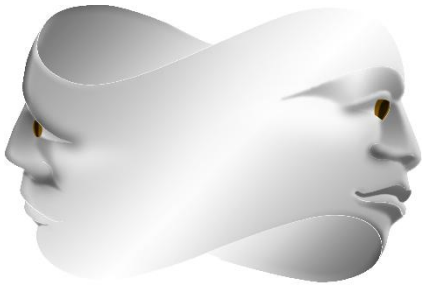


OPC-UAC 2.0

OPERATING MANUAL

OPC UA Client library for Windows applications.

INDEX

1 Introduction

OPC-UAC is a library for connecting Windows applications with OPC UA servers.

OPC-UAC is available in the RUNTIME commercial formula; the license is for a single application. The operation of the library is linked to the presence of a specific hardware or software activation; in the absence of such activation, the communication will be limited to 15 minutes only.
For activation management refer to the paragraph "Library License Activation" on page 5.

After installing the product, refer to the demo applications described in "Demo applications" on page 45 for a very first test.

2 Characteristics

The library provides a simple interface that supports reading, writing, and monitoring of changes in variable values.

OPC-UAC library supports:

- connection to an unlimited number of OPC UA servers
- discovery server su host
- connection mode:
  - Trasport: UA-TCP UA-SC UA-Binary
  - Security policy: None, Basic256, Aes128-Sha256-RsaOaep, Basic256Sha256, Aes256-Sha256-RsaPss
  - User Authentication: Anonymous, UserName
  - Management of self-signed certificates
- "Data Access" service
  - reading and writing in single and one dimension array mode of the following data types:
    - Boolean
    - Byte and SByte
    - UInt16 and Int16
    - UInt32 and Int32
    - Single (Float)
    - Double
    - String
    - ByteString
    - LocalizedText (only 'Default language')
  - Reading of optional information:
    - Data quality
    - server timestamp
    - source timestamp
  - writing management in "SafeWrite" mode
  - array writing with "IndexRange" mode
  - monitor of value changes (DataChange Subscriber)
- "Method call" service (only with .NET interface)

3 Minimum requirements

The minimum requirements for the running of the OPC-UAC library are those required by the .NET Framework 4.6.2 system.
According to the information provided by Microsoft, the minimum version of the Operating System is:

- Client - Microsoft Windows 7 SP1
- Server - Windows 2008 R2 SP1 Server

4 Limits

Compared to the basic functions, the following limits are noted for OPC-UAC:

- It does not support the following data types:
  - Int64
  - UInt64
  - DateTime
  - Guid
  - XmlElement
  - NodeId
  - ExpandedNodeId
  - StatusCode
  - QualifiedName
  - ExtensionObject
  - DataValue
  - Variant
  - DiagnosticInfo
  - Number
  - Integer
  - UInteger
  - Enumeration
- Array values: only one dimension arrays are supported

5 Skills

OPC-UAC requires the following skills for use:

- minimal knowledge of OPC UA technology and terminology
- minimal knowledge of certificate management
- minimal knowledge of Ethernet TCP/IP communications

6 Release notes from version 1.0 (March 2021)

6.1 Version 2.0 (July 2025)

New features:

- User authentication support with Username and Password
- Facilitation for connection with servers without security
- Support of the LocalisedText type (default language only)
- Support of Method call
- Array write support with IndexRange mode (partial writing)

Fixed issues:

- Configurator tool: possible hang situation when browsing server resources
- Runtime: possible temporary return of an old value after repeated writing operations

7 Installation

OPC-UAC is distributed on CD ROM and is also available from http://www.automa.it in the "FreeDownload" section.

To activate the installation procedure:

1. run the program "Setup_OPC-UAC_Runtime.exe". The program is present in the CD ROM or in the files downloaded from the Internet.

2. the installation is almost completely automatic. The user is asked for the folder where OPC-UAC should be installed, by default the folder "\Programs (x86)\Automa\Communication Tools\OPC-UAC 2.0" is suggested.
   The demo applications are installed in the folder "<APPDATA>\Automa\Communications Tools\OPC-UAC 2.0" to avoid problems related to UAC handling. The path where "<APPDATA>" folder is located depends on the Operating System you are using; to access it quickly you can open a "File explorer" window and enter the symbolic name "%APPDATA%" in the address bar.

Three subfolders will be created in the installation folder:

Folder	Content
DRIVER	OPC-UAC library files and configurator tool
USER GUIDE	This document in PDF format
TOOLS	Tools for managing hardware and software keys and redistribution files of Microsoft components

The following subfolder will be created in the folder "<APPDATA>\Automa\Communication Tools\OPC-UAC 2.0":

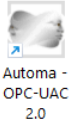
Folder	Content
DEMO	Demo applications and configurator tool

In the system folder "Program Data" (default: "C:\ProgramData") is also created the folder "Automa" where the "Certificate Store" is located; this is where the 'Application Certificate' generated for the driver and configurator tool and any certificates of the OPC UA servers used will be stored.
The store consists of a set of folders with the typical structure of a PKI (Public Key Infrastructure) system:

- Own folder: certificates relating to the configurator tool and runtime driver are present
- Rejected folder: this contains any server certificates that have not been accepted
- Trusted folder: the certificates of the servers that have been validated are present

The installer, if needed, will install the ".NET Framework 4.6.2 Runtime" and "Microsoft Visual C++ 2015-2022 Redistributable (x86)" components.

In the Windows "Start" menu the group "Programs" ▶ "Automa" ▶ "Communication Tools" ▶ "OPC-UAC 2.0" is created; in addition the element "Automa - Super-Flash OPC-UAC 2.0" containing the shortcut links for quick access is created on the Desktop.



8 Library use

The information needed to use the library is provided below.

8.1 Deployment

In the "DRIVER" installation folder there are the files needed to run the configurator tool and the applications.

Starting from the element on the Desktop:

The library folder can be accessed by selecting the shortcut named "Driver".

In the folder there are the following files:

Area	File	Description
Configuration tool	OPCUACCFG.EXE	OPC-UAC configuration tool
	Opc.Ua.ClientControls.dll	
	Opc.Ua.QuickstartsLibrary.dll	
	OPCUACCFG.Config.XML	File containing parameters that regulate the operation of the configurator tool.
Library	OPCUAC.DLL	Main module of the OPC-UAC library
	OPCUAC.Config.XML	File containing parameters that control the library operation.
Common	AOPCUALibrary.DLL	DLL files common to the library and configuration tool.
	BouncyCastle.Crypto.DLL	
	Opc.Ua.Client.ComplexTypes.DLL	
	Opc.Ua.Client.DLL	
	Opc.Ua.Configuration.DLL	
	Opc.Ua.Core.DLL	

To run the configuration tool you need to:

- Copy in a folder the **Configuration tool** and **Common**

To run the application you need to:

- Copy **Library** and **Common** to the same directory where the executable file is located
- Copy the configuration file OPCUAC.XML (prepared using the configurator tool) to the working directory of the application

You should also make sure that the following Microsoft components are present in your system:

- Microsoft .NET Framework 4.6.2
- Microsoft Visual C++ 2015-2022 Runtime Files

In the "TOOLS" installation folder there are two executables provided by Microsoft for the installation of the two components, respectively:

- NDP462-KB3151800-x86-x64-AllOS-ENU.exe
- VC_redist.x86.exe

The "Certificate Store", if not previously prepared, is automatically created the first time the configuration tool or the application is run.

8.2 Library License Activation

The continuous operation of the OPC-UAC library is linked to the presence of a specific hardware or software activation. In the absence of such an activation, operation is limited to 15-minute sessions.

Two types of keys are supported, each requiring a specific installation procedure:

Hardware key: USB hardware device that does not require the installation of any management software.

Software key: "OPCUAC.CTK" file containing the activation code.
The software key is strictly related to the single PC and must be activated on the PC where the application will be executed (target machine).
During the development phase, if a different PC is used, OPC-UAC can be used without the key in demo mode (15 minutes sessions); otherwise (for example if it is necessary to carry out a test of the application that requires continuous operation for a longer time than that of the single 15 minutes demo session) it is necessary to have a second software key.
The software protection key requires the presence of an Ethernet network card

The following procedure must be followed to use the software key:

1. on the PC where you want to activate the driver, run the tool "WRFCCode.exe" and detect the Customer Code
2. communicate the customer code via e-mail to Automa (sales@automa.it); the company will generate the activation code and provide the file "OPCUAC.CTK"
3. copy the file "OPCUAC.CTK" to the root directory of the application

The "WRFCCode.exe" tool is present in the "TOOLS" folder of the driver installation path and it is available in the free download section of the Automa site.

8.3 Library interfaces

The library consists of the main OPCUAC.DLL module and a number of other dependent DLLs based on the .NET Framework v.4.6.2.

The OPC-UAC library provides two interfaces:

1. .NET assembly interface (OPCUAC namespace): primary preferential interface
2. "C" interface (Win32 C function set): secondary interface with some limitations

The following accessory files in the OPCUACDemo_VC2010 demo projects are also provided for use of the "C" interface and .NET interface in older Visual Studio versions:

- OPCUACINTERFACE.CS: contains the definition of constants, structures, and methods to support the use of the DLL in C# Windows Form applications via P/Invoke.
- OPCUAC.H: contains constants, structures and function prototypes
- OPCUAC.LIB: Import Library file (for implicit link)

For simplicity the typical terms of C programming will be used also in the case of references to corresponding elements in other languages; example:

- Function -> function/method/delegate
- Structure -> structure, class
- Constant -> constant value defined by #define directive or by static field of a class

The following paragraphs explain the operating principle of the library, the database configuration and a description of the functions by category.

