SIEMENS

Application Description • 08/2014

Programming an OPC UA .NET Client with C# for the SIMATIC NET OPC UA Server

SIMATIC NET OPC UA Server

http://support.automation.siemens.com/WW/view/de/42014088

Warranty and Liability

Note

The Application Examples are not binding and do not claim to be complete with regard to configuration, equipment or any contingencies. The Application Examples do not represent customer-specific solutions; they are only intended to provide support for typical applications. You are responsible for ensuring that the described products are used correctly. These Application Examples do not relieve you of the responsibility of safely and professionally using, installing, operating and servicing equipment. When using these application examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these Application Examples at any time and without prior notice. If there are any deviations between the recommendations provided in this Application Example and other Siemens publications – e.g. catalogs – the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.

Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this application example will be excluded. Such an exclusion will not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). The compensation for damages due to a breach of a fundamental contractual obligation is, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change in the burden of proof to your disadvantage.

Any form of duplication or distribution of these Application Examples or excerpts hereof is prohibited without the expressed consent of Siemens Industry Sector.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens' products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit http://www.siemens.com/industrialsecurity.

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit http://support.automation.siemens.com.

Table of Contents

1 Automation Task

Purpose

With OPC Unified Architecture (UA), an additional, convenient and performant option of process coupling for PC systems with SIMATIC S7 now exists in SIMATIC NET OPC Server, which will successively replace the existing OPC Data Access (DA) and Alarms & Events (A&E) functions.

The main advantages of OPC UA over conventional OPC interfaces are:

- communication via the internet and across firewalls.
- optimized, robust and fault-tolerant protocol with integrated security mechanisms.
- OPC UA can be directly integrated in applications on different operating systems with different programming languages.
- all OPC information, such as data or alarms, is integrated in a namespace.
- information can be described with object-oriented means.

Target audience

This application is designed for end users who need a comprehensive introduction to this technology and who want to acquire experience with the professional creation of OPC UA clients in C# under .NET

Content

This is where you get an overview of the use of the OPC UA communication interface which offers the data, alarms and diagnostic information from the SIMATIC S7 controllers. You will learn about the components used, standard hardware and software components and the specially created user software.

The user software offers examples for the creation of OPC UA clients with C# under .NET. Included are a simplified, reusable API, a simple example and a complex example with a convenient user interface. The example also provides notes on the optimization and expansion of the application.

1.1 Overview

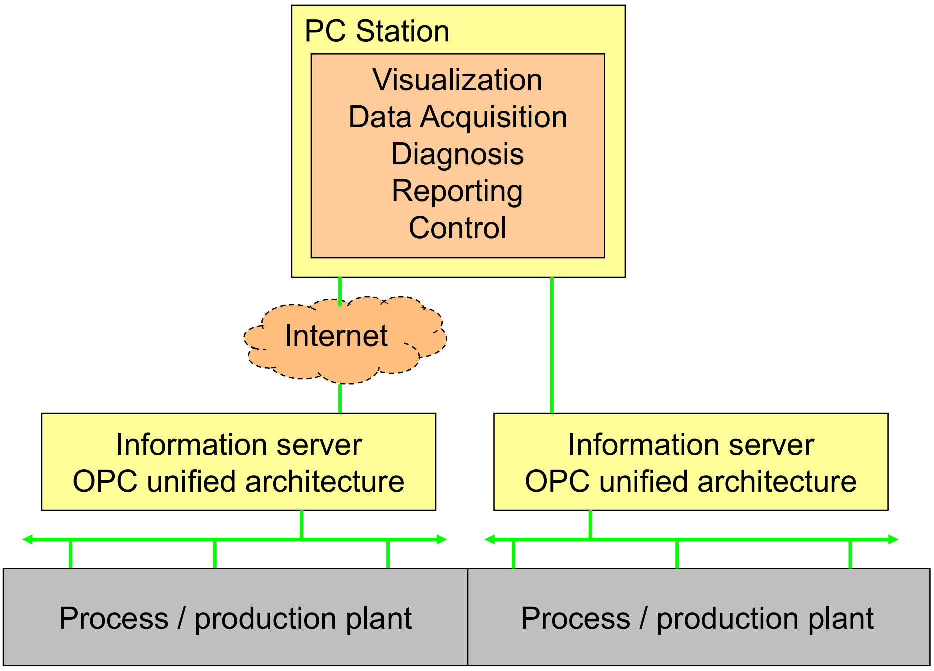
Introduction

To realize a data link, it is nowadays preferred to use standardized mechanisms in order to ensure that such a data exchange remains independent of the used bus system or protocol or even manufacturer. For the exchange of event and alarm messages, a standardized mechanism for connecting different subsystems will also be used. OPC UA combines this functionality and additionally offers authentication and encrypted data transmission and advanced diagnostic information.

Overview of the automation problem

The following figure provides an overview of the automation task.

Figure 1-1



Description of the automation task

In the automation system the OPC UA server shall be considered the information server, which can display and describe individual components but also the entire system. Due to the encrypted access, which is checked and secured with certificates, a link to other locations is also possible.

The core task of this example is access to process data with the OPC UA interface. This is explained by creating a simple, individually created visualization on the basis of the new OPC UA standard which is nevertheless suitable for real-life situations.

The application is to contain the following functionalities:

- server selection including security settings.
- navigation through the OPC UA namespace of the server and selection of process tags.
- reading of attributes including the values of the selected process tags.
- monitoring of the value of the selected process tags.
- writing the value of the selected process tags.
- use of block services via OPC UA.

Further data processing (e.g. saving in database or similar) is not discussed here.

1.2 Requirements

Requirements of the automation task

The sample application has been created in C# and uses the interfaces of .NET API of the OPC Foundation.

The user is explained the handling of the OPC UA interface under .NET in a real life situation. The basis interface is the .NET Client SDK of the OPC Foundation included in delivery on the SIMATIC NET installation.

This interface offers the full functional scope of OPC UA. To simplify the interface, a reduction to the functionality required for this example is performed. An efficient instruction which is suitable for real-life situations for the OPC UA services is developed.

The design of a simple GUI interface demonstrates the basic functionality of OPC UA. This shows the entire functional chain between S7 tag(s), OPC UA namespace and access from the client in C#:

- login, logout and authentication on the server
- searching the namespace for tags
- Reading, writing and monitoring tags
- simple error handling

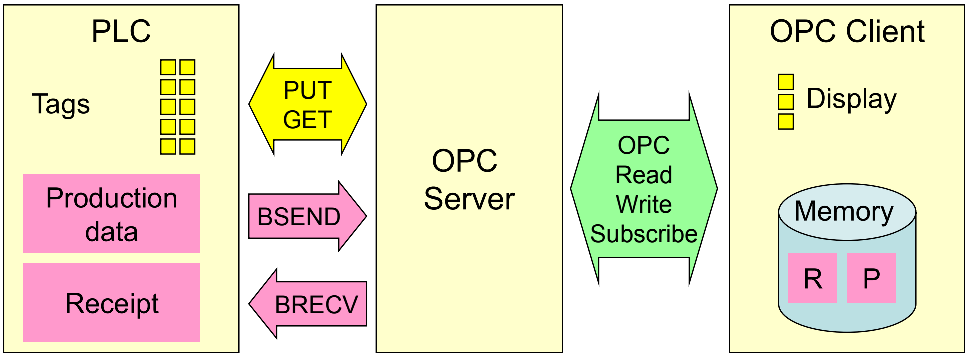
The example describes the symbolic and absolute addressing and the use of the tag services “read, write and monitor” for the S7 basic types as well as the use of the block-oriented services (receiving and sending of large data blocks).

