Functionality of the Consistent Message Transporter App

Siemens Industry Online Support

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.
Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products.
The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness, and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.
By using the application examples, you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.
The Siemens terms of use (https://support.industry.siemens.com) shall also apply.

Security information

Siemens provides products and solutions with Industrial Security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.
Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the Internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.
For additional information on industrial security measures that may be implemented, please visit https://www.siemens.com/industrialsecurity.
Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at: https://www.siemens.com/industrialsecurity.

Table of contents

Table of figures

1 Introduction

1.1 Overview

The Consistent Message Transporter is a REST server based on the HTTPS protocol and the REST API structure. It is available to establish the connection with the data source of WinCC (v7.5 or Unified). The REST API Server contains a web server to offer all the tags of the data source. They can be read, written, or subscribed. It also contains a web client that sends changed tag values to foreign REST web servers. This makes possible to exchange data and info with existing applications, for example, the OpCenter Server.

The Figure 1 shows the System Overview of the whole Project.

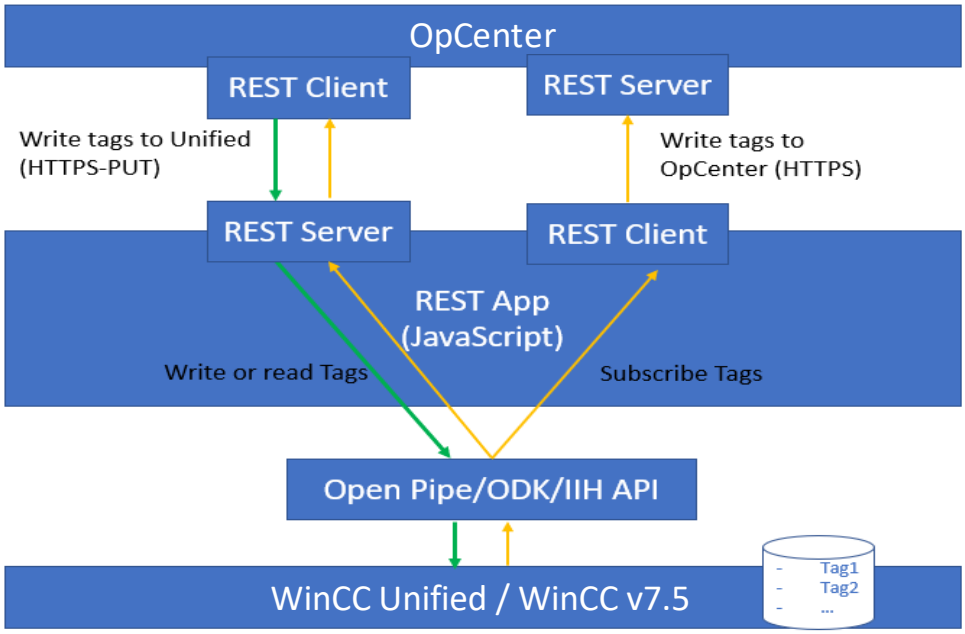


Figure 1 - System Overview

This guide will help you in the configuration of the Consistent Message Transporter as well as reviewing several scenarios to test manually if the server works according to the expectations.

1.2 Components used

This application example has been created with the following software components:

Table 1-1

Component	Number
Source code	You can find the files at the table 5-1 entry #7
OS	Windows 10
IDE	Visual Studio Code
Node.js	32-bit version
Node model	`npm install - -s express express-jwt log4js MD5 xml2js xmlhttprequest xslt-processor`
Data source	WinCC ODK and V7.5, WinCC Open Pipe and Unified
Certificate Generation Tool	Openssl 1.0.0f (32bit or64bit)

1.3 Use cases and benefits

The main advantages of REST APIs, compared to other tools, are the easiness to integrate in every type of systems, the high level of scalability and the independence with several sections of the project.

This system, as it is described before, is based on the REST API architecture with all the benefits of a regular REST API applications. But the added value of this tool, is the consistent message mechanism which is explained below in more detail and the possibility to communicate with WinCC v7.5 and WinCC Unified systems because, there is no other tool which can achieve this right now

Possible use cases where this tool can be useful are:

- Do a consistent order download from SAP or MOM via WinCC to PLCs.

- Several clients need to access to the same data at the runtime.

2 Prerequisites

2.1 Installation of node.js and npm

The tool used to run the server is node.js. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on a JavaScript Engine and executes JavaScript code outside a web browser, which was designed to build scalable network applications.

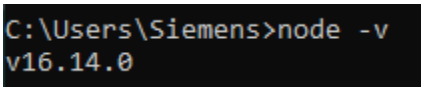
This tool it is available to download at the link #5 in the table 5-1. Notice that 32-bit is needed because of C++ components.

Note: In order to install node.js in the system, Administrator Rights are needed.

Once the download is completed, an executable file is generated and leaving the installation values as default is more than okay to achieve a successful installation. To check if the installation is successful type in the command line:

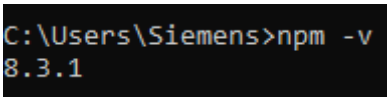
```
node -v
```

If everything is working as expected an answer like this it is returned.



The Node Package Manager (npm) is already installed with the node.js. To check the version of npm, type:

```
npm -v
```



Note: Both versions can be different due to updates and are the latest at the moment this document is being written.

2.2 Certificate generation

To enable SSL verification, we have to create the CA certificate and extend the domain name supported by the certificate. Here we are using Openssl 1.0.0f tool to create all the certificates. Openssl is available to download in the link #4 at the table 5-1.

Note: The certificates used in this guide were created using the version 1.0.0f of OpenSSL. This version is supported until 11th September 2023, but new LTS (Long Term Support) versions are available, e.g. the 3.0 series.

1. Open the installation file of OpenSSL (.exe) and click “next”, leaving the configuration values as default, until the installation is complete.

2. Open “**Environment values**” in “**Advanced system settings**” and add in the “**User variables**” inside the “**Path**” section, the path where the OpenSSL is installed. In this case, the default path is “**C:\OpenSSL-Win32\bin**”.

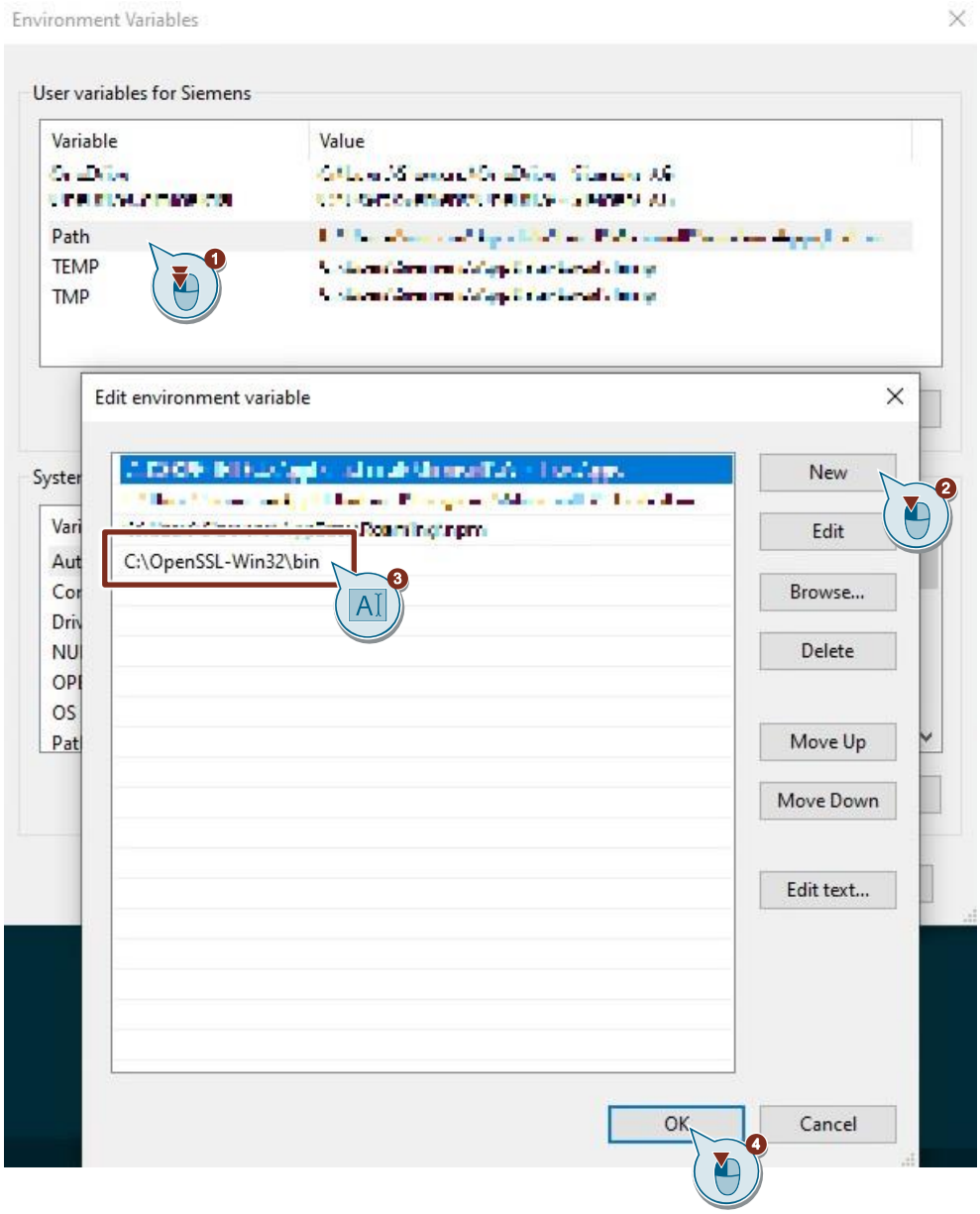


Figure 2 - OpenSSL configuration

3. Also, in the "**Environment values**", include the same path "**C:\OpenSSL-Win32\bin**" in the "**System variables**" inside the "**Path**" section. Furthermore, you must create a system variable.

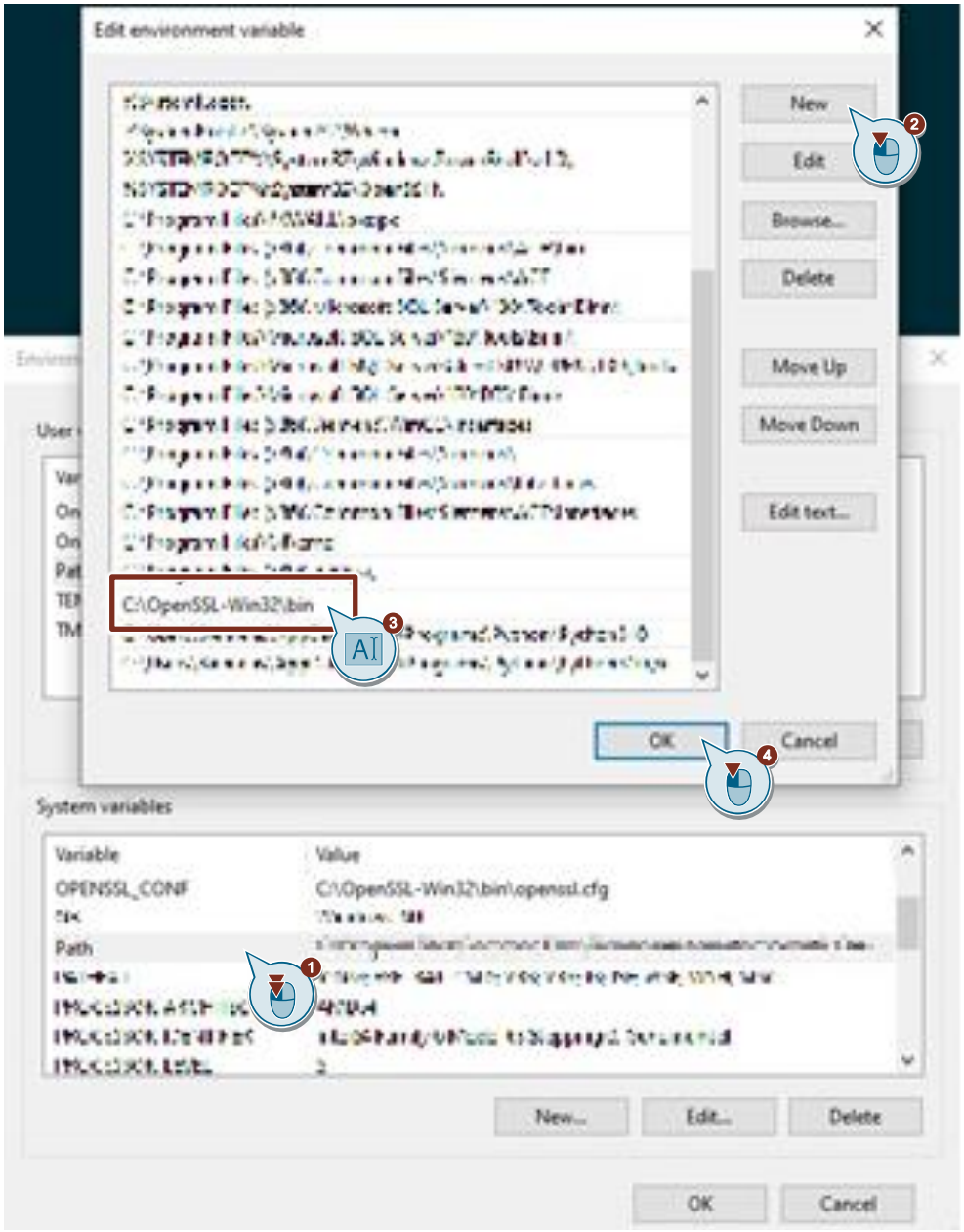


Figure 3 - OpenSSL configuration (2)

