GUIDE

OPC Router RESTful to OPC & SQL Quick Start

HANDS-ON APPLICATION GUIDE

Our mission is to provide you with the right software package to solve your industrial operation challenges.

Software toolbox®

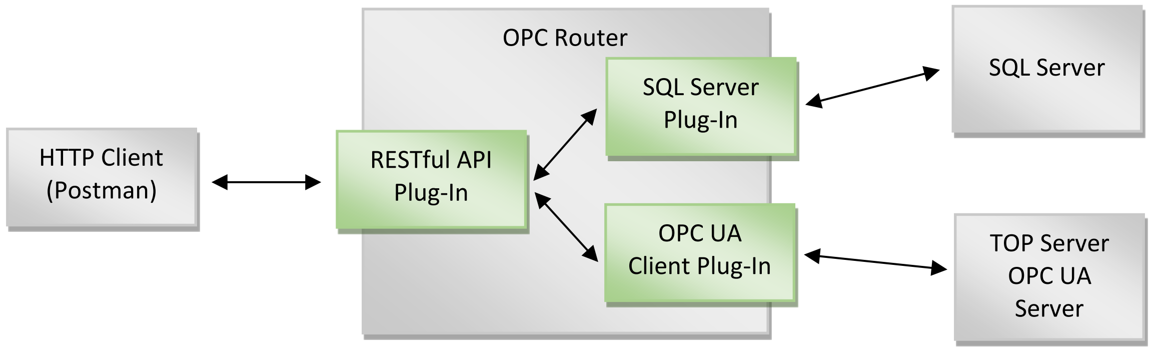
Table of Contents

Introduction

The OPC Router provides a flexible way to move data from point A to point B, while transforming, analyzing, and acting on said data. This guide is intended to serve as a guide on configuring the OPC Router to expose a RESTful API Endpoint that HTTP Clients can use to extract data from the OPC Router. It is assumed that the reader has a basic understanding of the OPC Router, particularly on how Plug-Ins and Transfer objects are used – the OPC Router Quick Start guide can be referenced for an introductory look at how to use the product.

This document will cover configuring the OPC Router with a RESTful HTTP Endpoint that will be used as a trigger return data that is accessible in the OPC Router. What values are returned will be dynamically determined based on the query parameters provided by the HTTP Client as part of the HTTP GET request.

The OPC Router will be configured in the following configuration:



Specifically, this document will discuss the following scenario.

1. The OPC UA Server is monitoring the speeds for two systems, each with two running machines.

2. When the HTTP Client does a request, it will be passing (as Query Parameters) the system and machine numbers for which it would like to know the current speed.

3. When the HTTP Client does a GET Request the OPC Router will (based on the system and machine numbers that are passed as query parameters) query a SQL Server database to determine the product ID currently running on the requested system and machine.

The resulting HTTP Response will contain the system and machine numbers (contain in the HTTP Request), the Product ID that was queried from SQL, and the current speed of the specified machine/system that was read from the OPC UA Server.

While this document will discuss creating all necessary components (plug-ins, transfer objects, and connections) it is not intended to be a replacement to the OPC Router helpfile and/or feature specific documentation. For additional information on any feature mentioned in this document please refer to the appropriate resource.

Creating the Plug-Ins

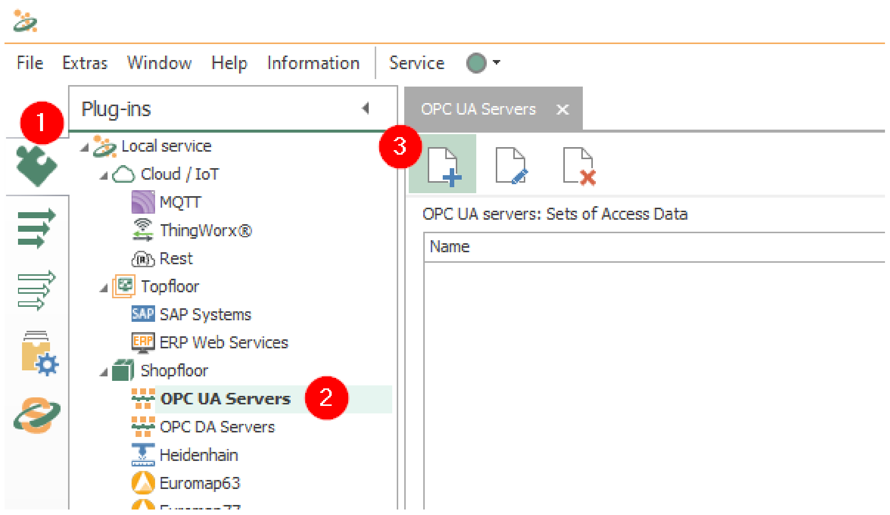
As a first step the various plug-ins – that define data sources and destinations with which the OPC Router will be interacting – must be defined. With this scenarios this is made up of three plug-ins; an OPC UA plug-in to read the machine speed from the OPC UA Server, a Microsoft SQL Server plug-in to query the current product ID being produced, and a REST/HTTP API Endpoint which will be used to expose the product data to the outside world using HTTP.

The OPC UA Plug-In

The OPC UA Server from which the machine data is being read is the TOP Server, which is exposing an OPC UA endpoint to extract data. Because the TOP Server UA Endpoint is accepting localhost connections only – the connection will not be secured and encrypted – in production environments it is recommended to follow IT best practices to keep data and machines secure. The exact steps to configure the OPC UA Server endpoints will vary with server and is outside the scope of this document – please refer to the appropriate server documentation.

To create the OPC UA Server Plug-in:

1. Open the Plug-Ins section
2. Double-click the *OPC UA Servers* plug-in button in the Plug-Ins tree view
3. Use the *Create new Plug-in Instance* button to create a new OPC UA Server plug-in instance



4. When Creating the OPC UA Server Plug-In instance:

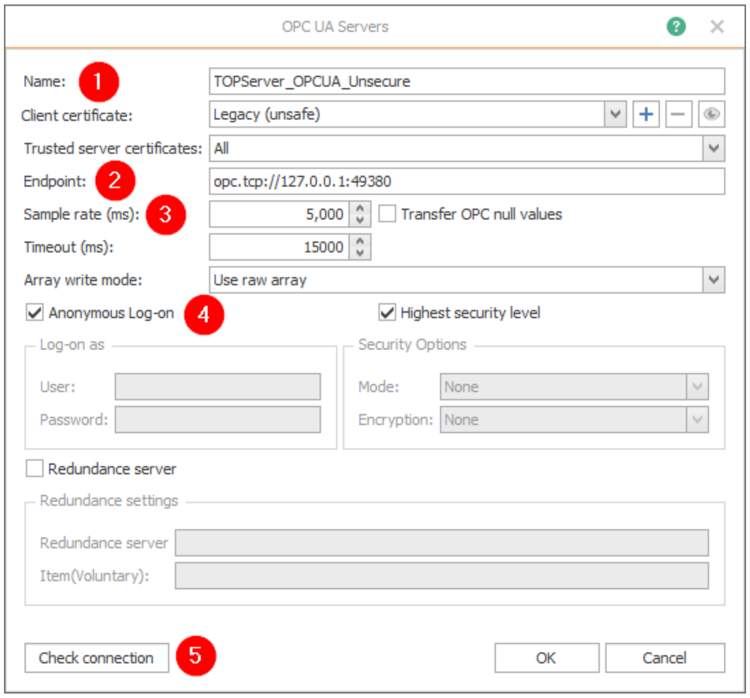
   (1) Give the plug-in a user-friendly name. This will be how the plug-in is referenced later when it is used in the OPC Router configurations.

   (2) Specify the OPC UA Endpoint to which the OPC Router will be connecting. The exact syntax here can vary depending on the OPC UA Server being used.

   (3) (Optional) Modify the rate at which the data will be read from the OPC UA Server. This setting will need to be set based on the amount of data that will be read, and the resolution at which the data is needed. In this scenario – for example – if the HTTP Client is never going to query the data often, or the machine speed is not expected to change rapidly, the sample rate can be greatly increased to reduce the loading on both the OPC UA Server and the downstream devices.

   (4) Set the OPC UA Security parameters of the OPC Router UA Client plug-in to match the settings specified in the OPC UA Server endpoint. Refer to the OPC UA Server documentation on how to configure the OPC UA Endpoint. In this example no security is used.

   (5) Use the *Check Connection* button to verify the connection to the OPC UA server is healthy, and then use the *OK* button to create the configured OPC UA Client Plug-in instance.



Name:	TOPServer_OPCUA_Unsecure
Client certificate:	Legacy (unsafe)
Trusted server certificates:	All
Endpoint:	opc.tcp://127.0.0.1:49380
Sample rate (ms):	5,000 ☐ Transfer OPC null values
Timeout (ms):	15000
Array write mode:	Use raw array
☑ Anonymous Log-on	☑ Highest security level
User:	
Password:	
Mode:	None
Encryption:	None
☐ Redundance server	
Redundance server	
Item(Voluntary):	

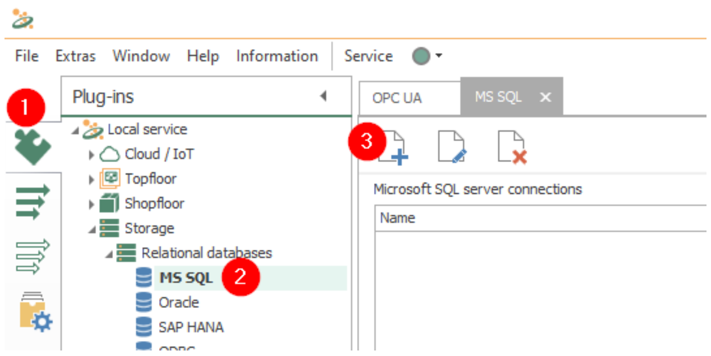
The SQL Server Plug-In

The SQL Server Plug-In is needed to be able to query the product ID from the SQL Database table. When configuring the OPC Router MS SQL Plug-In there is no need (or place) to configure the table and/or database columns. At this point in the configuration the plug-in simply represents a Microsoft SQL Server Database which contains the table that will later be queried.

