SIGMATEK
OPC_UA Server

Date of creation: 26.08.2016 | Version date: 26.11.2021 | Article number: XX-XXX-XXX-E

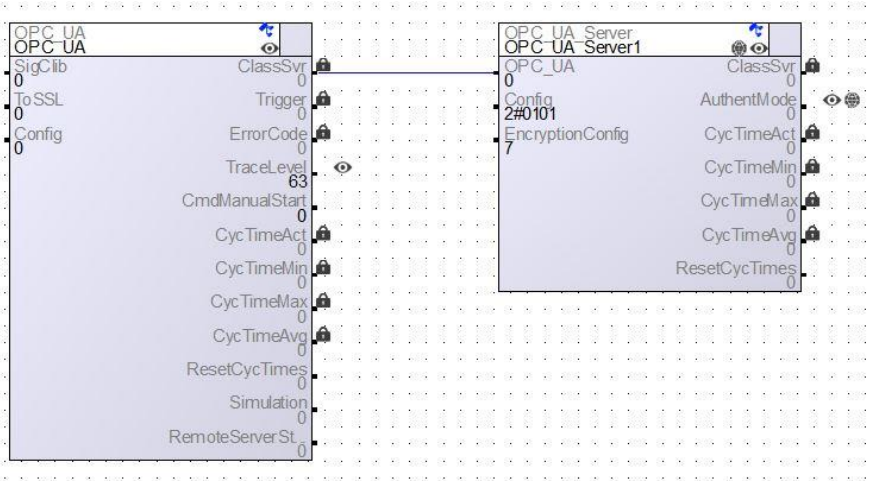
Publisher: SIGMATEK GmbH & Co KG
A-5112 Lamprechtshausen
Tel.: +43/6274/4321
Fax: +43/6274/4321-18
Email: office@sigmatek.at
WWW.SIGMATEK-AUTOMATION.COM

Translation from German

Contents

1 OPC-UA Server Introduction

For the OPC UA server functionality, the OPC UA class must be imported and placed in a network. Furthermore, an OPC_UA_Server class must be placed and connected to the OPC_UA class.



Important information can also be found in the documentation for the *OPC_UA* class.

2 Access Authorization

A simple user administration including access control is integrated.
Starting with version 3.5 the OPC_UA server also supports optional user roles.

For this purpose, an XML file called "**UserConfiguration.xml**" is used, which must be located in the "*C:\OPC_UA*" folder. If this folder is missing, the file must be in the root of drive C on the controller.

→ if this file is **not** available, **no** verification takes place and only anonymous registrations are possible. In this case, the client may execute the connect
only anonymously without a username and password.