4. Unzip the file attached to this document where the code is going to be.

5. Once the repository is unzipped, inside the folder "**C:\<…>\src\certificate**" go to the file "**san.cnf**" and put the IP address of the computer where the application is going to be installed, in the fields "**commonName**" and "**IP.1**". After changing the addresses, save the file.

Note:

The IP address can be checked by opening the terminal and type `ipconfig`. There, the IPv4 address of **Ethernet adapter** will be displayed.

6. Then, inside the same path as the previous point, "**C:\<…>\src\certificate**", open the cmd again and type:

```
openssl req -x509 -nodes -days 36500 -newkey rsa:2048 -keyout key.pem -out cert.pem -config san.cnf
```

7. You will get CA self-signed certificate, now create Server private key as typing:

```
openssl rsa -in key.pem -out server.key
```

8. Then, create your server certificate by typing:

```
openssl x509 -in cert.pem -out server.crt
```

9. Transfer .crt file into.pfx file and enter an "**export password**" for your certificate by typing:

```
openssl pkcs12 -export -out server_cert.pfx -inkey server.key -in server.crt
```

Note:

An executable file is provided to avoid writing these commands. The executable file name is certificateGenerator.bat

10. Create a new .txt file named "servercertpassword.txt" and put your Export password inside. And now, in new certificate folder, you should see these file:

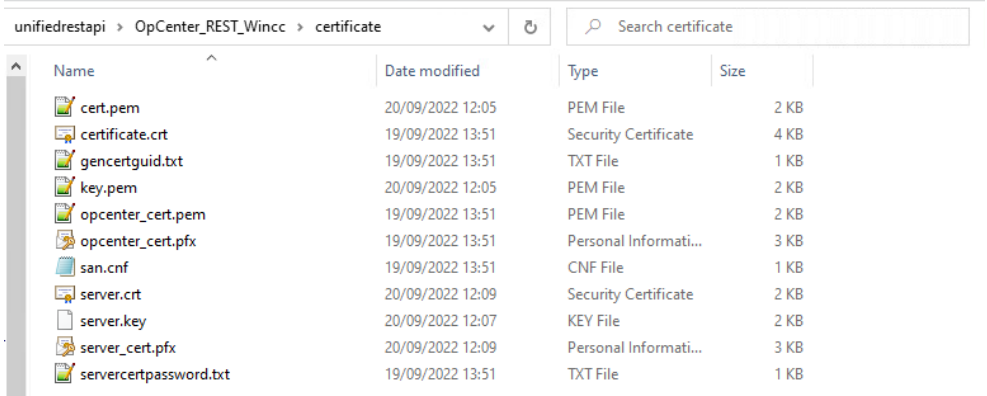


Figure 4 - The folder where the certificates are saved

It is important to store the password in plaintext because the Consistent Message Transporter will need it and it is needed in order to import the certificate in the step number 12.

Note:

Notice that you are only allowed to give cert.pem (CA certificate) and server_cert.pfx (your server certificate) to your client, do not give them key.pem because it is your CA private key!

11. The last step needed is adding the CA certificate to the CA storage. Press **Win+R** to enable “Run”, enter “certmgr.msc”, find the “**Trusted Root Certification Authorities**”, right click and find “**All Tasks**” and “**Import item**”, add “**cert.pem**” file (CA Certificate) inside like:

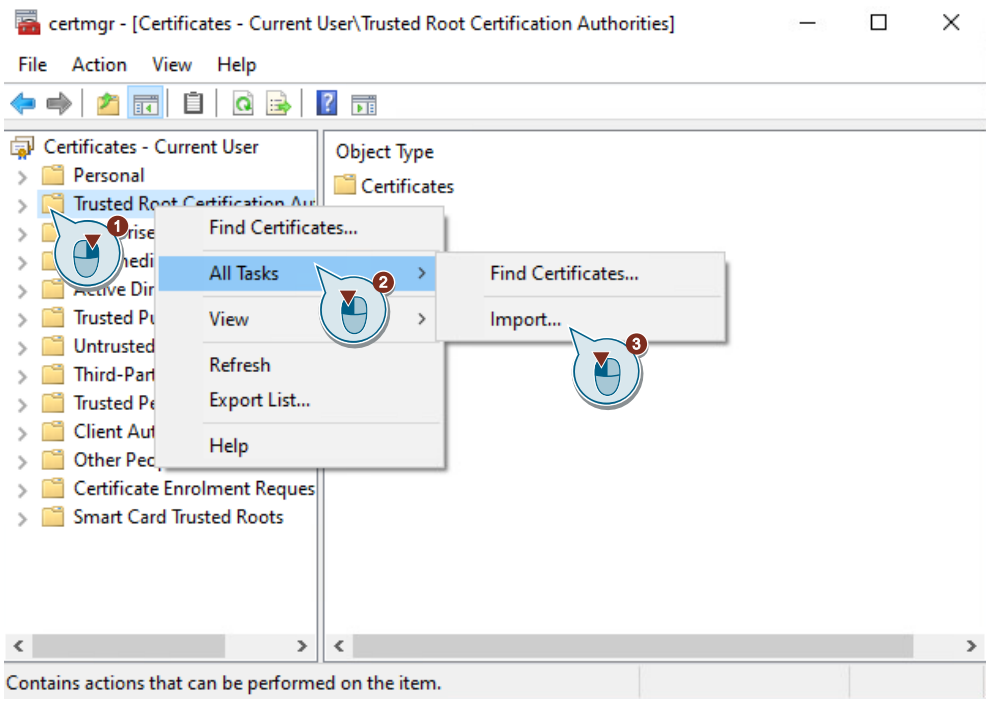


Figure 5 - Certificate manager (1)

12. Then, in the folder “**Personal**”, right click and find “**All Tasks**” and “**Import item**”, add “**server_cert.pfx**” file (your server Certificate) inside. In this case, it is needed to add the “**Export password**”. In case of being forgotten, it is available as plaintext in the “servercertpassword.txt” file.

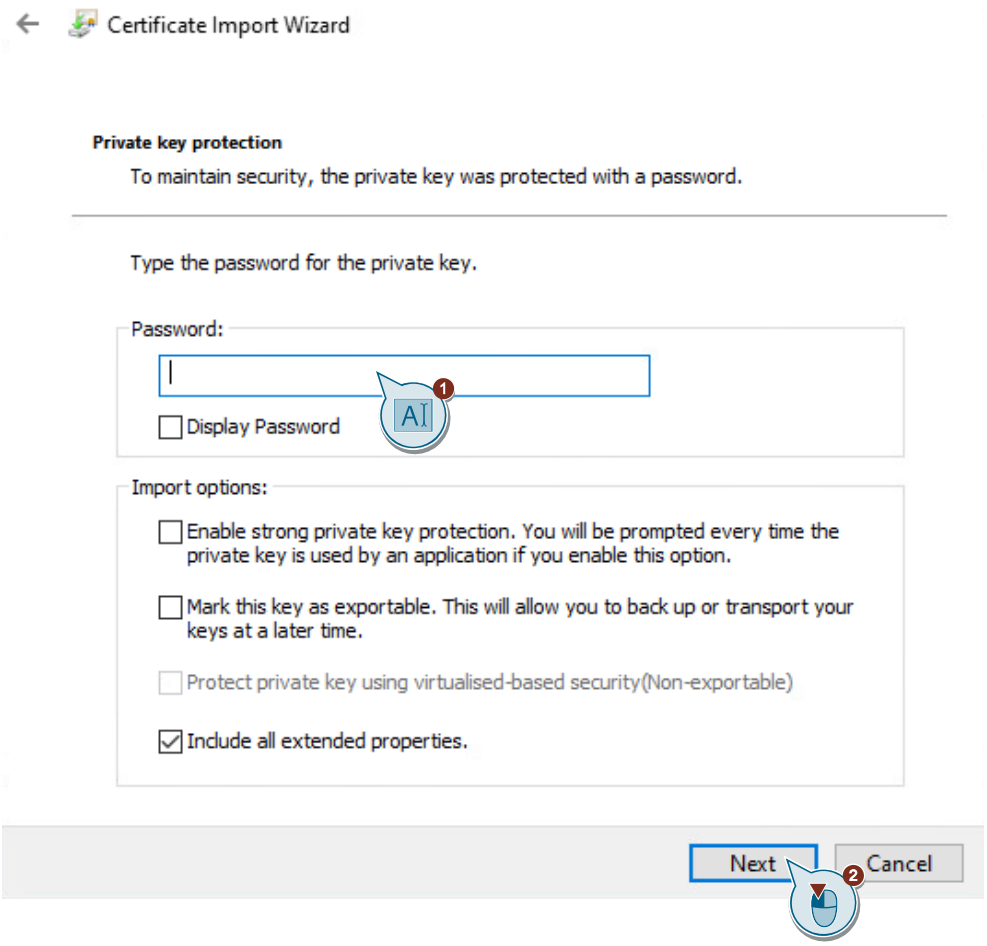


Figure 6 - Certificate manager (2)

13. Once this point is reached, the last step needed before starting the server, a few node modules are needed in order to launch the server successfully using these commands.

```
npm init -y
npm install
npm install - -s express express-jwt log4js MD5 xml2js
xmlhttprequest xslt-processor
```

2.3 Configuration of the system

2.3.1 The TagConfiguration.xml & WinccTagconfiguration.xml file

The “**TagConfiguration.xml**” file is used for transferring OpCenter Client defined Tag name into **WinCC Unified** Recognized Tag name. Similarly, the “**WinccTagconfiguration.xml**” file is used for transferring OpCenter Client defined Tag name into **WinCC V7.5** Recognized Tag name and adopted in scenarios 1, 2 and 3 (see chapter 3 below).

The Figure 7 and Figure 8 show the demo section of these two xml files.