1. Open the OPC Router Plug-Ins section

2. Open the list of Microsoft SQL Server Plug-Ins from the tree-view (found under Storage > Relational databases)

3. Use the *Create new Plug-in Instance* button to create a new Microsoft SQL Server plug-in instance



4. Configure the Microsoft SQL Server Plug-In instance as appropriate

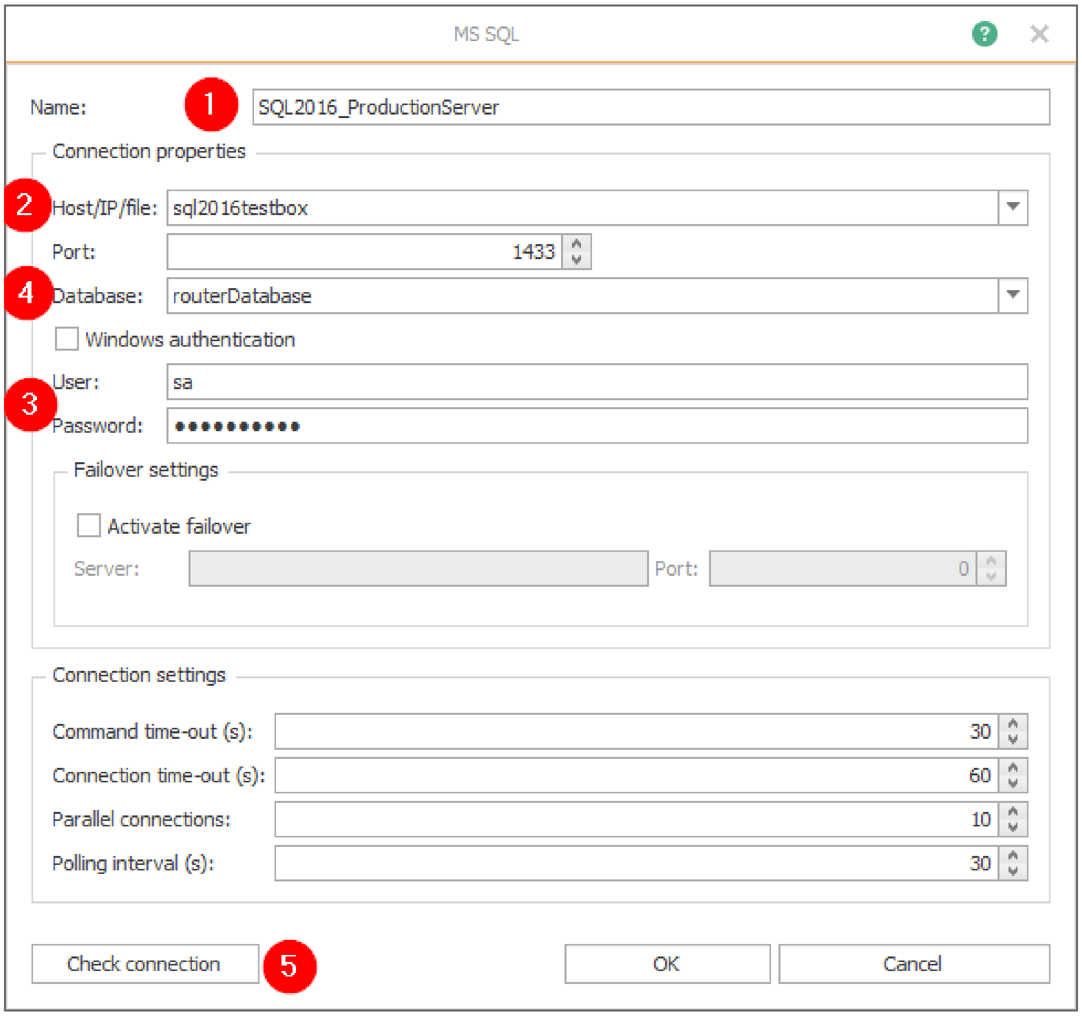
   (1) Give the SQL Server Plug-in instance a name. As with the OPC UA Plug-In this will be how the plug-in instance is referenced through the OPC Router configuration.

   (2) Specify the hostname or IP address of the server hosting the SQL Database

   (3) Specify the user credentials that should be used to connect to the server. These credentials must have adequate permissions to access the necessary database objects, as well as query data out of the necessary tables.

   (4) Use the *Database* dropdown to select the database containing the table from which the product data will be queried.

(5) Use the *Check Connection* button to confirm the connection to the SQL Server database is successful, then use the *OK* button to complete the plug-in instance configuration.



Name: SQL2016_ProductionServer
Host/IP/file: sql2016testbox
Port: 1433
Database: routerDatabase
Windows authentication
User: sa
Password: ●●●●●●●●●●●
Activate failover
Server:
Port: 0
Command time-out (s): 30
Connection time-out (s): 60
Parallel connections: 10
Polling interval (s): 30

The REST API Plug-In

Arguably the most important plug-in in this configuration is the REST API Plug-in. This plug-in will not only be used to expose the RESTful webservice endpoint (from the OPC Router to the rest of the world) that will be used to query the data out of the OPC Router, but will serve as the 'data flow' trigger to set the connection in motion.

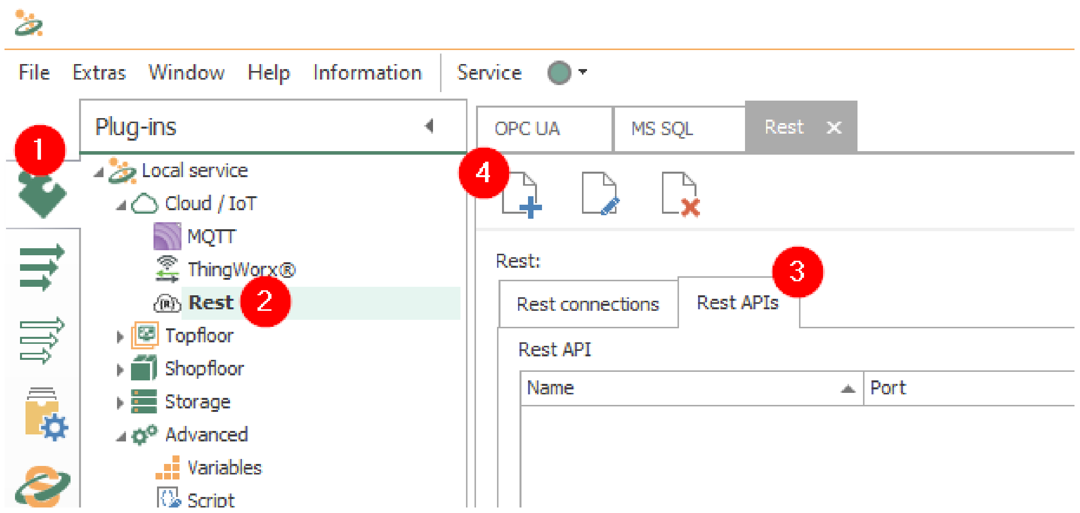
As with the other plug-ins create a new plug-in instance:

1. Open the Plug-In toolbox in the OPC Router

2. Open the Rest Plug-Ins (Found under the Cloud/IoT section of the Plug-In toolbox)

3. Navigate to the Rest APIs tab

4. Use the *Create new Plug-In instance* button to create a new Rest API plug-in instance



5. Configure the Rest API plug-in instance parameters

   (1) Specify the plug-in instance name – as with the other plug-ins this will be how the plugin is referenced in the OPC Router configuration.

   (2) Since this Plug-In is exposing an HTTP/REST Endpoint for clients to use – the port must be specified on which the plug-in should listen. This port must be unique to other application on the machine.

   (3) Specify an optional route prefix. This prefix will be mandatory on all URLs referencing this HTTP Endpoint. For example

      i. Without a route prefix the URL would become:

         http://<hostname/ipaddress>:<portnumber>

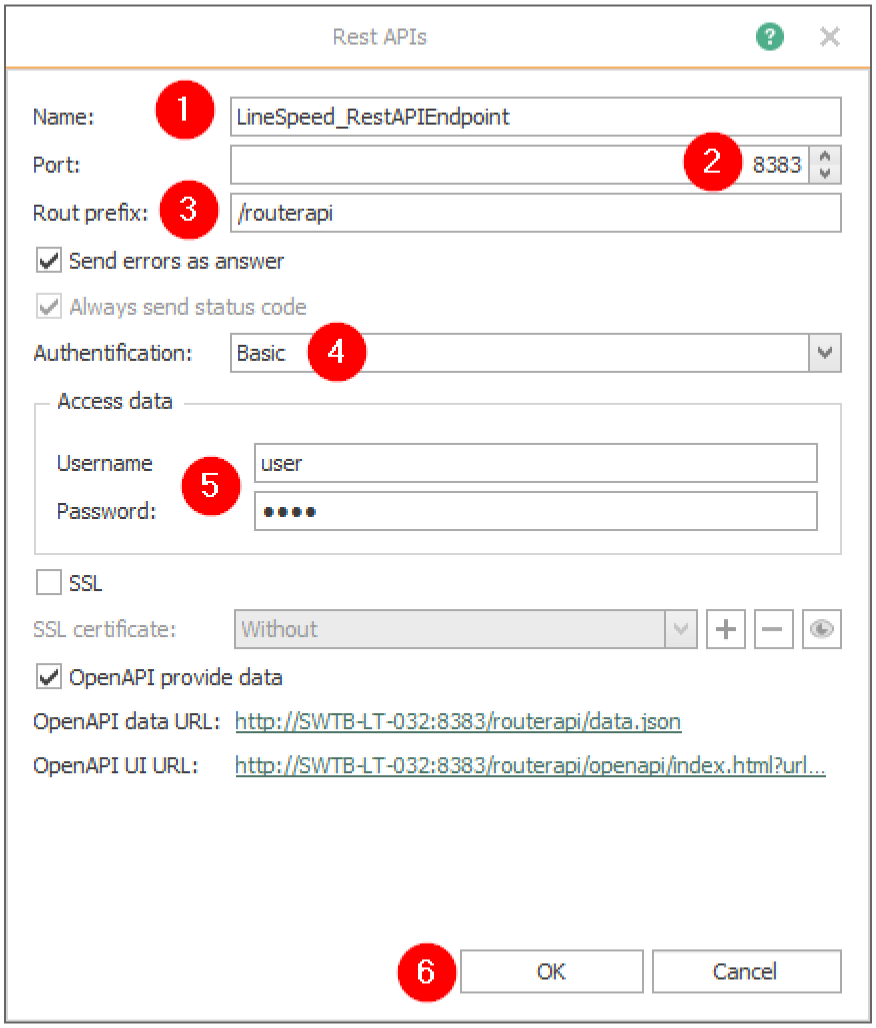
      ii. When using a route prefix of "/routerapi" the URL would become:

         http://<hostname/ipaddress>:<portnumber>/routerapi

(4) (Optional) Specify the authentication type that should be used with the API Endpoint. In this case Basic Authentication will be used.

(5) (Required with Basic Authentication) Because Basic authentication is being used a username and password must be specified. This information must be provided in any client request to this endpoint.

(6) Press the *OK* button to complete the plug-in instance creation.