→ if the file exists, the OPC UA server automatically checks with each client-side connection whether the combination of username and password received with the connect is stored in the file. Please note, that this is case-sensitive.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Config Version="1.0">
    <Users>
        <User Name="BertiBediener" Password="x123" Userrole="5" />
        <User Name="EmilExperte" Password="y345" Userrole="-1" />
        <User Name="RosiRatlos" Password="z987" Userrole="0" />
    </Users>
</Config>
```

→ by means of the class server "AuthentMode" the registration can be switched to anonymous. For example, authentication can be temporarily suspended by setting the "AuthentMode" class server to #1.

→ The optional attribute "Userrole" is to be interpreted as a bit mask, it can be used for up to 32 write permissions.
When writing access to a node, the system checks whether a "Userrole" was also specified for this node – if this is the case, at least one of the 32 possible bits must match for the write permission.

`Userrole="0"` … the user has no write permissions
`Userrole="-1"` … the user has all write permissions
`Userrole="x"` … x = value for any bit mask (5 = role 1 and role 3)

3 Connection Setup and Limits

The server provides so-called endpoints for establishing connections. An endpoint that meets the appropriate encryption requirements is selected for a connection setup. If the server does not provide a corresponding endpoint, the encryption settings must be adjusted accordingly.
The endpoints provided by a server can be configured with respect to the operating system. See also **Fehler! Verweisquelle konnte nicht gefunden werden.** (**Fehler! Verweisquelle konnte nicht gefunden werden.**).
The limit values given here are those specified by the software. However, additional attention must be paid to which application runs on which hardware. This influences the response times of the server and it should be noted that, if necessary, application-specific smaller values should be used.

3.1 Sessions

After opening a connection, a session is opened by the client. The number of concurrent sessions on the server is limited. A client can open a maximum of 25 sessions and the total number of sessions on the server is 50. If the maximum number of sessions is open on the server, further attempts are rejected with error 0x80560000 (BadTooManySessions).

3.2 Subscriptions

Per session the number of possible subscriptions is limited to 20. Further attempts to create a subscription result in error 0x80770000 (BadTooManySubscriptions).

3.3 MonitoredItems

The maximum number of MonitoredItems per subscription is 1000. Creating MonitoredItems in addition leads to error 0x80DB0000 (BadTooManyMonitoredItems).

4 SSL Encryption

4.1 General Information

If encryption is basically supported on a PLC (see 3.3), a certificate is created for each interface when OPC-UA is started if these are not available. The certificates are located in the C:\opc_ua folder on the PLC. For the first interface two files named cert.der and key.pem are created. The files for further interfaces are then called cert<num>.der and key<num>.pem, where num starts at 1 and is increased continuously. However, it is not mandatory that all numbers are assigned away from 1.
The file with the extension *pem* contains the private key for the certificate. The file with the extension *der* of contains the certificate. The certificates must comply with the X.509 v3 standard in order to be used.

However, the certificates can also be provided by the administrator and are stored on the control as described above.

Certificates contain information from when and until when they are valid.

Attention: It is necessary to make sure that the time of the PLC is set correctly when creating the certificate.
In both cases, if a certificate is not yet or no longer valid, it is rejected by the remote peer. The invalid certificate must be deleted and replaced with a new one, or regenerated.

Attention: It must be ensured that the private key is not accessible to third parties, otherwise the encrypted data traffic can be read.

Note: Currently, certificates can only be used for the purpose of connection security. It is not possible to create or use a signed certificate on the control. Likewise, there is no possibility to use chained certificates (CA certification).

Note: Currently, no certificates can be used for authentication, but only username and password.

Note: If a certificate is to be created via the method CreateApplCertificate (see also class documentation of the OPC_UA class), it is necessary to specify data of the certificate issuers. Please note that the country must be entered as a 2-letter code (for example "AT" for Austria). Otherwise OPC_UA acknowledges the call with BadInvalidArgument.

4.2 Certificate Management

To manage and display certificates on the control, the AddOn **SSLCertificate** is available. This AddOn allows to move the certificates between the folders and delete them.

Note: An application should provide the possibility to delete, move and create/download certificates, because a secure connection via OPC-UA is no longer possible after a certificate has expired.

4.3 Security of the OPC UA Connection

Since version V.3.0 of the OPC UA server SSL encryption is implemented.
The following security policies are currently supported:

- None
- Basic256 [1]
- Basic256Sha256
- Aes128_Sha256_RsaOaep

From the following message security modes can be chosen:

- None: No encryption
- Sign: Message is only signed
- SignAndEncrypt: Message is encrypted and signed.

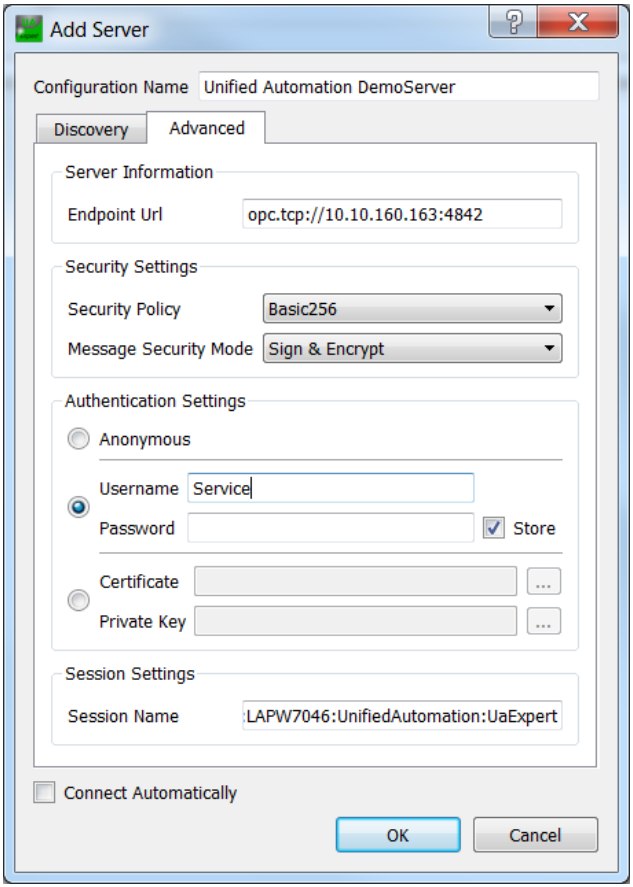
[1] This security policy is no longer assumed to be secure and therefore should no longer be used.

4.4 PLC Specific Notes

- → SSL encryption is **NOT supported under RTOS-OS**
- → SSL is supported by Salamander-OS since version 09.03.116 SSL
- → The folder structure required for the certificates is automatically created within the OPC_UA folder for the client SSL connect
- → A new certificate is automatically stored in the folder "rejected"
- → An accepted certificate must be moved to the folder "trusted".
  As long as the certificate is not in this folder, the corresponding client cannot establish a connection to the OPC_UA server.
- → If a certificate should be revoked later, it must be moved to the folder "revoked"
- → Since July 2018 SIGMATEK provides an AddOn for the administration of SSL certificates
- → Before creating the certificates, make sure that the PLC time is set correctly
- → The SecurityPolicys Basic256Sha256 and Aes128_Sha256_RsaOaep are supported from Salamander OS version 09.03.160.

4.5 SSL Settings on the Client

UA-Expert is used as an example



5 Simple Data Exchange

This chapter deals with the exchange of base data types.

The normal data exchange in SIGMATEK systems is realized with server variables of one or more classes.

At the moment simple data is supported:
DINT, UDINT, REAL and STRING

as well as complex data:
BYTESTRING and structures

Accessing the SIGMATEK OPC-UA server is done via its endpoint Url.

For communication port 4842 is used.

Example for endpoint Url: `opc.tcp://10.10.160.41:4842`

5.1 Principle for Simple Data

→ The user has to declare all OPC-UA variables in the PLC project as such. It makes sense to create an own class for the OPC-UA variables and place it in a network. See explanation for implementing below.

→ All variables to be transferred are entered in a XML file. This is done fully automatically by the SIGMATEK system! The XML file is transferred to the CPU while downloading the project.

→ When booting the PLC the OPC-UA server reads and interprets the XML file and from this generates an OPC-UA address space for easy data exchange.

→ The OPC-UA server is available for the outside world, so an OPC-UA client can connect to the server and read and write all OPC-UA variables and their properties.

5.2 Implementing in the PLC Project

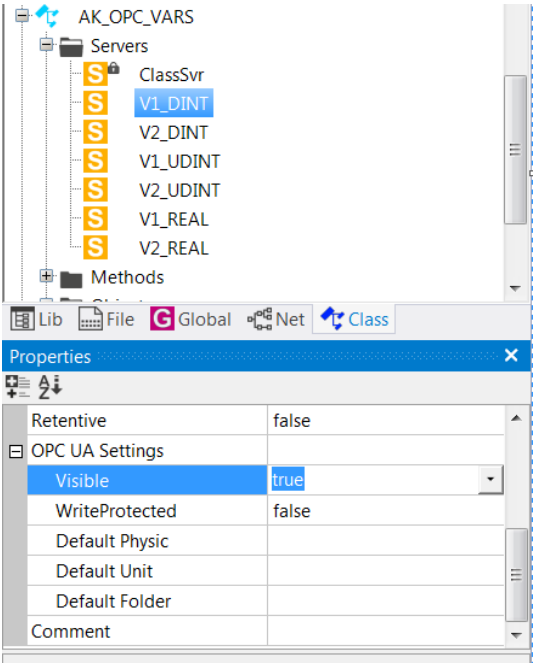
As mentioned before, the normal data exchange in SIGMATEK systems is realized with server variables (types DINT, UDINT, REAL and STRING).

The SIGMATEK system generates the OPC_UA.XML file for the OPC server.

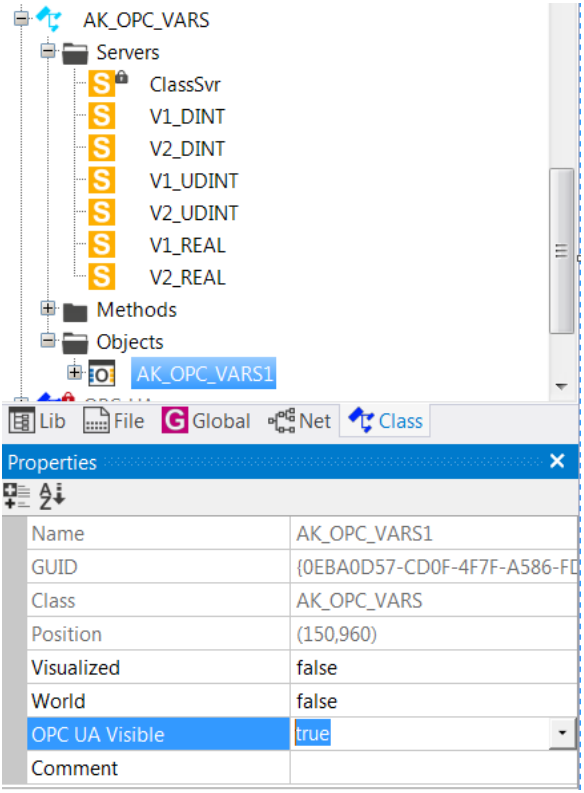
Here the following requirements have to be met:

→ In the properties of all desired **servers** the OPC attribute “Visible” has to be “true”

→ The OPC attribute “WriteProtected” depends on the application



→ In all instantiated **objects** containing servers of OPC-UA data transfer, the attribute "**OPC UA Visible**" has to be "**true**"



→ Transfer project to the PLC

The XML file is generated when compiling the program and transferred to the control only with the download. When resetting/starting the CPU the file is neither created nor changed. Now the OPC-UA client should be able to access the variables.

The program UaExpert can be used to test the program (see according chapter in this documentation).

6 Configuration Using an XML File

One (or more) user-specific configuration files are needed to inform the OPC UA server of the corresponding data points (variables), including attributes and directories.

As mentioned before, LASAL Class 2 automatically generates the OPC_UA.XML file. When downloading the project, this XML file is transferred to the CPU. Currently (LASAL V.02.02.153) the file is saved in the root. If a folder "*C:\OPC_UA*" exists, the file must be copied there manually.

However, a manually generated XML file can help to enable extended functionality. In this case the project setting "Enable OPC UA" must be switched off!

An XML file must be stored in the corresponding PLC, in which the OPC-UA server is running. An OPC-UA client can only access data points configured this way!

By default, the file is named "OPC_UA.xml", which must be located in the folder "*C:\OPC_UA*". If this folder is missing, the file must be in the root of drive C. With overwriting the virtual method OPC_UA::FunctSetUp name and path can be changed.

6.1 General Structure

OPC_UA.XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Config Version="1.0">
    <Trace TraceLevel="48" />
    <Release>
        <ReleasePath Path="C:\OPCUA\"/>
        <ReleasePath Path="E:\OPCUA\"/>
    </Release>
    <DataSet>
        <DataElement Hostname="ClassSvr"        Type="DINT"    Writeprotected="true"  Physic="" Unit="" Folder="MO1\Analyse"   Label="MyTask1.ClassSvr"/>
        <DataElement Hostname="ErrorCode"       Type="DINT"    Writeprotected="true"  Physic="" Unit="" Folder="MO1\Analyse"   Label="MyTask1.ErrorCode"/>
        <DataElement Hostname="CycleCounter"    Type="DINT"    Writeprotected="true"  Physic="" Unit="" Folder="MO1\Analyse"   Label="MachineData.CycleCounter"/>
        <DataElement Hostname="Test32"          Type="DINT"    Writeprotected="false" Physic="" Unit="" Folder="MO1\Analyse"   Label="MachineData.Test32"/>
        <DataElement Hostname="TestString"      Type="STRING"  Writeprotected="false" Physic="" Unit="" Folder="MO1\Analyse"   Label="TestString.Data"/>
    </DataSet>
</Config>
```

All configurations are mad in the "Config" tag. Here the sub tags "Trace", "Release", "Server" and "DataSet" are distinguished.