The different diagnostics options and the processing of error scenarios by the program are explained. The errors can also be triggered by simulating disconnections between the different components.

Requirement for data storage

The controller is to be able to offer the necessary data structures and data volumes and simulate value changes. There is no concrete control task, only the access to the data is to be illustrated. The data areas and the interaction with other components are displayed in the figure below.

Figure 1-2



The STEP7 program in the S7-CPU simulates the individual values which are to be received and displayed by the client (tag services). Different data types are used as individual tags.

The PLC program simulates and generates the necessary structures and values for the bi-directional transmission of larger data volumes and calls the block-oriented services accordingly (BSEND, BRECV). This is used for STRUCT or ARRAY tags with a total of several 100 bytes (recipe data, production data blocks or similar.).

To send data, PLC actively triggers the transmission of a block-oriented production data record to the OPC UA server. The PLC receives a block-oriented data set (e.g. recipe) sent by an OPC UA client and stores it in the respective structure in a data block.

The necessary tag tables are furthermore provided in STEP 7 for test purposes.

Requirement for the PC station

The PC station must have the necessary physical connection to the respective hardware and software for the communication with the controller. The application for the visualization and control should only use the OPC UA interface to be able to use any OPC UA servers.

The application example is to show what has to be generally projected on the server/client PC station and the S7 controllers in order to solve the communication task.

In STEP 7 the SIMATIC NET OPC server is configured for the task (protocol, security settings, certificates, etc.) in the configuration console for the PC station and in the respective configuration files.

The underlying S7 protocol and the necessary connections to the controllers are configured, including all corresponding steps which are to be projected and configured on the server PC for the OPC UA operation.

Under Windows a secure communication between client PC and server PC is created by OPC UA means.

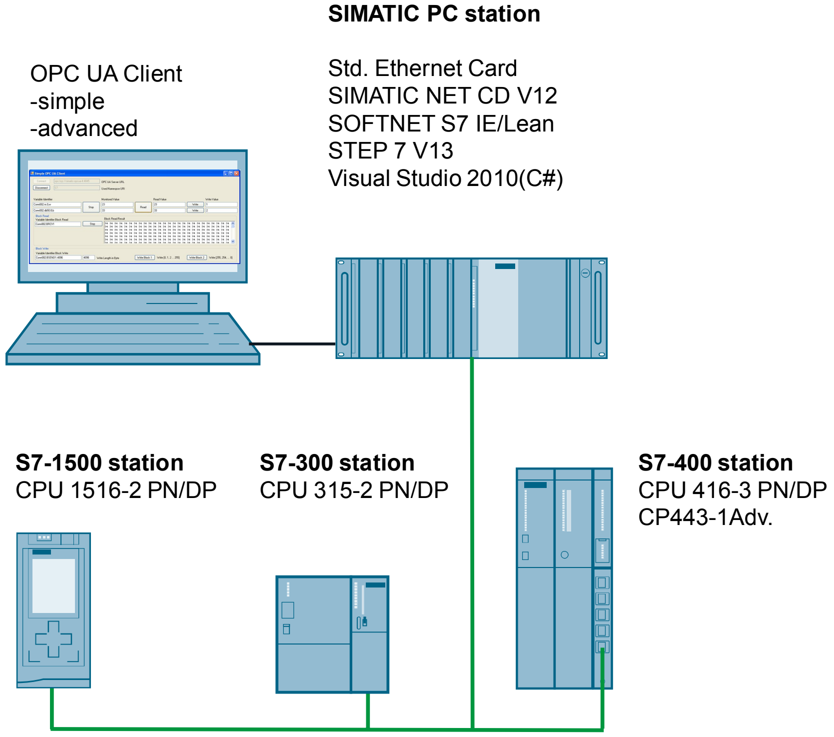
2 Automation Solution

2.1 Solution overview

Overview

The figure below shows a schematic overview of the most important components of the solution:

Figure 2-1



Structure

A PC station is connected to a CPU 315-2 PN and a CPU 414-2 via Ethernet.
A standard Ethernet card is used in the PC.

OPC-UA Client software

The OPC-UA client in the PC station is realized at two levels of complexity. A very simply designed client (**Simple OPC UA Client**) shows you all basic functions for getting started in OPC UA. A more complex client (**OPC UA .NET Client**) with a convenient interface will demonstrate professional handling with reusable classes.

The functionality of these sample clients will be explained in the next section.

2.2 Description of the core functionality

Overview

SIMATIC NET OPC UA Server forms the main functionality part of this example. It simplifies the functions and information of the classic OPC server for Data Access and Alarm & Events in one single namespace and permits access to information via a service-oriented architecture. Communication via the Internet and across firewalls is secure and performant.

This figure below shows the functional chain for a data access:

Figure 2-2

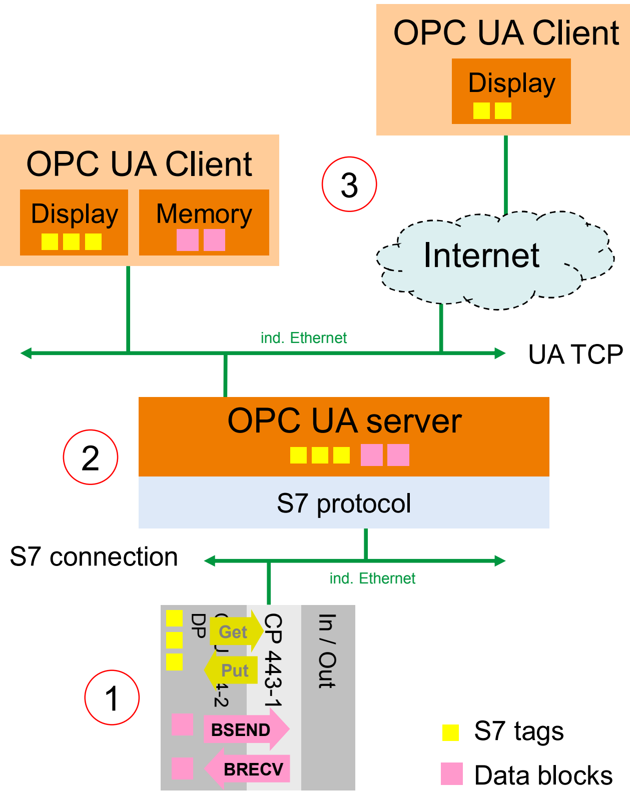


Table 2-1

No.	Component	Description
1.	S7 station	The S7 CPU provides S7 tags for data areas such as flags or data blocks. Via the block-oriented services BSEND and BRECV, larger data blocks can also be actively sent and received from the user program.
2.	OPC UA server	The OPC UA server transposes the S7 tags and the block services to the OPC UA tags and provides OPC services such as browse, read, write and data monitoring.
3.	OPC UA client	The OPC UA Client can establish a secure connection to the server, navigate through the namespace of the server and read, write and monitor selected tags.