Rest APIs

Name: LineSpeed_RestAPIEndpoint

Port: 8383

Rout prefix: /routerapi

☑ Send errors as answer

☑ Always send status code

Authentification: Basic

Access data

Username: user

Password: ●●●●

☐ SSL

SSL certificate: Without

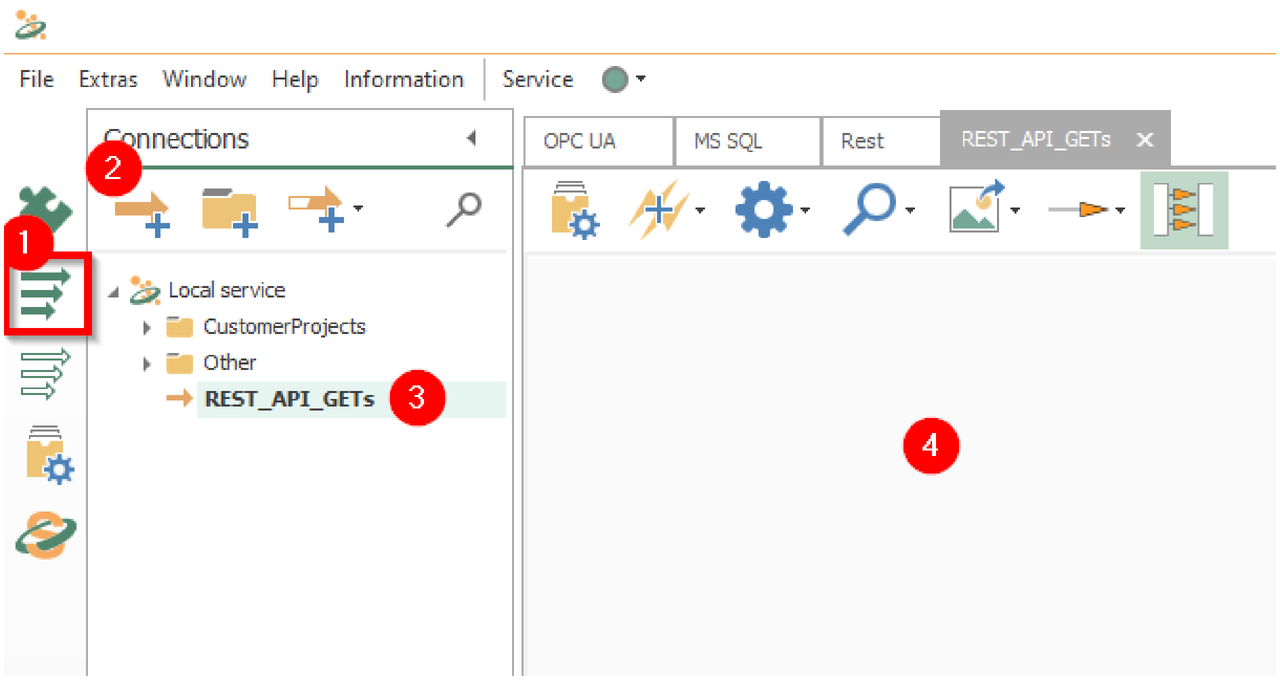
☑ OpenAPI provide data

OpenAPI data URL: http://SWTB-LT-032:8383/routerapi/data.json

OpenAPI UI URL: http://SWTB-LT-032:8383/routerapi/openapi/index.html?url...

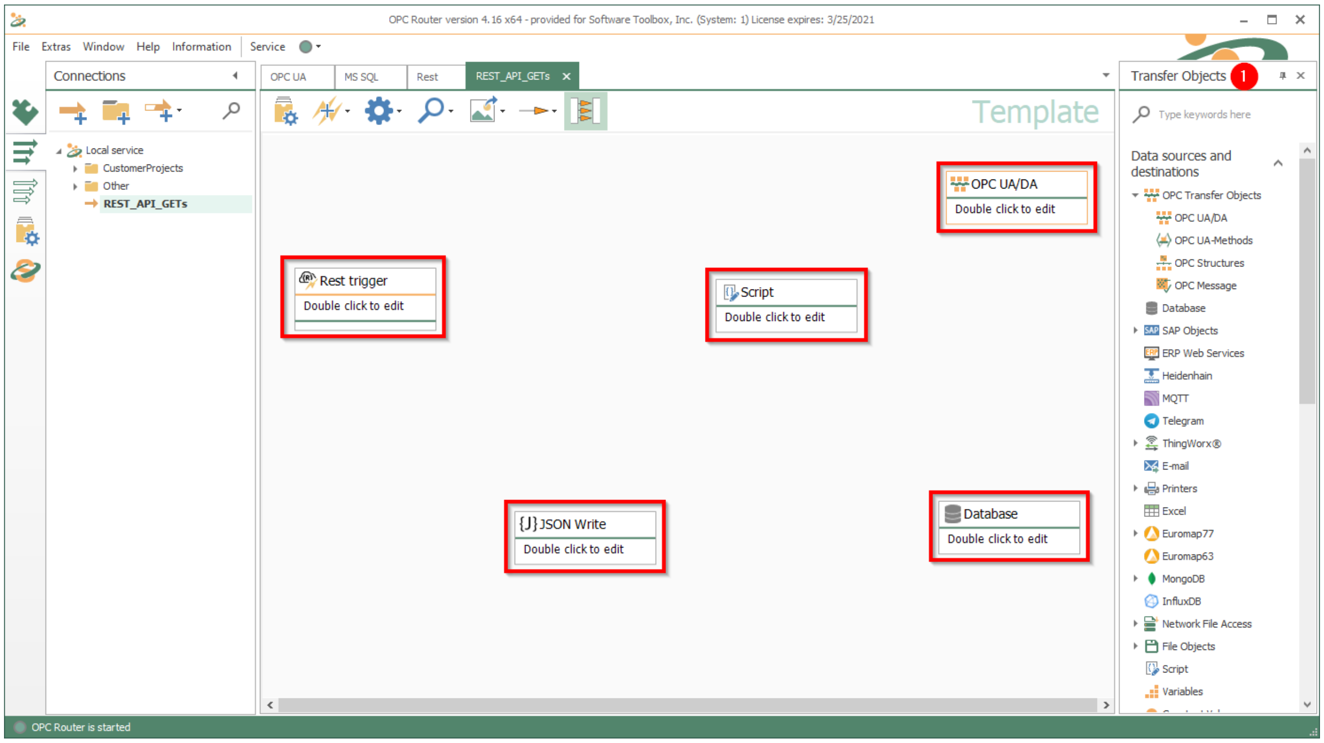
Configuring the Connection

The connection can now be created; any other transfer objects will be configured as needed during the connection configuration steps. Create a new connection by clicking on the Connections tab (1), clicking the *New Connection* button (2), and giving the new connection a name (3). A blank connection workspace will be created (4).



Using the Transfer Objects toolbox (1) find and add the following transfer obejcts to the blank connection by clicking and dragging the transfer object to the connection workspace:

1. Transfer Objects Toolbox > Triggers > **Rest Trigger**
2. Transfer Objects Toolbox > Data sources and destinations > **Script**
3. Transfer Objects Toolbox > Data sources and destinations > **JSON Write**
4. Transfer Objects Toolbox > Data sources and destinations > OPC Transfer Objects > **OPC UA/DA**
5. Transfer Objects Toolbox > Data sources and destinations > **Database**



With the generic Transfer objects added to the connection – these objects can now be configured.

REST API Trigger Transfer Object

Double-click the Rest Trigger transfer object to open its configuration.

1. Use the Rest API Plug-in Dropdown to select the REST API plug-in instance that was created previously in this document.

2. Provide an optional query endpoint.

   Without specifying the router endpoint, the URL for HTTP clients to connect to would be:

   http://<hostname/ipaddress>:<portnumber>/routerapi

   If an optional endpoint is specified (e.g. /QuerySpeed), the URL becomes

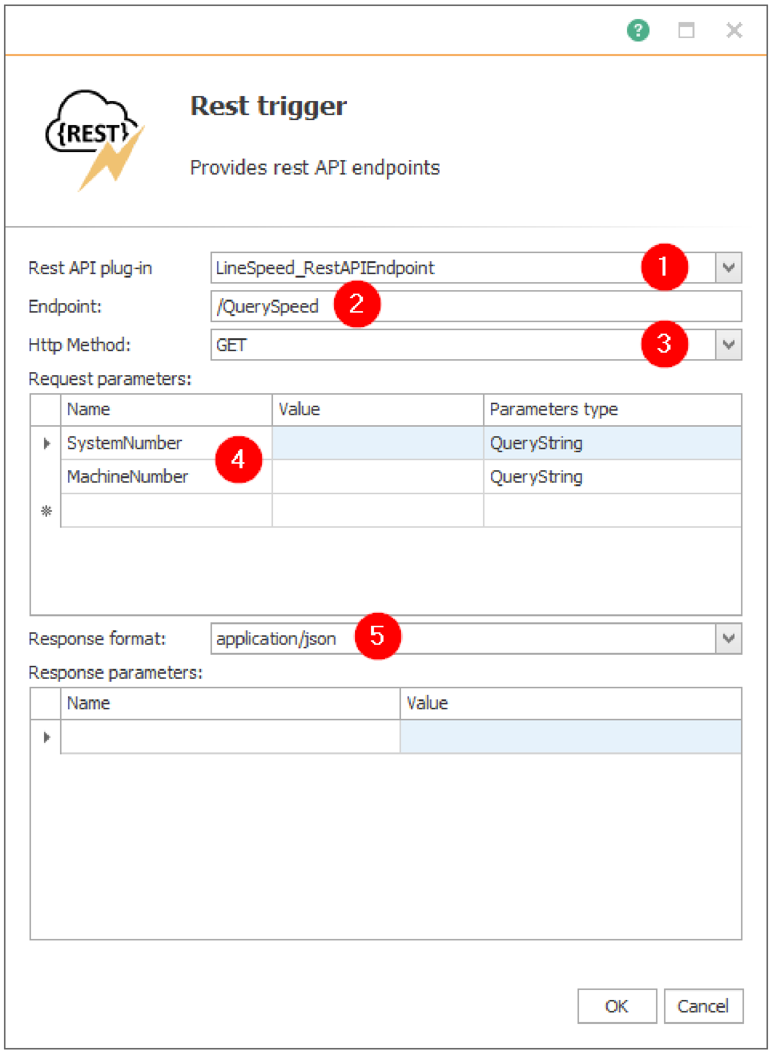
   http://<hostname/ipaddress>:<portnumber>/routerapi/QuerySpeed

3. Specify the HTTP method that the OPC Router should support on this endpoint. In this example, the OPC Router will only accept GET Requests on the API Endpoint – which specific methods are supported will vary by system and application.

4. Specify the Request Parameters. These are Query Parameters that must be provided by any HTTP Client issuing a request to this endpoint. In this case the SystemNumber and MachineNumber must be passed in by the HTTP Client so that the OPC router knows which system it should be returning the data for.

5. Specify the format of the response payload. In this example the response payload will be a JSON Document.

Press OK to complete the Rest Trigger Configuration.