Trace	Element is optional. It defines the level for trace outputs. For possible values see the class description OPC-UA, area server - TraceLevel. Tag "Trace" attribute *TraceLevel* defines the trace level for trace outputs
Release	defines one or more shared directories for file up-/download Tag "ReleasePath" attribute *Path* defines the possible path(s) for file transfers
Server	Element is optional. It defines one or more "external" OPC-UA servers, that can be accessed by the "own" OPC-UA class as clients. See chapter "Client Data Transfer … "
DataSet	defines the individual data points in the PLC with the corresponding attributes. OPC-UA clients can access these data points! Tag "DataElement" with attributes *Type* (DINT, UDINT, REAL, STRING) *Hostname* data element name to write *Writeprotected* (true, false) *Physic* user specific text - optional *Unit* user specific text - optional *Folder* user specific folder - optional *Label* actual name of the data element (Object.Server) *IsAvailable* optional - specifies whether the node is visible in the address space `IsAvailable="False"` … fixed invisible `IsAvailable="True"` … fixed visible `IsAvailable="Object.Server"` … via class server (0, 1) *Userrole* optional - specifies a maximum of 32 write permissions (bit 0 to bit 31), also see user management OPC_UA-AddOn) `Userrole="0"` … no write permissions set `Userrole="-1"` … all write permissions set `Userrole="x"` … x = value for any bit mask There also are additional attributes for the definition of the nodes in the "external" OPC-UA servers in case of the optional client data transfer. See chapter "Client Data Transfer … "

6.2 Example for OPC-UA Variables with 2 Configuration Files

Config1.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Config Version="1.0">
    <DataSet>
        <DataElement Hostname="Vendor"      Type="STRING"  Writeprotected="true"  Physic=""  Unit=""  Folder="Info"  Label="Vendor"          />
        <DataElement Hostname="Version"     Type="STRING"  Writeprotected="true"  Physic=""  Unit=""  Folder="Info"  Label="Version"         />
        <DataElement Hostname="TestString"  Type="STRING"  Writeprotected="false" Physic=""  Unit=""  Folder="Info"  Label="TestString.Data" />
    </DataSet>
</Config>
```

Config2.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Config Version="1.0">
    <Release>
        <ReleasePath Path="C:\OPCUA\"/>
        <ReleasePath Path="E:\OPCUA\"/>
    </Release>
    <DataSet>
        <!-- Comment -->
        <DataElement Hostname="Counter" Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Shot"                  Label="MyData1.CycleCounter" />
        <DataElement Hostname="Test32"  Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Shot"                  Label="MyData1.Test32"       />
        <DataElement Hostname="TestU32" Type="UDINT" Writeprotected="false" Physic="" Unit="SEC" Folder="Shot"                  Label="MyData1.TestU32"      />
        <DataElement Hostname="TestF32" Type="REAL"  Writeprotected="false" Physic="" Unit="SEC" Folder="Shot"                  Label="MyData1.TestF32"      />

        <DataElement Hostname="M0_00"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1\Level2\Level3" Label="Machine0.Server0"     />
        <DataElement Hostname="M0_01"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1\Level2\Level3" Label="Machine0.Server1"     />
        <DataElement Hostname="M0_02"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1\Level2"        Label="Machine0.Server2"     />
        <DataElement Hostname="M0_03"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1\Level2"        Label="Machine0.Server3"     />
        <DataElement Hostname="M0_04"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1"               Label="Machine0.Server4"     />
        <DataElement Hostname="M0_05"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1"               Label="Machine0.Server5"     />
        <DataElement Hostname="M0_06"   Type="DINT"  Writeprotected="false" Physic="" Unit="SEC" Folder="Level1"               Label="Machine0.Server6"     />
    </DataSet>
</Config>
```

Note about strings

Labels for strings must always be specified with the "Data" server of the String object.

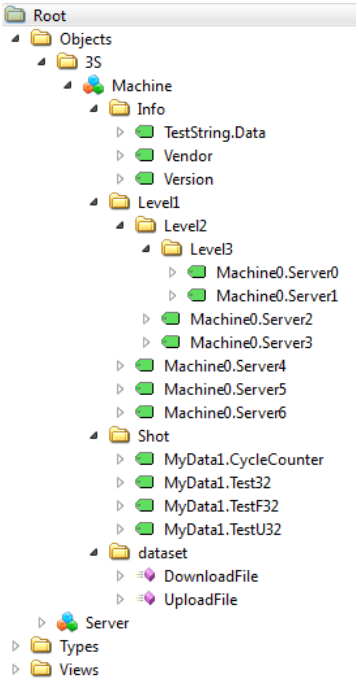
`Label="StrSerialNumber.Data"`

This is especially important for nested objects where the data server was "pulled out".

`Label="Testvars.StrSerialNumber.Data"`

Address Space of the Server

The two configuration files shown above define the following address space:



6.3 Transfer of Complex Data: ByteString

This paragraph deals with the transfer of constant and variable length ByteStrings.

6.3.1 ByteStrings of Constant Length

Since version 4.0 the transfer of complex data types is also possible. Here the ByteString is explained. The following points refer to a ByteString of fixed length. The differences to a variable length ByteString are shown below.

- → A ByteString can be used to transfer binary data that needs to be interpreted in the application.
- → The required variable must be defined within a user class as a member variable.
- → The address of the member variable must be written to a class server using a user program.
  This class server is to be declared as attribute **`Label`** in the XML file.
- → The **`Type`** here is "BYTESTRING"
- → In addition to the attributes already explained, an attribute called **`Size`** must be entered in the XML file. It defines the length of the ByteString and cannot change during runtime, independent of the transferred data. This can result in data either being truncated or, if the length is unknown, the data being inconsistent. See ByteStrings with variable length, further below.
- → Another attribute with the name **`MemoryAccessMode`** is required. This defines the access and has to be set to "1".
  - Server (0): If the mode is set to "MemoryAccessMode_Server", the received value is written directly to the server specified in the corresponding mapping.
  - ServerToMemory (1): If the "MemoryAccessMode_ServerToMemory" mode is set, the server holds in the class the address of a memory block. The user must ensure that sufficient memory is allocated to write the specified data to this address in memory.
  - Memory (2): The "MemoryAccessMode_Memory" mode cannot be used in a mapping. When calling a Get function, for example, the parameter lasal id should directly be the memory block without offset. In this case, no more checks are performed on the address.

Example for an XML entry:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Config Version="1.1">
    <Release>
        <ReleasePath Path="C:\OPCUA\"/>
    </Release>
    <DataSet>
        <DataElement Hostname="TestByteString"    Type="BYTESTRING" Mode="RW" Unit=""
            Label="TestBytestring.Addr" Size="100" MemoryAccessMode="1"/>
    </DataSet>
</Config>
```

6.3.2 ByteStrings of Variable Length

For a ByteString of variable length an object of the class OPC_UA_ByteString must be placed. This is connected with the object of the class OPC_UA. The **Label** attribute in this case does not point to a server, but to the name of the object directly. In addition, no length (**Size**) is specified. The MemoryAccessMode can be omitted for this case or is specified with 0.
The data of the ByteString can be accessed via the methods of MerkerEx. Further explanations can be found in the corresponding class documentation.

Example for an XML entry:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Config Version="1.1">
    <Release>
        <ReleasePath Path="C:\OPCUA\"/>
    </Release>
    <DataSet>
        <DataElement Hostname="TestByteStringDyn" Type="BYTESTRING" Mode="RW" Unit=""
            Label="TestBytestringDyn"/>
    </DataSet>
</Config>
```

6.4 Transfer of Complex Data: Structured Variable

Since version 4.0 the transfer of complex data types is also possible, here a structured variable is explained.