TagConfiguration.xml

```
<?xml version="1.0" encoding='utf-8'?>
<TagList>
  <WebClient>
    <CertificateID>.\certificate\</CertificateID>
    <CommandName>https://10.20.230.21:443/sit-svc/Application/AppU4DM/odata/DVLApp_WriteWinCCTag/</CommandName>
    <CommandMode>POST</CommandMode>
    <TemplateBody>{"command":{"TagName":"Order_Tag_4","TagValue":"${Order_Tag_5}"}}</TemplateBody>
    <Tags>
     <Tag>
       <Trigger>Onchange</Trigger>
       <PlaceHolder>Order_Tag_4</PlaceHolder>
       <Name>HMI_Tag_4</Name>
       <Filter>&gt;10</Filter>
     </Tag>
     <Tag>
       <Trigger>Onchange</Trigger>
       <PlaceHolder>Order_Tag_5</PlaceHolder>
       <Name>HMI_Tag_5</Name>
     </Tag>
     <Tag>
       <Trigger>Onchange</Trigger>
       <PlaceHolder>Order_Tag_6</PlaceHolder>
       <Name>HMI_Tag_6</Name>
       <Filter>&lt;100</Filter>
     </Tag>
    </Tags>
  </WebClient>
```

Figure 7 - TagConfiguration.xml

WinccTagConfiguration.xml

```

<?xml version="1.0" encoding='utf-8'?>
<TagList>
  <WebClient>
    <CertificateID>.\certificate\</CertificateID>
    <CommandName>https://10.20.230.21:443/sit-svc/Application/AppU4DM/odata/DVLApp_WriteWinCCTag/</CommandName>
    <CommandMode>POST</CommandMode>
    <TemplateBody>{"command":{"TagName":"wincc1","TagValue":"${wincc1}"}}</TemplateBody>
    <Tags>
       <Tag>
        <PlaceHolder>wincc1</PlaceHolder>
        <Name>testBool</Name>
      </Tag>
      <Tag>
       <Trigger>Onchange</Trigger>
        <PlaceHolder>wincc2</PlaceHolder>
        <Name>testSigned32</Name>
         <Filter>&gt;100</Filter>
      </Tag>
       <Tag>
        <Trigger>Onchange</Trigger>
        <PlaceHolder>wincc3</PlaceHolder>
        <Name>testFloat64</Name>
        <Filter>&gt;100</Filter>
       </Tag>
```

Figure 8 - WinccTagConfiguration.xml

Table 2-1 lists all XML nodes and its Meaning:

Table 2-1

XML nodes	Meaning
<CertificateID>	The path of 'certificate' folder, where OpCenter certificate stored inside.
<CommandName>	The URL of the OpCenter Server.
<CommandMode>	The method that is used to visit the OpCenter Server. Currently, the POST method is possible.
<TemplateBody>	The JSON body that needs to be followed when offering Request to OpCenter Server. This can be adapted completely to any other structure. This is one of the benefits of our app.
<Tags>	An XML node contains several tags.
<Tag>	An XML node contains several tag parameters.
<Trigger>	If a tag contains the <Trigger> node and its value is "OnChange", then the value of this tag will be subscribed by the Consistent Message Transporter from WinCC Unified/V7.5. If a tag does not contain the <Trigger> node, then its value needs to read from WinCC Unified/V7.5.
<PlaceHolder>	The tag names that are known by the OpCenter Server, which is corresponding to the content in the <Name> node.
<Name>	The tag names that are known by WinCC Unified/V7.5, which is corresponding to the content in the <PlaceHolder> node.

<Filter>	If the tag is subscribed from WinCC Unified/V7.5 and the value compared to the <Filter> node is true, we send the latest value to the OpCenter Server. If the tag does not contain the <Filter> but has the <Trigger> with the value of "OnChange", we send the latest value with no filter.
	Examples:
	Filter greater than 100
	`<Tag>`
	`<Trigger>Onchange</Trigger>`
	`<PlaceHolder>wincc2</PlaceHolder>`
	`<Name>testSigned32</Name>`
	`<Filter>>100</Filter>`
	`</Tag>`
	Filter lower than 100
	`<Tag>`
	`<Trigger>Onchange</Trigger>`
	`<PlaceHolder>wincc2</PlaceHolder>`
	`<Name>testSigned32</Name>`
	`<Filter><100</Filter>`
	`</Tag>`

2.3.2 The RestServerConfiguration_OpCenter.xml file

The “RestServerConfiguration_OpCenter.xml” file is used to identify the OpCenter Clients Request Body to enable the consistent message mechanism with WinCC V7.5. This file will be adopted in scenarios 1 and 2 (see chapter 3 below).

Figure 9 shows the demo section of “RestServerConfiguration_OpCenter.xml”.

RestServerConfiguration_OpCenter.xml

```
<TagList>
 <WebClient>
  <CommandName>/WinCCRestService/TagManagement/Values</CommandName>
  <CommandMode>PUT</CommandMode>
  <TemplateBody>{"OrderName": "${OrderName}", "Equipment": "${Equipment}", "PlannedStartTime": "${PlannedStartTime}", "PlannedEndTime": "${PlannedEndTime}"}
  </TemplateBody>
  <AcknowledgeTag>NewOrder_ack</AcknowledgeTag>
   <!-- subscribe only to this on startup -->
   <SequenceTag>NewOrder_seq</SequenceTag>
    <!-- read once at startup and count up with each message -->
   <Tags>
    <Tag>
     <PlaceHolder>OrderName</PlaceHolder>
     <Name>testText8</Name>
    </Tag>
    <Tag>
     <PlaceHolder>Equipment</PlaceHolder>
     <Name>testText16</Name>
    </Tag>
    <Tag>
     <PlaceHolder>PlannedStartTime</PlaceHolder>
     <Name>testSigned32</Name>
    </Tag>
    <Tag>
     <PlaceHolder>PlannedEndTime</PlaceHolder>
     <Name>testFloat64</Name>
    </Tag>
```

Figure 9 - RestServerConfiguration_OpCenter.xml

Table 2-2 lists all XML nodes and its Meaning

Table 2-3

XML nodes	Meaning
<CommandName>	The write URL of the REST Server.
<CommandMode>	The method that is used to visit our REST Server. Currently, the PUT method is possible.
<TemplateBody>	The template JSON body each Opcenter Client offers.
<AcknowledgeTag>	The Tag Subscribed Tag from WinCC V7.5, telling whether the consistent message succeeded
<SequenceTag>	The Tag which will write down to WinCC V7.5 while REST Server received Req
<Tags>	An XML node contains several tags.
<Tag>	An XML node contains several tag parameters.
<Trigger>	If a tag contains the <Trigger> node and its value is “OnChange”, then the value of this tag needs to be subscribed from WinCC V7.5. If a tag does not contain the <Trigger> node, then its value needs to read from WinCC V7.5.
<PlaceHolder>	The tag names that can be known by the OpCenter Server, which is corresponding to the content in the <Name> node.

XML nodes	Meaning
<Name>	The tag names that can be known by WinCC V7.5, which is corresponding to the content in the <PlaceHolder> node.
<Filter>	If the tag needs to be subscribed from WinCC V7.5 and the value is larger than the value in the <Filter> node, we send the latest value to the OpCenter Server. If the tag does not contain the <Filter> but has the <Trigger> with the value of “OnChange”, we send the latest value with no filter.

2.3.3 OpCenter configuration

Opcenter Execution Process provides OData API to commands call, for getting data stored in the Opcenter Execution database. To call OData methods and get the data user should be authenticated and authorized. To do this user calls dedicated endpoint URI with “user” and “password” provided and receives bearer token as a response. This token is valid for 10 minutes and will be used by the Rst App with every OData method call.

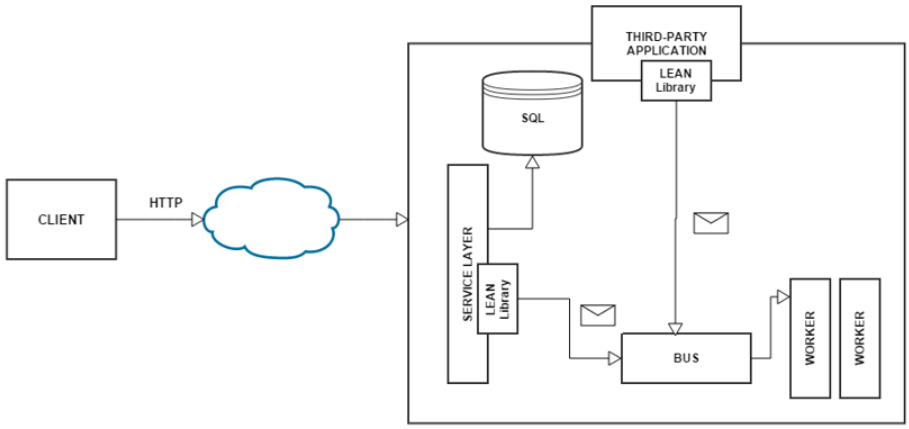


Figure 10 - OpCenter architecture

In this case Opcenter works as a server, but it can also work as a client. Developer can create custom applications with .Net Framework and use HTTPS client library to call a external source to get the data. In our scenario we used authorization based on the JWT token combined with X.509 certificates. Both token and authorization certificate should be generated by server and provided for Opcenter in advance. Additionally, to make it possible execute requests using secured HTTPS connection, additional server certificate should be issued and stored in the local certificate store of machine with Opcenter.

2.4 Add token into the client

The Rest Server issues the token for the user and verify it while accessing. For testing purposes, in order to get the token and adding it to Postman, please follow these steps:

1. Open the terminal, and go into the folder "**C:\<…>\src**"

2. Type the command:

```
node createtoken.js
```

And the token will be generated in console automatically like:

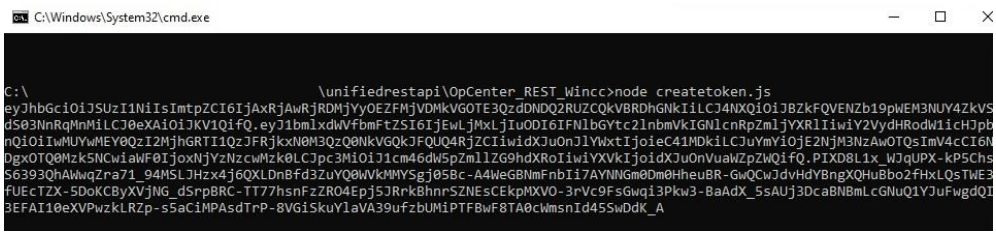


Figure 11 - Generating the token

3. Copy this token into Postman in the 'Authorization' tab and choose the 'Type' as 'Bearer token' like:

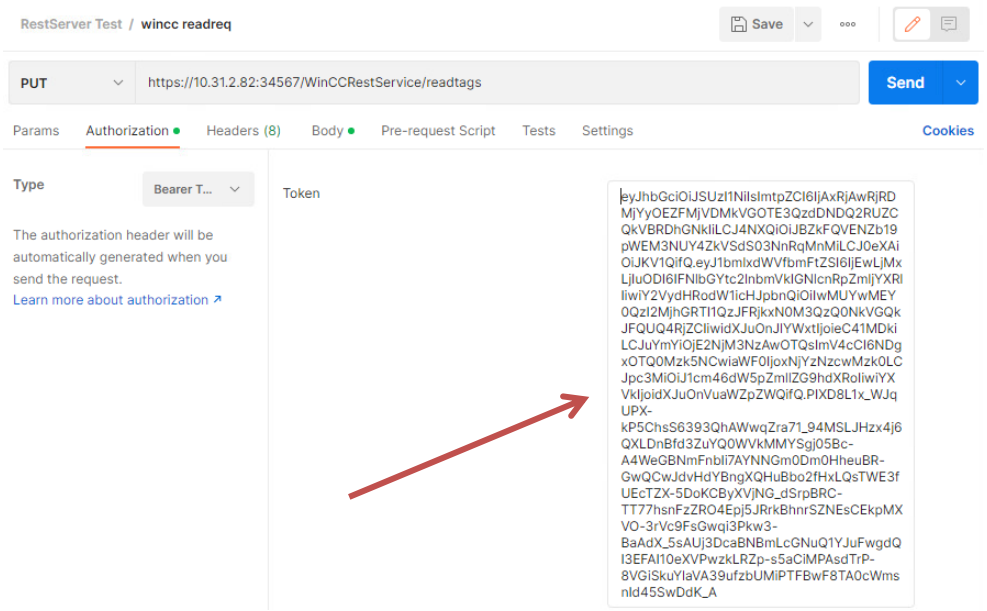


Figure 12 - Adding the token into the client

3 How to start the server?

3.1 Start the server automatically

In WinCC v7.5

There is a way to start the server after the start of the WinCC v7.5 runtime. Below, the steps needed to achieve this configuration will be written.