Rest trigger

Provides rest API endpoints

Rest API plug-in	LineSpeed_RestAPIEndpoint
Endpoint:	/QuerySpeed
Http Method:	GET

Request parameters:

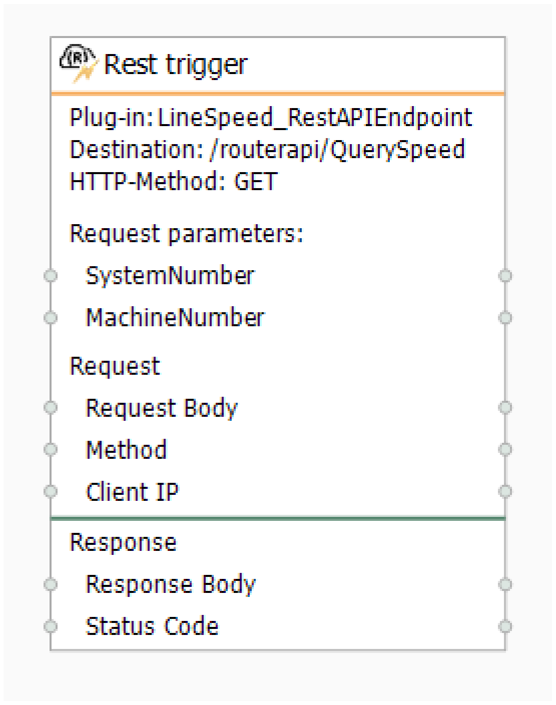
Name	Value	Parameters type
SystemNumber		QueryString
MachineNumber		QueryString

Response format: application/json

Response parameters:

Name	Value

The completely configured Rest trigger transfer object should now appear in the OPC Router Configuration Workspace.


Rest trigger
Plug-in: LineSpeed_RestAPIEndpoint
Destination: /routerapi/QuerySpeed
HTTP-Method: GET
Request parameters:
SystemNumber
MachineNumber
Request
Request Body
Method
Client IP
Response
Response Body
Status Code

OPC UA Client Transfer Object

Double-click the OPC UA/DA transfer object to open its configuration.

1. Use the OPC access data dropdown to select the OPC Client object instance that was created previously in this document.

   If the item IDs that will be read from this server are browsable, continue with step 2. If the item IDs will be manually entered skip to step 4.

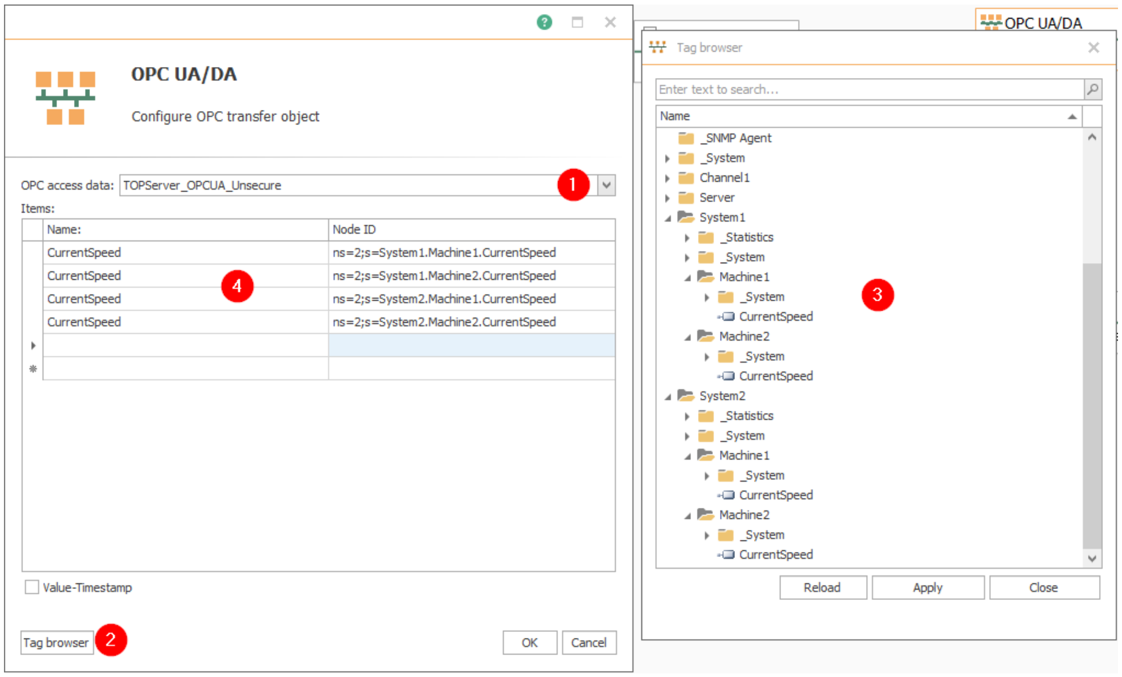
2. Press the *Tag browser* button to open the browsing utility

3. Find and double-click all tags in the OPC Server which the OPC Router should monitor through this OPC Client Plug-in Instance. By double-clicking the tag, the corresponding item will appear in the *Items* list of the OPC UA/DA transfer object configuration dialog.

4. Review the tags that have been added to the OPC Client object. The Name field can be modified here as needed. This this example the Name field defaulted to the same value for all tags being monitored (since the tag naming convention was identical on the server side) so the Name filed should be made more descriptive and unique – to make it easier to work with the item names later on.

   If the decision was made in step 1 to not browse to tags, but to manually type out the node ID. This can be done by specifying a Name (how the tag is referenced in the OPC Router), and the corresponding Node or Item ID (the path to the item in the OPC Server).

   Press OK to complete the OPC Client object configuration.


OPC UA/DA

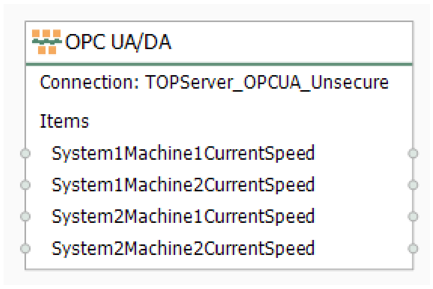
Configure OPC transfer object

OPC access data: TOPServer_OPCUA_Unsecure

Items:

Name:	Node ID
CurrentSpeed	ns=2;s=System1.Machine1.CurrentSpeed
CurrentSpeed	ns=2;s=System1.Machine2.CurrentSpeed
CurrentSpeed	ns=2;s=System2.Machine1.CurrentSpeed
CurrentSpeed	ns=2;s=System2.Machine2.CurrentSpeed

The OPC Client Transfer Object will now show any configured items/tags as accessible data points in the OPC Router Configuration workspace:


OPC UA/DA

Connection: TOPServer_OPCUA_Unsecure

Items

System1Machine1CurrentSpeed
System1Machine2CurrentSpeed
System2Machine1CurrentSpeed
System2Machine2CurrentSpeed

SQL Server Transfer Object

Double-click the Database transfer object to open its configuration.

1. Use the DB access data dropdown to select the Microsoft SQL Server object instance that was created previously in this document.

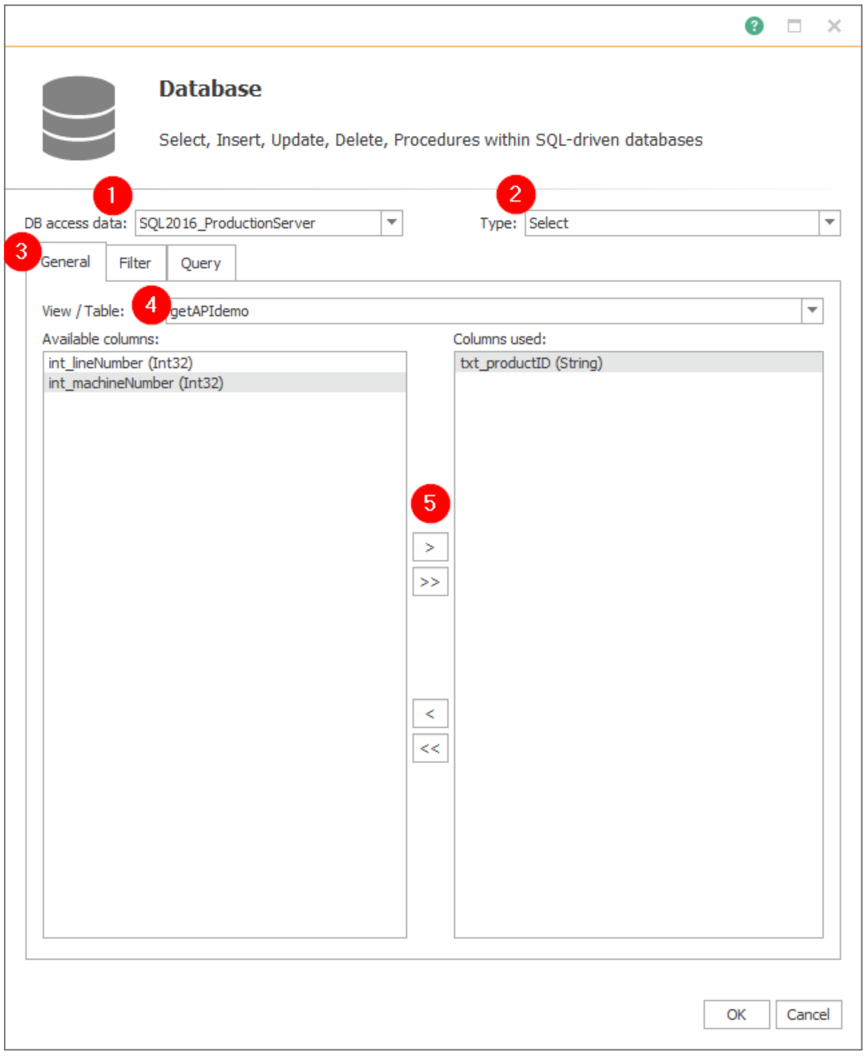
2. Use the Type dropdown to select how the Database transfer object will be interacting with the specified database i.e. what kind of query will be executed. In this case data is being queried out of the database table, so a SELECT statement must be constructed.

3. Navigate to the General Tab

4. Specify the View or Table from which the data will be retrieved – i.e. the view/table against which the query will be executed (this determines the FROM clause of the query)

5. Select the column(s) that will be used in the OPC Router, i.e. the fields that should be queried from the specified table. In this example this will determine what fields are Selected in the query. Highlight any column(s) and use the arrow to add them to the *Columns used* section.

Do NOT click OK yet – the configuration is not completed.



Database

Select, Insert, Update, Delete, Procedures within SQL-driven databases

DB access data: SQL2016_ProductionServer

Type: Select

General | Filter | Query

View / Table: getAPIdemo

Available columns:
int_lineNumber (Int32)
int_machineNumber (Int32)

Columns used:
txt_productID (String)

With the basic SELECT <Fields> FROM <table> having been written, the query can be narrowed down some using a WHERE Clause – to only return rows of data the meet specific criteria.