8.4 Operating principle

8.4.1 General

The OPC-UAC library plans to operate with a pre-configured database contained in the "OPCUAC.XML" file.
The configuration of the database is done with the configurator tool in which you define the connections with the servers, the subscriptions with the list of "Variable" type items you want to access and possible methods.
Each item is associated with a unique alphanumeric identifier that masks and virtualizes references to NodeId (unique identifier of the item in the server).
The library provides a very simple interface that allows you to read, write and, optionally, receive notifications of variations in "Variable" type item values and execute method calls.
Connections to servers are managed automatically and transparently, including possible reconnections.
Certificate management is provided in a simplified mode: during initialization the library generates, if necessary, a self-signed certificate and during connection it automatically accepts the self-signed certificates of the servers.

For the continuous working of the library it is also necessary to make sure that you have correctly activated the license: refer to the paragraph "Library License Activation" on page 5.

Notes
Please note that the connection to the OPC UA servers is an operation that may take a few seconds. Read requests made during this initial connection phase may fail with a specific connection not established error.
For a description of the possible error codes, please refer to "OPC UA Communication errors" on page 51.

8.4.2 Values reading

The library keeps a cache with the last available value of each item.
After having established the connection with the server, the library executes for each item a reading to initialize the value and activates the subscription service which, when the value changes, will send an update notification to the client.

The update frequency of an item is defined by the "Sampling Interval" of the individual item and the "Publishing Interval" parameter configured in the subscription containing the item

If the server does not support the subscription service at each request the library will read the data from the server.
The library also allows you to force the re-reading of the value from the server.

8.4.3 Values writing

The write operation is carried out in blocking mode for the time necessary to guarantee the delivery of the value to the peripheral or to determine its eventual impossibility.
Only in special cases, when the "Safe Write" mode is also active, the procedure may take additional time. See the notes under "Modify an item" on page 16.
In the case of array-type items, the library uses the following methods, depending on the elements to be updated and the capabilities of the server:

- immediate writing of the entire value: this is carried out when the modification of a number of elements equal to or greater than the current size of the array is required.
- partial write with the "IndexRange" mode: is carried out when only some elements need to be modified and the server supports this mode.

- write with RMW mode: this is carried out when only some elements need to be modified and the server does not support the "IndexRange" mode. The procedure involves a prior read of the entire value, the modification in memory of the elements and the rewriting of the entire value.

Note

Please note that starting from version 2.0 the driver uses the "IndexRange" mode by default (if available on the server).
If necessary, it is possible to disable this functionality by directly editing the "OPCUAC.XML" configuration file with a standard text editor, adding or modifying the "WriteFullArrayOnly" parameter with the value "true" for the server concerned.
To locate the edit point, find the line with the "UserDescription" parameter containing the description of the server (in the example "Plant PLC") and add or edit the parameter as shown below.

```
<COPCCServerCfg>
  <DisableDomainCheck>false</DisableDomainCheck>
  <SafeWriteTimeout>1000</SafeWriteTimeout>
  <SessionTimeout>20000</SessionTimeout>
  <UseSecurity>true</UseSecurity>
  <UserDescription>Plant PLC</UserDescription>
  <WriteFullArrayOnly>true</WriteFullArrayOnly>
```

In the configurator tool, inside the top right-hand pane, the "Array" attribute indicates whether or not the value supports partial writing.

Example:

8.4.4 Method calls

The library provides an interface for making method calls and handling the input/output arguments and the result.

8.5 Database configuration

The OPC-UAC library bases its functionality on the information contained in the configuration file "OPCUAC.XML".
The configuration activity is performed through the configuration program "OPCUACCFG.EXE" and must be carried out mainly in an on-line condition with the OPC UA servers you intend to use

The configuration consists of the following elements:

Server:	it represents the connection to an OPC UA server
Subscription:	it's an element containing items of type (node class) "Variable", associated with the server. The server can contain multiple subscriptions.
Methods:	it's an element containing items of type (node class) "Method", associated with the server.
Item:	identifies a resource of type (node class) "Variable" present in an OPC UA server whose value can be read and written. Items are contained in a subscription.
Method:	identifies a resource of type (node class) "Method" present in an OPC UA server representing a function which can be executed on an object. The methods are contained in the default group "Methods".

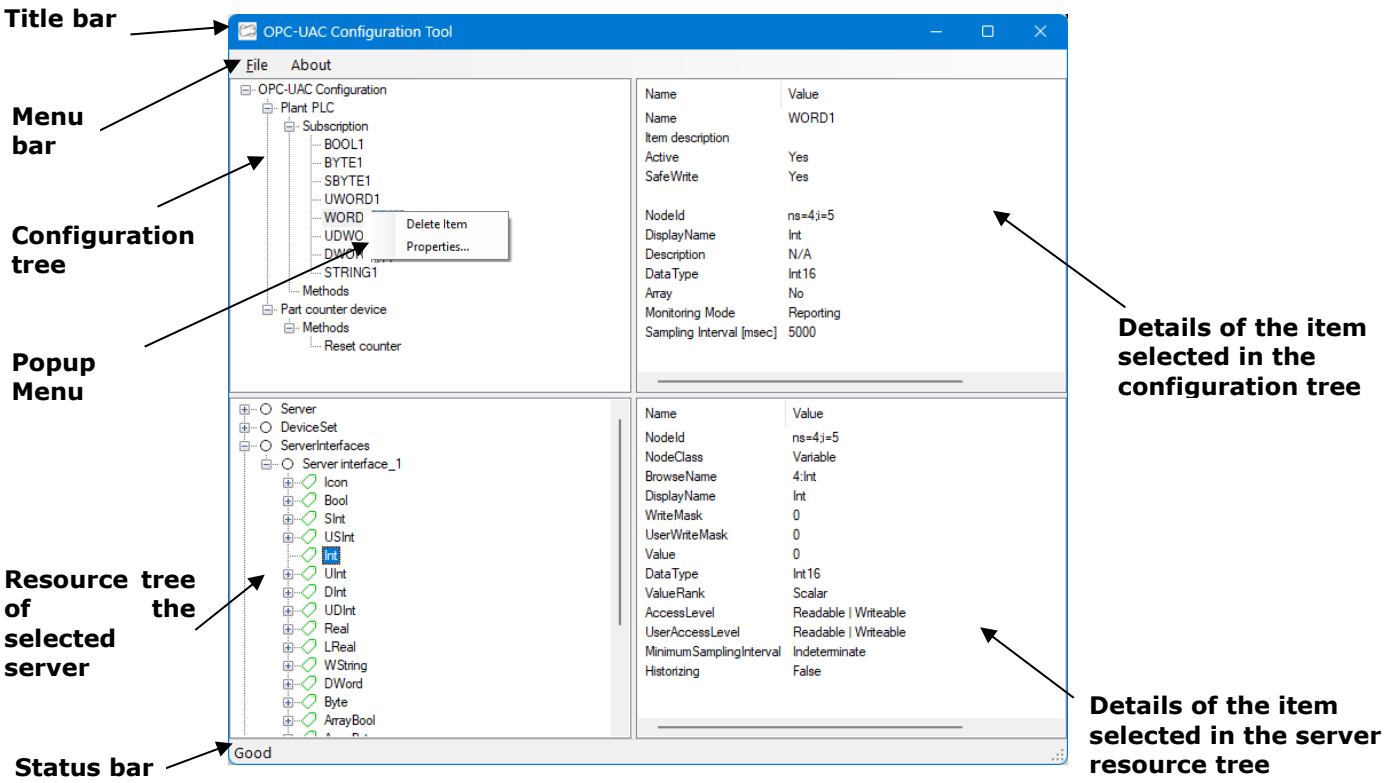
8.5.1 Configuration tool

To run the configuration tool "OPCUACCFG.EXE" refer to the instructions in the "Deployment" paragraph on page 4.

During the initialization phase the configurator loads the current configuration contained in the "OPCUAC.XML" file; in case of missing file or reading error the configurator creates an empty configuration.

When the configurator starts, the main window is displayed and it is composed of:

- Title bar
- Menu bar
- Status bar
- Configuration tree area: this area displays the tree of servers/subscriptions/items/methods that make up the configuration
- Configuration element details area: in this area the detail information related to the selected configuration element is displayed
- Server resource tree area
- Server resource details area
- Popup menu



Throughout the execution time the configurator program tries to establish and maintain a connection with all the servers present in the configuration in order to allow browsing the items available in the server database and adding them to the driver database.

Selecting any element of a server in the configuration tree displays the connection status information in the status bar.
If the connection is active ("Good" status), in the lower panels the tree of the items available in the server are displayed on the left and on the right a detail of some attributes of the selected item.

If the connection is not active ("Server not connected" status), the boxes remain blank; please refer to "Troubleshooting" on page. 53.

All the available commands are listed below with the relative access modes.
In the following paragraphs the functions of the single commands are described.

Command	Activation mode	Key
Save	Menu bar: "File" ▶ "Save"	CTRL+S
Exit	Menu bar: "File" ▶ "Exit"	ALT+F4
Information	Menu bar: "About" ▶ "Information"	CTRL+X

Command	Activation mode	Key
Server creation	Root element popup menu ("OPC UAC Configuration"): "New server..."	INS (*)
Subscription creation	Server element popup menu: "New subscription..."	INS (*)
Item creation	Drag&Drop operation of the single element of type (class node) "Variable" from the resource tree to the configuration tree (inside a suscription associated to the same server).	INS (*)
Method creation	Drag&Drop operation of the single element of type (class node) "Method" from the resource tree to the configuration tree (inside the "Method" group associated to the same server).	INS (*)
Delete server / subscription /item/method	Selected element popup menu: "Delete..."	DEL (*)
Server / subscription / item /method properties	Selected element popup menu: "Properties..."	CTRL+P (*)

Notes
(*) Contextual commands with reference to the selected element.

