the specific data type as string.



Configure the **Inject** node as manually triggered timestamp.

Insert the code below with `msg.action = "info"`. `msg.topic` must be the node id concatenated with the data type (`datatype=Double`).

Finally configure the **OPC UA Client** node as usual.

Edit OpcUa-Client node

Delete Cancel Done

Properties

Endpoint

Action

Certificate

Local file

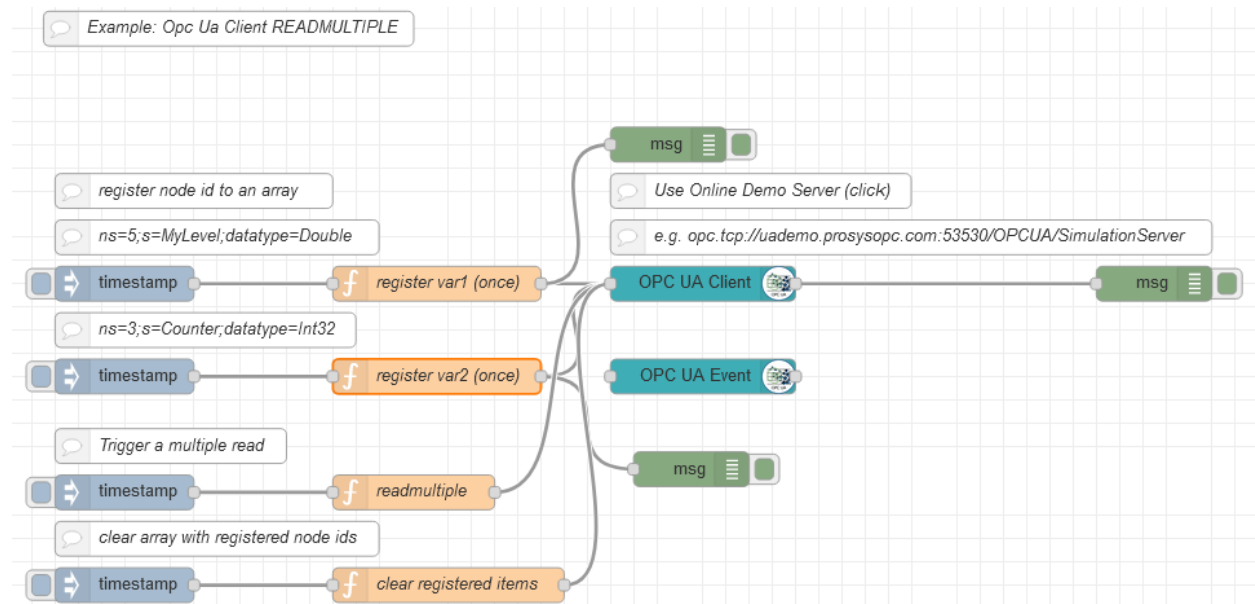
Name

OPC UA Client node outputs a message containing information like (focus on payload):

```
▼ object
  topic: "ns=5;s=MyLevel"
  ▼ payload: object
    ▶ description: object
      browseName: "MyLevel"
      userAccessLevel: 1
      type: "11"
    action: "info"
    datatype: "Double"
    _msgid: "b48d483.197bdb8"
```

2.2.5. Read multiple

This action is very useful when you want to keep your cycle times generic and your list of variables to acquire is long. The procedure is similar to **subscribe**. There are some small differences to be taken into account.

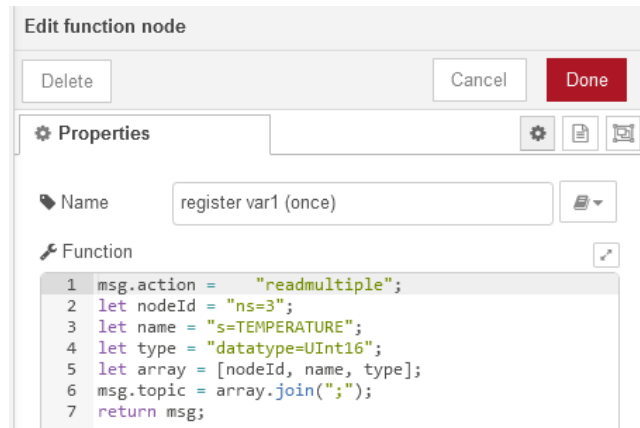


All **Inject** nodes within this example are configured like this:

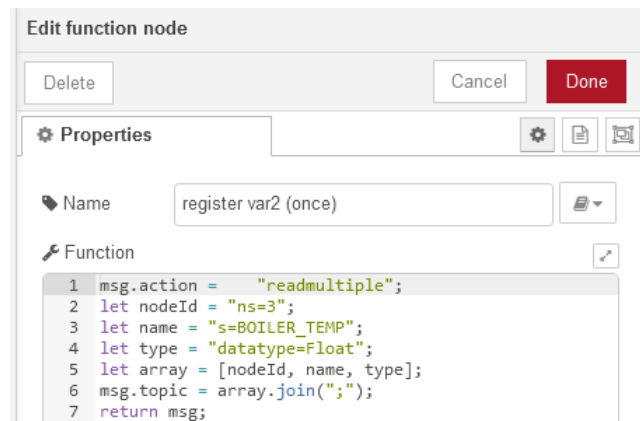
The 'Edit inject node' dialog shows the configuration for an inject node. The 'Properties' tab is active, showing the following settings:

- Payload:** dropdown menu set to 'timestamp'.
- Topic:** empty text field.
- Inject once after:** checkbox is unchecked, followed by '0.1 seconds, then'.
- Repeat:** dropdown menu set to 'none'.
- Name:** text field with the value 'Name'.