1. Write click in **Computer** and select **Properties:**

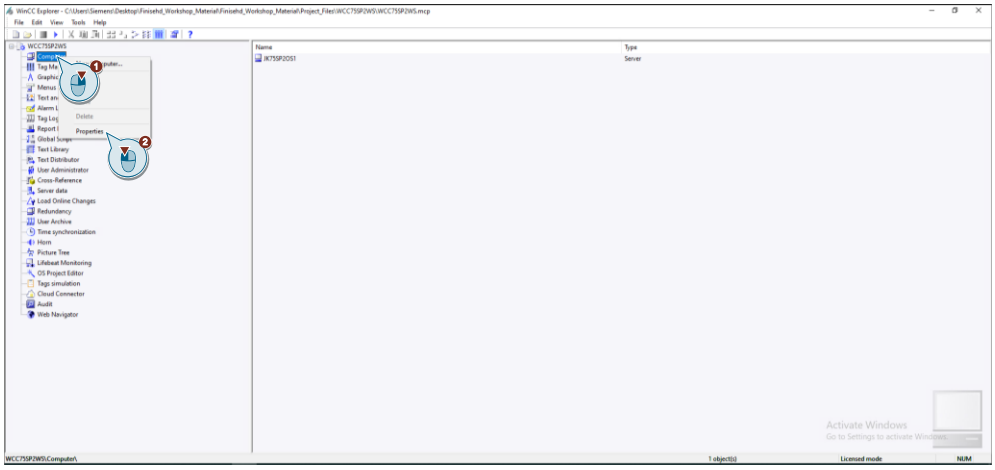


Figure 13 - Start the server automatically in WinCCv7.5 (1)

2. Select again **Properties**:

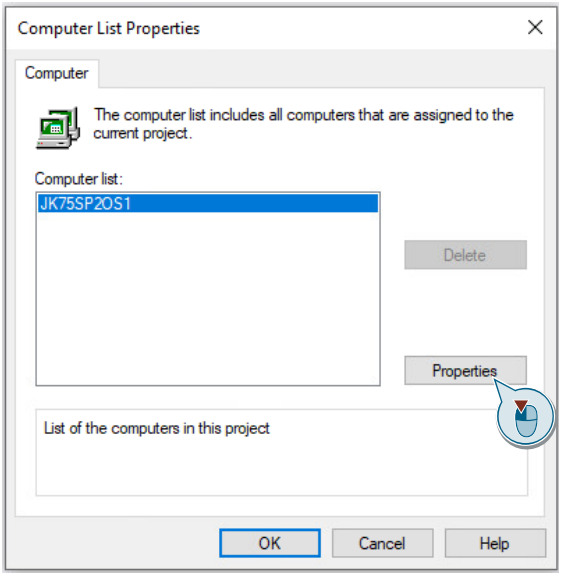


Figure 14 - Start the server automatically in WinCCv7.5 (2)

3. Go to the tab **Startup** and add in the **Applications** area the path to the **StartV7.bat** program which will launch the server.

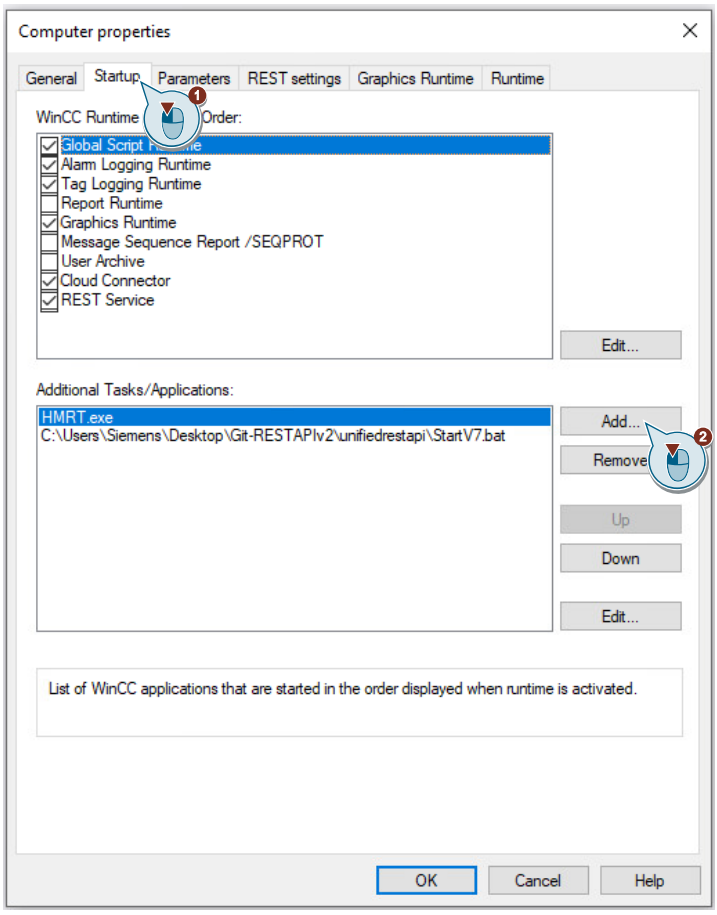


Figure 15 - Start the server automatically in WinCCv7.5 (3)

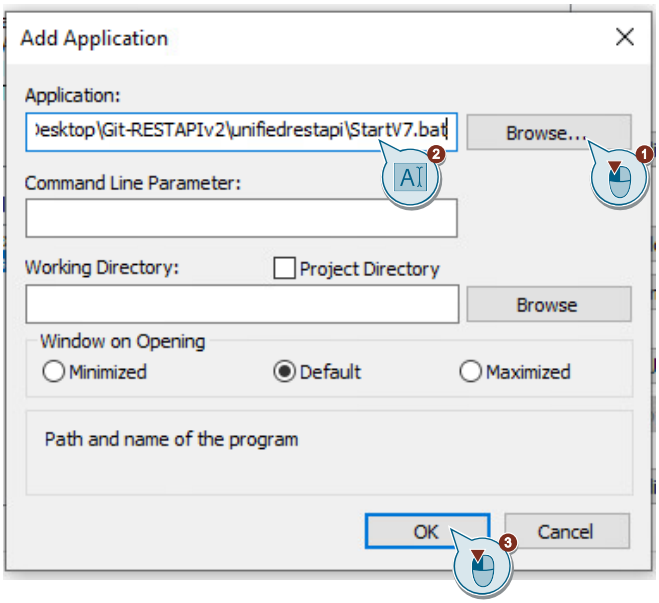


Figure 16 - Start the server automatically in WinCCv7.5 (4)

In WinCC Unified

Step 1

Create a button with click event to raise a windows event like this:

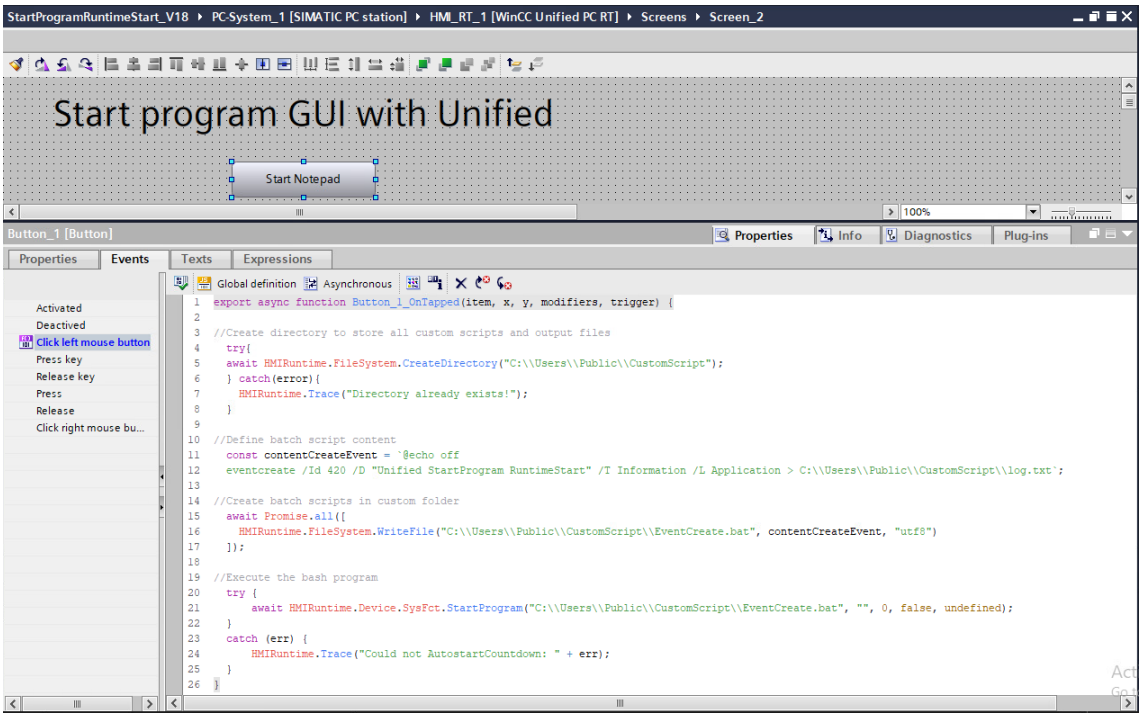


Figure 17 - Start the server automatically in Unified (1)

In the script, 4 different operations are going to be carried out once the button is clicked.

1. First of all, it creates a folder where the files are going to be located.

2. Creates a variable and stores the commands that are going to be launched. This command is going to be used to trigger a Windows Event.

3. Writes the text which is stored in the variable to a file called "*EventCreate.bat*"

4. Executes that "*EventCreate.bat*" which actually triggers the event.

Step 2

Create a Windows Event which will open the program:

1. Go to the "**Task Scheduler**":

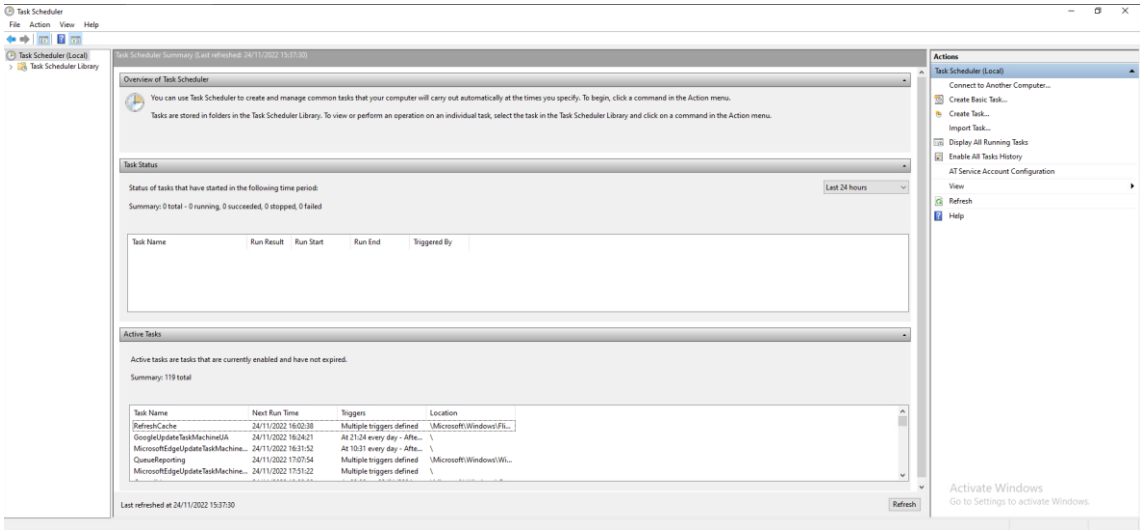


Figure 18 - Start the server automatically in Unified (2)

2. In the right panel, click in “**Create Task**”:

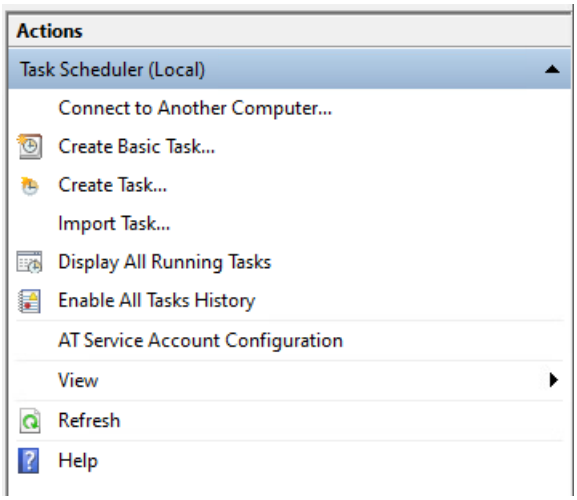


Figure 19 - Start the server automatically in Unified (3)