Starting from an empty configuration, the minimum necessary steps are:

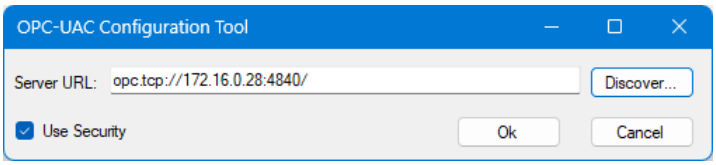
1. Create a server
2. Create a subscription in the server
3. Create items in the subscription

8.5.2 Server configuration

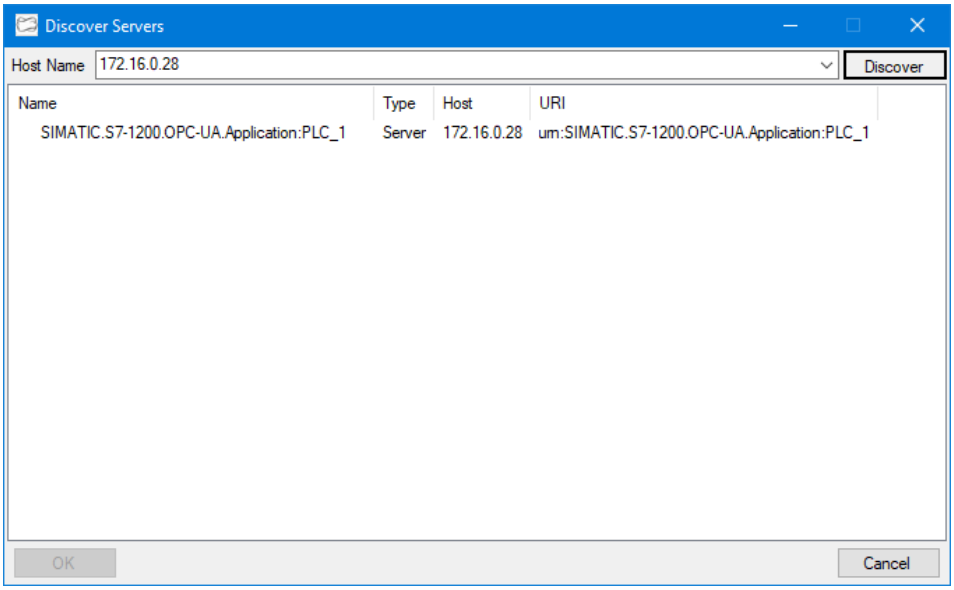
From the configuration tree it is possible to access the commands to create a new server or modify or delete an existing server.

8.5.2.1 Create a server

The following window is displayed for creating a new server:



You can directly enter the URL string of the server you want to contact in the "Server URL" field, or you can access the "Discover…" option that allows you to search for available OPC UA servers on a given host



In the "Discover Servers" window enter the name or IP address of the host hosting the server you wish to contact and select the "Discover" command.
If the search is successful in the list you will see the available servers; selecting the server and continuing with "OK" you return to the main window where the string URL of the selected server is automatically inserted.

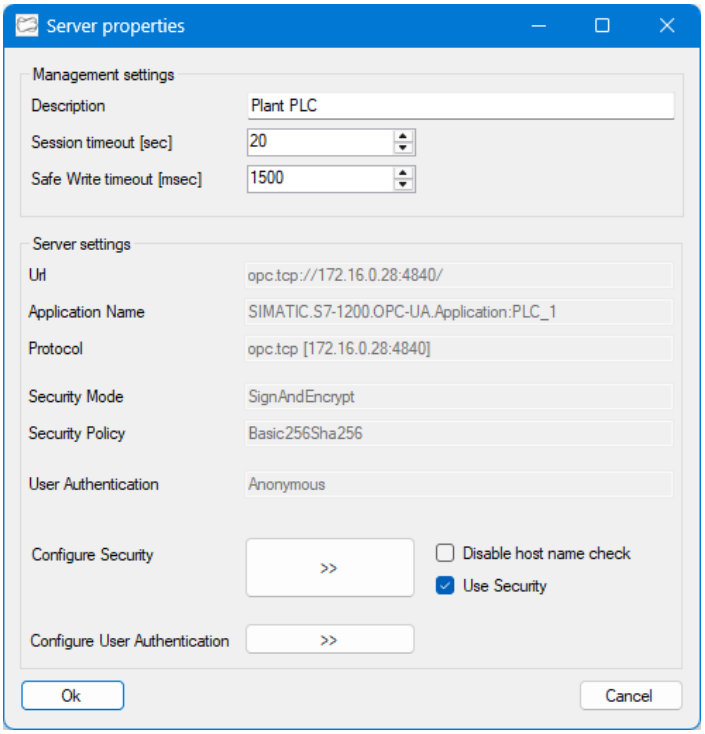
In the main window the 'User Security' flag allows you to indicate your preference on whether or not to use security: by confirming with "OK" the syntactic correctness of the inserted URL is verified and in the case of a positive outcome an attempt to connect to the server is triggered.
The operation may take a few seconds; the mouse cursor takes the form of a 'wait' and an informative message is displayed in the status bar.

If the connection attempt fails, an error message is displayed and the server creation operation is cancelled, otherwise the procedure continues with the display of the properties window in which it is possible to enter other parameters and modify the settings assigned by default during the initial connection.
For a description of the properties window refer to the following paragraph.

If, at the end of the configuration, a server with the same characteristics is already present, an error will be reported and the server will not be added to the configuration.

8.5.2.2 Modify a server

After the command to create or edit a server you will see the properties window:



Properties are divided into two groups:

- "Management settings": parameters that regulate some aspects of client-side management
- "Server settings": parameters for configuring the connection with the server (protocol, security settings of the transport, application and user levels)

"**Management settings**" parameters.

Description
Free description to be associated with the server.
Session timeout
Maximum waiting timeout for creating a communication session with the server. The creation of a session requires the execution of various operations and the exchange of numerous information between client and server; typically the time necessary is in the order of some seconds.
Safe Write timeout
Maximum waiting timeout for re-reading the updated data following a write operation.

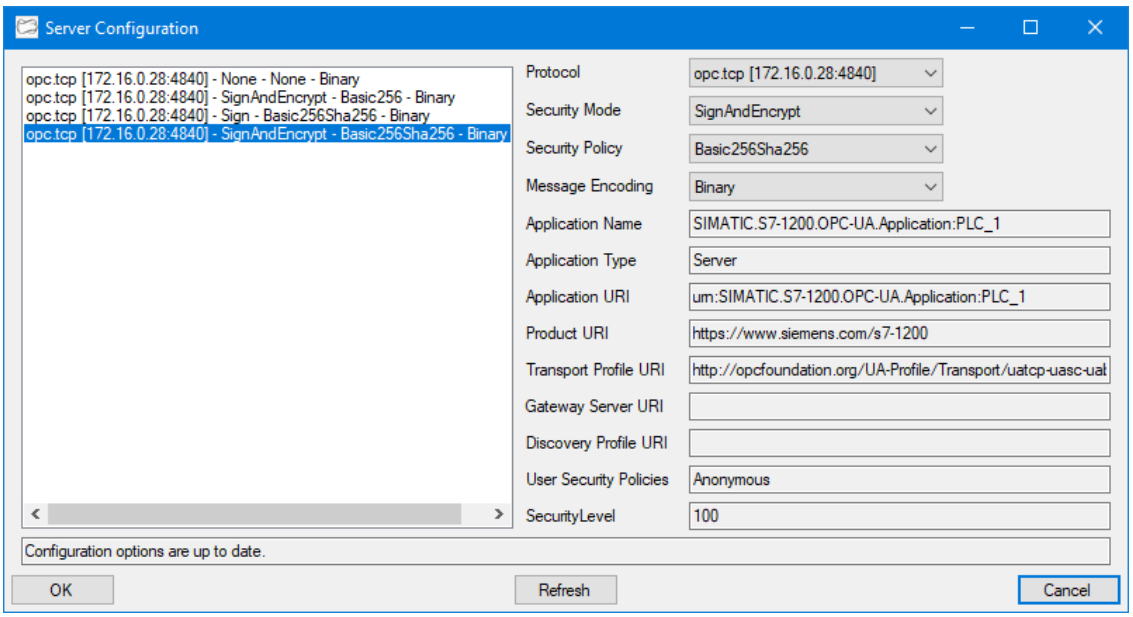
"**Server settings**" parameters.

Use Security

The "Use Security" flag specifies a preference as to whether or not to use security; based on this preference and its own capabilities, the server selects and proposes a compatible connection Endpoint.
In the case of servers that do not support security, disabling the flag should make it easier to select the Endpoint without security with the "Security mode" and "Security Policy" set to "None".

Configure Security

The option shows the "Server Configuration" window:



In the list on the left are displayed the connection modes (the so-called "Endpoints") available on the server **(*)**; in the area on the right are displayed the detailed information.
The mode can be selected directly in the list on the left or by setting the parameters individually in the area on the right:

- Protocol
- Security Mode
- Security Policy
- Message Encoding

Confirming with "OK" will return you to the main properties window where the updated information will be displayed according to the selected mode.

Notes

(*)If this window is accessed in off-line conditions, i.e. when there is no active connection w the server, it will not be possible to make modifications.
The information displayed will be exclusively those currently stored in the configuration (the will be only one line in the list on the left). At the bottom of the window a warning message w be displayed.

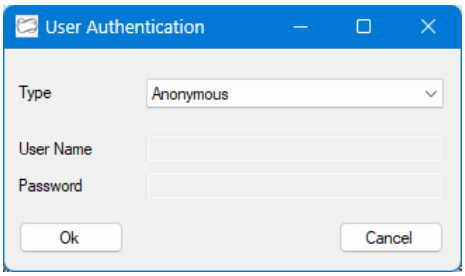
Disable host name check

This setting allows you to disable the hostname check in the server certificate.