→ To be able to transfer structured variables, the design of the structure must also be known in the OPC_UA address space!
The structure of the structured variable is described in a **model file** (e.g. created with UaModeler).

In addition to a **Model.xml** you also need a **Mapping.xml**, which must be read using an **OPC_UA_AddressSpace module**.

More information and an example can be found in the client documentation and in the client demo project.

→ Also arrays and structure arrays are possible

→ The structure of the variable must be absolutely identical on the server side and on the client side

→ Pointers to data and texts within the variable are not supported.
Texts must therefore be created in the variable as ARRAY OF CHAR.

→ The required variable must be defined within a user class as a member variable.

→ The address of the member variable must be written to a class server using a user program.
This class server is to be declared as **Label** in the XML file.

Example for a XML entry in the mapping file:

```
<?xml version="1.0" encoding="utf-8"?>
<DataMapping>
    <NamespaceUris>
        <Uri>http://www.sigmatek-automation.com/CH/</Uri>
    </NamespaceUris>
    <Mappings>

        <Mapping NodeId="ns=1;i=6023" Label="OPC_UA_AddessSpace_Demo1.DataPtr" StructureTypeId="3002" Count="1" Size="26518" DataTypeId="22" MemoryAccessMode="1" />

    </Mappings>
</DataMapping>
```

→ Under **<NamespaceUris>** the namespace must be specified.
This must match the namespace that was assigned for the address space instances during XML creation (see Model.xml).

→ The mapping entries must be set in the element **<Mappings>**.
Here, a separate line must be created for each necessary link.
The format must be strictly adhered to.

→ The **NodeId** is the one of the respective address space instance

→ The **Label** specifies the name of the variable on the own station
This is the name of the class server where the address of the member variable can be found (this address must be written to the server by the user in the application).

→ The **MemoryAccessMode** specifies the abbess – here fixed to "1"
- Server (0): If the mode is set to "MemoryAccessMode_Server", the received value is written directly to the server specified in the corresponding mapping.
- ServerToMemory (1): If the "MemoryAccessMode_ServerToMemory" mode is set, the server holds in the class the address of a memory block. The user must ensure that sufficient memory is allocated to write the specified data to this address in memory.
- Memory (2): The "MemoryAccessMode_Memory" mode cannot be used in a mapping. When calling a Get function, for example, the parameter LasalID should directly be the memory block without offset. In this case, no more checks are performed on the address.

→ The **StructureTypeId** specifies the numeric ID of the data type in the Model.xml (can be chosen freely)

→ The **Count** specifies, how many elements exist in the array (1 to x)

→ The **Size** specifies, how many bytes a single array element has (1 to x)

→ The **DataTypeId** specifies the data type on the OPC_UA side
These IDs are defined by OPC_UA - here fixed "22" for Struct

7 OPC-UA Functions

7.1 Reading/Writing Variables

In this documentation reading and writing of variables has already be dealt with.

All servers of a LASAL project can be provided via configuration to the OPC UA server. Through the configuration, which variables can be read or written by the user is set.

Currently, the data types DINT, UDINT, REAL and STRING.

Recommendation:

Since access to the data entries from the OPC UA server cannot be triggered by user specifications, an appropriate interface between the OPC UA server and PLC is recommended.

7.2 MonitoredItems

For each OPC-UA variable that can be read by the client, a MonitoredItem can also be created on the client side. The monitoring of such items however works on the server side.

The refresh time for subscription/monitored items is negotiated between client and server. Our server supports a minimum of 50 ms.

Per created subscription it is possible to add 1000 MonitoredItems. If more than 1000 elements have to be monitored, they have to be divided into several subscriptions.

Recommendation:

Each monitored item must be internally monitored by the server for changes, which increases the load on the entire system.

There should only be as many MonitoredItems created as absolute necessary.
As rule, there is one trigger variable in the entire system. Using this variable, whether or not data has changed can be detected (e.g. cycle counter).

A MonitoredItem should be created for this field, so that the client can be automatically informed of changes by the server with an event. After a change trigger change, the client read the changed data with a single read process.

7.3 File Download

With this standard OPC-UA method, a single file can be sent from the OPC-UA client to the OPC-UA server.

No background knowledge concerning the server application is needed.

CLIENT: StatusCode = DownloadFile([input]String Filename, [input]ByteString Data)

Filename	Name, under which the file is stored for on the PLC. Optionally an absolute path can be given with the file name. If no path is defined, the path "C:\OPCUA\" is used. When entering an absolute path, it is validated according to the shared paths in the configuration. Only paths from the configuration and "C:\OPCUA" are valid.
Data	Contains the file content in the form of a byte string.
StatusCode	Return value indicates the success / failure of the method call.

7.4 File Upload

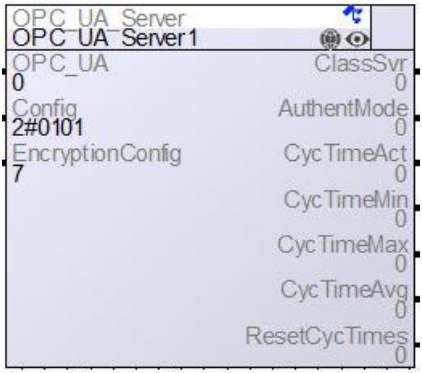
With this standard OPC-UA method, a single files can be read from the OPC-UA server and sent to the OPC-UA client.

No background knowledge concerning the server application is needed.

CLIENT: StatusCode = UploadFile([input]String Filename, [output]ByteString Data)

Filename	name, under which the file is searched for on the PLC. Optionally an absolute path can be given with the file name. If no path is defined, the path "C:\OPCUA\" is used. When entering an absolute path, it is validated according to the shared paths in the configuration. Only paths from the configuration and "C:\OPCUA" are valid.
Data	Contains the file content in the form of a byte string.
StatusCode	Return value indicates the success / failure of the method call.

8 OPC_UA_Server Class



For the functionality of the OPC-UA server an instance of this class has to be placed in a network.

On program start an own thread is created, in which the OPC-UA server runs. No further programming is necessary.

8.1 Interface Connections

8.1.1 Server

ClassSrv	ClassSrv
AuthentMode	Authentfication mode 0 ... authentification via user name/password (with file UserConfiguration.xml) 1 ... authentication is set to "anonymous"
CycTimeAct	current cycle time [µs]
CycTimeMin	minimum cycle time [µs]
CycTimeMax	maximum cycle time [µs]
CycTimeAvg	average cycle time [µs]
ResetCycTimes	command "RESET Cycle Times" Setting to #1 will reset the above cycle times

8.1.2 Clients

OPC_UA	Object channel to the OPC_UA object
config	Bit pattern for configuration Bit 0 .. Enable historical data 0 = "historical data is disabled" / 1 = "historical data is enabled" Bit 1 .. Enable historical events 0 = "historical events disabled" / 1 = "historical events enabled" Bit 2 .. Reserved Bit 3 .. Reserved Bit 4 .. Use old namespace URI 0 = "use the current URI" / 1 = "use the old URI" Note: The old URI shall only be used for compatibility reasons. In any other case this bit shall be set to 0!
EncryptionConfig	Bit pattern for configuration of the encryption support 0 = "Disabled" / 1 = "Enabled" Bit 0 ... SecurityPolicy None/SecurityMode None Bit 1 ... SecurityPolicy Basic256/SecurityMode Sign Bit 2 ... SecurityPolicy Basic256/SecurityMode Sign and Encrypt Bit 3 ... SecurityPolicy Basic256Sha256/SecurityMode Sign Bit 4 ... SecurityPolicy Basic256Sha256/SecurityMode Sign and Encrypt Bit 5 ... SecurityPolicy Aes128Sha256Rsa0aep/SecurityMode Sign Bit 6 ... SecurityPolicy Aes128Sha256Rsa0aep/SecurityMode Sign and Encrypt Bit 7 ... Reserved Bit 8 ... Reserved Note: only those encryptions may be activated which are actually supported by the OS used! RTK does not support SSL! SSL is basically supported from Salamander version 09.03.116. Basic256Sha256 and Aes128Sha256Rsa0aep are supported from Salamander version 09.03.160.