Function node **register var1 (once)** holds the following JavaScript code. The intention of this code is it to enter the nodeId, name, and type of the variable and connecting it using semicolons as separator. It is the first of two variables which should be read out as read multiple. This node works like the **subscribe** example. Every new execution leads to duplicate registrations of node ids. Could cause performance issues.



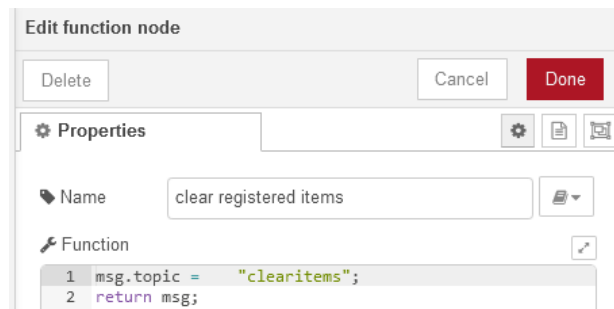
The code below is partially the same as above. Different is the nodeId, name and type of the variable at which we want to point. It is the second of two variables which should be read out as readmultiple. This node works like the **subscribe** example. Every new execution leads to duplicate registrations of node ids. Could cause performance issues.



msg.topic set to readmultiple triggers the read out. Every subscribed node id will be returned in a separate message. We expect here after execution two result msg containing our values.

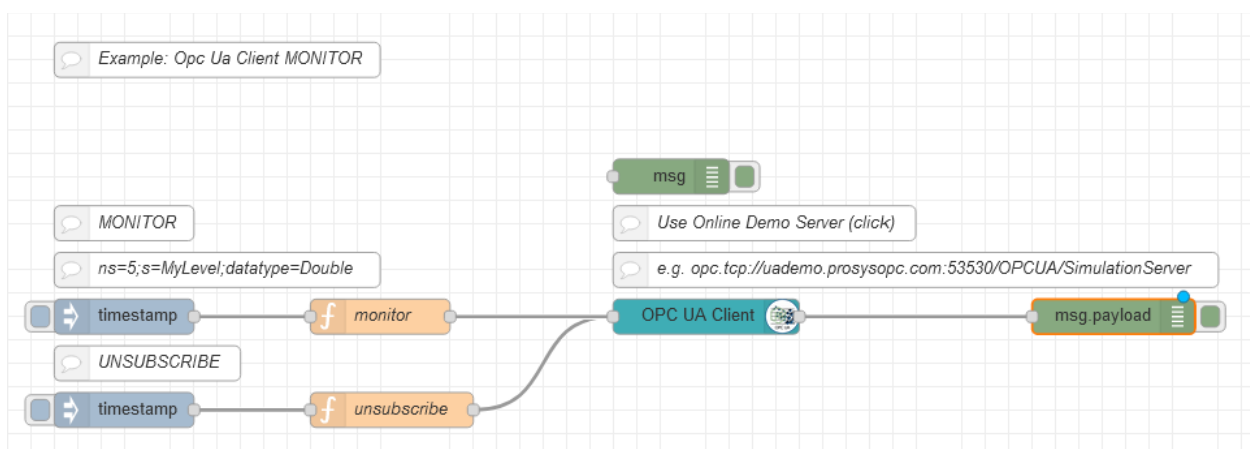


msg.topic set to clearitems triggers clears all registered node ids.



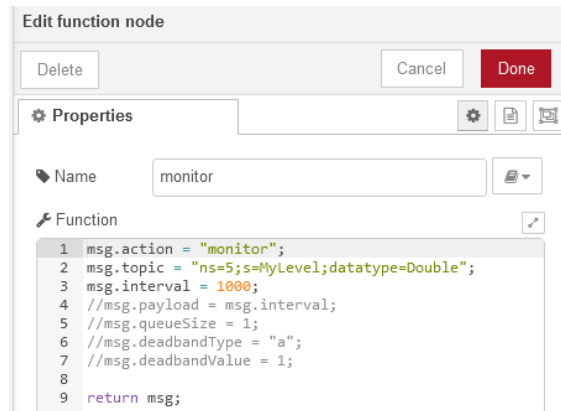
2.2.6. Monitor

Monitor uses the same function (under the hood) as subscribe. Therefore, it is recommended to refer to the section 2.2.3.



Application Note for using node-red-contrib-opcua

msg.action with monitor and msg.topic with the specific node id subscribes/registers the desired variable.



Unsubscription works according to section 2.2.3.