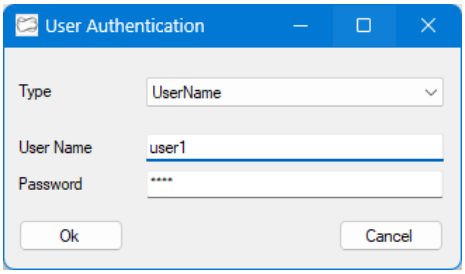
The setting may be necessary if the status bar displays the message "Server not connected – BadCertificateHostNameInvalid"; for further information, please refer to paragraph "Troubleshooting" on page 53.

Configure User Authentication

The option shows the "User Authentication" window:



The "Anonymous" and "Username" levels are supported; the list allows you to choose between the levels supported by the server.
In the case of the "Anonymous" level, no further information is required, whereas with the "Username" level, "User name" and "Password" must be entered.



8.5.2.3 Delete a server

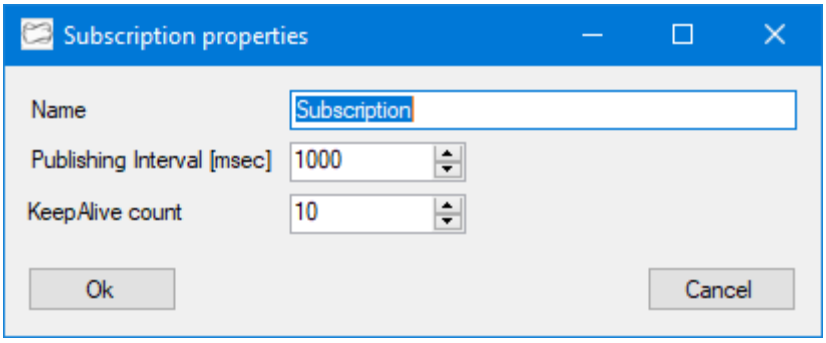
Before executing the delete operation of a server you are asked for confirmation.

8.5.3 Subscription configuration

From the configuration tree you can access the commands to create a new subscription or edit or delete an existing subscription.

8.5.3.1 Create or modify a subscription

After the creation or modification command of a subscription, the window of the related properties will be displayed:



Name
Subscription name.
Publishing Interval
Time required to update items (how often the server sends notifications). **Limits** ▪ the value is expressed in milliseconds and the range is between 0 and 1000000000
KeepAlive Count
Number of consecutive 'empty' publishing cycles (that have not generated notifications) that trigger the sending of a KeepAlive message to the client. **Limits** ▪ the range of the value is between 0 and 1000

The value assigned to the parameters is to be considered as 'indicative/desired'; the server can apply adjustments. If set to 0, the server will adopt the minimum value according to its capabilities.

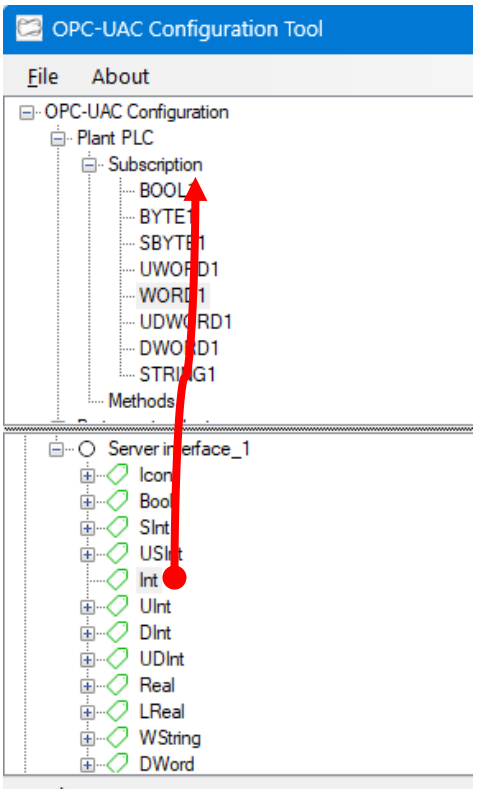
8.5.3.2 Delete a subscription

Before executing the delete operation of a subscription you are asked for confirmation.

8.5.4 Item configuration

8.5.4.1 Create an item

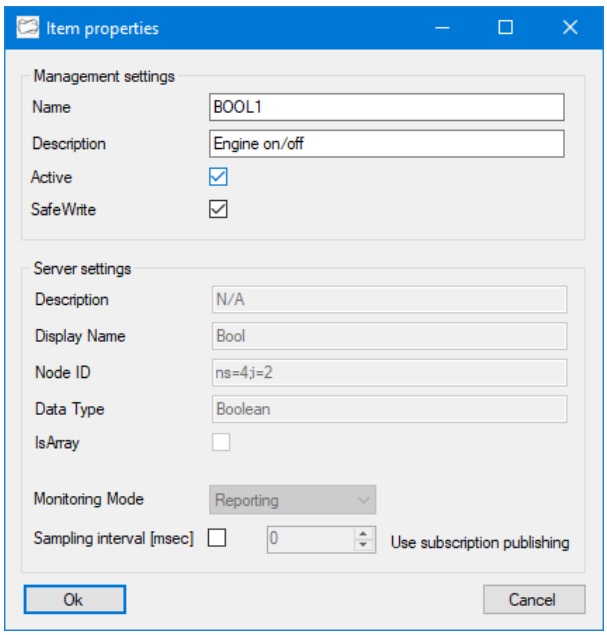
The addition of an item to the configuration is done through a drag&drop operation by dragging the resource of type "Variable" selected in the resource tree of the server into the destination subscription (the drop point can be the subscription node or one of the items already contained). Elements of type "Variable" are identified with this green icon .



The drop is allowed only within a subscription assigned to the source server; if the operation is successful the properties window is immediately displayed and it is possible to complete the item settings.

8.5.4.2 Modify an item

After the creation or modification command of an item, the properties window appears:



Properties are divided into two groups:

- "Management settings": parameters that regulate some aspects of item management
- "Server settings": parameters that report some of the item properties as declared in the server and that regulate its operation. Of particular importance is the "Node ID" property which uniquely identifies the item in the server's database.

"**Management settings**" parameters.

Name
Name to be used to identify the item: must be unique in the whole configuration.
Description
Free description to be associated with the item.
Active
Item enabling.

Notes The inactive items are kept in the configuration but are not managed in the runtime; in case they are used in the variables a specific error code will be generated. For details about error codes refer to the "OPC UA Communication errors" paragraph on page 51

Safe Write

Activates "safe" writing mode.

Generally, the write operation is carried out in blocking mode for the time necessary to guarantee the delivery of the value to the peripheral or to determine its eventual impossibility. In the case of a system structured on more levels the server could however deliver the value to an intermediate system that completes the operation in asynchronous mode (in this case the server informs the client through a specific status); it can therefore happen that a possible close reading can return a value not yet updated.
Activating the "Safe Write" mode, in the situation just described, after sending the write request to the server the driver performs some readings until the read value matches the one just set or until the timeout value is reached (parameter "Safe Write timeout" set in the server element). In case of termination due to timeout an error will be generated.
For details of the error codes refer to "OPC UA Communication errors" paragraph on page 51

Notes This mode can only be used if the data is written exclusively by a single client application.

"**Server settings**" parameters

Sampling interval

Indicates the sampling time of the item in the server (acquisition from the device).

If the check is disabled (standard default condition) the parameter is assigned equal to the value of the "Publishing Interval" of the subscription

Enabling check instead allows you to set a custom time.

The value assigned to the parameters is to be considered as 'indicative/desired'; the server can apply adjustments. If set to 0, the server will adopt the minimum value according to its capabilities.

Limits ▪ the value is expressed in milliseconds and the range is between 0 and 1000000000

Upon confirmation with "OK" a check is made on the "Name" field and in case of errors the properties window remains active; the following errors can be reported:

Error	Description
The item name cannot be empty	Name field cannot be empty
The name xxx is already used	The mentioned item is already present in the configuration

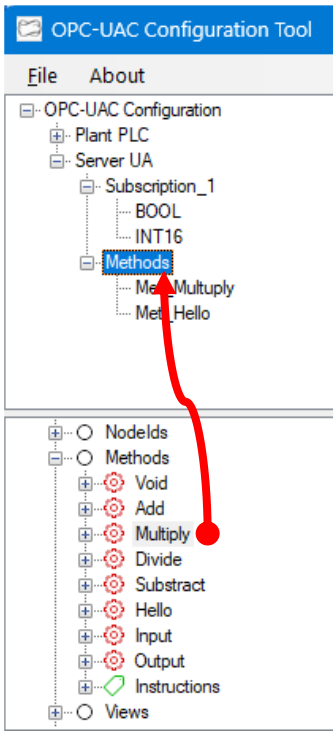
8.5.4.3 Delete an item

Before executing the delete operation of an item you are asked for confirmation.

8.5.5 Method configuration

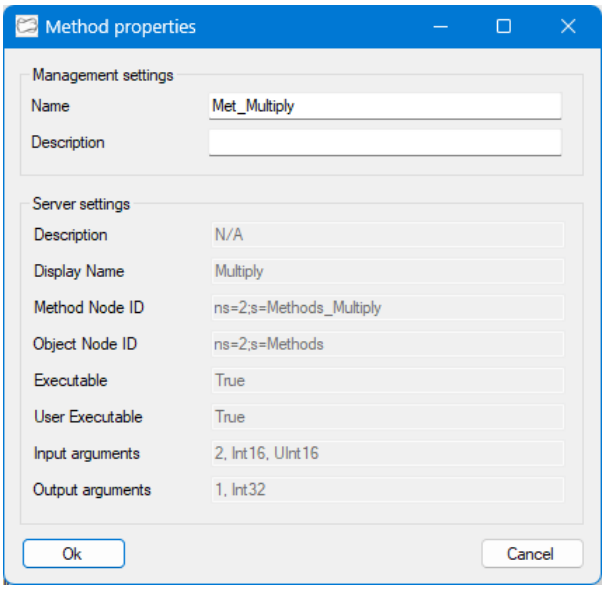
8.5.5.1 Create a method

The addition of a method to the configuration is done through a drag&drop operation by dragging the resource of type "Method" selected in the resource tree of the server into the predefined "Methods" group (the drop point can be the "Methods" node or one of the methods already contained).
Elements of type "Method" are identified with this red icon .