Software components of the application (OPC UA .NET client)

The figure below shows the software components used for the more complex application (OPC UA .NET client). The OPC UA server and the basic libraries for the OPC UA communication on the client side are from the SIMATIC NET CD.

The software components created in C# for the application can be divided in reusable modules and sample code.

Figure 2-3

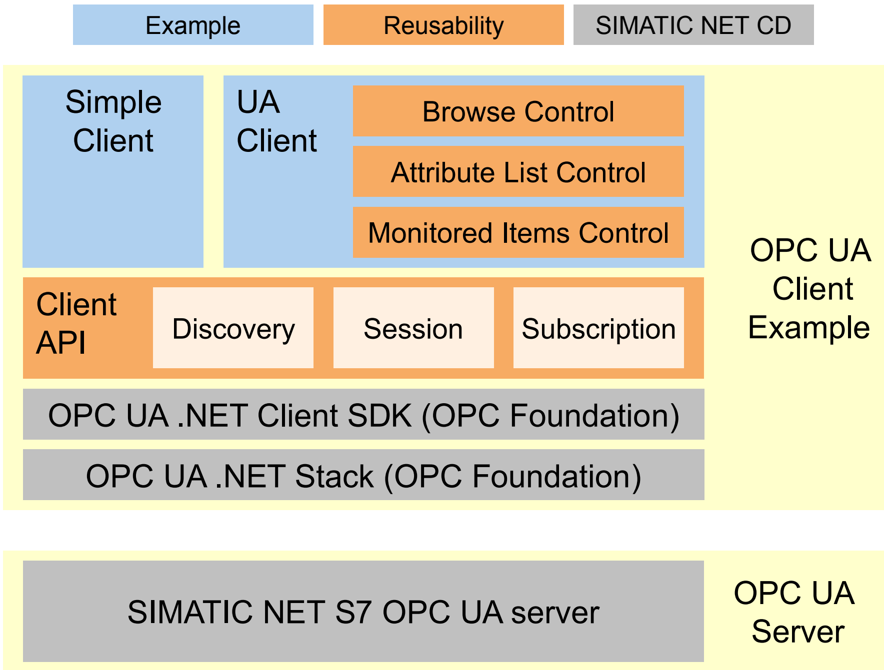


Table 2-2

Module	Description
OPC UA .NET Stack	The .NET OPC UA stack from the OPC Foundation for the realization of the network communication.
.NET Client SDK	The .NET OPC UA client SDK of the OPC foundation. The two DLLs of the OPC foundation are part of the delivery of the SIMATIC NET CD.
Client API	Reusable, simplified and tailored to this .NET Client API task. It offers reusable C# classes for discovery, session and subscription handling.
Simple Client	Simple user interface for the use of the Client API with the functions Connect, Disconnect, Read, Write and Data Monitoring. This example also shows direct addressing and the handling of namespaces.
UA Client	Convenient OPC UA client with the functions: discovery, connect, disconnect, browse, read of all attributes, write and data monitoring. General functions such as browse, listing attributes and monitoring of data tags are encapsulated in reusable controls. In this example the symbolic tags can be browsed and can be used directly from the browser.
S7 OPC UA server	The SIMATIC NET OPC UA server implements the necessary server logic for sessions and subscriptions and the data connection to the S7 stations.

User interface of the simple example (Simple OPC UA Client)

The user interface of the Simple OPC UA Client is operated via buttons for the individual functions. The simple example shows the use of the direct addressing of S7 tags.

Figure 2-4

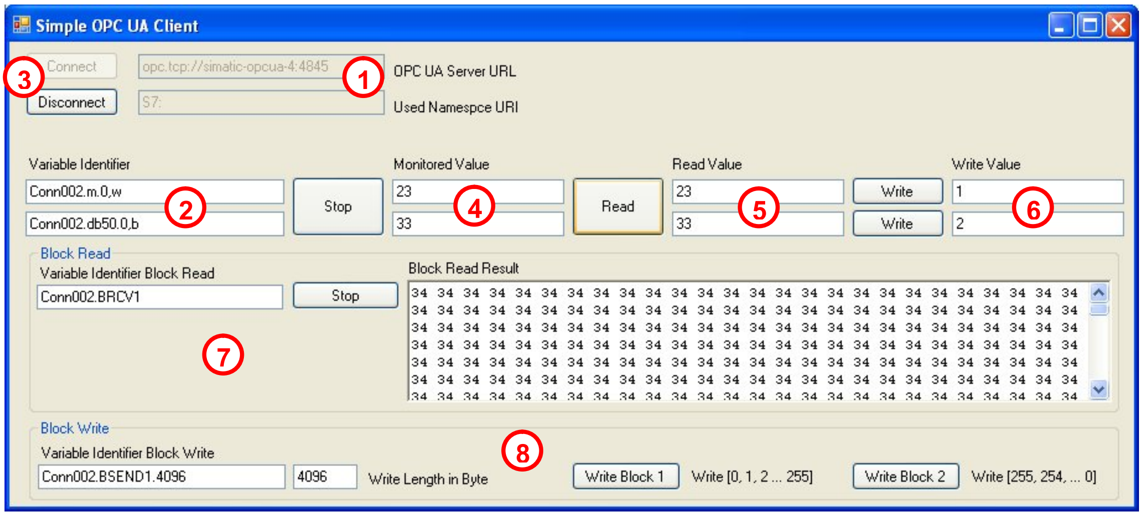


Table 2-3

No.	Description
1.	The server URL can be specified in the text box for the **Server URL**. For the SIMATIC NET OPC Server this is composed of **opc.tcp://<computername>:4845**. In the **Namespace URI** text box the namespace used is indicated. This is **S7:** for direct addressing, **S7COM:** for direct addressing via the OPC DA compatible Syntax and **SYM:** for symbolic addressing.
2.	In the text boxes for the Tag Identifier the identification code of the NodeID is indicated. For namespace **S7:** this is composed of **<S7connection>.<dataarea>.<offset>,<datatype>** The NodeID for reading and writing is made up of identification and namespace.
3.	Via the **Connect** and **Disconnect** buttons, the connection to the OPC server can be established or disconnected. The connection is only established without security. Secure connection establishment is explained in the next example.
4.	A subscription is created via the **Monitor** button and two **Monitored Items** are created in the **Subscription** with both **NodeIDs**. The data changes are displayed in the text boxes next to the button. Errors are displayed instead of the values.
5.	The **Read** button reads the values (Attribute value) of both tags with the specified **NodeIDs** and displays them in the text boxes next to the button.
6.	The **Write** button writes the value from the text box next to the button onto the tag identified by the **NodeID**. In order to write, “read” has to be called first since the text from the text box has to be converted in the data type suitable for the tag. The conversion is on the basis of the data type which is supplied at “read”.
7.	In the “**Block Read**” group, data can be received which is actively sent by the S7 with the **BSEND** block service. This can be, for example, used for the sending of result data from the S7 to a PC application.
8.	In the “**Block Write**” group, data blocks can be sent to the S7 which are there received by the **BRECV** block service. Two blocks with different contents can be sent. This can be used, for example, for the download of recipe data for the S7.