3. Enter the name of the New Task:

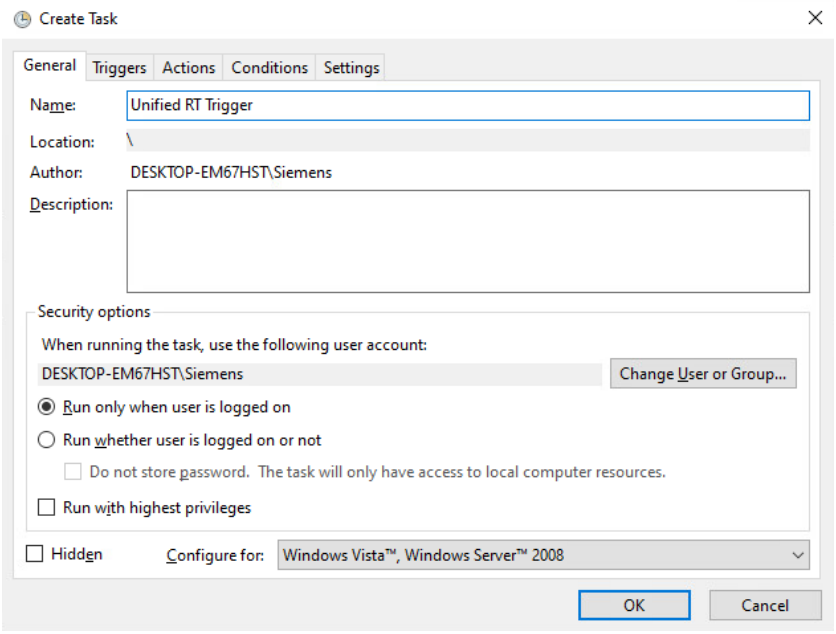


Figure 20 - Start the server automatically in Unified (4)

4. Go to the “**Trigger**” tab and click on “**New**”:

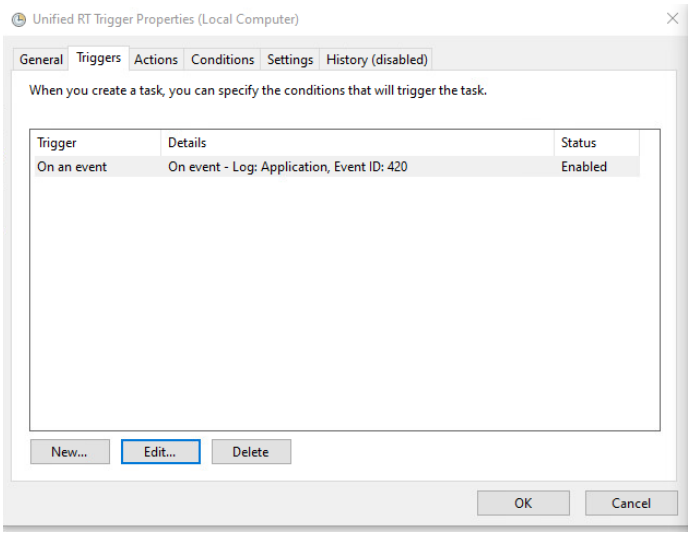


Figure 21 - Start the server automatically in Unified (5)

On the “**New Event**” panel, choose to **Begin the task** “On an event”, in the “**Log**” field, select “**Application**” and, finally, write a random “**Event ID**”.

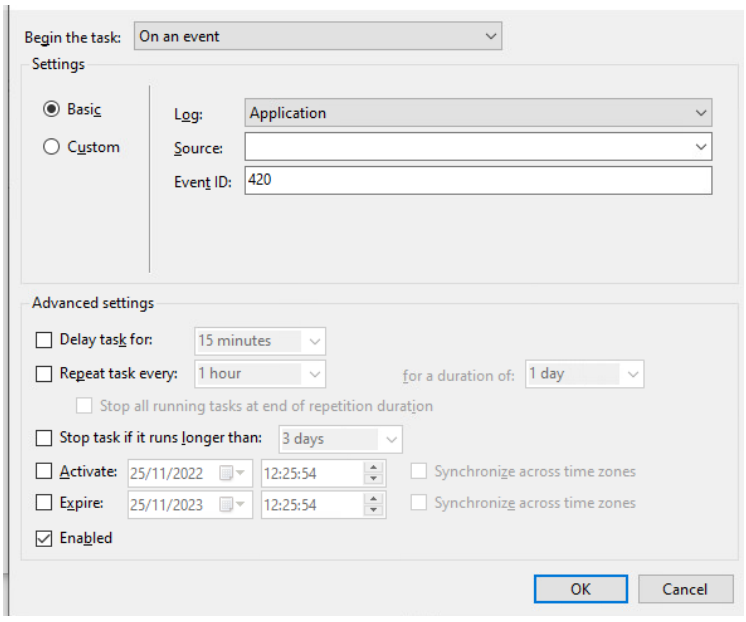


Figure 22 - Start the server automatically in Unified (6)

5. Go to the “**Actions**” tab and click on “**New**”:

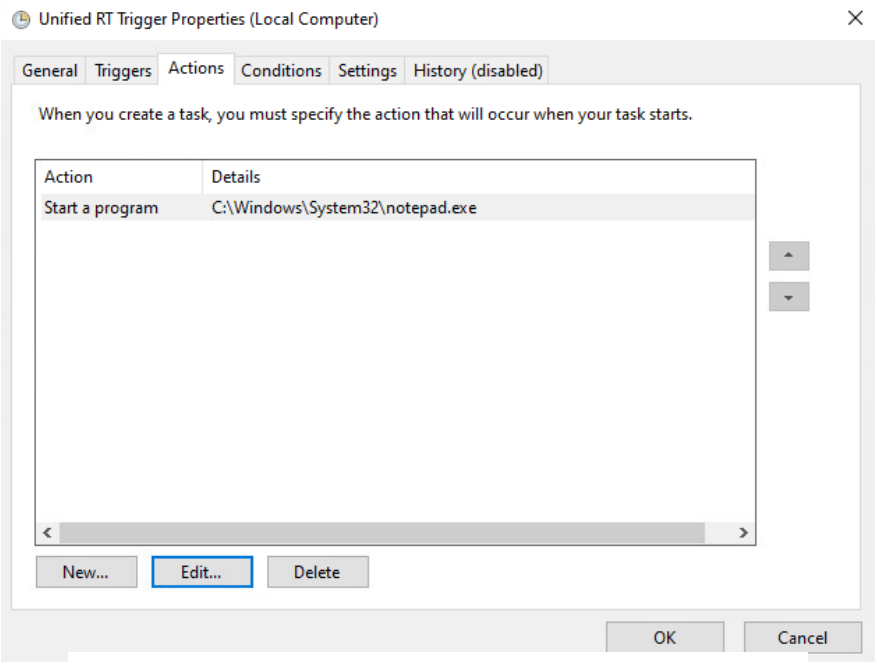


Figure 23 - Start the server automatically in Unified (7)

On the “**New Action**” panel, select in the “**Action**” field, “Start a program”, and in the Settings section, in the field “**Program/script**” select the path of the program which will be started by WinCC Unified. In this case, the Runtime will open the notepad.exe

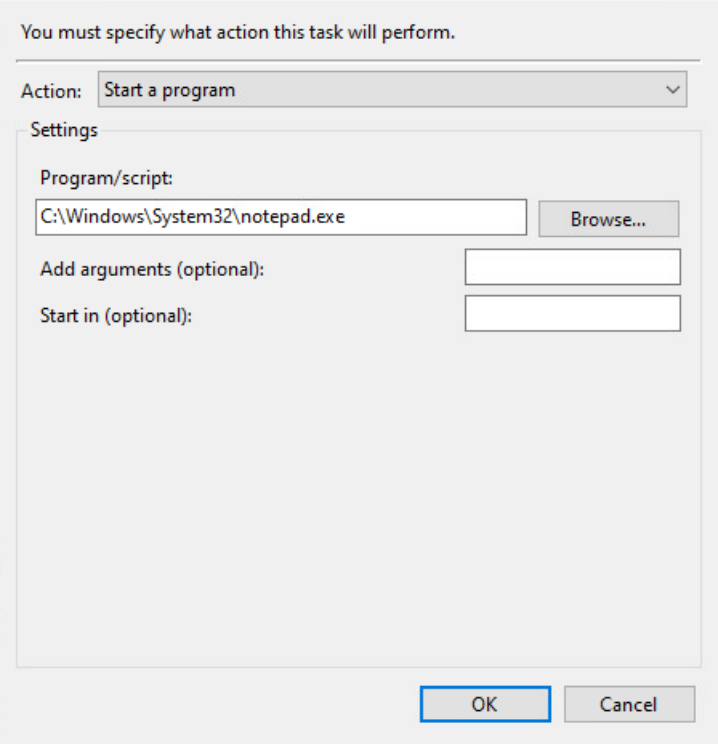


Figure 24 - Start the server automatically in Unified (8)

6. Last but not least, in the "**Settings**" tab, in the last field, select "**Stop the existing instance**" if the task is already running.

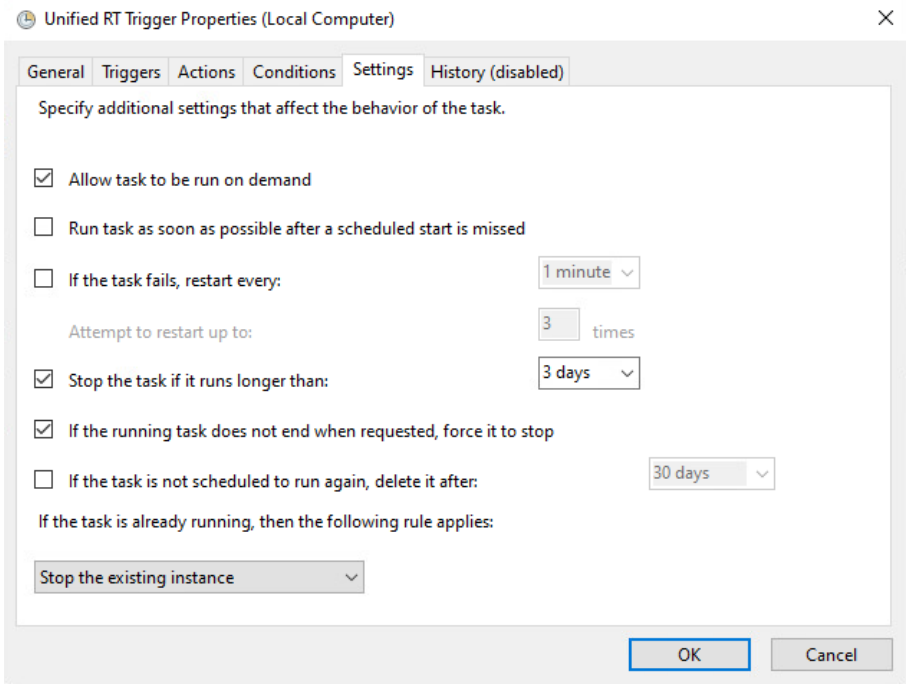


Figure 25 - Start the server automatically in Unified (9)

Step 3

Start the WinCC Unified Runtime and see how the WinCC Unified Runtime opens notepad when clicking the button.