8.5.5.2 Modify a method

After the creation or modification command of a method, the properties window appears:



Name
Name to be used to identify the method: must be unique in the whole configuration.
Description
Free description to be associated with the method.

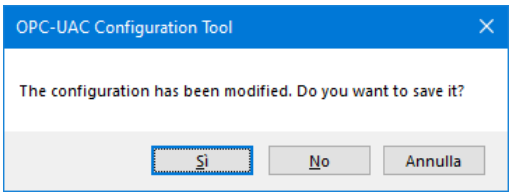
Note
A method always refers to an object; the container node of the method itself (referred to as "Object Node ID") is considered as the reference object during creation.

8.5.5.3 Delete a method

Before executing the delete operation of a method you are asked for confirmation.

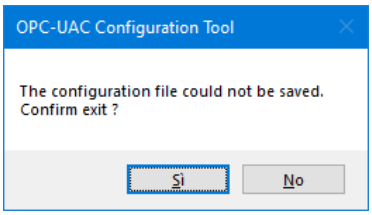
8.5.6 Configuration Saving

This command stores the configuration on disk in the "OPCUAC.XML" file.
When exiting the configurator, if the configuration has been modified but not saved, the following request will be made:



- **"Sì":** saves the configuration
- **"No" :** does not save the configuration, losing all changes
- **"Annulla" :** cancels the exit command

If saving the file fails, e.g. due to a problem accessing the file, a confirmation prompt is displayed to continue with closing the application.



8.6 Description of functions by category

Functions can be grouped into the following categories:

- initialization and termination
- Data Access (reading, writing and notification of values)
- method call
- information

8.6.1 Initialization and termination

OpcuacInit	Initializes the library by loading the application configuration file, checking certificates and loading the item database.
OpcuacSetItemNotifier	Set the callback function for notification of changes in item values.
OpcuacStart	Starts server connection and item update.
OpcuacStop	Closes the connection to the servers, stops items update and discharges the configuration.

Functions must be called in the following order:

1. OpcuacSetItemNotifier (opzionale)
2. OpcuacInit
3. OpcuacStart
4. …
5. OpcuacStop

Example

C/C++

```
#include "OPCUAC.H"

static void Notify(OPCUACItemNotification *pNotif) ;

void InitOPCUAC()
{
	OpcuacSetItemNotifier(Notify) ;
	unsigned int err = OpcuacInit() ;

	if (err == ERR_None)
		OpcuacStart() ;

	InitMessage(err) ;
}

void Notify(OPCUACItemNotification *pNotif)
{
	//Manage Notification
}

void TerminateOPCUAC()
{
	OpcuacStop() ;
}
```

C#

```
private void InitOPCUAC()
{
	OPCUACItemNotifyDelegate del = Notify;
	OpcuacInterface.OpcuacSetItemNotifier(del);

	uint err = OpcuacInterface.OpcuacInit();

	if (err == OpcuacInterface.Opcuac_ERR_None)
		OpcuacInterface.OpcuacStart();

	InitMessage(err) ;

}

private void Notify(OPCUACItemNotification notif)
{
	//Manage Notification
}

private void TerminateOPCUAC()
{
	OpcuacInterface.OpcuacStop();
}
```

8.6.2 Data Access management

OpcuacGetValue	Gets the last updated value of an item from the internal library cache.
OpcuacReadValue	Reads the value of an item from the server.
OpcuacWriteValue	Writes the value of an item.
OpcuacFreeReadValue (*)	Releases the item reading management memory.
OpcuacFreeNotification (*)	Releases the item notification management memory.

(*) Only for C/C++ interface

Example

C/C++

```
#include "OPCUAC.H"

void unsigned int ReadWriteItem(wchar_t *name)
{
	OPCUACItemReadInfo tmpr ;

	int numval = 0 ; //all
	int index = 0 ;
	unsigned int err ;

	::ZeroMemory(&tmpr, sizeof(tmpr)) ;
	if ((err = OpcuacGetValue(name, numval, index, &tmpr)) == ERR_None)
		{
		OPCUACItemWriteInfo tmpw ;

		::ZeroMemory(&tmpw, sizeof(tmpw)) ;
		tmpw.Value = tmpr.Value ;
		tmpw.ValueSize = tmpr.ValueSize ;

		err = OpcuacWriteValue(name, tmpr.ValueNum, 0, &tmpw) ;
		}
	OpcuacFreeReadValue(&tmpr) ;

}
…

void Notify(OPCUACItemNotification *pNotif)
{
	if (pNotif != NULL)
	{
		UpdateItem(pNotif->ItemName, &pNotif->Value) ;
		OpcuacFreeItemNotification(pNotif) ;
	}
}
```

C#

```
private unsigned int ReadWriteItem(string name)
{
	OPCUACItemReadInfo tmpr;
	uint err;

	OPCUACItemReadInfo tmpr = new OPCUACItemReadInfo();
	uint err;

	string name = m_ItemName.Text;
	name.TrimEnd();

	if ((err = OpcuacInterface.OpcuacGetValue(name, 0, 0, tmpr)) ==
		OpcuacInterface.Opcuac_ERR_None)
	{
		OPCUACItemWriteInfo tmpw = new OPCUACItemWriteInfo();

		tmpw.Value = tmpr.Value;
		err = OpcuacInterface.OpcuacWriteValue(name, tmpr.ValueNum, 0, tmpw);
	}

	return (err) ;
}

private void OnNotify(OPCUACItemNotification notif)
{
	UpdateItem(notif.ItemName, notif.Value) ;
}
```

8.6.3 Method execution management

OpcuacCallMethod (*)	Executes a method call

(*) Available only via C# interface.

Example

C#

```
private unsigned int CallMethodMultiply()
{
	OPCUACMethodInfo met = new OPCUACMethodInfo();
	object[] input = new object[2];

	System.Int16 p1 = 10;
	System.UInt16 p2 = 5;
	input[0] = p1;
	input[1] = p2;
	met.InputPar = input;
	uint err = OpcuacInterface.OpcuacCallMethod("Multiply", met);

	if (err == 0 && met.OutputPar != null)
	{
		string msgresult = String.Format("{0} x {1} = {2}", met.InputPar[0],
		met.InputPar[1],
		met.OutputPar[0]);
		MessageBox.Show(msgresult, "Method Multiply result");
	}
	else
	{
		string msgresult = String.Format("Method result: {0:X} [{1}]", err,
		OpcuacInterface.OpcuacErrorDescription(err));
		MessageBox.Show(msgresult, "Method Multiply result");
	}
	return (err) ;
}
```

8.6.4 Information

OpcuacVersion	Provides the library version
OpcuacProtection	Provides the library's operation mode (demo / full)
OpcuacErrorDescription	Provides the descriptive string associated with the error code/status (in English)
OpcuacValueStatusInfo	Provides the usability status of the data (Good/Uncertain/Bad)

Example

C/C++

```
#include "OPCUAC.H"

void InitMessage(unsigned int st)
{
	CString msg, msgst, msgver, msglic, strst;

	unsigned int Ver = OpcuacVersion();
	msgver.Format(L"Version: %d.%02d", Version / 100, Version % 100);

	unsigned int Prot = OpcuacProtection();
	msglic = Prot != 0 ? L"Licence: Ok" : L"Licence Demo";

	strst = OpcuacErrorDescription(st);
	msgst.Format(L"OPCUAC Initialization: [%x = %s]", st, strst);

	msg.Format(L"%s\r\n%s\r\n%s", msgver, msglic, msgst);

	MessageBox(msg, L"OPCUAC demo", MB_OK);
}

void DisplayValueStatus(OPCUACItemReadInfo *info)
{
	unsigned int stcat = OpcuacValueStatusInfo(info->ValueStatus) ;
	wchar_t *str = L"Unknown";

	if (stcat ==_DS_Good)
		str = L"Good" ;
	else if (stcat == DS_Uncertain)
		str = L"Uncertain" ;
	else if (stcat == DS_Bad)
		str = L"Bad" ;

	m_Quality = str ;
}
```

C#

```
private void InitMessage(uint st)
{
	String msg, msgst, msgver, msglic, strst;

	UInt32 Version = OpcuacInterface.OpcuacVersion();
	msgver = String.Format("Version: {0}.{1}", Version / 100, Version % 100);

	UInt32 Prot = OpcuacInterface.OpcuacProtection();
	msglic = Prot != 0 ? "Licence: Ok" : "Licence Demo";

	strst = OpcuacInterface.OpcuacErrorDescription(st);
	msgst = String.Format("OPCUAC Initialization: [{0:X} = {1}]",st, strst);

	msg = String.Format("{0}\r\n{1}\r\n{2}", msgver, msglic, msgst);

	MessageBox.Show(msg, this.Text);

}

private void DisplayDataStatus(OPCUACItemReadInfo info)
{
	uint stcat = OpcuacInterface.OpcuacValueStatusInfo(info.ValueStatus) ;
	string str = "Unknown";

	if (stcat == OpcuacInterface.Opcuac_DS_Good)
		str = "Good" ;
	else if (stcat == OpcuacInterface.Opcuac_DS_Uncertain)
		str = "Uncertain" ;
	else if (stcat == OpcuacInterface.Opcuac_DS_Bad)
		str = "Bad" ;

	m_Quality.Text = str ;
}
```

9 Functions in alphabetical order

This chapter describes one by one, in alphabetical order, all available functions.
The tab contains the following information:

Name	Function name
Versioni obsolete	Eventual obsolete names of the same function
Complete interface	• Type of the function return value • Function name • Parameters in order and their type
Parameters	Summary meaning of each parameter
Description	Function purpose
Notes	Notes on some particular behaviors and on some attentions to be observed in the use of the function
Return value	Meaning of the function return value
Identification table	See below

The function tab is completed by an identification table that contains the following information:

State	• Ok: function is active and usable • Obsolete: the function is not to be used
Product	Product in which the function is available
Category	Function category
References	List of functions in the same category.
Examples	Examples of functions category usage.
In OPC_UAC	Version of OPC-UAC in which the function was inserted. If the function is obsolete, the version of OPC-UAC in which it has been declared obsolete is mentioned.