8.1.3 Global Methods

Init	Initializing and creating the OPC_UA thread.
OpcUaThread	OPC-UA services are processed in this thread.
FunctStart	Called once while starting the OPC UA server and signals the user that this service was started.
FunctRun	Called cyclically, if the service was started.
FunctSetUp	With this method, the OPC UA Server is declared the corresponding configuration files (.XML). It is also called directly once after starting. Several different configuration files can also be declared. This occurs through the function call. OUT retcode 0= OK otherwise error code (server is not started in this case!) The base implementation here loads the standard configuration file "OPC_UA.XML". So retcode complies to the return value of AddXmlConfig.
RegisterProvider	Registers the providers of all modules OUT retcode 0 ... OK / <> 0 ... RegisterProvider() incorrect
ProviderRun	Server provider function - is called by the OPC_UA service once per cycle.

SetParameter	Can be used to set process specific parameters IN ParaNr parameter number IN Value new value for the wanted parameter OUT retcode 0 = Parameter successfully set -1 Parameter was not set (wrong value, ...) valid parameter: 0 = **OPC_SET_OPTION** … setting an option Value: pointer to the option string Option strings for hiding functions from the OPC_UA address space: "DISABLE:ACTIVATE_DATASET_EXT_INFO" "DISABLE:PREPARE_DATASET_EXT_INFO" "DISABLE:DOWNLOAD_FILE_EXT_INFO" "DISABLE:UPLOAD_FILE_EXT_INFO" "DISABLE:GET_ALL_ACTIVE_STANDARD_ALARMS" 1 = **OPC_UA_PAR_SET_PORTNR** ... Port number for IP-Adresse Value: 1 = 1024 bis 49151 **Attention**: must be called in the **Init BEFORE** the **_FirstScan**! 2 = **OPC_UA_PAR_SET_SENDNOCERT** Service don't send Server Certificate for non-secure connections 3 = **OPC_UA_PAR_SET_DATALOGGERMAXITEMS** Defines the maximum number of elements for data logging
GetNamespaceIndexByUri	Returns the namespace index for a given namespace URI. IN namespaceUri Namespace URI string from requested namespace OUT nameSpaceIndex Namespace index for the requested namespace -1 = The requested namespace is not registered in the server namespace array
SetDisplayName	Sets the DisplayName of the destination node. IN nodeId Id of the destination node IN locale Locale of the DisplayName IN displayName Translated DisplayName (corresponding to the locale) OUT result 0 = no error / <> 0 = OpcUa error code

SetBrowseName	Sets the BrowseName of the destination node. IN nodeId Id of the destination node IN namespaceIndex Namespace index of the BrowseName IN browseName BrowseName String OUT result 0 = no error / <> 0 = OpcUa error code
GetCycTimeMin	returns the current value of the server "CycTimeMin"
GetCycTimeMax	returns the current value of the server "CycTimeMax"
GetCycTimeAvg	returns the current value of the server "CycTimeAvg"
ResetCycleTimes	calling this method resets all cycle times
GetCurrentSessionCount	OPC_UA Diagnosis - returns CurrentSessionCount
GetCumulatedSessionCount	GetCumulatedSessionCount
GetReadCount	OPC_UA Diagnostics - returns the sum of all ReadCounts of all currently opened sessions of all clients
GetWriteCount	OPC_UA Diagnostics - returns the sum of all WriteCounts of all currently opened sessions of all clients
GetBytesRead	OPC_UA Diagnostics - returns the sum of all Read Bytes of all currently opened sessions of all clients
GetBytesWritten	OPC_UA Diagnostics - returns the sum of all Written Bytes of all currently opened sessions of all clients
GetTotalNodesCount	OPC_UA Diagnostics - returns TotalNodesCount
GetRejectedSessionCount	OPC_UA Diagnostics - returns RejectedSessionCount
GetRejectedRequestsCount	OPC_UA Diagnostics - returns RejectedRequestsCount
GetSessionAbortCount	OPC_UA Diagnostics - returns SessionAbortCount
GetSessionTimeoutCount	OPC_UA Diagnostics - returns SessionTimeoutCount
GetCumulatedAccessCount	OPC_UA Diagnostics - returns "CumulatedAccessCount" (sum of all types of client accesses, such as read, write, call,...)
AlarmChanged **StandardAlarmChanged**	With this method, a change in an alarm can be sent to the OPC UA server. Each change of an alarm has to be reported. Both activation and deactivation of an alarm. This method keeps the list of current alarms up to date. Together will all other alarm related functions, this call has to be executed threadsafe. IN alarm information about the changed alarm OUT state 0

AlarmChangedUC	Complies to the method "AlarmChanged". The difference is that strings in this method are transferred as an array of 16-bit values. So any UniCode characters can be transferred. Together will all other alarm related functions, this call has to be executed threadsafe. IN alarm information about the changed alarm OUT retcode 0
DatasetPreparationFinished	With this method, the OPC UA server can be informed that the preparation of a data set for transmission to a client has been completed. IN status status of the activation (0: error free, !=0: error code) IN datasetId unique Id of the transfer / data set IN datasetName name of the settings data set IN path path where the according file was saved OUT state 0 Using OPC UA Event, OPC UA clients can be informed when settings data has been prepared. A client can then read the settings data via the “DownloadFile”, method from the OPC UA server.
DatasetActivationFinished	With this method, the OPC UA server can be informed when settings data is activated. IN status status of the activation (0: error free, !=0: error code) IN datasetId unique Id of the transfer / data set IN datasetName name of the settings data set IN path path where the according file was saved OUT state 0 Using OPC UA Event, OPC UA clients can be informed when settings data is activated.
AllActiveAlarms	With this method, the list of active alarms can be sent to the OPC UA server. This method must be called only once during the program start to initialize the list of active alarms. Together will all other alarm related functions, this call has to be executed threadsafe. IN alarmList pointer to the list of active alarms IN listCount number of alarms in the list OUT state 0 Via the "GetAllActiveAlarms" function, OPC UA clients can query the list of active alarms.
RefreshUserConfiguration	The call leads to the reloading of the user XML file.

GetClientDiagnosticInfos	OPC_UA Diagnostics - returns client/session specific information to a structured memory using a pointer. ATTENTION: After calling this method, the "ClearClientDiagnosticInfos()" method must be called after evaluating the data so that the memory is cleared again.
ClearClientDiagnosticInfos	Clears the memory allocated by the "GetClientDiagnosticInfos()" method.
CreateAddressSpaceDescription	Method to create and export the “address space information” to an XML file named "model_definition.xml".
LogHistoryData	This function gets called whenever a datapoint must be logged for historical access, in this function the DataLoggerBase class gets called if connected. IN primaryKey — The primary key of the node to store IN statusCode — The status code of the value to be stored IN sourceTime — Source timestamp of the value to store IN serverTime — Server timestamp of the value to be stored IN valueType — The type of the value to be stored IN dataLength — The length of the byte string to store IN data — The pointer to the byte string to store IN userData — Additional parameter which can be used later in order to be able to use the correct logger for the given data value or any further information which might be of help. OUT retcode — Return code for storing 0 = storing fine / <> 0 = failure with error number