Overview and description of the comfortable user interface (OPC UA .NET Client)

The figure and table below describe the interface of the generic OPC UA client example with which the information of the namespace of an OPC UA server can be conveniently accessed.

The interface also permits browsing the symbolic S7 tags.

Figure 2-5

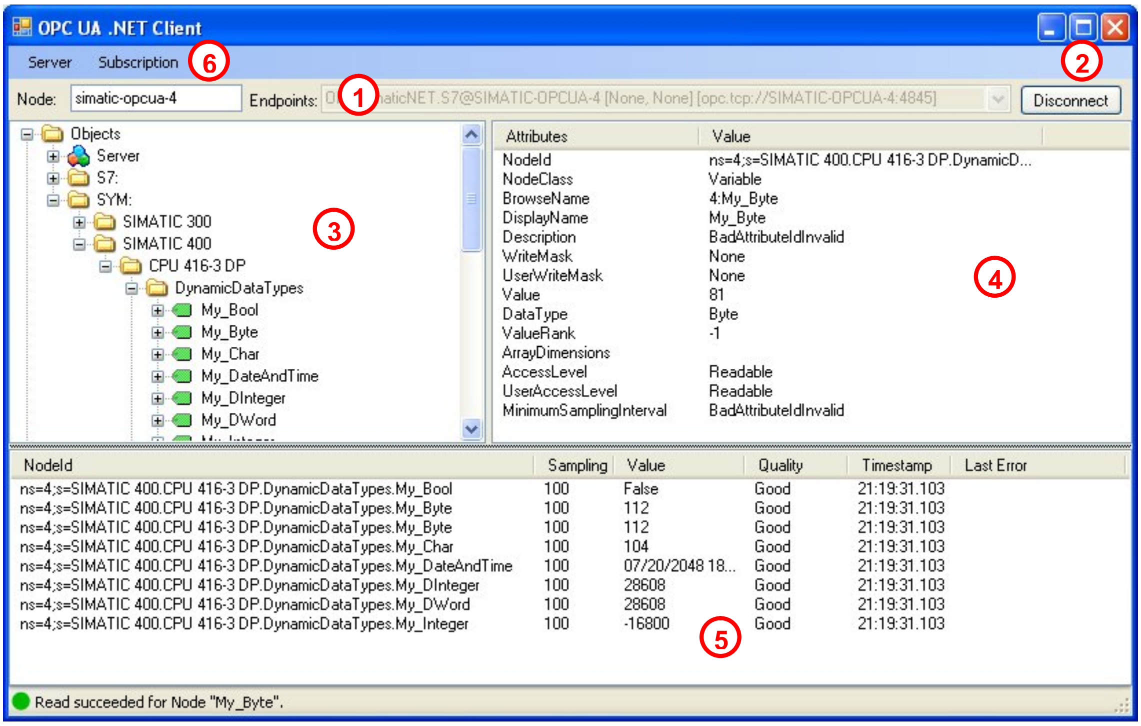


Table 2-4

No.	Description
1.	The server can be selected via the **Endpoints** selection list. For this purpose the list of the available OPC UA servers from the corresponding network node is determined. The computer, from which the list is to be prompted, can be entered in the **Node** text field. If the field is empty, the list will be determined on the local computer. The URL of the OPC UA server can also be entered manually. For the SIMATIC NET OPC UA Server the URL is composed of **opc.tcp://<computername>:4845**.
2.	The connection to the server can be established or terminated via the **Connect** button.
3.	In Browse Control the entire address space of the connected server is shown in a hierarchical tree view. Only hierarchical references are displayed.
4.	For the selected nodes the attributes are read in Browse Control and they are displayed in this control.
5.	With drag&drop the tags can be dragged from Browse Control to the monitoring window. For the tag, the NodeID, the sampling interval, the value, the time stamp and the status code is displayed.
6.	The properties of the subscription and monitored items can be changed via the context menu in the monitoring window or via the application menu. This is how e.g. the sampling interval can be changed. The dialog for writing can also be opened. Doing this, accepts the tags marked in the monitoring window in the dialog.

Advantages of this solution

The solution presented here offers you the following advantages:

- easy introduction to OPC UA technology
- programming in C# for .NET
- easy expandability of the example
- reusable program components
- access possible via internet and across firewalls
- access rights can be assigned individually for users
- handling with certificates, encryption and authentication
- demonstration of S7 communication

Delimitation

This application does not contain a description for processing or saving data in the OPC UA client e.g. in databases.

Assumed knowledge

Basic knowledge of the handling of the SIMATIC configuration and programming tool STEP7 as well as of the Microsoft Visual Studio 2008 development environment and the programming language C# and object-orientated programming is assumed.

2.3 Hardware and software components used

The application was created with the following components:

Hardware components

Table 2-5

Component	No.	Article number	Note
S7-400 CPU 416-3 PN/DP	1	6ES7416-3XR05-0AB0	Any other S7-400 CPU can also be used.
CP 443 -1 Advanced	1	6GK7443-1GX20-0XE0	Alternatively, any other S7-capable Ethernet CP can also be used.
S7-300 CPU 315-2 PN/DP	1	6ES7315-2EH14-0AB0	Alternatively, any other S7-300 with PNIO interface can also be used.
S7-1500 CPU 1516-3 PN/DP	1	6ES7 516-3AN00-0AB0	Alternatively, any other S7-1500 can be used.
SIMATIC PC station as OPC UA server	1	6AG4104-1AA22-0BB0	Standard PC (e.g. PGs) under Windows Vista or Windows XP.
Standard PC as OPC UA client	1	6AG4104-1AA22-0BB0	Alternatively, the client can also be operated locally on the SIMATIC PC station.

Software components

Table 2-6

Component	No.	Article number	Note
SIMATIC NET DVD V8.2 SOFTNET IE S7	1	6GK1704-1LW08-2AA0 6GK1704-1CW08-2AA0	LW=8 S7 connections (Lean), CW=64 S7 connections
STEP 7 Professional V13	1	6ES7822-4AA03-0YA5	For the configuration of bilateral S7 connections on S7-CPUs
Microsoft Visual Studio 2010	1	Express Edition Standard Edition Professional Edition	Obtainable in the Microsoft store (http://emea.microsoftstore.com)
.NET Framework 3.5	1	Free download at http://www.microsoft.com/	Installed by SIMATIC NET

Sample files and projects

The following list includes all files and projects that are used in this example.