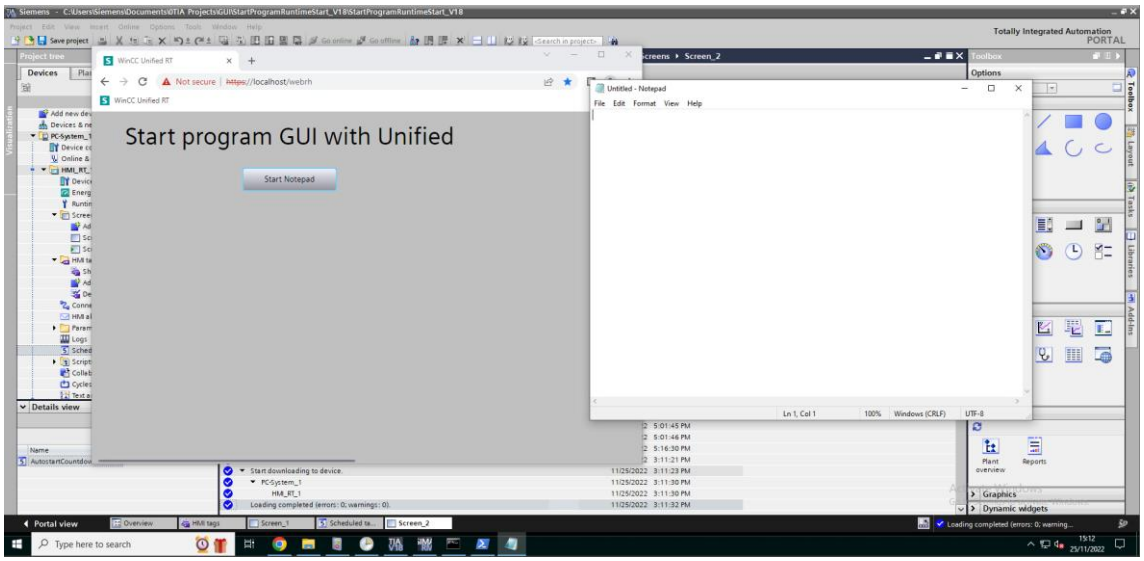


Figure 26 - Start the server automatically in Unified (10)

3.2 Start the server from the Command Line

To start-up Consistent Message Transporter, it is needed to type the next command in the terminal with:

```
node Server.js
```

The table below lists all the available parameters after the command `node Server.js` in order to specify the server which is the configuration of the system.

Table3-4

Scenario	Parmeter 1 & 2	Parmeter 3	Parmeter 4	Parmeter 5
1 (Unified)	`--southboundservice=unified`	`--clientconfig=unified2opcenter`	`--serverconfig=TagConfiguration.xml`	`--log=verbose`
				`--log=info`
				`--log=error`
2 (WinCC V7.5)	`--southboundservice=classic`	`--clientconfig=wincc2opcenter`	`--serverconfig=WinccTagConfiguration.xml`	`--log=verbose`
				`--log=info`
				`--log=error`

Example:

Connect OpCenter to WinCC Unified

```
node Server.js --southboundservice=unified --clientconfig=unified2opcenter
--serverconfig=TagConfiguration.xml --log=verbose
```

Connect OpCenter to WinCC V7.5

```
node Server.js --southboundservice=classic --clientconfig=wincc2opcenter
--serverconfig=WinccTagConfiguration.xml --log=verbose
```

4 How does the Consistent Message Transporter work?

In order to explain how the Consistent Message Transporter works, different scenarios, within the WinCC v7.5 environment, are going to be used. The behavior with the Unified is similar so, it is going to be omitted

It is important to notice that there are several functions that the server does on startup. The most important one is that the server subscribes to the tags that have a OnChange property in the configuration XML, in this case, “WinccTagConfiguration.xml”. This scenario is going to be shown in the scenario nº3.

After preparing the certificate and Token, we can test each Scenario with response message via Postman. All of the following scenarios are from the point of view of the OpCenter server.

4.1 Scenario 1: Read a tag

In the scenario 1, it is shown the execution of the read tag command. In this case, the OpCenter Client firstly offer a Read Request to the Rest Server with the purpose of reading a tag from the PLC.

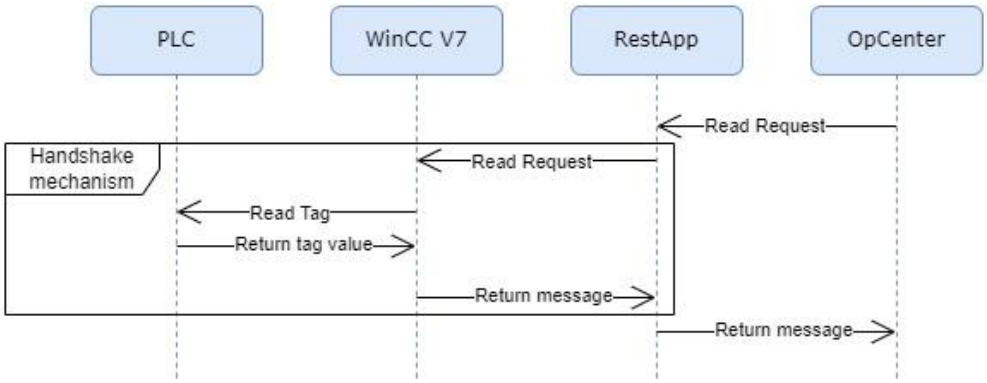


Figure 27 - Scenario 1: Read a tag

The table 3-1 specifies the required parameters for the read query and the response if the request was successful.

Table 3-1

Request Type	URL	Method	Send JSON body Request	Response
Read Request	https://192.168.56.1:34567/WinCCRestService/readtags	PUT	{ "variableName":["testBool", "testText8"]}	[{ "name": "testBool", "value": "true" }, { "name": "testText8", "value": "ON00050" }]

The steps for a manual test will be the following:

1. Start the server with the command:

```
>> node Server.js --southboundservice=classic
--clientconfig=wincc2opcenter
--serverconfig=WinccTagConfiguration.xml --log=verbose
```

2. Send the read request:

Figure 28 - Read request in the Postman App

3. Check the response:

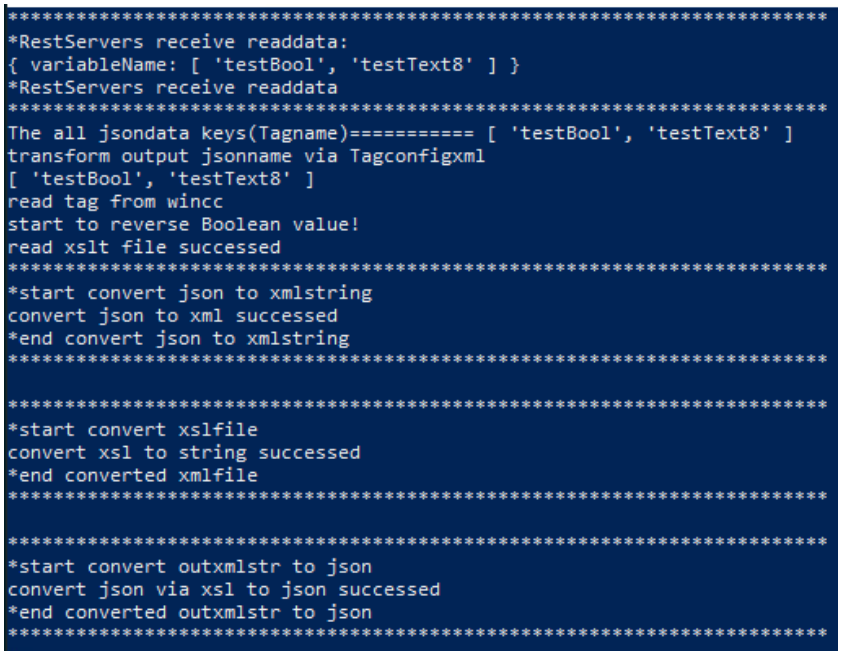


Figure 29 - Server response to the read request

4.2 Scenario 2: Write a tag

In the scenario 2, it is shown the execution of the write tag command. In this case, the OpCenter Client firstly offer a Write Request to the Rest Server.

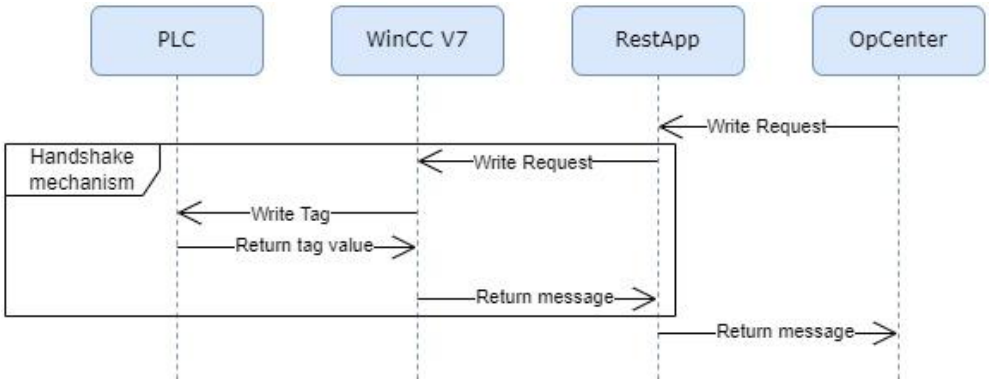


Figure 30 - Scenario 2: Write a value to a tag

In this procedure, it is used the consistent message mechanism which makes sure that the message arrives, but this mechanism is explained with details in the chapter nº4

The table 3-2 specifies the required parameters for the write query and the response if the request was successful.

Table 3-2

Request Type	URL	Method	Send JSON body Request	Response
Write Request	https://192.168.56.1:34567/WinCCRestService/tagManagement/Values	PUT	{"OrderName": "SIE1001", "Equipment": "Potatoes", "PlannedStartTime": 1902, "PlannedEndTime": 1930}	{ "@odata.context":"https://10.31.2.82:34567/WinCCRestService/TagManagement/Values", "Succeeded":"true", "Error": {"Errorcode":0,"Errormessage":"" }}

The steps for a manual test will be the following:

1. Start the server with the command:

```
>> node Server.js --southboundservice=classic
--clientconfig=wincc2opcenter
--serverconfig=WinccTagConfiguration.xml --log=verbose
```

2. Send the write request:

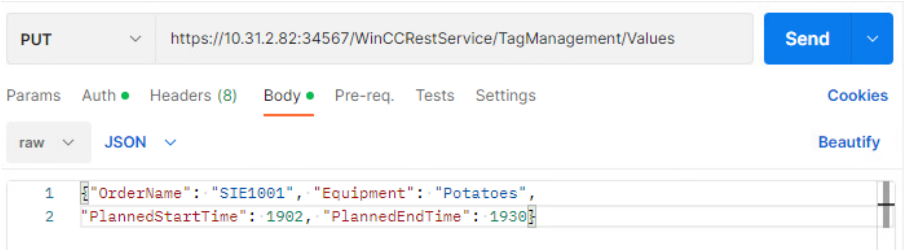


Figure 31 - Write request in the Postman App

3. If the handshake is successful, an output like this will be displayed:

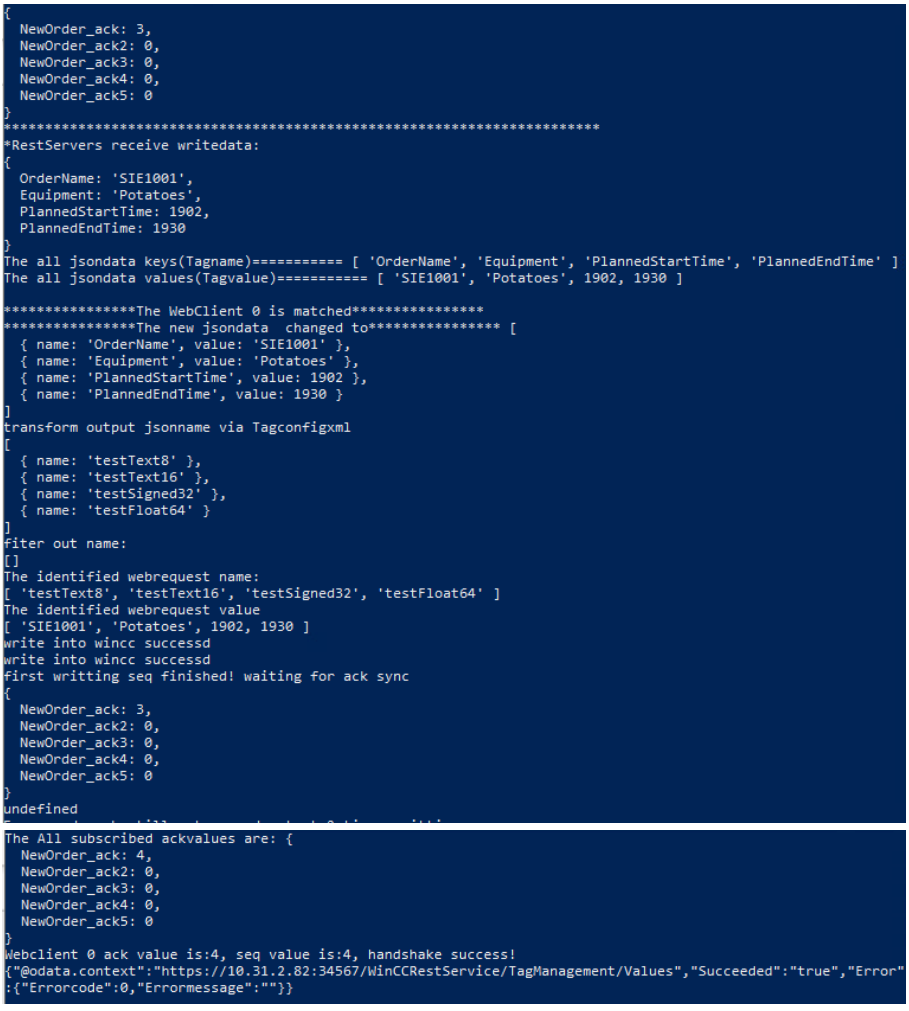


Figure 32 - Server response sequence of the consistent message, when succesful

If the handshake was not successful, the output will be:

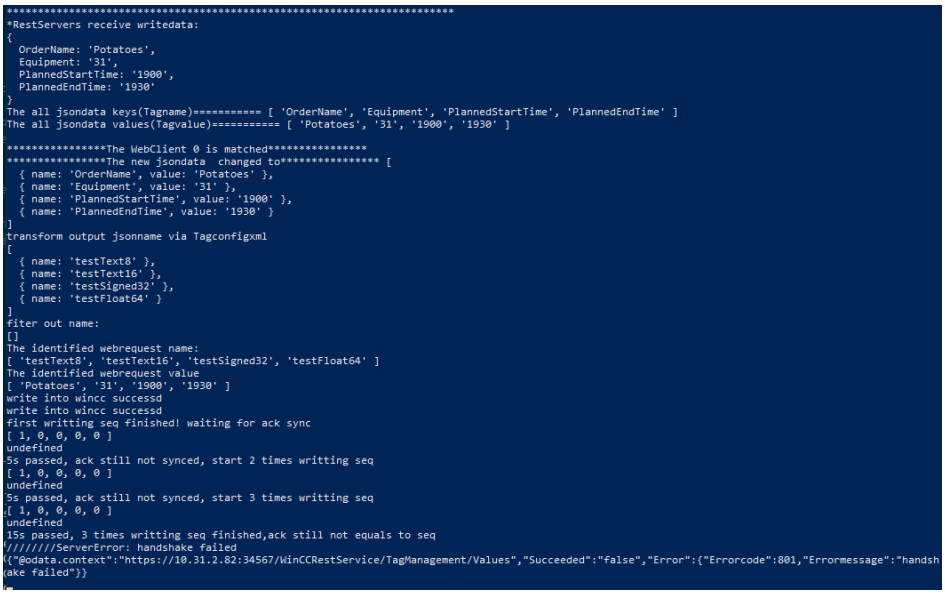


Figure 33 - Server response sequence of the consistent message, when failure

4.3 Scenario 3: Tag value changed

The scenario nº3 is about noticing when a certain tag has changed. OpCenter can define tags of the PLC in order to notice a value change from a tag defined in the xml files.As explained at the beginning of this chapter, there is a file called “WinccTagConfiguration.xml” which stores a property called OnChanged.

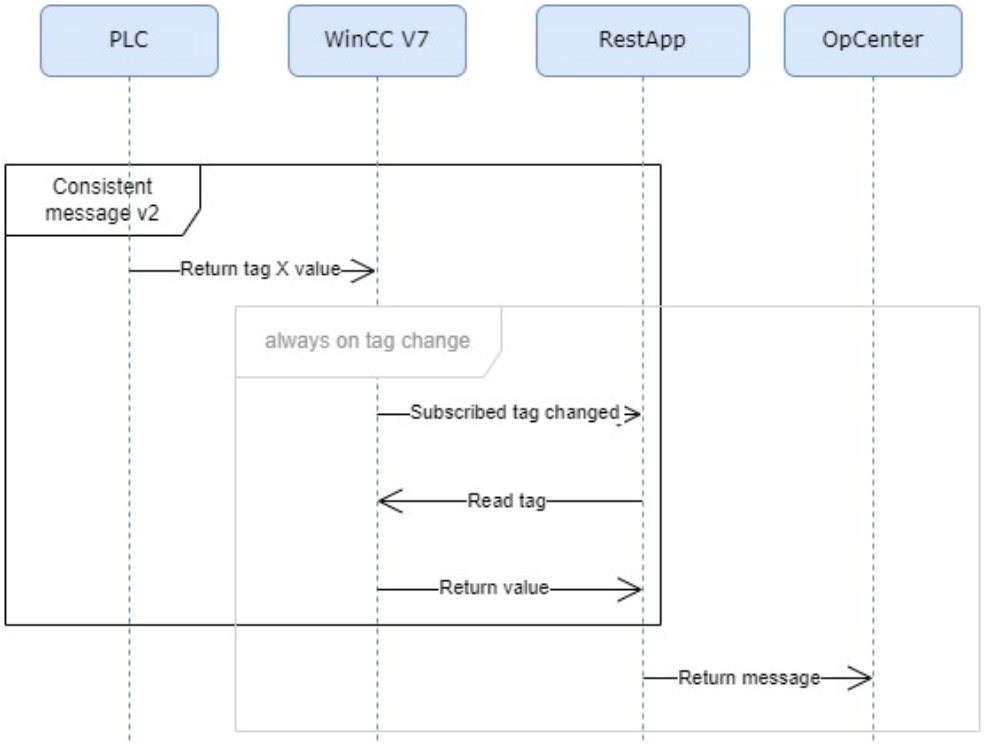


Figure 34 - Scenario 3: Tag value changed

This method below is in charge of being alert of changes on the value of the variables subscribed and display the new values of all the variables subscribed, despite only one changed.

Once the values of WinCC V7.5 subscribed tags are changed and over the **threshold** value (for wincc2 & wincc3 the threshold is 100), Rest Server will read the need-to-read tags from it. After getting values, it sends a request to OpCenter Server.

You can get the response message from Opcenter Server in console like:

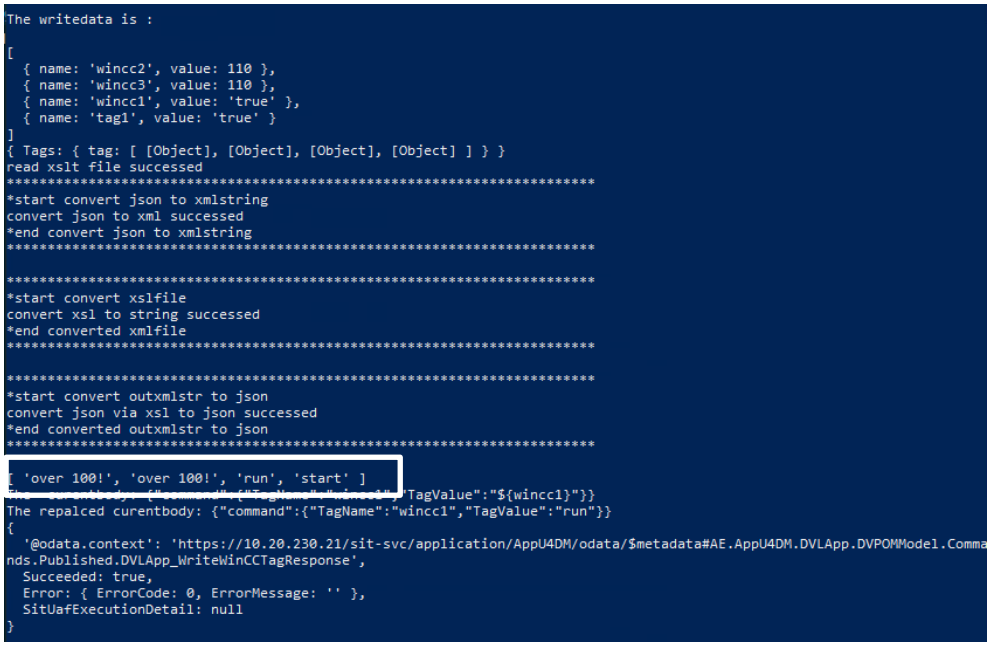


Figure 35 - Response when the values are greater than the thresold

h

The steps for a manual test will be the following:

1. Start the server with the command:

```
>> node Server.js --southboundservice=classic
--clientconfig=wincc2opcenter
--serverconfig=WinccTagConfiguration.xml --log=verbose
```

2. Change the value of a variable (testSigned32):

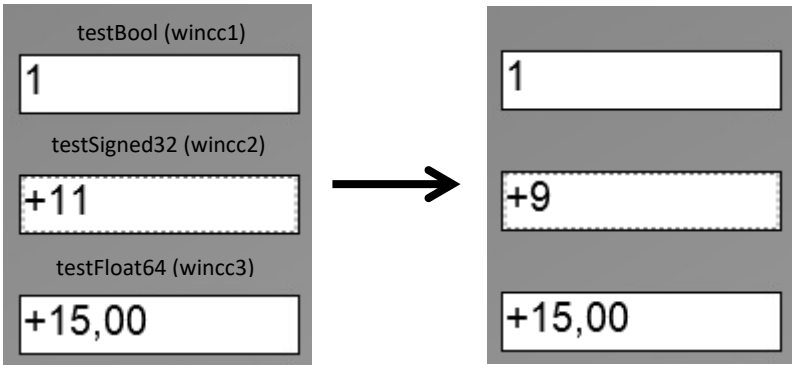


Figure 36 - WinCC RT screenshot (before)

Figure 37 – WinCC RT screenshot (after)

3. See the output in the App and this message appears:

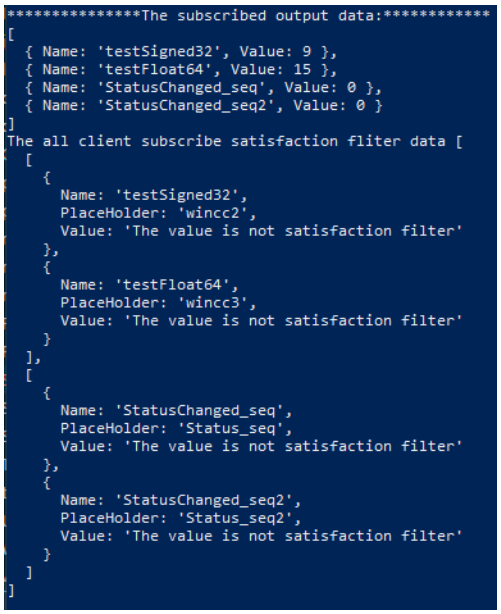


Figure 38 - Server response to the change of a subscribed tag

5 Consistent message

The **consistent message** mechanism (previously called handshake mechanism) implemented here, makes possible to buffer and exchange messages of arbitrary structure tag based and consistently. Thereby, incoming as well as outgoing messages can be processed sequentially.

The **consistent message** is an improved version of the handshake mechanism, and it is the version implemented in this application. This mechanism will be available soon in the SIOS platform.

The scenario in which this application is using this consistent message mechanism is the “machine as a receiver”. The generic inbound handling function is called cyclically and notices when the sequence number in the handshake interface is changing. This is the signal that a partner wants to send a message and needs the confirmation that he may write to the message interface. The function checks if there is a free place in the buffer. If this is true, it sets the acknowledge number to the sequence number so the partner can write. After the partner wrote its message into the message interface content, the partner increments the sequence number again which is the signal for the PLC that writing is complete, and the message can be written to the message buffer. After filling the message header with information and storing it in the buffer, the PLC sets the acknowledge number to the sequence number again. Before the sequence number is not equal to the acknowledge number, the partner must not write data nor ask for sending a new message.

This updated version will be available in SIOS soon.