ReadHistoryData	This function gets called whenever a history read has to be performed. IN primaryKey — Primary key of the node to read IN startTime — Start time of values for the node to read IN endTime — End time of values for the node to read IN isInverse — If the data should be read in inverse direction IN numValues — Pointer to the number of values to be read IN results — Pointer to where the results should be stored, the memory is already allocated IN continuationPoint — Pointer to the continuation point which needs to be set if needed The continuation point contains a timestamp where the last values has been written from and an continuation offset, which indexes the number of values that have been read at the exact same time IN continuationOffset — The continuation offset from the request. If set, it indexes the number of values that have been read at the exact same start time IN userData — Additional parameter which can be used later in order to be able to find the correct logger to read from or any further information which might be of help. OUT retcode — Returncode of reading 0 = reading fine / <>0 = error (for example out of memory)
LogHistoryEvent	This function gets called whenever a event has to be logged for historical access, in this function the DataLoggerBase class gets called if connected. **Attention**: This method is currently not implemented! IN primaryKey — Primary key of the event to be stored IN noOfEvents — Number of events to store IN variants — The variant containing the events IN userData — Additional parameter which can be used later in order to be able to use the correct logger for the given data value or any further information which might be of help. OUT retcode — Return code of the function 0 = success / <> 0 = failure with error code
RegisterDataLogger	Registers a data logger to the server which is used to store data for historical access. The data logger must be an implementation of the OPC_UA_DataLoggerBase. The method is usually called by the concrete implementation of the DataLogger, for example in Init. IN pDataLogger — The pointer to the implementation of an OPC_UA_DataLoggerBase. OUT retcode — 0 = success / -1 = the pointer to the logger is invalid

RegisterEventSource	With this method, a nodeId is registered as event source. This is relevant if the events shall be filtered, i.e. the notifier is different to the Server node. Without registering, the event will be sent from the Server node as notifier. If registered, the closest parent of the given nodeId will be the notifier if an event occurs. If registration fails, the event will still be notified by the Server node. IN NodeId — The nodeId which should be registered as event source for the closest parent which is a notifier. OUT retcode — The return code indicates if adding the node as source was successful (OpcUa_Good = 0) or not. For detailed error description see the list of error codes at the end of the OPC_UA_Client's documentation.

8.1.4 Private Methods

OPC_UA_Server	Constructor initializes the OPC-UA interface
AddServerFacets	Adds the specified strings as facets to the ServerProfileArray (i=2269). The method does not check whether the specified URIs are unique or valid, but only takes the content. IN facets The array of strings with facet URIs to be added. IN noOfFacets The number of strings to transfer and add to the array. OUT result Returns 0 on success or an OPC_UA error number.
SetOptions	With this function option can be set that change the program sequence. IN options "USE:HOSTNAME-AS-BROWSENAME" "USE:ALPHANUMERIC-IDENTIFIERS" "DISABLE:ACTIVATE_DATASET_EXT_INFO" "DISABLE:PREPARE_DATASET_EXT_INFO" "DISABLE:DOWNLOAD_FILE_EXT_INFO" "DISABLE:UPLOAD_FILE_EXT_INFO" "DISABLE:GET_ALL_ACTIVE_STANDARD_ALARMS" OUT retcode 0
Setvalue32Changed	With this method, changes (of type DINT) to the settings data can be sent to the OPC UA server. Together will all other “Setvalue..Changes” functions, this call has to be executed threadsafe. IN change general properties of the parameter change IN oldValue value before the change IN newValue current value / value after the change OUT state 0 Using OPC UA Event, OPC UA clients can be informed of changes in the settings data.
SetvalueU32Changed	Equivalent to the "Setvalue32Changed" method for the data type "UDINT".
SetvalueF32Changed	Equivalent to the "Setvalue32Changed" method for the data type "REAL".
SetvalueStringChanged	Equivalent to the "Setvalue32Changed" method for the data type "CHAR". Together will all other “Setvalue..Changes” functions, this call has to be executed threadsafe. IN change general properties of the parameter change IN oldValue value before the change IN newValue current value / value after the change OUT state 0

SetvalueStringChangedUC	Complies to the method "SetvalueStringChanged". The difference is that the input parameter in this method is transferred as an array of 16-bit values. So any UniCode characters can be transferred. Together will all other “Setvalue..Changes” functions, this call has to be executed threadsafe.
InitDatasetWorkingPath	With this method, the default path for operations with settings data for runtime can be defined. This path is, in addition to paths from the configuration, valid for all file operations. If this path is set, it is used as the default path for file operations without path specifications. E.g.: If the DatasetWorkingPath was set to c:\datenset\, the file is stored in the c:\datensatz\text.txt“ directory when UploadFile is called with the "test.txt" parameter. IN path default path specification OUT retcode 0
InitFileSystemCallback	With this method, the OPC UA server can provided with a callback function. IN f_CB_fileSystem Callback for FileSystem changes OUT retcode 0 The "f_CB_fileSystem" callback is called when a client triggers a change in the file system with a function call. All file functions (Upload File, Download File, Activate Dataset, Prepare Dataset) for example, thereby trigger changes in the file system and subsequently call this callback function.
InitDatasetCallback	With this method, the OPC UA server can be provided with two callback functions. IN f_CB_activateDS CallBack for activating settings data sets f_CB_prepaireDS Callback for providing settings data sets OUT retcode 0 The Callback (CB) "f_CB_activateDS” is called, if a client wants to transfer and activate a data set to the control. This callback is used in the control program as a trigger for reading and activating the desired settings data set. The "f_CB_ prepaireDS" callback is called when a client requests a data set. This callback is used in the control program as a trigger for providing desired settings data set.
InitAlarmCallback	With this method, the OPC UA server can provided with a callback function. The OPC UA server calls this function during initialization. With this method, the OPC UA server requests the list of all active alarms. IN f_CB_alarmList pointer to the Callback function OUT retcode 0
IniGetStringArrayCallback	This method can be used to provide the OPC UA server with a callback function for reading string arrays
IniSetStringArrayCallback	This method can be used to provide the OPC UA server with a callback function for writing string arrays

8.2 FAQs on performance data and memory requirements

→ **How much memory is required?**
The OPC_UA server (classes OPC_UA and OPC_UA_Server) requires memory for setting up the OPC_UA standard address space and OPC_UA environment, approx.:
… 5.42 MB code memory
… 4.96 MB RAM (3.72 MB of them for UserHeap)
The further memory requirement depends on the number of data points.

→ **How many data points does the class support as OPC_UA server?**
Technically unlimited, memory dependent

→ **How many clients can connect to the OPC_UA server?**
The number is limited to 50 or depends on the available memory.

→ **Memory requirement per server data point for external clients?**
approx. 1,244 bytes per data point for reading in the XML file, etc.

→ **CPU load per server data point and per connection to external client?**
1: The start time of the OPC_UA server is strongly dependent on the number of OPC_UA variables and the CPU used - it can be between a few seconds up to more than one minute.
2: The duration of a connection establishment to an external client is to be evaluated exactly as under point 1.
3: The CPU load for data exchange after connecting a client depends mainly on the client.

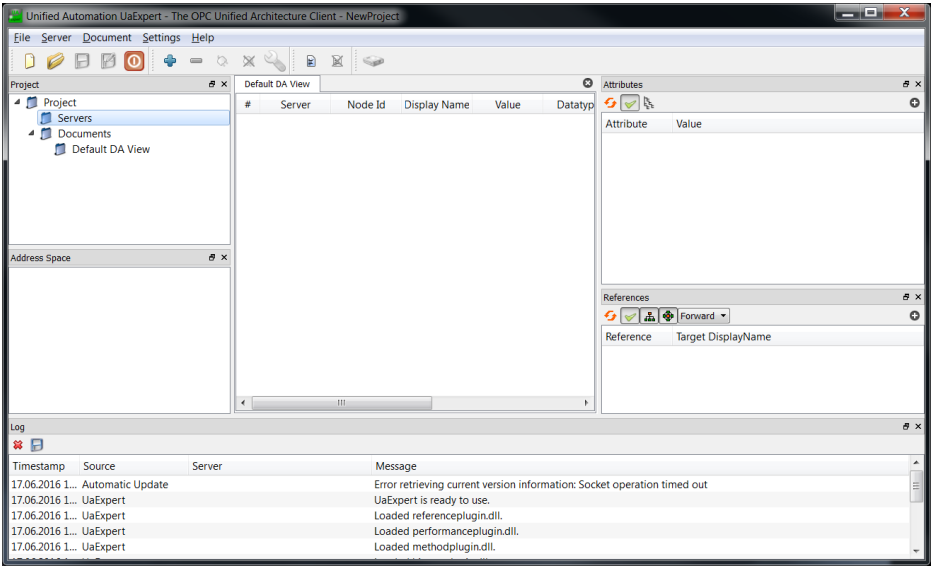
→ **Does the class work anonymously or with logon?**
Anonymous, with username/password and with encryption

9 Win Program UaExpert

This program is not from SIGMATEK and not necessary for the real operation of an OPC-UA communication.