Table 2-7

Component	Note
OPC_UA_DOKU_v1_1_e.pdf	This document.
OPC_UA_CODE_v1_1.zip	This zip-file contains the OPC UA Client and the sources.
OPC_UA_STEP7_v1_1.zip	This zip file contains the STEP 7 V13 project.

2.4 Alternative solutions

OPC Data Access on the basis of COM

Today, this automation task is typically solved with the COM based classic OPC data access interface.

Advantages of the solution with COM OPC Data Access:

- Wide distribution of the interface.
- Many applications for different tasks support the interface.
- Easy access for local applications.

Disadvantages of the solution with COM OPC data access:

- Complicated DCOM configuration for remote access.
- No communication possible across firewall or internet boundaries.
- OPC clients can only be operated on Windows PC systems.
- Restricted security mechanisms and user authentication only within the framework of the DCOM configuration.
- No user-defined access rights possible.

3 Basics

3.1 Basics on OPC

Overview

In recent years, the OPC Foundation (an interest grouping of well-known manufacturers for the definition of standard interfaces) has defined a large number of software interfaces to standardize the information flow from the process level to the management level. According to the different requirements within an industrial application, different OPC specifications were developed in the past: Data Access (DA), Alarm & Events (A&E), Historical Data Access (HDA) and Data eXchange (DX). Access to process data is described in the DA specification, A&E describes an interface for event-based information, including acknowledgement, HDA describes functions for archived data and DX defines a lateral server to server communication.

Based on the experience with this classic OPC interface, the OPC Foundation defined a new platform, called OPC Unified Architecture (UA). The aim of this new standard is the generic description and uniform access to all information which is to be exchanged between systems or applications. This includes the functionality of all previous OPC interfaces. Furthermore, it is to generate the possibility of natively integrating the interface in the respective system, irrespective of which operating system the system is operated on and irrespective of the programming language in which the system was created.

This example discusses the OPC Unified Architecture interface. A detailed documentation is available on the SIMATIC NET CD. For more information, please go to www.opcfoundation.org.

What is OPC?

In the past, OPC was a collection of software interfaces for data exchange between PC applications and process devices. These software interfaces have been defined according to the rules of Microsoft COM (Component Object Model) and can therefore be easily integrated into Microsoft operating systems. COM or DCOM (Distributed COM) provides the functionality of inter process communication and organizes the information exchange between applications, even across network boundaries (DCOM). Using mechanisms of the Microsoft operating system, an OPC client (COM client) can use it to exchange information with an OPC server (COM server).

The OPC server provides process information of a device at its interface. The OPC client connects itself with the OPC server and can access the offered data.

The use of COM or DCOM causes OPC servers and clients to run only on a Windows PC or in the local network and that the communication to the respective automation system has to be realized mainly via proprietary protocols. Additional tunneling tools have to be used for the network communication between client and server in order to get through firewalls or to avoid the complicated DCOM configuration. The interface can furthermore only be accessed natively with C++ applications; .NET or JAVA applications can only gain access via a wrapper layer. In practice, these restrictions lead to additional communication and software layers which increase the configuration workload and complexity.

Due to the widespread use OPC, the standard is increasingly used for the general connection of automation systems and no longer only for the original application as driver interface in HMI and SCADA systems to access process information.

To solve the mentioned restrictions in real-life situations and to fulfill the additional requirements, the OPC Foundation has defined a new platform in the last five years, called OPC Unified Architecture, which offers a uniform basis for the exchange of information between components and systems. OPC UA will also be available as an IEC 62541 standard and therefore forms the basis for other international standards.

OPC UA offers the following features:

- Summary of all previous OPC features and information such as DA, A&E and HDA in a generic interface.
- Use of open and platform-independent protocols for inter-process or network communication.
- Internet access and communication by means of firewalls.
- Integrated access control and security mechanisms on protocol and application level.
- Extensive representation options for object-oriented models; objects can have tags and methods and can trigger events.
- Expandable type system for objects and complex data types.
- Transport mechanisms and modeling rules form the basis for other standards.
- Scalability of small embedded systems up to business applications and from simple DA address spaces up to complex, object-oriented models.

3.2 Basics on OPC Unified Architecture

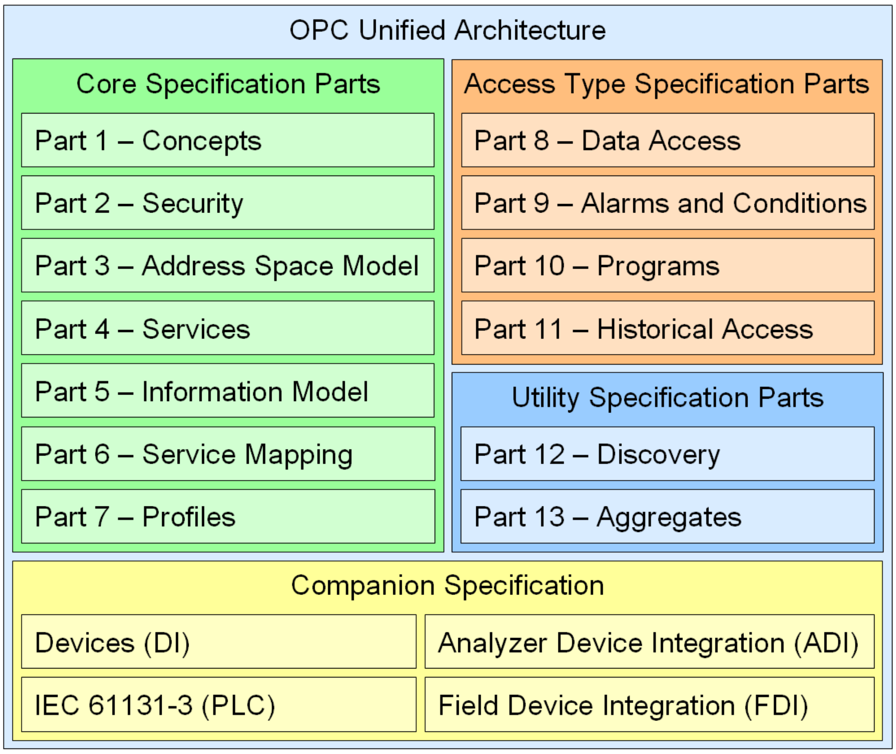
This chapter explains the basis of the OPC Unified Architecture necessary for the example.

3.2.1 OPC UA specifications

Overview

The OPC UA specifications are divided in different parts due to the IEC 62541 standardization. Figure 3-1 gives an overview of the various parts.

Figure 3-1



Part 1 to 7 form the basis of the technology and the realization of OPC UA applications. It is mainly parts 3 to 5 which form the core of the standard.