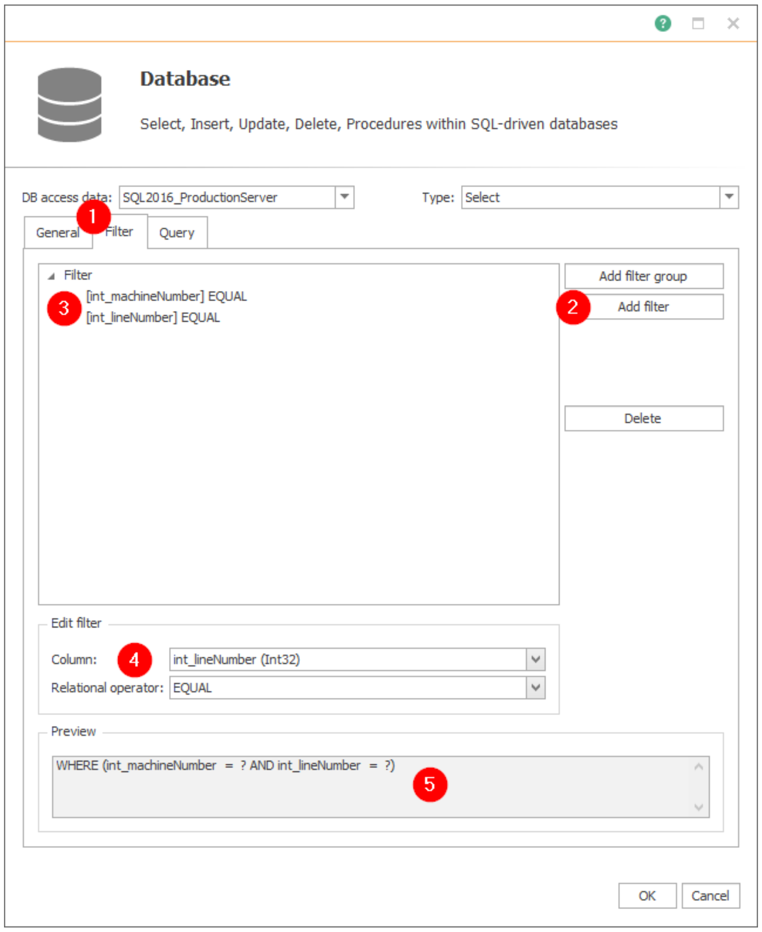
1. Navigate to the *Filter* tabe to configure the data filter

2. Add two filters to the clause by pressing the *Add filter* button.

3. These filters will appear in the filter list and workspace. Click on each filter, and…

4. Use the Edit filter fields to specify the type of filter. In this scenario both the lineNumber and machineNumber will be used to filter, and only data that matches a specifie line and machine number should be returned. So both filters us the EQUAL relational operator.

5. The resulting WHERE clause of the query can be viewed at any time at the bottom of the configuration window. Any field showing a question mark will be passed in at runtime as a paramters in the connection.



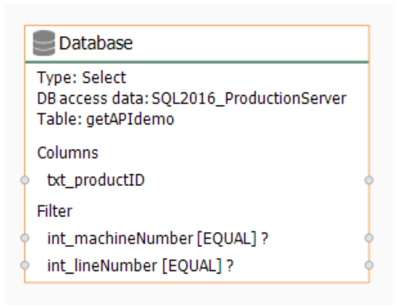
DB access data: SQL2016_ProductionServer
Type: Select

Filter
[int_machineNumber] EQUAL
[int_lineNumber] EQUAL

Edit filter
Column: int_lineNumber (Int32)
Relational operator: EQUAL

Preview
WHERE (int_machineNumber = ? AND int_lineNumber = ?)

Press OK on both Database transfer object configuration dialogs to complete the configuration. The Database transfer object in the configuration will now give access to both the productID field resulting from the constructed query, as well as the machineNumber and lineNumber filter fields that need to be provided as input parametesr in order for the query to return the correct result set.



Database
Type: Select
DB access data: SQL2016_ProductionServer
Table: getAPIdemo
Columns
txt_productID
Filter
int_machineNumber [EQUAL] ?
int_lineNumber [EQUAL] ?

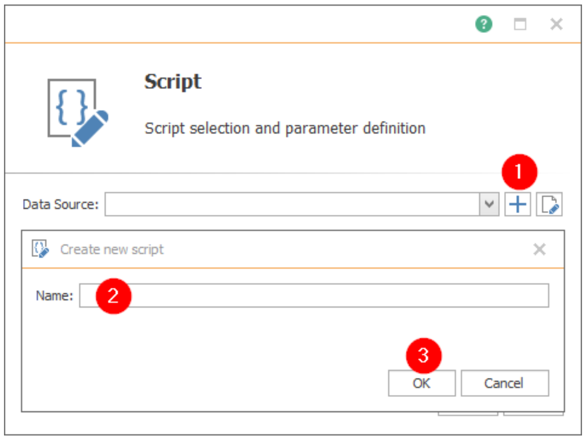
With the transfer objects corresponding to the three plug-ins, that were originally created, now added to the connection and configured, two more objects are needed.

Script Transfer Object

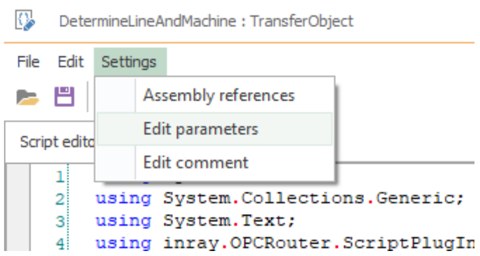
The first additional transfer object is a script object that will be used to run some logic on the four CurrentSpeed tags, that are being read from the OPC Server, in order to determine which should be returned.

Double click on the Script transfer object in the connection workspace to begin the configuration.

1. Click the add button to create a new script.
2. Give the script a name. This will be how the script is referenced in the OPC Router configuration, within any connection that references the script.
3. Press the OK button to complete the creation and launch the script editor.



Within the Script editor navigate to *Settings > Edit parameters* (on the top toolbar) to modify the input and output parameters for the script.



```
using System.Collections.Generic;
using System.Text;
using inray.OPCRouter.ScriptPlugIn
```

Create the following parameters by providing a parameters name (1), the parameter data type (2), and the direction in which the parameters should be available (3)

1. RequestedSystem – this will be the System ID that the HTTP Client passed into the OPC Router.

2. RequestedMachine – this will be the Machine ID that the HTTP Client passes into the OPC Router.

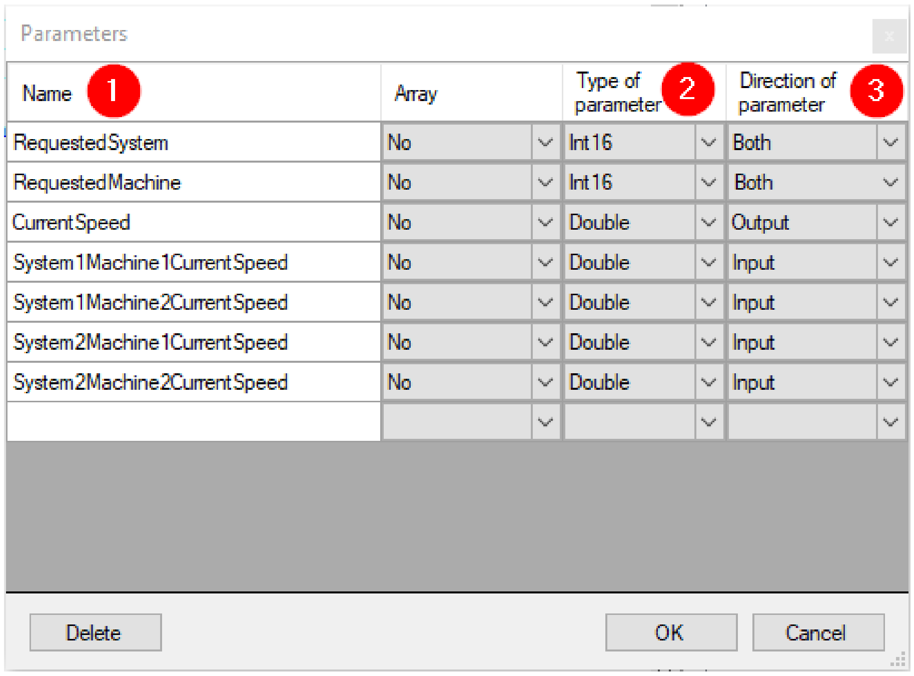
3. CurrentSpeed – this will be the output of the script, and will reflect the current value of one of the four speed values read from the OPC Server

4. System1Machine1CurrentSpeed – this will be the Current speed read from the OPC Server for System 1 Machine 1.

5. System1Machine2CurrentSpeed – this will be the Current speed read from the OPC Server for System 1 Machine 2.

6. System2Machine1CurrentSpeed – this will be the Current speed read from the OPC Server for System 2 Machine 1.

7. System2Machine2CurrentSpeed – this will be the Current speed read from the OPC Server for System 2 Machine 2.



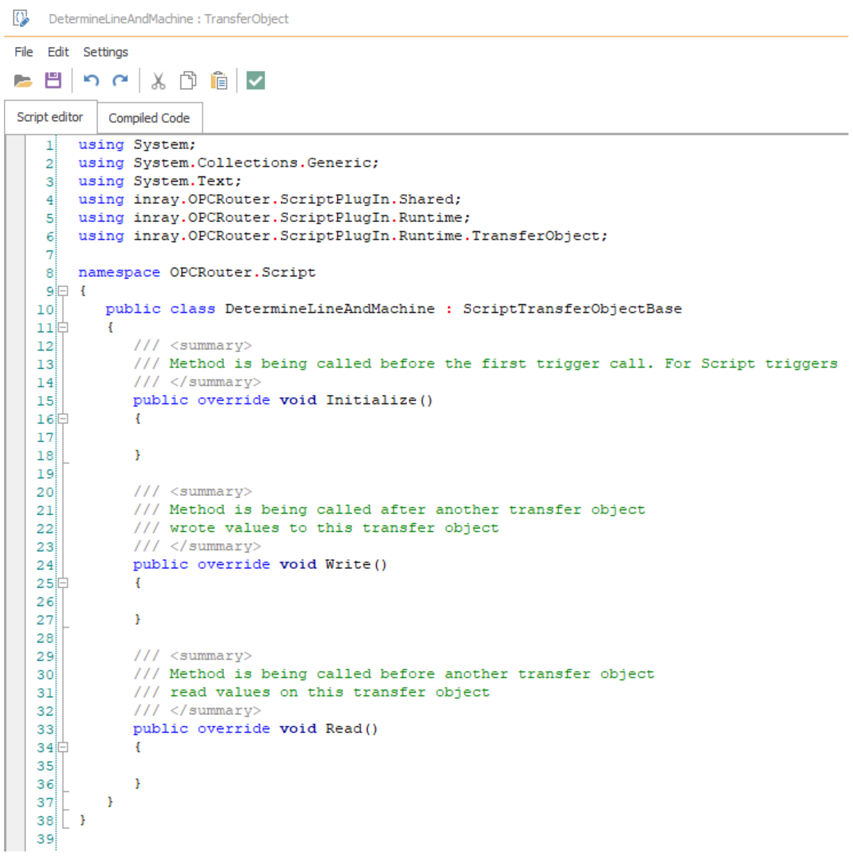
Name	Array	Type of parameter	Direction of parameter
RequestedSystem	No	Int16	Both
RequestedMachine	No	Int16	Both
CurrentSpeed	No	Double	Output
System1Machine1CurrentSpeed	No	Double	Input
System1Machine2CurrentSpeed	No	Double	Input
System2Machine1CurrentSpeed	No	Double	Input
System2Machine2CurrentSpeed	No	Double	Input