9.1 OpcuacCallMethod

C#

unsigned int OpcuacInterface.OpcuacCallMethod(string item, OPCUACMethodInfo pdBuf)

The *OpcuacCallMethod* function executes the method associated to the item.

item	Alphanumeric identifier of the item associated to the method to be executed (method "Name" attribute).
pdbuf	Structure containing the input and output arguments and the detailed code for the result of the call operation.
Notes	Input and output parameters are handled as arrays of System::Object; for details on how to handle a System::Object object with respect to the OPC UA data type, please refer to the remarks on the *OpcuacGetValue* function (notes referring to C#). If the operation result is the specific error of failed execution, the **ExecutionStatus** detail code contained in the **pdbuf** structure can be consulted; the possible values are listed in the paragraph "Method execution status" on page 49.
Return value	The result of the operation; 0 (zero) if the operation was successful otherwise an error code. For a list of possible error codes, please refer to "OPC UA Communication errors" paragraph on page 51.

State	Ok
Product	OPC-UAC
Category	Method execution functions on page **Errore. Il segnalibro non è definito.**
References	On page 25
Examples	On page 25
In OPC-UAC	From version 2.0

9.2 OpcuacErrorDescription

C/C++

wchar_t* OpcuacErrorDescription(unsigned int err)

C#

string OpcuacInterface.OpcuacErrorDescription(uint err)

The *OpcuacErrorDescription* function returns the descriptive string associated with the specified error code.

err	Error code returned by the item read and write functions
Notes	C/C++: the function uses a static area; each call overwrites the result of the previous one.
Return value	The error descriptive string.

State	Ok
Product	OPC-UAC
Category	Information functions page 25
References	On page 25
Examples	On page 25
In OPC-UAC	From version 1.0

9.3 OpcuacFreeItemNotification

C/C++

void OpcuacFreeItemNotification(OPCUACItemNotificationInfo* notif)

The *OpcuacFreeItemNotification* function frees the memory used by the structure of type OPCUACItemNotificationInfo for the management of the notification of value change.

notif	Structure received via the value change notification callback function.

Notes	The function exists only in the C/C++ interface.
Return value	None.

State	Ok
Product	OPC-UAC
Category	Value management functions page 23
References	On page 23
Examples	On page 23
In OPC-UAC	From version 1.0

9.4 OpcuacFreeReadValue

C/C++

void OpcuacFreeReadValue(OPCUACItemReadInfo* read)

The *OpcuacFreeReadValue* function frees the memory relative to the item value contained in the structure of type OPCUACItemReadInfo.

read	Structure containing the item value returned by the read function.
Notes	The function exists only in the C/C++ interface.
Return value	None.

State	Ok
Product	OPC-UAC
Category	Value management functions page 23
References	On page 23
Examples	On page 23
In OPC-UAC	From version 1.0

9.5 OpcuacGetValue

C/C++
unsigned int OpcuacGetValue(wchar_t* item, int numval, int index, OPCUACItemReadInfo* pdBuf)

C#
uint OpcuacInterface.OpcuacGetValue(string item, int numval, int index, OPCUACItemReadInfo pdBuf)

The *OpcuacGetValue* function returns the last available value of the item stored in the library's internal cache.
If the activation of the subscription service was not successful the value is read from the server.

item	Alphanumeric identifier of the item to be read (item "Name" attribute)
numval	Number of values to return: <=0 returns the entire value (single or array) >0 returns the required number of values
index	Index of the first value to return, valid values are: 0 for single values (“Scalar”) 0/n-1 for array values, where n is the number of elements in the array
pdbuf	Structure in which to load the value and associated information.

Notes

The managed and unmanaged versions of the OPCUACItemReadInfo structure differ only in the way the value is handled.

ValueRank specifies whether the value is single or an array.
The values that it can assume are those indicated in the constants RV_xxx.

ValueType specifies the DataType of the value (Boolean, Byte, etc.).
The values that it can assume are those indicated in the constants DT_xxx.

ValueNum 1 if the value is single or the total number of elements in the array.

ValueStatus the status associated with the value.
To check the validity of the value you can use the function *OpcuacValueStatusInfo.*

ServerTS the server timestamp associated with the value.

SourceTS the source timestamp associated with the value.

Value the value.

C#: in the managed structure, **Value** consists of an object of type System::Object (it contains any type of data). The object type depends on the characteristics **ValueRank** and **ValueType** of the item.
Es:
Single value, Boolean: bool
Array value, Boolean: bool[]

Single value, Byte:	byte
Array value, Byte:	byte[]
Single value, SByte:	sbyte
Array value, SByte:	sbyte[]
Single value, UInt16:	ushort
Array value, UInt16:	ushort[]
Single value, Int16:	short
Array value, Int16:	short[]
Single value, UInt32:	uint
Array value, UInt32:	uint[]
Single value, Int32:	int
Array value, Int32:	int[]
Single value, Float:	float
Array value, Float:	float[]
Single value, Double:	double
Array value, Double:	double[]
Single value, Bytestring:	byte[]
Array value, Bytestring:	byte[][]
Single value, String:	string
Array value, String:	string[]
Single value, LocalizedText:	string
Array value, LocalizedText:	string[]

C/C++: in the unmanaged structure, **Value** is a generic void* pointer that must be interpreted on the basis of the other information, as described below.

The number of values contained in **Value** is **ValueNum** if the request was made for the whole item (numval <= 0) otherwise it is the requested number **numval**.

The type of the value is given by the **ValueType** field.

Types Bytestring (sequence of bytes), String or LocalizedText: **Value** contains structures of type OPCUACStringData; one for each returned value.
The OPCUACStringData structure contains the **StrValue** field and the **StrValueSize** size information expressed in bytes.
If the data is of type String or LocalizedText, **StrValue** should be interpreted as an array of wchar_t with size **StrValueSize/2.**
If the data is empty **StrValue** is NULL and **StrValueSize** is 0.
Warning: the value of String or LocalizedText type does not include the terminator character.

All other types: **Value** is a byte buffer containing all values; the **ValueSize** field indicates the total size.

For single values the size is:

Boolean, Byte, Sbyte	**ValueSize** = 1
Int16,UInt16	**ValueSize** = 2
Int32, UInt32, Single (float)	**ValueSize** = 4
Double	**ValueSize** = 8
ByteString,String,LocalizedText	**ValueSize**= sizeof(OPCUACStringData)

For values of array type the size is:
ValueSize = *single size * number of values*
The only exception is for the array type of Boolean in which the bits are compacted and therefore the total size corresponds to:
ValueSize = *(number of values + 7) / 8*

Warning: at the end of the use of the unmanaged structure it is necessary to pass it to the function *OpcuacFreeReadValue* to free the memory allocated by the library for the **Value** field.

Return value

The result of the reading; 0 (zero) if the operation was successful otherwise an error code.
For a list of possible error codes, please refer to "OPC UA Communication errors" paragraph on page 51.

State	Ok
Product	OPC-UAC
Category	Value management functions page 23
References	On page 23
Examples	On page 23
In OPC-UAC	From version 1.0

9.6 OpcuacInit

C/C++
unsigned int OpcuacInit()

C#
uint OpcuacInterface.OpcuacInit()

The *OpcuacInit* function performs the initialization of the library.
Initialization consists of these operations:

- loading the application configuration file "OPCUAC.Config.xml"
- uploading (eventual generation or renewal) of the "Application Certificate"
- loading the database from the "OPCUAC.XML" file

Notes None.

Return value The result of the initialization procedure; 0 if the operation was successful otherwise an error code.
For a list of possible error codes, please refer to "General errors" paragraph on page 50.

State	Ok
Product	OPC-UAC
Category	Initialization functions page 21
References	On page 21
Examples	On page 21
In OPC-UAC	From version 1.0

9.7 OpcuacProtection

C/C++
unsigned int OpcuacProtection()

C#
uint OpcuacInterface.Protection()

The *OpcuacProtection* function returns the operation mode of the library with respect to the presence of the protection system (hardware key or software activation code).

Notes In case of absence of protection the library works in demo mode for a time of 15 minutes. After the demo time the read and write functions return the error code FFFFFF92H and the notification mechanism is interrupted.

Return value Value other than 0 (zero): protection has been detected and the library works in normal mode.
Value 0 (zero): protection has NOT been detected and the library is running in demo mode.

State	Ok
Product	OPC-UAC
Category	Information functions page 25
References	On page 25
Examples	On page 25
In OPC-UAC	From version 1.0

9.8 OpcuacReadValue

C/C++

unsigned int OpcuacReadValue(wchar_t* item, int numval, int index, OPCUACItemReadInfo* pdBuf)

C#

uint OpcuacInterface.OpcuacReadValue(string item, int numval, int index, OPCUACItemReadInfo pdBuf)

The *OpcuacReadValue* function reads the item value from the server.

item	Alphanumeric identifier of the item to be read (item "Name" attribute)
numval	Number of values to return: <=0 returns the entire value (single or array) >0 returns the required number of values
index	Index of the first value to return, valid values are: 0 for single values ("Scalar") 0/n-1 for array values, where n is the number of elements in the array
pdbuf	Structure in which to load the value and associated information.
Notes	Please refer to the notes of the *OpcuacGetValue* function.
Return value	The result of the reading; 0 (zero) if the operation was successful otherwise an error code. For a list of possible error codes, please refer to "OPC UA Communication errors" paragraph on page 51.