But this tool can help in first commissioning.

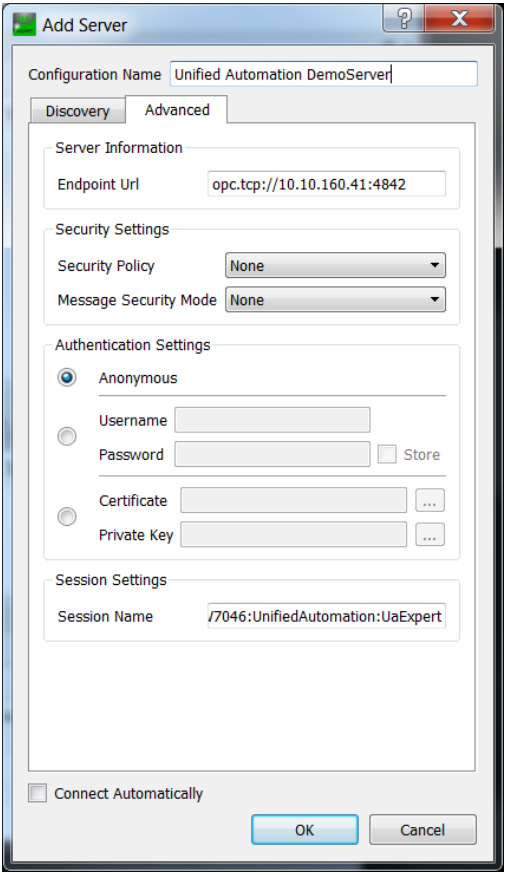
9.1 Setup Connection



After the first start no project exists.

Right click on “servers” in the section Project - click on “Add...”

continue on the next page ...



here the entpoint URL has to be entered

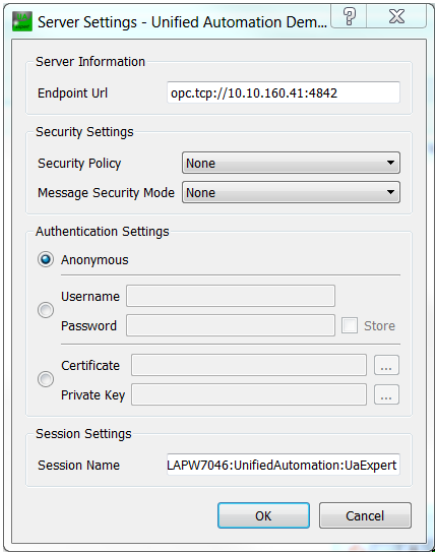
```
opc.tcp://10.10.160.41:4842
```

… the IP address is that of the PLC
… the port is 4842 for SIGMATEK

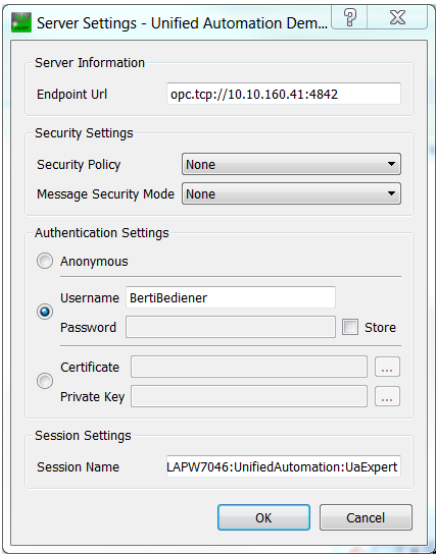
9.2 Establish Connection

Before establishing the connection, the type of authentication should be set in the server settings.

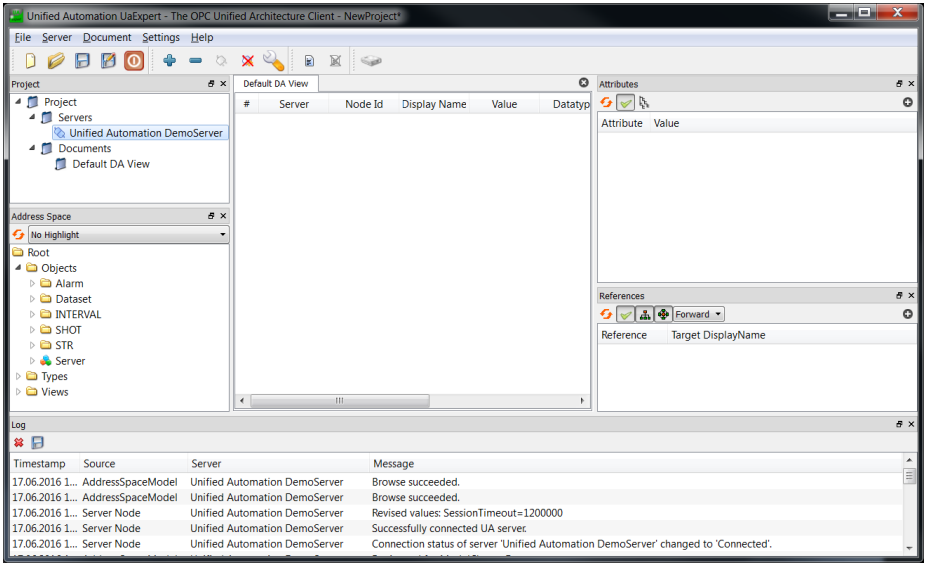
A: anonymous login



B: login with user/password



This setting must match the type of authentication of the OPC UA server!
See chapter "Access Authorization" within this documentation.



The connection is established by right-clicking on the server and then clicking on "Connect".

If the login with user/password is selected, you will be asked to enter the password before the actual connect.

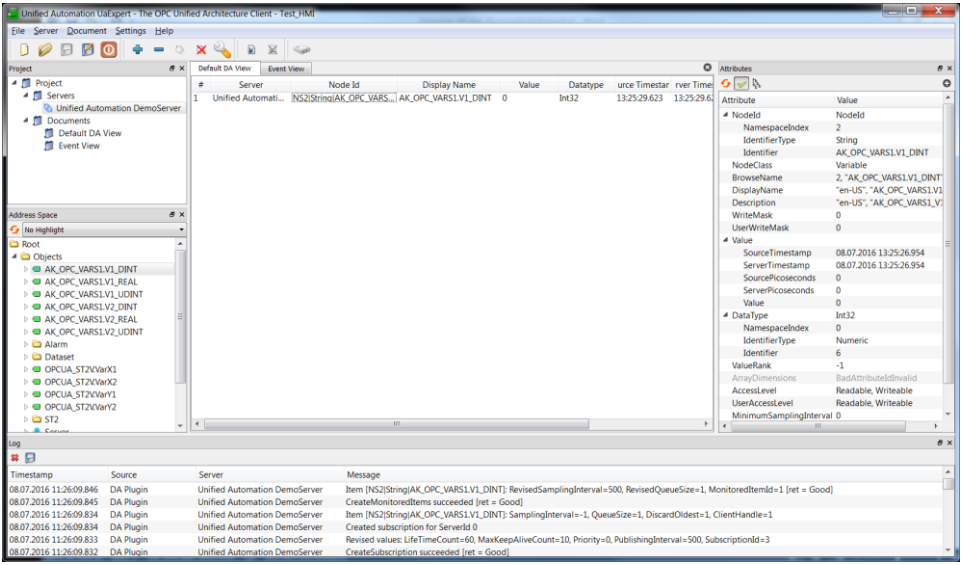
Now automatically the OPC-UA client requests the variable list from the OPC server and displays it here.

The OPC server generates this variable list according to its own OPC XML file.

9.3 Data Exchange

The normal data exchange in SIGMATEK systems is realized with server variables.