The empty script contains 3 methods by default;

1. The Initialize method which is called when the transfer object is first initialized (when the connection is first set productive)

2. The Write method, which is called when an external Transfer object writes/pushes data to the Script (i.e. when the data flow arrows are pointed into the script object)

3. The Read method, which si called when an external Transfer object reads/pulls data from the Script (i.e. when the data flow arrows are pointed out of the script object)

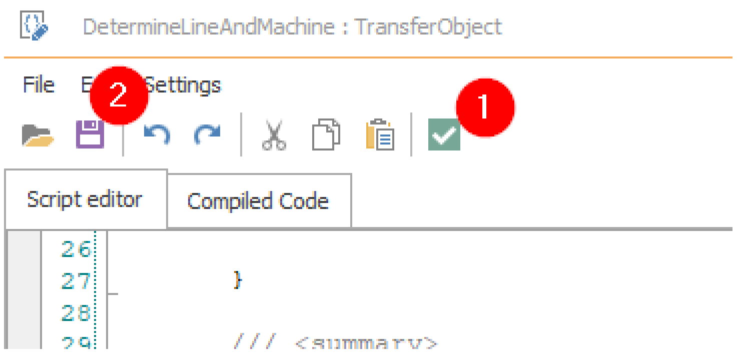


```
using System;
using System.Collections.Generic;
using System.Text;
using inray.OPCRouter.ScriptPlugIn.Shared;
using inray.OPCRouter.ScriptPlugIn.Runtime;
using inray.OPCRouter.ScriptPlugIn.Runtime.TransferObject;

namespace OPCRouter.Script
{
    public class DetermineLineAndMachine : ScriptTransferObjectBase
    {
        /// <summary>
        /// Method is being called before the first trigger call. For Script triggers
        /// </summary>
        public override void Initialize()
        {

        }

        /// <summary>
        /// Method is being called after another transfer object
        /// wrote values to this transfer object
        /// </summary>
        public override void Write()
        {

        }

        /// <summary>
        /// Method is being called before another transfer object
        /// read values on this transfer object
        /// </summary>
        public override void Read()
        {

        }
    }
}
```

In this case the script should fire whenever the current speed is requested by a client – which will serve as a read request. There are many ways to write the logic for this script – the following is but one of many possible implementations of the Read method. The basic code checks the system ID, and then the machine ID, and sets the CurrentSpeed output tag to be the current value of the corresponding System/Machien speed.

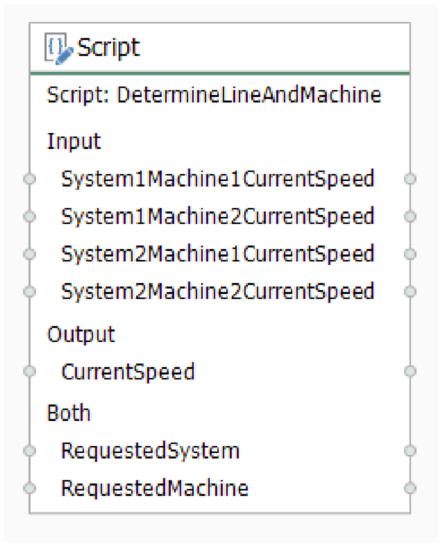
```
public override void Read()
{
        switch(RequestedSystem)
        {
                case 1:
                        switch(RequestedMachine)
                        {
                                case 1:
                                        CurrentSpeed = System1Machine1CurrentSpeed;
                                        break;
                                case 2:
                                        CurrentSpeed = System1Machine2CurrentSpeed;
                                        break;
                                default:
                                        CurrentSpeed = 0;
                                        break;
                        }
                        break;
                case 2:
                        switch(RequestedMachine)
                        {
                                case 1:
                                        CurrentSpeed = System2Machine1CurrentSpeed;
                                        break;
                                case 2:
                                        CurrentSpeed = System2Machine2CurrentSpeed;
                                        break;
                                default:
                                        CurrentSpeed = 0;
                                        break;
                        }
                        break;
                default:
                        CurrentSpeed = 0;
                        break;
        }
}
```

With the script written, validate that it compiles successfully (1) and then save (2) and close the script.



```
        }

        /// <summary>
```

The script transfer object will now appear in the connection workspace, with the expected input and output parameters accessible as such.


Script
Script: DetermineLineAndMachine
Input
System1Machine1CurrentSpeed
System1Machine2CurrentSpeed
System2Machine1CurrentSpeed
System2Machine2CurrentSpeed
Output
CurrentSpeed
Both
RequestedSystem
RequestedMachine

JSON Write Transfer Object

Finally, the JSON Write object can be configured to consolidate all parameters that will be returned to the HTTP client and construct the JSON Document that will be the payload of the HTTP response.

Double click the JSON Write transfer object to begin the configuration.

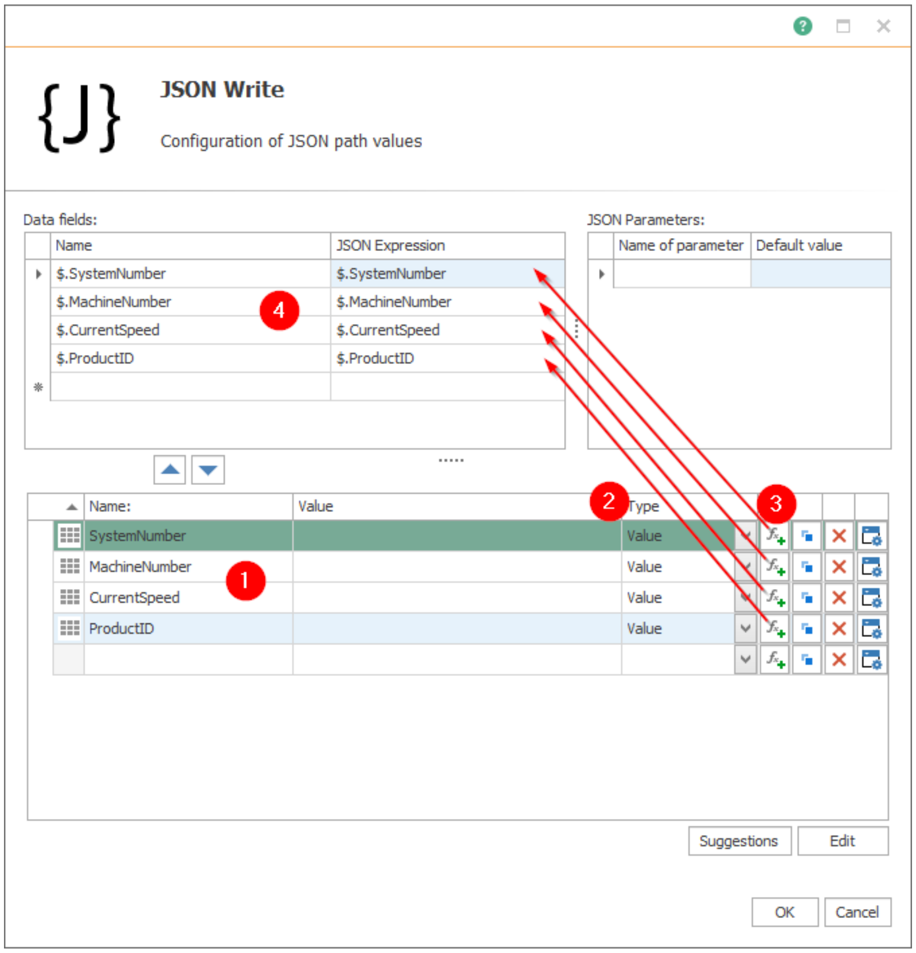
1. Add any parameters that should be returned as part of the JSON payload along with any default values that should be used. These are the parameters that are accessible in the OPC Router configuration.

2. Specify the type of JSON value

3. Use the function button to automatically add the configured parameters to the data fields section that will be used in the JSON Document/payload

4. Verify that all desired parameters (that should be returned to the HTTP Client) are added to the Data fields section, and that they have a valid JSON Expression associated with them. When using the function button, the JSON Expression is configured correctly automatically.

Press OK to finish the JSON Write transfer object configuration.



Data fields:

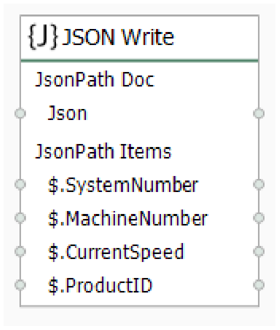
Name	JSON Expression
$.SystemNumber	$.SystemNumber
$.MachineNumber	$.MachineNumber
$.CurrentSpeed	$.CurrentSpeed
$.ProductID	$.ProductID

JSON Parameters:

Name of parameter	Default value

Name:	Value	Type
SystemNumber		Value
MachineNumber		Value
CurrentSpeed		Value
ProductID		Value

The JSON Write Transfer object will now be accessible in the connection workspace, and expose the configured parameters, as well as a data field to access the constructed JSON document.



Linking Transfer Objects

With all transfer objects now instantiated and configured in the connection, the data flow between object fields can now be drawn – these arrows will represent the flow of data between transfer objects.