State	Ok
Product	OPC-UAC
Category	Value management functions page 23
References	On page 23
Examples	On page 23
In OPC-UAC	From version 1.0

9.9 OpcuacSetItemNotifier

C/C++

void OpcuacSetItemNotifier(void (*pfun)(OPCUACItemNotificationInfo* fun))

C#

void OpcuacInterface.OpcuacSetItemNotifier(OPCUACItemNotifyDelegate fun)

The *OpcuacSetItemNotifier* function sets the callback function that will be used to notify when the value of an item has been updated.

fun	Management function for notifications of changes in item values
Notes	The OPCUACItemNotificationInfo structure passed to the callback function contains the following fields: **ItemName** Identifier of the updated item. **Value** updated item value. The field consists of a structure of type OPCUACItemReadInfo described in the function *OpcuacGetValue.* The function must be set before calling the *OpcuacInit* initialization function. **Warning**: at the end of the use of the unmanaged structure it is necessary to pass the pointer to the function *OpcuacFreeNotification* to free the memory allocated by the library.
Return value	None.

State	Ok
Product	OPC-UAC
Category	Initialization functions page 21
References	On page 21
Examples	On page 21
In OPC-UAC	From version 1.0

9.10 OpcuacStart

C/C++
void OpcuacStart()

C#
void OpcuacInterface.OpcuacStart()

The *OpcuacStart* function starts the connections to the servers and the mechanism for updating and notifying changes in the item values.

Notes None.

Return value None.

State	Ok
Product	OPC-UAC
Category	Initialization functions page 21
References	On page 21
Examples	On page 21
In OPC-UAC	From version 1.0

9.11 OpcuacStop

C/C++
void OpcuacStop()

C#
void OpcuacInterface.OpcuacStop()

The *OpcuacStop* function interrupts server connections and the mechanism for updating and notifying changes in item values.

Notes None.

Return value None.

State	Ok
Product	OPC-UAC
Category	Initialization functions page 21
References	On page 21
Examples	On page 21
In OPC-UAC	From version 1.0

9.12 OpcuacValueStatusInfo

C/C++
unsigned int OpcuacValueStatusInfo(unsigned int status)

C#
uint OpcuacInterface.OpcuacValueStatusInfo(uint status)

The *OpcuacValueStatusInfo* function returns the usability indicator of the value, read or notified, of an item.

status	Status associated with the item value.
Notes	The possible values that **status** can take are listed in the paragraph "Value status" on page 48.
Return value	The usability indicator of the value. The indicator can take three values identified by the following constants: • DS_Good • DS_Uncertain • DS_Bad

State	Ok
Product	OPC-UAC
Category	Information functions page 25
References	On page 25
Examples	On page 25
In OPC-UAC	From version 1.0

9.13 OpcuacVersion

C/C++
unsigned int OpcuacVersion()

C#
uint OpcuacInterface.Version()

The *OpcuacVersion* function returns, in the form of a numerical value, the version of OPC-UAC.

Notes

The version is expressed as follows:
- major version number * 100
- plus minor version number

Example: version 120 = 1.20

Return value

OPC-UAC version.

State	Ok
Product	OPC-UAC
Category	Information functions page 25
References	On page 25
Examples	On page 25
In OPC-UAC	From version 1.0

9.14 OpcuacWriteValue

C/C++

unsigned int OpcuacWriteValue(wchar_t* item, int numval, int index, OPCUACItemWriteInfo* pdBuf)

C#

uint OpcuacInterface.OpcuacWriteValue(string item, int numval, int index, OPCUACItemWriteInfo pdBuf)

The *OpcuacWriteValue* function writes the item's value.

item	Alphanumeric identifier of the item to be read (item "Name" attribute)
numval	Number of values to be written: 1 for single values ("Scalar") 1/n for array values: number of elements to be modified, where n is the current number of elements in the array -1 for array values: triggers writing the entire array (i.e. dynamic arrays)
index	Index of the first value to be written, valid values are: 0 for single values ("Scalar") 0/n-1 for array values, where n is the current number of elements in the array
pdbuf	Structure containing the value to be written.
Notes	Use of **numval** parameter in the case of array value: • to change one or more elements of the current array value assign numval a value >= 1 • to write the entire array assign **numval** a value -1; this mode can be used for dynamic arrays in order to replace the value completely (e.g., in the case of changing the number of elements). **Value** field of the OPCUACItemWriteInfo structure: contains the value to write. C#: In the managed structure, **Value** consists of an object of type System::Object. The object type must be consistent with the **ValueRank** and **ValueType** characteristics of the item. In the case of writing only one element of an array, a single value can also be specified. Example: Single value, Boolean: bool Array value, Boolean: bool[] Single value, Byte: byte Array value, Byte: byte[] Single value, SByte: sbyte Array value, SByte: sbyte[] Single value, UInt16: ushort Array value, UInt16: ushort[] Single value, Int16: short Array value, Int16: short[] Single value, UInt32: uint Array value, UInt32: uint[] Single value, Int32: int Array value, Int32: int[] Single value, Float: float

Array value, Float:	float[]
Single value, Double:	double
Array value, Double:	double[]
Single value, Bytestring:	byte[]
Array value, Bytestring:	byte[][]
Single value, String:	string
Array value, String:	string[]
Single LocalizedText, String:	string
Array LocalizedText, String:	string[]

C/C++: in the not managed structure, **Value** is a generic void* pointer to the memory area that must be predisposed, as the **ValueSize** field, following the indications described in the function *OpcuacGetValue.*

Return value

The result of the writing; 0 (zero) if the operation was successful otherwise an error code.
For a list of possible error codes, please refer to "OPC UA Communication errors" paragraph on page 51.

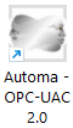
State	Ok
Product	OPC-UAC
Category	Value management functions page 23
References	On page 23
Examples	On page 23
In OPC-UAC	From version 1.0

10 Demo applications

Four demonstration projects with similar functionality are provided; the first three are developed in C++ and C# with Microsoft Visual Studio 2022 and 2010 while the fourth is implemented with Python:

OPCUACDemo_VC2022\OPCUACDemoCS	C# Windows Form application using the assembly type interface.
OPCUACDemo_VC2010\OPCUACDemoCpp	C++ MFC dialog-based application that uses the C/C++ interface.
OPCUACDemo_VC2010\OPCUACDemoCS	C# Windows Form application using the C/C++ interface via P/Invoke. In the demo there is the OpcuacInterface.cs module in which the OpcuacInterface container is available with the definition of constants, structures and methods to support the use of the DLL.
OPCUACDemo_Python312	Application developed with Python 3.12

Starting from the element on the Desktop:



The folder can be accessed by selecting the shortcut named “Demo applications”.

Otherwise it is possible to access the folder directly by inserting the following path in the Windows "File Explorer" window:

%APPDATA%\Automa\Communication Tools\OPC-UAC 2.0\Demo

The path corresponding to %APPDATA% depends on the Operating System you are using.

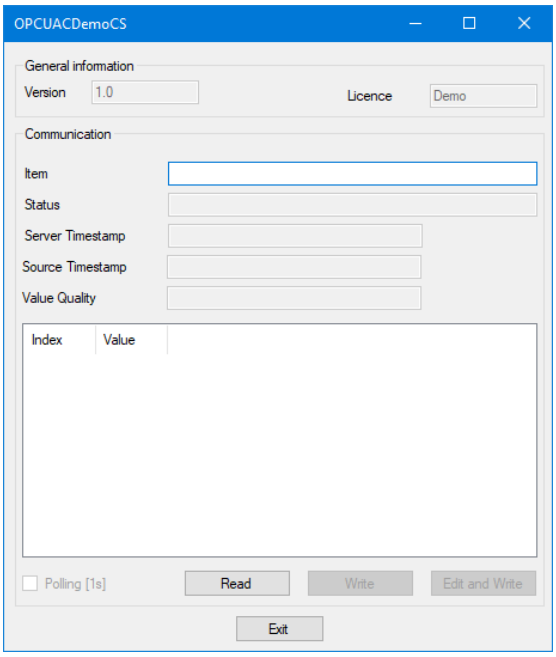
In addition to the demo folders, there is also the OPCUACFG service folder:

OPCUACCFG	Folder containing all the files needed to use the configurator program.

10.1 Microsoft Visual Studio demo

All projects are set up with the two configurations "Debug" and "Release".
In the respective target folders, also set in the projects as Working Directory, the executable file and other necessary files are already present.
For a very first test, even outside the development environment, please refer to the "Release" build of the "OPCUACDemo_VC2022\OPCUACDemoCS\bin\Release" demo.

By default the configuration file "OPCUAC.XML" is empty.
Use the configuration program, available in the OPCUACCFG folder and described in the "Database configuration" paragraph at page 8, to set up a minimal configuration (even with only one item); copy the file "OPCUAC.XML" into the demo working folder and start the application.



General information is displayed at the top:

- Library version
- Licence state: "Demo" if the activation has not been carried out or "Ok". In demo mode at each launch the application manages the communication for a maximum of 15 minutes.

Enter in the "Item" field the identifier of one of the items defined in the database (to confirm the value press the TAB key); if communication with the server has been established and the subscription has been correctly activated, the information relating to the current value of the item will be displayed in the fields below.

Available commands are:

- "Read" option: reads from the server the current value of the item and updates the data on the screen.
- "Write" option: retrieves the current item value from the cache and rewrites it entirely; the "Status" field displays the result of the operation.
- "Edit and Write" option: manages the input of one of the values (in the case of an array it is necessary to select the element to be modified) and writes the item; the "Status" field displays the result of the operation.
- "Polling" option: every second retrieves the current item value from the cache.