Parts 8 to 11 define OPC specific information models for the provision of classic OPC information such as current process data or alarms.

Additional tools are defined in part 12 and 13.

Moreover, so called companion specifications are generated which define additional information models, together with other standardization organizations, based on OPC UA. The models and information in other standards form the basis and the companion specification defines how this information is described and transported with OPC UA.

Note

For this application the parts three to five and part eight are relevant. The description of the other parts is included to provide a comprehensive overview of the OPC Unified Architecture.

List of specifications

Table 3-1 explains the list of specifications and their contents. The currently relevant specifications for the SIMATIC NET server are highlighted here

Table 3-1

Specification	Description
Part 1 – Concepts	This non-normative part gives an overview of the standard.
Part 2 – Security	The requirements to security and an introduction to the basics are described in the second part which is also non-normative.
Part 3 – Address Space Model	This part defines the basic rules and elements for the set-up of the address space of an OPC UA server. These rules form the basis for the information models in part 5, 8 to 11 and the companion specifications.
Part 4 – Services	This document is the only part which defines the interface for the access to all OPC UA information. It specifies a list of methods, the so called services. These services are generic and form the basis for all information models.
Part 5 – Information Model	The basis information model defines the access points in the address space and basic types such as, e.g. data types or object types. This part, together with part 3 and 4 forms the core of OPC UA.
Part 6 – Service Mapping	The services in part 4 are independent of the defined transport mechanism used. This part specifies the realization of the services in different ways of serialization, security and transport protocols for messages between OPC UA client and server. This part forms the basis for the implementation of communication stacks and is not relevant for the users of the technology.
Part 7 – Profiles	A profile specifies subset of OPC functionalities for different applications which are offered by an OPC UA server or which can be used by an OPC UA client. This part defines the list of profiles for OPC UA.
Part 8 – Data Access	This part defines the tag types, properties and quality status codes for process data. All other necessary concepts are already contained in the parts 3 to 5.
Part 9 – Alarms and Conditions	This part defines the model for the description of condition monitoring and process alarms and the signaling of status changes via events. All other necessary concepts for events are already contained in the parts 3 to 5.
Part 10 – Programs	This part defines how actions which are running over a longer period of time can be started and monitored. This is performed on the basis of state machines whose handling is defined in part 5 in OPC UA.
Part 11 – Historical Access	Here, the access to historical data and events is defined.
Part 12 – Discovery	Defines how the OPC UA server can be found in the network.
Part 13 – Aggregates	This part defines aggregate functions for data compression such as average or maximum value over a time range. The aggregates can be used for current or historical data.
Devices (DI)	This companion specification defines a generic model for the configuration and diagnostics of devices.
IEC 61131-3 (PLC)	This companion specification defines a mapping of the IEC 61131-3 software model and of the standardized control programming languages on an OPC UA server address space.
Analyzer Device Integration (ADI)	This companion specification defines a model for the configuration and data linking for complex devices for process analysis based on DI
Field Device Integration (FDI)	This companion specification defines a model for the complete engineering of field devices on the basis of Electronic Device Description Language (EDDL) and Field Device Tool (FDT).

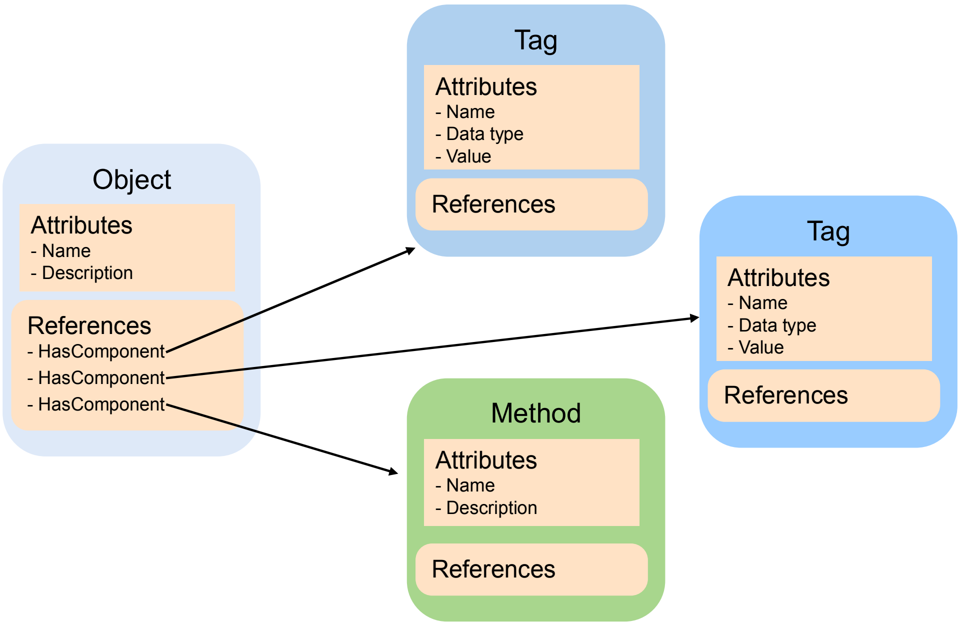
3.2.2 Structure of the OPC UA Server address space

Node in the address space

A node in the OPC UA address space is of a certain type such as e.g. object, tag or method and is described by a list of attributes. All nodes have joint attributes such as name or description and specific attributes such as, e.g. the value of a tag. The list of attributes cannot be extended. Additional information on the node can be added as property. Properties are a special type of tag.

The nodes are interconnected with references. The references are typified. There are two main groups, hierarchical references such as, e.g. HasComponent for the components of an object or non-hierarchical references such as, e.g. HasTypeDefinition for a connection of an object instance to an object type. Figure 3-2 offers an example for a node and the connection references.

Figure 3-2



Available types of nodes in the address space

The defined node types are listed in Table 3-2. The list of types cannot be extended.

Table 3-2

Node type	Description	Example
Object	An object is used as typified container for tags, methods and events.	The objects which represent a S7 connection always have the same structure.
Tag	Tags represent the data of objects or as property, the properties of a node.	S7 tag in a data block.
Method	Methods are components of objects and can have a list of input or output parameters. The parameters are described via defined properties.	BlockRead( ) method on a S7 connection object with which a block can be read out from the S7.

Node type	Description	Example
View	Views represent a part of the address space. The node is used as access point and as filter when browsing.	Views are not available in the SIMATIC NET server.
Object type	Object types supply information on the structure or the components of an object.	S7ConnectionType describes the components which are present in a S7 connection object.
Tag type	Tag types typically describe which properties or data types can be found in an instance of the type (tag).	The AnalogItemType defines that a tag of this types provides the EngineeringUnits properties and the EURange.
Reference type	Reference types define the possible types of references between nodes.	A method is referenced by an object with HasComponent.
Data type	Data types describe the content of the value in a tag.	The value of a tag can have the Double data type.