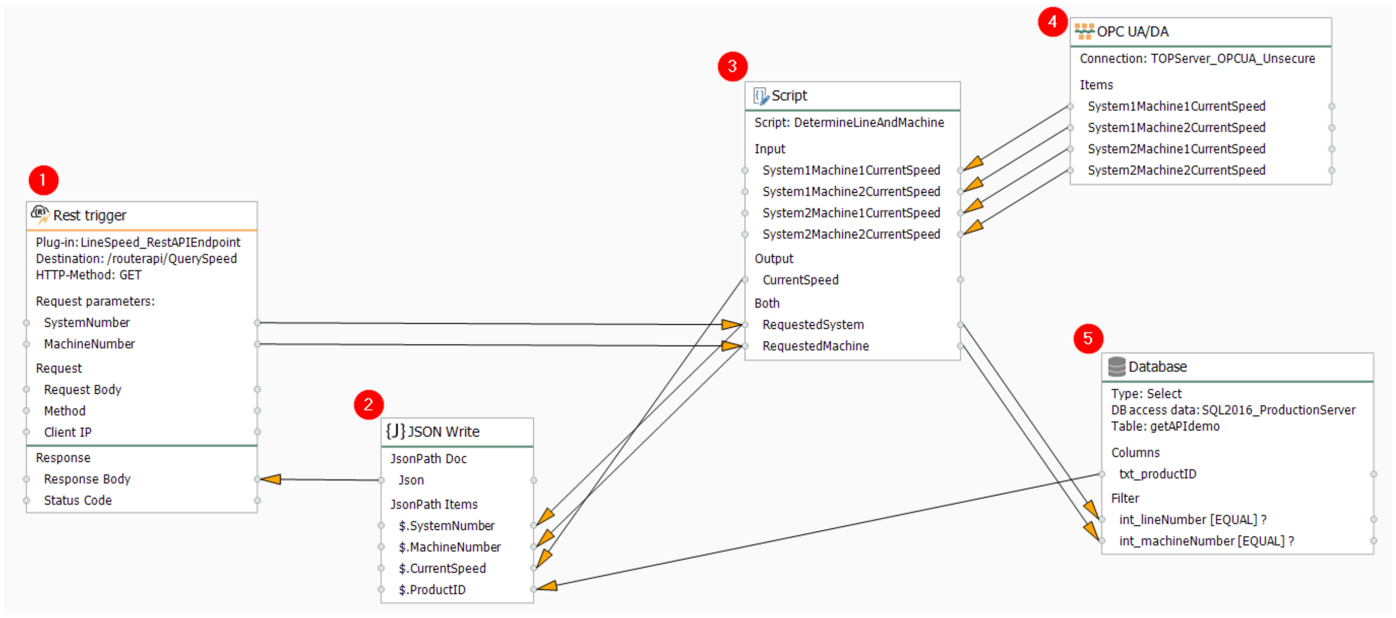
1. **Rest trigger fields** – The SystemNumber and MachineNumber fields are provided as query parameters by the HTTP Client and will be used to drive the logic in the connection. The Response Body will be the content of the Json Document that is constructed in the JSON Write transfer object (2)

2. **JSON Write fields** – The Json Document serves as the only output of the JSON Write transfer object, and is constructed from the SystemNumber (passed from the Script Object), MachineNumber (also passed from the Script Object), CurrentSpeed (Calculated in the script object), and ProductID (passed from the database object) fields. The MachineNumber and SystemNumber fields are provided from the Script object as opposed to being passed from the Rest trigger directly purely from an organization standpoint; to make the data flow easier to read; having these fields come from the Rest trigger directly is also valid.

3. **Script fields** – The script serves as the primary hub of data in the configuration. The System and Machine numbers are passed from the Rest trigger (where they are provided by the HTTP Client) and serve as the logic triggers to determine which of the four speeds passed from the OPC Server object should be provided to the requesting HTTP Client. The system and machine numbers are then also passed to other objects in the configuration to be used as needed.

4. **OPC UA/DA fields** – The OPC UA/DA object has four output fields – which correspond to the four line/machine speed combinations. This are passed to the Script object for analysis and processing.

5. **Database fields** – The txt_productID field is queried from the SQL database, and is passed directly to the JSON Write object. The only input parameters to the database object are the two values that are used in the WHERE clause of the query – the line/system number and the machine number. Just like the JSON Write Object, the MachineNumber and SystemNumber fields are provided from the Script object as opposed to being passed from the Rest trigger directly to help with the data flow organization, and to make it easier to read; Having these fields come from the Rest trigger directly is also perfectly valid.


1
Rest trigger
Plug-in: LineSpeed_RestAPIEndpoint
Destination: /routerapi/QuerySpeed
HTTP-Method: GET
Request parameters:
SystemNumber
MachineNumber
Request
Request Body
Method
Client IP
Response
Response Body
Status Code
2
{J} JSON Write
JsonPath Doc
Json
JsonPath Items
$.SystemNumber
$.MachineNumber
$.CurrentSpeed
$.ProductID
3
Script
Script: DetermineLineAndMachine
Input
System1Machine1CurrentSpeed
System1Machine2CurrentSpeed
System2Machine1CurrentSpeed
System2Machine2CurrentSpeed
Output
CurrentSpeed
Both
RequestedSystem
RequestedMachine
4
OPC UA/DA
Connection: TOPServer_OPCUA_Unsecure
Items
System1Machine1CurrentSpeed
System1Machine2CurrentSpeed
System2Machine1CurrentSpeed
System2Machine2CurrentSpeed
5
Database
Type: Select
DB access data: SQL2016_ProductionServer
Table: getAPIdemo
Columns
txt_productID
Filter
int_lineNumber [EQUAL] ?
int_machineNumber [EQUAL] ?

Going Productive

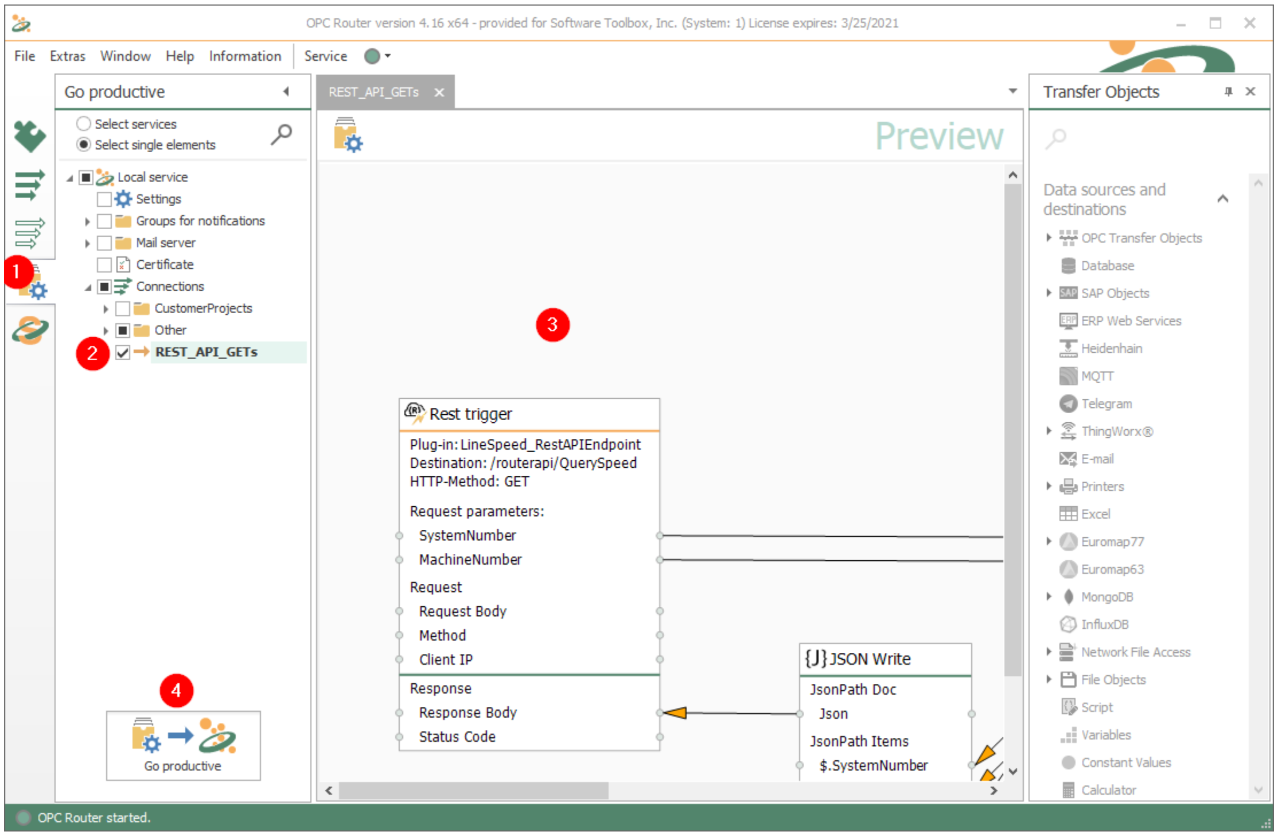
In order to deploy the configuration to the runtime service it must be set productive.

1. Navigate to the production tab

2. Check the connections that should be deployed to the runtime service for production.

3. Review the connection that will be deployed to confirm it is the right one(s)

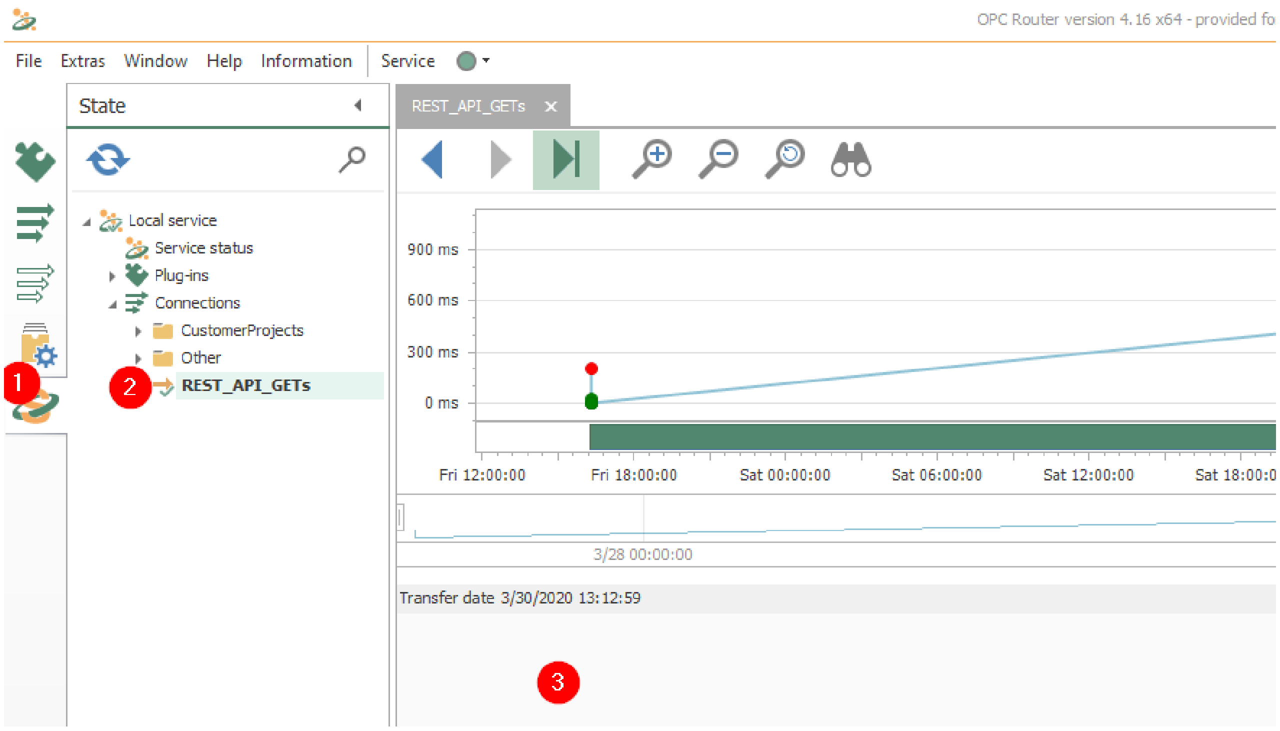
4. Use the *Go productive* button to deploy the connection to the runtime service. Read and acknowledge any resulting prompts.