Supported at the moment: DINT, UDINT, REAL and STRING



→ in the area “AddressSpace” on the left side the variable list read from the PLC can be found

→ the value of the according variable can be found to the right under “Value”

→ updating here is only done when selecting

→ with Drag & Drop a variable can also be moved to the center view field

→ variables placed here are updated cyclically

→ the values can be changed

9.4 Alarms

For alarms OPC-UA provides a flexible alarm handling.

The base class cannot access the SIGMATEK alarms.

For this SIGMATEK provides an expanded OPC-UA class.

9.5 Events

OPC-UA offers a flexible event system for events.

The base class cannot access the SIGMATEK events.

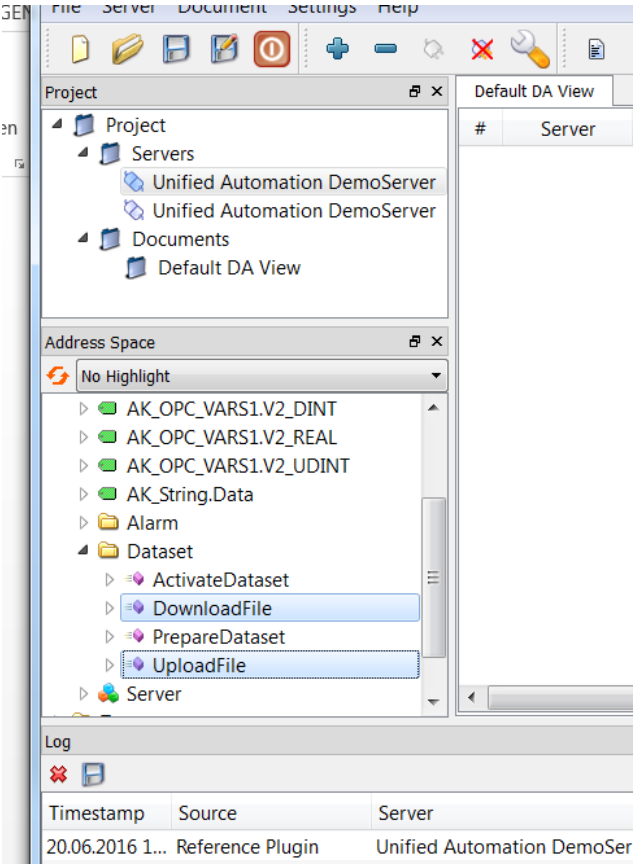
An expansion to also support this functionality is already planned.

9.6 File Transfer

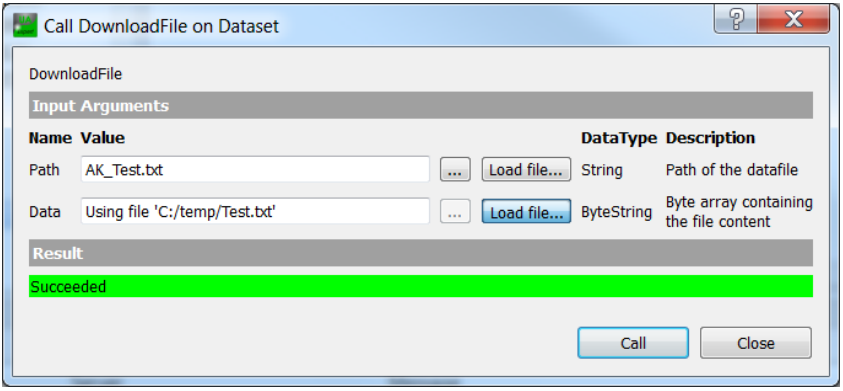
A file transfer can be executed in both directions. But in both cases the client writes.

In the base class the server has no influence on the time of the transfer.

In the program UaExpert, these functions can be found on the left side in the section “Address Space” in the entry “Dataset”. The functions are called by right-clicking on them and selecting "Call".



9.6.1 File from Client (Win) to Server (PLC)

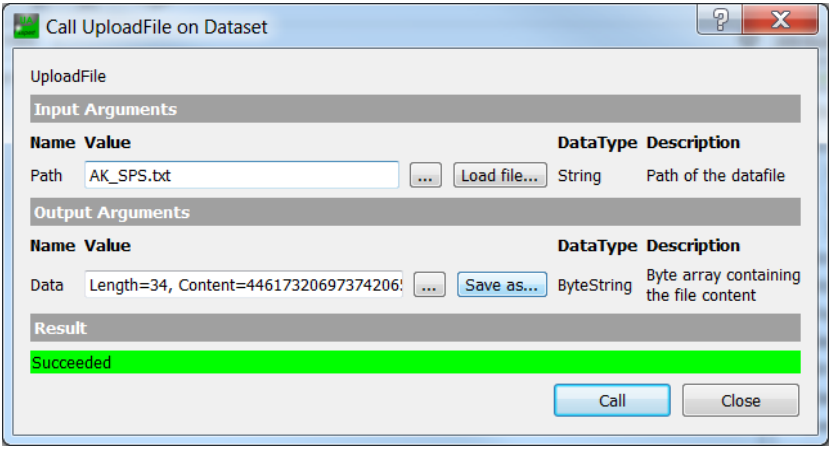


Path	defines the target path on the PLC including the file name. This path has to be entered in the PLC XML file in the section Release! Default: "C:\OPCUA\" Without path definition the first DIR entered in the XML is used.
Data	defines the source on the client.

An error indicates, that e.g. the OPC-UA DIR is missing on the PLC.

It is only possible to write to this DIR!

9.6.2 File from Server (PLC) to Client (Win)



Path	defines the source path on the PLC including the file name. This path must be entered in the PLC XML file in the section Release! Default: "C:\OPCUA\" Without path definition the first DIR entered in the XML is used.
Data	The data field remains empty until the "Call" button is pressed - this transfers the file to the client, but does not yet save it. The "Save as" button opens a file browser for saving to the target file.

An error indicates, that e.g. the OPC-UA DIR is missing on the PLC.

It is only possible to read from this DIR!

10 Appendix A

	Features List Sigmatek OPC-UA Server				
Function	**v02.01.005**	**v02.02.001**	**v02.03.001**	**V02.05.004**	**Comment**
	Connections & calls				
Connection of multiple clients simultaneously	yes	yes	yes	yes	Up to 50 sessions possible
ReadNode	yes	yes	yes	yes	
WriteNode	yes	yes	yes	yes	
Subscriptions/MonitoredItems	yes	yes	yes	yes	
CallMethods Client=>Server	only EM77	yes	yes	yes	
Triggering events	only EM77	yes	yes	yes	
	Encryption				
Encrypted connection via Basic256 S&E	yes	yes	yes	yes	
Encrypted connections via Basic256Sha256, and Aes128Sha256Rsa	no	no	yes	yes	
Encrypted connections via Aes256Sha256Rsa	no	no	no	no	
Creating Self-Signed Certificates	yes	yes	yes	yes	
Certificate management with manual/auto-trusting	yes	yes	yes	yes	
Ignore certain certificate failure	no	no	yes	yes	This is not foreseen for productive use!
	Data types				
Basic data types	yes	yes	yes	yes	
Arrays of basic data types	no	yes	yes	yes	
Structures without strings	yes	yes	yes	yes	
Arrays of structures without strings	no	yes	yes	yes	
Structures with strings (fixed length)	no	yes	yes	yes	
Arrays of structures with strings (fixed length)	no	no	no	no	
Structures with strings	no	no	no	no	
Arrays of structures with strings	no	no	no	no	
	Browsing				
BrowseNodes	yes	yes	yes	yes	
TranslateBrowsePathsToNodeIDs	yes	yes	yes	yes	
	Further Services				
User administration with role system	yes	yes	yes	yes	
File transfer	yes	yes	yes	yes	
Discovery GetEndpoints	yes	yes	yes	yes	
Historical Access data	no	no	yes	yes	
Historical Access events	only EM77	only EM77	only EM77	only EM77	
Discovery	no	no	no	no	Concept phase
PubSub	no	no	no	no	Concept phase