Structure of the address space

The basic structure of the OPC UA address space is defined in part 5. Figure 3-3 shows one part of this structure and SIMATIC NET shows specific parts. The different areas are described in Table 3-3.

Figure 3-3

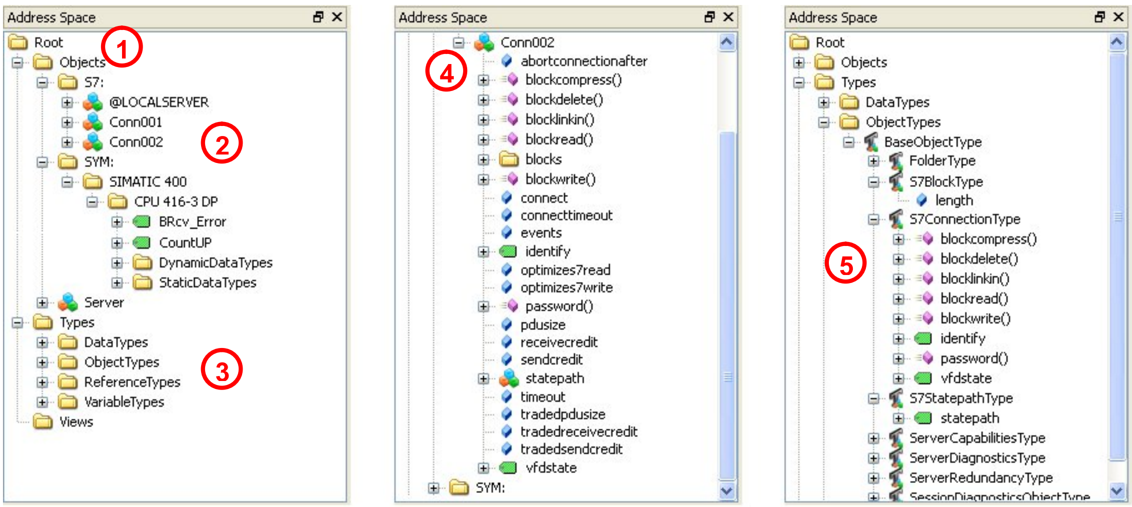


Table 3-3

No.	Description
1.	In the Objects directory, instances such as objects and tags can be found. In this directory a data access client can find the tags for data access Apart from the specific SIMATIC NET directories you can also find the server object here which was defined by OPC UA. It contains information on the range of function and the status of the server.
2.	The two directories S7: and SYM: under Objects, are specific for the SIMATIC NET OPC UA server. Under S7: the configured S7 connections are listed as objects. SYM: contains the symbols from the STEP 7 project.
3.	In the Types directory are the different type nodes for DataTypes, ObjectTypes, ReferenceTypes and TagTypes.
4.	An S7 connection object provides various status information and methods. You can, e.g. process or read out blocks in the S7 via methods. Apart from the methods, the properties supply information on the configuration of the S7 connection.
5.	The S7ConnectionType belonging to the S7 connection object, can be found in the ObjectTypes directory. It describes the minimum of methods and tags, present at the instance. The rules for the type system are described in detail in /2/.

Namespaces and NodeID

Each node in the OPC UA address space is uniquely identified by a NodeID. This NodeID is made up of a namespace to distinguish codes from different subsystems and a code which can either be a numerical value, a string or a GUID.

Strings are typically used for the code. This is analog to OPC Data Access, where the itemID as code is also a string. Numerical values are used for statistical namespaces such as, e.g. type system.

OPC UA defines a namespace for the nodes defined by OPC. The OPC UA servers additionally define one or several namespaces. Table 3-6 lists the relevant namespaces for the SIMATIC NET OPC UA Server.

Table 3-4

Namespace	Description
http://opcfoundation.org/UA/	Used for nodes which are defined in the OPC UA part 5. These are nodes which form the basic structure of the address space and nodes which represent types defined by OPC UA.
S7:	Namespace for direct addressing of S7 tags with an optimized syntax.
S7COM:	Namespace for direct addressing of S7 tags with syntax compatible to the OPC Data Access Server.
SYM:	Namespace for symbolic addressing of S7 tags. The symbol information is exported from the STEP 7 project.

Attributes of the nodes

The most important attributes of nodes are listed as an example in the table below. The main emphasis is on the tag node type.

Table 3-5

Attributes	Node type	Description
NodeID	All	Unique node address.
DisplayName	All	Localized display name for the node. The language depends on the language requested by the client for the connection and on the languages supported by the server.
BrowseName	All	Non-localized name for the node. The name contains a namespace and is mainly relevant for the use of types.
NodeClass	All	Type of node such as, e.g. object, tag or method.
Description	All (optional)	Optional localized description of the node.
Value	Tag	Value of the tag. Just like for all other attributes, time stamp and status of the value are delivered together with the value of the attribute when reading them.
DataType	Tag	Data type of the Tag or the Value attribute. Data types are, e.g. OPC UA defined data types such as Int32, Double or String or also structured data types.
ValueRank	Tag	Indicates whether the value (value attribute) is a scalable value, an array or a multi-dimensional array.
AccessLevel	Tag	Indicates whether the tag can be read or written.

3.2.3 Interface for access to the OPC UA Server address space

Communication channel and application objects

Figure 3-4 shows the different objects which can be created during data exchange between OPC client and server. The objects are described in Table 3-6.

Figure 3-4

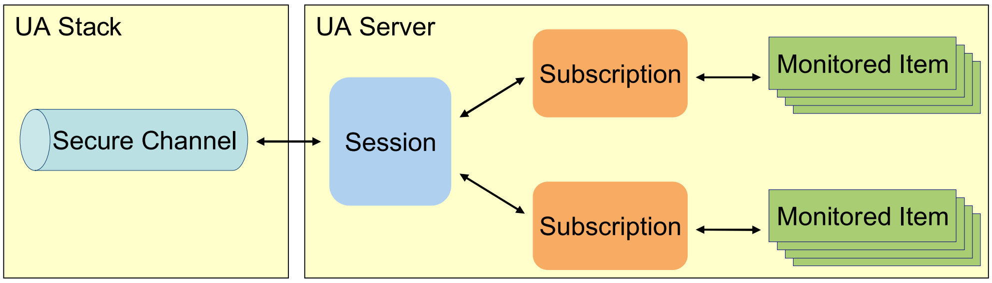


Table 3-6

Object	Description
Secure Channel	The secure communication channel is realized in the OPC UA stack. The objects on application level are independently viable. However, they can only be created, used or changed within the context of a secure channel. If a new secure channel is established after an interrupted connection, it has to be assigned to the session on application level.
Session	The session in the server is the logic connection between OPC UA client and server. It contains user information and language settings for the connection. The session is deleted from the server if no calls are received by the client within the timeout. The timeout is specified by the client. The session is linked to a secure channel but can be assigned a new secure channel if the communication was interrupted.
Subscription	A subscription object can be created by the client to group monitored items. Monitored items are used to monitor value changes or to receive event messages. The subscription is deleted by the server if no data or KeepAlive messages could be sent to the client within the timeout. The timeout is specified by the client.