Testing the configuration

In order to check the running state of the connection the state view can be used.

1. Navigate to the state view in the OPC Router
2. Select the connection that should be monitored from the tree view
3. Use the workspace here to monitor triggered occurrences, as well as the resulting values that were used throughout the connection



Transfer date 3/30/2020 13:12:59

The HTTP Client that is being used in this case is Postman. This is a test client only, and how the HTTP call is 'constructed' in product will depend on what HTTP Client is being used.

Postman was configured with the following settings:

1. The request method is set to GET
2. The URL and endpoint reflect the settings specified in the OPC Router REST API plugin and REST Trigger object configurations

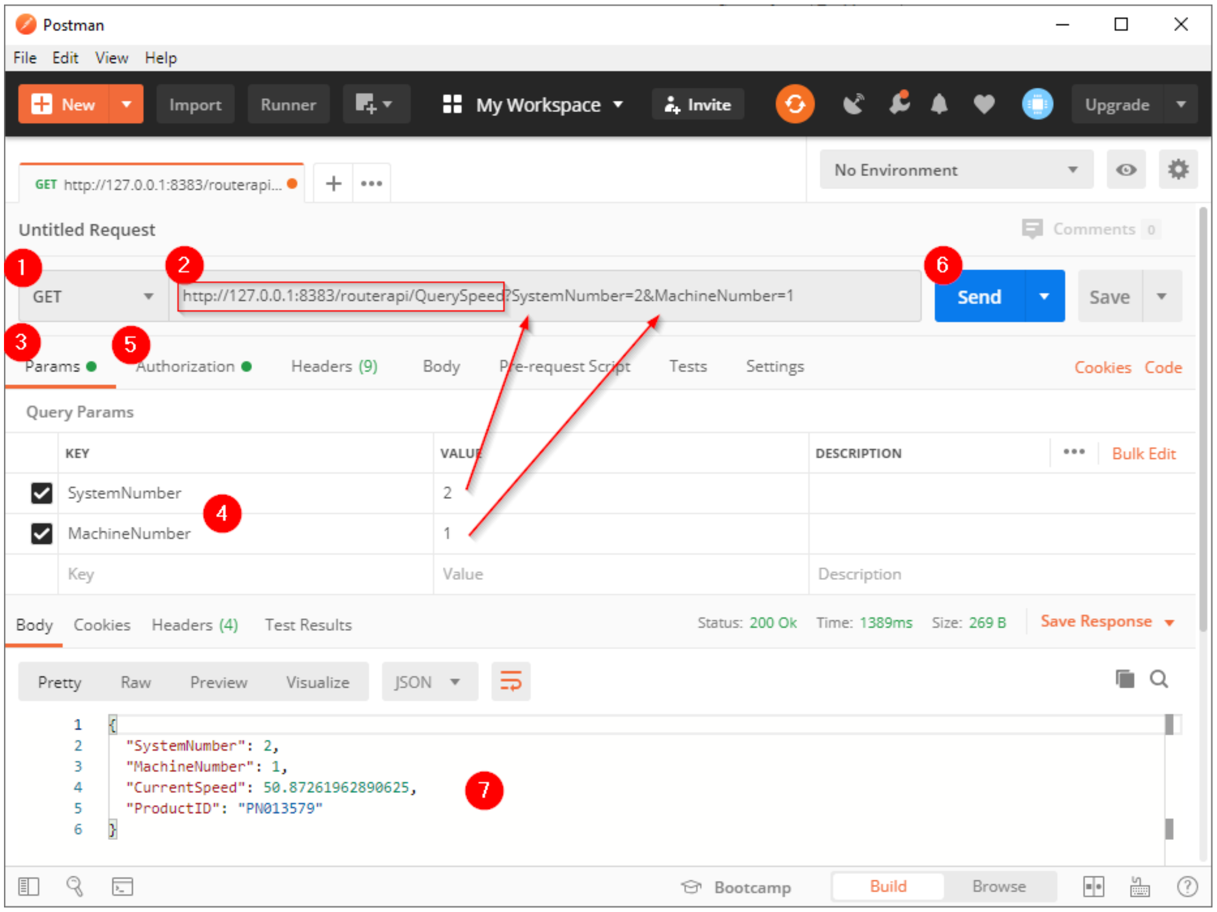
3. On the Parameter tab the query parameters that will be passed as part of the query can be specified

4. Create the two Query Parameters for SystemNumber and MachineNumber, and specify the values that should be passed as part of the GET request to the OPC Router. As it is configured here the Speed of System 2 and Machine 1 will be returned. Updating the keys here will automatically update the request URL

5. On the Authorization tab the security/authorization settings must be set to match those set in the OPC Router. (in this demo; Basic Authentication with a username of user and a password of 1234)

6. Use the Send button the send the query

7. If successful, the results will be shown. As expected (and configured in the JSON Write transfer object, the SystemNumber, Machine Number, CurrentSpeed, and ProductID fields are returned)



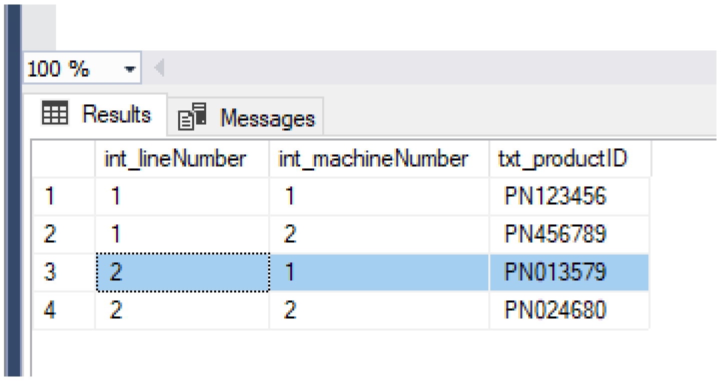
GET http://127.0.0.1:8383/routerapi/QuerySpeed?SystemNumber=2&MachineNumber=1

	KEY	VALUE	DESCRIPTION
✓	SystemNumber	2	
✓	MachineNumber	1	
	Key	Value	Description

Status: 200 Ok Time: 1389ms Size: 269 B

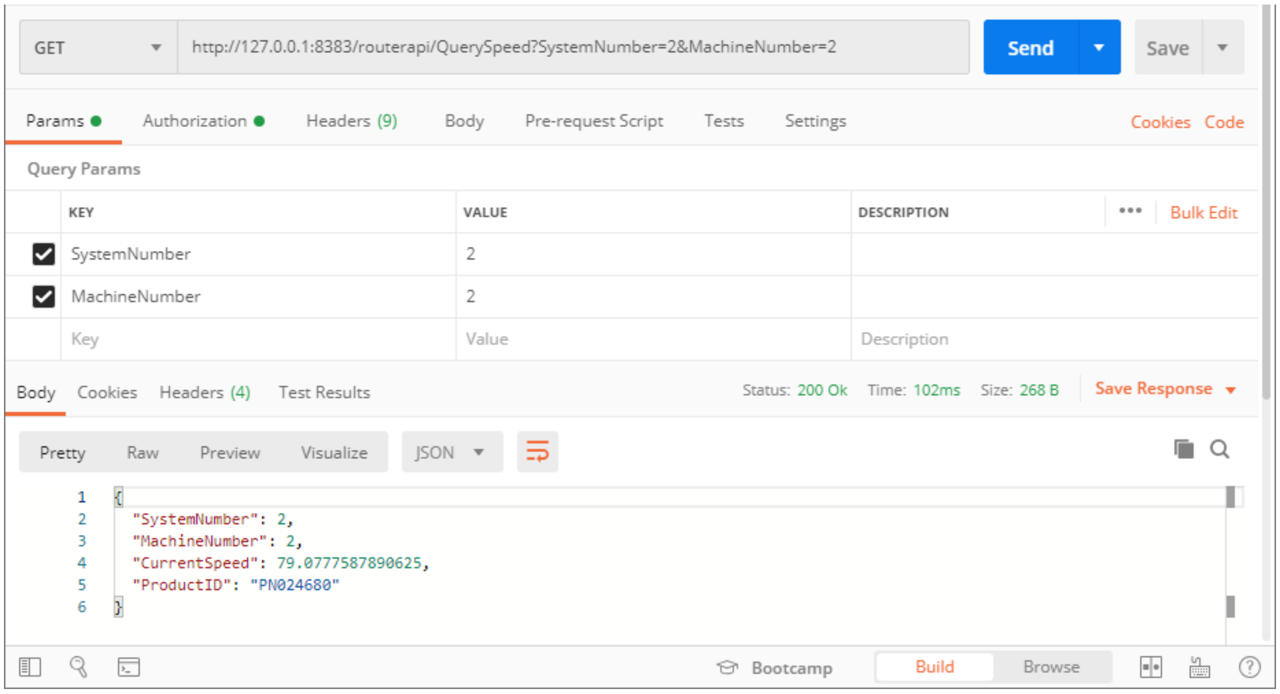
```
{
  "SystemNumber": 2,
  "MachineNumber": 1,
  "CurrentSpeed": 50.87261962890625,
  "ProductID": "PN013579"
}
```

When compared to the content of the SQL Table. It can be confirmed that the productID returned *was,* in-fact, queried from SQL based on the System/line and Machine Numbers.



	int_lineNumber	int_machineNumber	txt_productID
1	1	1	PN123456
2	1	2	PN456789
3	2	1	PN013579
4	2	2	PN024680

Modifying the SystemNumber and/or MachineNumber query parameters is accurately reflected in the returned ProductID field.



http://127.0.0.1:8383/routerapi/QuerySpeed?SystemNumber=2&MachineNumber=2

	KEY	VALUE	DESCRIPTION
✓	SystemNumber	2	
✓	MachineNumber	2	
	Key	Value	Description

Status: 200 Ok Time: 102ms Size: 268 B

```
{
    "SystemNumber": 2,
    "MachineNumber": 2,
    "CurrentSpeed": 79.0777587890625,
    "ProductID": "PN024680"
}
```

Conclusion

This document has been meant to serve as a step by step guide to configure the OPC Router to expose an HTTP RESTful API endpoint which can then be queried by HTTP clients to read data from the OPC Router. Hopefully the configuration steps taken here can be applied to other – similar – OPC Router installations and applications. While the OPC Router documentation is excellent, the Software Toolbox team is ready to help with any questions. Please don't hesitate to reach out to the technical team at the information below:

Software Toolbox Hours: Monday - Friday, 08:00 - 17:00 US EST

Phones: +1 704 849 2773

Email: support@softwaretoolbox.com

Website: https://support.softwaretoolbox.com (The *Ask a Question* button can be used to quickly, and easily submit a question to the technical team)

888 665 3678	TOLL FREE
+1 704 849 2773	GLOBAL
Charlotte, NC USA	GLOBAL HQ
www.softwaretoolbox.com	WEB

Our mission is to provide you with the right software package to solve your industrial operation challenges.