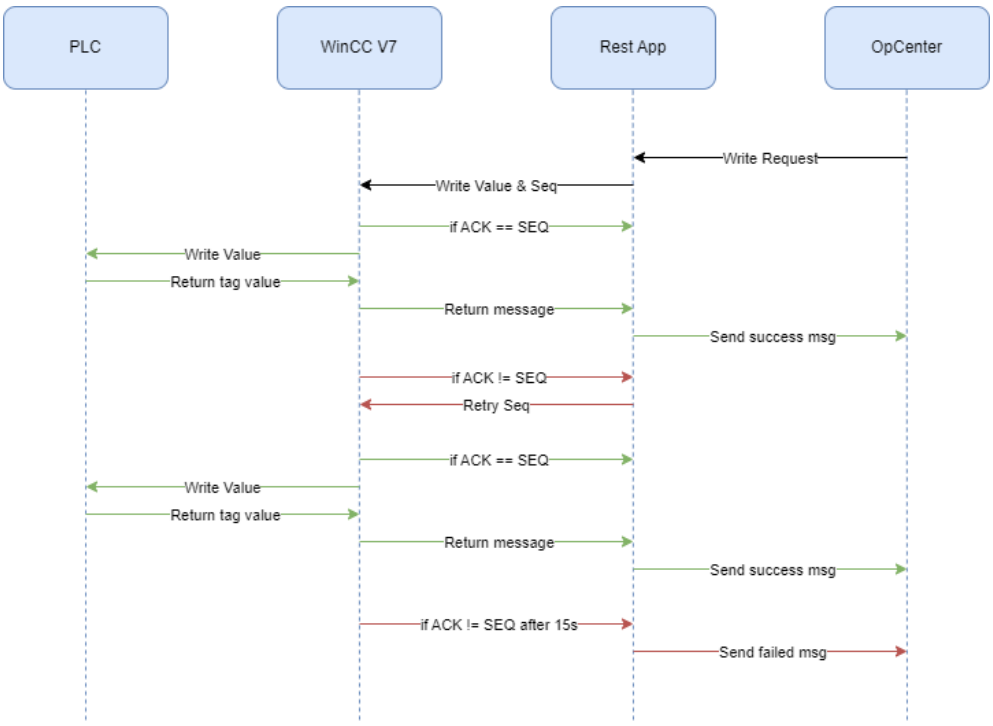


Figure 39 - Consistent message mechanism

6 Error handling

The Table 5-1 list all the possible errors occurred in different system and accordant solutions.

Table 5-1

System	Description	ErrorCode	Errormessage	Solutions
Webclient	Client Req error	400	“Webclient request error”	Check Req JSON format
RestServer	token recognized failed	401	“token recognized failed”	Get token first
RestServer	Get token failed	402	“Unidentified user”	Check request body for Gettoken
RestServer	Server not start	404	“connect ECONNREFUSED”	restart server
RestServer	Server internal error	500	“Server Internal Error”	Debug and restart server
RestServer	Request property error (less or more two property) for app.write	501	“Object: [“errobj1”,”errobj2”],error:should has two properties.”	Check Req JSON obj
RestServer	Request property error (no property) for app.write	501	“Property lacks”	Check Req JSON obj
RestServer	Request body matches Template body failed for app.write	502	“Request property matching Templatebody failed”	Check template body column in restserverconfiguration.xml and req property
RestServer	Incorrect property (transfer property with tagconfig.xml failed) for app.write	503	“unidentified Property [“tag1”, “tag2,...]”	Check PlaceHolder column in wincctagconfig or restserverconfig.xml
RestServer	Req JSONbody is not obj for app.read	504	“JSON body should be a JSON Object”	Check jsonbody is JSON Obj
RestServer	No property for app.read	504	“property lacks”	Check jsonbody with Property
RestServer	No parameter for app.read	504	“parameter lacks”	Check jsonbody with parameter
RestServer	Parameter is not JSON Array for app.read	504	“Parameter has to be a JSON Array”	Check jsonbody with parameter type(array)
RestServer	Property is not”variableName” or “TagName” for app.read	504	“unacceptable property: <errorproperty>“	Check jsonbody with acceptable property
RestServer	More than one Property for app.read	504	“Unacceptable for more than one property”	Check jsonbody with one property

System	Description	ErrorCode	Errormessage	Solutions
RestServer	type error value in Triger column of wincctagconfiguration or tagconfigureation.xml	505	{“what”: “The Trigger column of webclient[<i>] in <WinccTagConfiguration.xml>“, “value”: “<errTrigger>“ “reson”: “Trigger name has to be Onchange” }	check trigger column, the value has to be Onchange
RestServer	IIH write data error	506	“All properties match variable name in assets.xml failed”	Check property in jsonbody with “PlaceHolder&Name” columns in RestServerConfiguration_OpCenter.xml and “variableName “column in assets.xml
RestServer	IIH write data error	506	“Properties: [<error property>] matches variable name in assets.xml failed” “	Check property in jsonbody with “PlaceHolder&Name” columns in RestServerConfiguration_OpCenter.xml and “variableName “column in assets.xml
OpenPipe	Unified runtime connection error for openpipe.write	600	“connect ENOENT \\\\. \\pipe\\HmiRuntime”	Check unified runtime connection statues
OpenPipe	Unified get variable name error for openpipe.write	600	[{“variable”:”errvariable”,”error”:”The given tag does not exist”}]	Check input variable with placeholder and name columns in tagconfiguration.xml
OpenPipe	Valuetype error for Unified for openpipe.write	600	[{“value”:”errvalue”,”error”:”Conversion not allowed”}]	Check configtag.xslt and Unified eachtag valuetype
OpenPipe	Unified runtime connection error for for openpipe.read	601	“connect ENOENT \\\\. \\pipe\\HmiRuntime”	Check unified runtime connection statues
OpenPipe	Unified get variable name error for openpipe.read	601	{“variable”:[“variable1”,”variable2”],”error” :The given tag does not exist.”}	Check input variable with PlaceHolder and Name columns in tagconfiguration.xml
OpenPipe	Unified runtime connection error for openpipe.read	601	“connect ENOENT \\\\. \\pipe\\HmiRuntime”	Check unified runtime connection statues
OpenPipe	Unified runtime connection error for openpipe.subscribe	602	“connect ENOENT \\\\. \\pipe\\HmiRuntime”	Check unified runtime connection statues
OpenPipe	Unified get variable name error for openpipe.subscribe	602	{“variable”:[“variable1”,”variable2”],”error” :The given tag does not exist.”}	Check Name column which has Trigger column in tagconfiguration.xml

System	Description	ErrorCode	Errormessage	Solutions
OpCenter_Server	OpCenter_Server response error (TemplateBody error)	700	“An unexpected “StartArray” node was found when reading from the JSON reader. A “PrimitiveValue” node was expected.”	check <TemplateBody> column in wincctagconfiguration or tagconfiguration.xml,make sure type correct JSON body which Opecnter_Sever offered.
OpCenter_Server	OpCenter_Server response error (templatebody error)	700	“The property “<errproperty>“does not exist on type “AE.AppU4DM.DVLApp.DVPOMModel.Commands.Published.DVLApp_WriteWinCCTag”. Make sure to only use property names that are defined by the type.”	check <TemplateBody> column in wincctagconfiguration or tagconfiguration.xml,make sure type correct JSON body which Opecnter_Sever offered.
OpCenter_Server	OpCenter_Server response error (templatebody error)	700	“The parameter “<errparameter>“in the request payload is not a valid parameter for the operation “DVLApp_WriteWinCCTag”.”	check <TemplateBody> column in wincctagconfiguration or tagconfiguration.xml,make sure type correct JSON body which Opecnter_Sever offered.
OpCenter_Server	OpCenter_Server response error (CommandMode or CommandName error)	701	“OpCenter_Server resource Not found	check <CommandMode>or <CommandName> columns in wincctagconfiguration or tagconfiguration.xml,CommandMode should alway be POST,and make sure type correct URL of OpCenter_Server” in CommandName
OpCenter_Server	OpCenter_Server request error	ECONNRESET	“Client newwork socket disconnected before secure TLS connection was established”	check RestServer network and firewall, check Token (OpCenter_Server”s certificate)
OpCenter_Server	OpCenter_Server connection error	702	“connect ECONNREFUSED <Opcenter IP adress>”	check if OpCenter_Server is running
ODK	Wincc connection error (write)	800	{“error”: “wincc Runtime connection failed”}	Check wincc runtime connection statues
ODK	Wincc get variable name error(write)	800	{“variable”:[“variable1”,”variable2”],”error” :”cannot find variable names in wincc” }	Check Name column in wincctagconfig /restserverconfig.xml and wincc DM
ODK	Bool valuetype error for wincc	800	{“value”:”errvalue”,”error”:”valuetype is not Bool”}	Check req value type and winccvalueconfig.xslt file
ODK	unsigned short valuetype error for wincc	800	{“value”:”errvalue”,”error”:”valuetype is not unsigned short”}	Check req value type

System	Description	ErrorCode	Errormessage	Solutions
ODK	long value over max or min limit for wincc	800	{“value”:”errvalue”,”error”:”value over limited”}	Check req value
ODK	long valuetype error for wincc	800	{“value”:”errvalue”,”error”:”valuetype is not signed long”}	Check req value type
ODK	unsinged long value type error for wincc	800	{“value”:”errvalue”,”error”:”valuetype is not unsigned long”}	Check req value type
ODK	Float valuetype error for wincc	800	{“value”:”errvalue”,”error”:”valuetype is not float”}	Check req value type
ODK	double valuetype error for wincc	800	{“value”:”errvalue”,”error”:”valuetype is not double”}	Check req value type
ODK	Date valuetype error for wincc	800	{“value”:”errvalue”,”error”:”incorrect Date format”}	Check req value type& format
ODK	Date value error for wincc	800	{“value”:”errvalue”,”error”:”Date value unnormal”}	Check req value
ODK	Handshake failed for wincc	801	“Handshake failed”	Check wincc Runtime if ack tag is synchronized
ODK	Wincc runtime connection error for odk.readtag	802	{“error”:”wincc Runtime connection failed”}	Check wincc runtime connection statues
ODK	Wincc get variable name error for odk.readtag	802	{“variable”:[“variable1”,”variable2”],”error” :”can not find variable names in wincc” }	Check placeholder column, or Name column which has Trigger column in wincctagconfigureation, xml
ODK	Wincc runtime connection error for odk.subscribetag	802	{“error”:”wincc Runtime connection failed”}	Check wincc runtime connection statues
ODK	Wincc get variable name error(subscribe)	802	{“variable”:[“variable1”,”variable2”],”error” :”can not find variable names in wincc” }	Check placeholder column ,or Name column which has Trigger column in wincctagconfigureation, xml
IIH Server	IIH POST failed because IIH_Server connection error	900	“connect ECONNREFUSED <IIH ip adress>”	check if IIH is running
IIH Server	IIH post variable valuetype error	901	“Column < errorvalue>does not exist”	Check Date Service Variables type and req value
IIH Server	IIH post variable valuetype error	901	“syntax error at or near < errorvalue >“	Check Date Service Variables type and req value
IIH Server	IIH post variable value over limited	901	“integer out of range”	Check req value (in range of integer MAX)
IIH Server	IIH Service connection error	902	“IIH Service Unavailable”	Log in Industrial Edge and open IIH Data service

7 Appendix

7.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:

support.industry.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts.

Please send queries to Technical Support via Web form:

siemens.com/SupportRequest

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:

siemens.com/sitrain

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:

support.industry.siemens.com/cs/sc

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" APP. The app is available for iOS and Android:

support.industry.siemens.com/cs/ww/en/sc/2067

7.2 Industry Mall

The Siemens Industry Mall is the platform on which the entire siemens Industry product portfolio is accessible. From the selection of products to the order and the delivery tracking, the Industry Mall enables the complete purchasing processing – directly and independently of time and location:
mall.industry.siemens.com

7.3 Application support

Siemens AG
Digital Factory Division
Factory Automation
SUP HMI Innovation
DF FA S SUP HMI
Frauenauracher Str. 80
91056 Erlangen, Germany

mailto: simatic.industry@siemens.com

7.4 Links and literature

Table 7-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\3\	Application Example Unified Rest API https://code.siemens.com/restapi/unifiedrestapi/-/blob/master/OpCenter_REST_Wincc/UnifiedRestAPI_AWB.docx
\4\	OpenSSL download link: https://slproweb.com/products/Win32OpenSSL.html
\5\	Node.js download page https://nodejs.org/en/
\6\	SIOS page for consistent message https://support.industry.siemens.com/cs/document/109795979/consistent-message-based-communication-via-opc-ua?dti=0&lc=en-WW

7.5 Change documentation

Table 7-2

Version	Date	Modifications
V1.0	09/2022	First version
V1.1	10/2022	Improvements from the feedback
V1.2	11/2022	Release version