Notes
Modifying the OPCUACDemo_VC2022\OPCUACDemoCS application requires having .Net Framework 4.6.2 Developer pack installed on the PC.

10.2 Python demo

By default the configuration file "OPCUAC.XML" is empty.

Use the configuration program, available in the OPCUACCFG folder and described in the "Database configuration" paragraph at page 8, to set up a minimal configuration (even with only one item); copy the file "OPCUAC.XML" into the demo working folder and start the application.

General information is displayed at the top:

- Library version
- Licence state: "Demo" if the activation has not been carried out or "Ok". In demo mode at each launch the application manages the communication for a maximum of 15 minutes.

Enter in the "Item name" field the identifier of one of the items defined in the database and press the "Read" button: If communication with the server has been established and the subscription has been correctly activated, the information relating to the current value of the item will be displayed in the fields to the right.

To write the value, edit a value consistent with the Item type in the "Write value" field and press the "Write": the outcome of the operation will be displayed in the "Status" field.

Notes

A **32-bit** version of Python and the "pythonnet" library must be installed on the PC for this example to work.

11 Appendix

11.1 Value status

The possible values of the status associated with the data value are listed in the table below. Through the *OpcuacValueStatusInfo* function it is possible to obtain the usability indicator associated with a given code (Good/Uncertain/Bad).

Value Status	Codice Dec/Hex	Descrizione breve	Descrizione
BAD	2147483648 80000000H	Bad	
	2156462080 80890000H	BadConfigurationError	There is a problem with the configuration that affects the usefulness of the value.
	2156527616 808A0000H	BadNotConnected	The variable should receive its value from another variable; but has never been configured to do so.
	2150694912 80310000H	BadNoCommunication	Communication with the data source is defined; but not established; and there is no last known value available.
	2156724224 808D0000H	BadOutOfService	The source of the data is not operational.
	2156593152 808B0000H	BadDeviceFailure	There has been a failure in the device/data source that generates the value that has affected the value.
	2156658688 808C0000H	BadSensorFailure	There has been a failure in the sensor from which the value is derived by the device/data source.
	2150760448 80320000H	BadWaitingForInitialData	Waiting for the server to obtain values from the underlying data source.
	2156789760 808E0000H	BadDeadbandFilterInvalid	The deadband filter is not valid.
UNCERTAIN	1073741824 40000000H	Uncertain	
	1083113472 408F0000H	UncertainNoCommunicationLastUsableValue	Communication to the data source has failed. The variable value is the last value that had a good quality.
	1083179008 40900000H	UncertainLastUsableValue	Whatever was updating this value has stopped doing so.
	1083244544 40910000H	UncertainSubstituteValue	The value is an operational value that was manually overwritten.
	1083310080 40920000H	UncertainInitialValue	The value is an initial value for a variable that normally receives its value from another variable.
	1083375616 40930000H	UncertainSensorNotAccurate	The value is at one of the sensor limits.
	1083441152 40940000H	UncertainEngineeringUnitsExceeded	The value is outside of the range of values defined for this parameter.

Value Status	Codice Dec/Hex	Descrizione breve	Descrizione
	1083506688 40950000H	UncertainSubNormal	The value is derived from multiple sources and has less than the required number of Good sources.
GOOD	9830400 00960000H	GoodLocalOverride	The value has been overridden.

11.2 Method execution status

The main codes for the status are listed in the table below in the "Codice" column.
Any other negative codes not listed in the table should be considered as a "BAD" outcome.

Result	Codice Dec/Hex	Descrizione breve	Descrizione
BAD	-2136276992 80AB0000H	BadInvalidArgument	At least one input argument broke a constraint (e.g. wrong data type, value out of range)
	-2145452032 801F0000H	BadUserAccessDenied	User does not have permission to perform the requested operation.
	-2143617024 803B0000H	BadNotWritable	The access level does not allow writing to the Node.
	-2143289344 80400000H	BadNotImplemented	Requested operation is not implemented.
	-2139881472 80740000H	BadTypeMismatch	The value supplied for the attribute is not of the same type as the attribute's value.
	-2139750400 80760000H	BadArgumentsMissing	The client did not specify all of the input arguments for the method.
GOOD	000000000 00000000H	Good	

11.3 Error codes

Error codes are divided into the following categories:

- General errors (initialization)
- OPC UA Communication Errors

11.3.1 General errors

These are the errors that can be reported during driver initialization.

Dec Error	Hex Error	Description
-0001	FFFF	Internal memory management error. Possible causes/actions: • The library has not been correctly installed in the application: see "Deployment" on page. 4.
-0110	FF92	End of demonstration run. Possible causes/actions: • The library license is not active; refer to "Library License Activation" on page 5.
0001	0001	The library has not been initialized. Possible causes/actions: • Make sure you have called the *OpcuacInit* function.
4097	1001	OPCUAC.Config.XML file is missing. Possible causes/actions: • The library has not been correctly installed in the application. Restore the file from the "DRIVER" folder of the installation path: see "Deployment" on page 4.
4098	1002	OPCUAC.Config.XML file is not valid. Possible causes/actions: • The library has not been correctly installed in the application. Restore the file from the "DRIVER" folder of the installation path: see "Deployment" on page 4.
4099	1003	The application certificate is missing
4100	1004	The OPCUAC.XML database configuration file is missing
4101	1005	The OPCUAC.XML database configuration file is not valid

11.3.2 OPC UA Communication errors

These errors can be related to:

- communication problems with the server
- problems with the parameters in the item reading and writing requests

Dec Error	Hex Error	Description
8193	2001	The connection with the server is not active. Possible causes/actions: • the server is not active • the connection URL is not correct • increase the value of the "Session Timeout" parameter • fix any firewall and/or antivirus interference • use the configuration tool and check for possible error reports in the status bar • check that the server has accepted the OPC-UAC library certificate
8194	2002	The connection with the server is not active due to a certificate problem. Possible causes/actions: • the server certificate is no longer valid: activate the OPCUACCFG.EXE configurator tool, access the "Configuration" option in the properties of the server concerned, perform the "Refresh" operation and restore the configuration
8195	2003	The required item does not exist.
8196	2004	The required item is not active
8197	2005	Reading of the item has failed (ReadValue service unavailable or in error).
8198	2006	The value of the item is not yet available.
8199	2007	Writing the item has failed (WriteValue service unavailable or in error).
8200	2008	The writing of the item has failed (write error on device). One possible cause is inconsistency between the data type declared in the item and the data type passed to the .NET function.
8201	2009	The SafeWrite procedure failed (the client did not read back the modified value within the SafeWrite timeout). Possible causes/actions: • increase the value of the server "Safe Write timeout" parameter • disable the "Safe Write" option in the item
8202	200A	Read/write item: the item does not contain a value of array type. Possible causes/actions: • check **numval** and **index** parameters (for single values index must be 0 and numval can be 0 or 1)
8205	200D	Read/write item: OPC UA item data type is not supported.
8206	200E	Read/write item: invalid parameters. Possible causes/actions: • check that **item** and **pdBuf** parameters are not NULL.

Dec Error	Hex Error	Description
8208	2010	Read/write item: the index specified for the array access is out of range. Possible causes/actions: • check **index** parameter.
8209	2011	Scrittura item da interfaccia C/C++: the size declared for the data buffer is inconsistent with the number of data items specified. Possible causes/actions: • check **numval** parameter and **ValueSize** field.
8210	2012	Read/write item: the number of values required is greater than those contained in the item. Possible causes/actions: • check **numval** parameter.
8211	2013	Read/write item: the current item value is not valid (DataValue is NULL).
8214	2016	The method call has failed. • Please refer to the method result code for detailed information.

12 Troubleshooting

Configurator tool: the error "Server not connected" is displayed
- check that the server's IP address is reachable - check the connection URL string - if a DNS name is being used in the URL, ensure that the name is resolved with the server's IP address (e.g. by declaration in the HOSTS file) - check that the server has accepted (trusted) the configurator's OPCUACCFG certificate - check any limits of the server: e.g. maximum number of subscriptions supported, maximum number of items in subscriptions, etc. - increase the "Session timeout" parameter value

Configurator tool: the error "Server not connected - BadCertificateHostNameInvalid" is displayed
According to security management, the "Subject Alternative Name" field of the server certificate should contain the IP address or DNS name of the server itself. A URL consistent with this information must then be used to connect to the server, so for example if the hostname is stated in the certificate as:: - IP address "192.168.0.100": the URL to use will be like "opc.tcp://192.168.0.100:4840" - DNS name "ServerOPCUA": the URL to use will be like "opc.tcp://ServerOPCUA:4840" OPC UA's standard security procedure requires that when connecting, the client performs a consistency check between the URL and what is in the certificate, and in the event of a mismatch, aborts the connection with a "BadCertificateHostNameInvalid" error. In this situation, it is necessary to: - check the URL you are using - check the content of the server certificate If the problem is that an incorrect IP address is declared in the server certificate and you are unable to have a correct certificate regenerated, you can enable the "Disable host name check" flag to inhibit the check.

Configurator tool: the error "Server not connected - check certificate - BadCertificateUntrusted" is displayed.
Both the configurator program and the driver are set up to automatically accept self-signed certificates; in principle, therefore, there should be no connection problems related to the non-acceptance of certificates. In case of problems: - access the "Configuration" option in the properties of the server, perform the "Refresh" operation and save the configuration - check for the presence of any certificates in the Certificate Store's "Rejected" folder (Automa\pki\rejected folder) and move them to the "Trusted" folder if necessary (this operation must be done while the software is not active)

Runtime: reading, writing or method calls operations give bad results
- check error codes in paragraph "Error codes" on page 50. - in case of connection problems, try using the configurator tool to investigate the problem further

13 User notes