Methods for establishing the connection

Table 3-7 explains the most important methods of the OPC UA interface for establishing a connection.

Table 3-7

Method	Description
OpenSecureChannel	Opens a secure communication channel between client and server. To open the connection, the server URL, the application certificates and the security settings are necessary.
CreateSession	Creating an application session within the context of a secure channel.
ActivateSession	Activating the session by transferring the user authentication and language settings. This method is also used to assign an existing session to a new secure channel or to change the user.
CloseSession	Closes the application sessions.

Methods of the session object

Table 3-8 explains the most important methods of the OPC UA interface regarding the session.

Table 3-8

Method	Description
Browse	Supplies the list of nodes which can be obtained from a start node via a reference. The quantity of nodes can be restricted by filters. For each node, information is delivered which is, e.g. necessary for the display in a tree view.
Read	Reads a list of node attributes. With this method, values of tags (value attribute) and also meta data such as, e.g. the data type of a tag (DataType attribute) can be read.
Write	Writes a list of node attributes. This is a typical method for writing values of tags. If the server permits it, other attributes can also be written.
CreateSubscription	Creating a subscription for the receipt of data changes or event messages. The subscription is used for the grouping of information which is to be monitored. All new data or events are delivered as a package in adjustable time intervals for a subscription.
DeleteSubscription	Deletes a subscription.

Methods of the subscription object

Table 3-9 explains the most important methods of the OPC UA interface regarding the subscription.

Table 3-9

Method	Description
ModifySubscription	Changes the settings of a subscription, such as e.g. the publish interval in which new data for the client is collected and jointly sent.
CreateMonitoredItems	Creating a list of monitored items in a subscription. A monitored item is either used to monitor a value of a tag or to monitor event messages. Both types of monitored items can be combined to this method in one call. In this application, only data changes are monitored.
ModifyMonitoredItems	Changes the settings of a list of MonitoredItems, such as e.g. the sampling interval for the monitoring of value changes.
DeleteMonitoredItems	Deletes a list of monitored items in a subscription.
Publish	Method for transferring data packages for a subscription with value changes and event messages in the publish interval. This method is not visible in the Client API. The functionality there is realized as callback to the client application.

3.2.4 Protocols and security mechanisms

OPC UA communication architecture

The services for the access to the information in an OPC UA server address space such as browse, read and write are abstract and specified independent from the transport protocol in part 4.

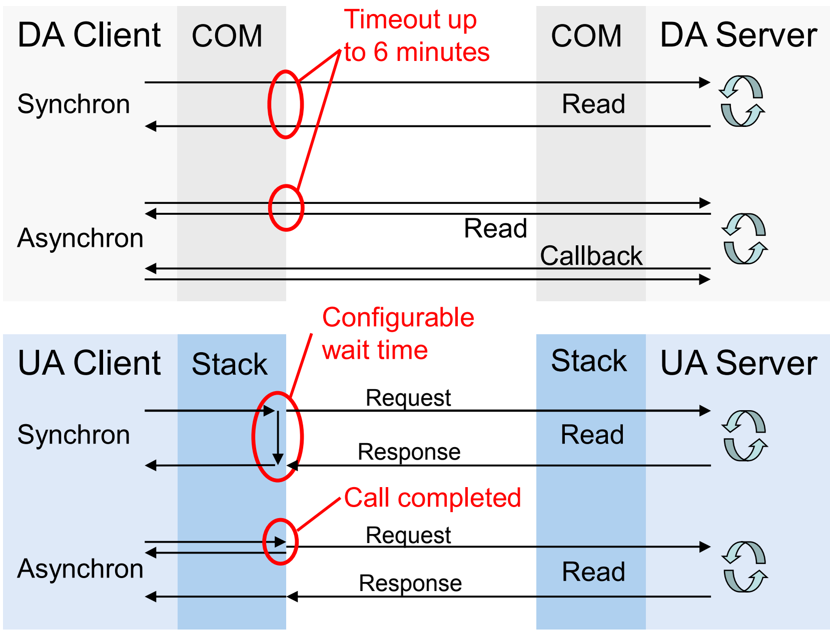
The different bindings for the transmission of service messages between OPC UA client and server are defined in part 6. A binding is made up of protocol, security mechanisms and serialization type for the data.

The bindings are implemented in communication stacks. At the moment there are three implementations from the OPC Foundation, namely in ANSI C, C# / .NET and JAVA. In this application, C# / .NET Stack is used.

The methods on the API of the stacks for the application correspond to the services in part 4 with concrete data types from the respective programming language. This is how in application development a native API can be accessed in the respective programming language. The application can also be implemented independent from the binding used. New bindings can be expanded by exchanging the OPC UA stacks.

Synchronous and asynchronous calls

Figure 3-5



For COM all calls to the server are synchronous. This is why additional asynchronous functions were defined for few actions such as read and write. A synchronous call starts the action in the server. After completing the action, the server sends a synchronous callback to the client. Due to the synchronous call to start the action, asynchronous calls may also block when the network connection is interrupted.

In the case of OPC UA all calls to the server are asynchronous. There is no differentiation between synchronous and asynchronous methods in the specification. Once the request message was written on the network, the asynchronous call is returned to the client application.
This is why an asynchronous call cannot be blocked. Since an asynchronous call can always be made synchronous, the stacks offer all OPC UA methods also as synchronous calls. For this purpose, the call is held in the stack until the response message has arrived from the server or until the timeout has expired. The timeouts can be adjusted individually per call. For the server there is no difference between synchronous and asynchronous calls

Safety layers

The different security layers of OPC UA are described in Figure 3-6 and Table 3-10.

Figure 3-6

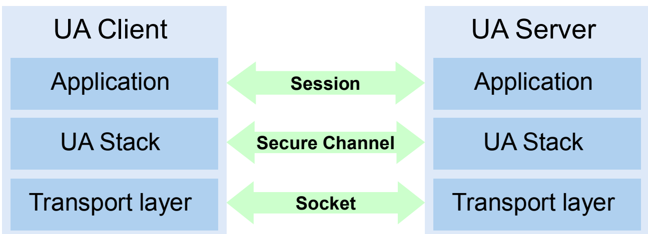


Table 3-10

Layer	Description
Socket	On the socket level, a connection-oriented security of the socket connection via Secure Socket Layer (SSL) or via Virtual Private Network (VPN) can be used in addition or as an alternative to the secure channel.
SecureChannel	On the SecureChannel level, mutual authentication of the applications and a message-based security of the communication are performed. Each message is signed and encrypted to ensure the integrity and secrecy of the messages. Basis of these mechanisms are certificates which uniquely identify the applications based on a Public Key Infrastructure (PKI) system. A detailed description of this mechanism is located in /2/. Exchanging these certificates as an important step in the security configuration is described in the next section.
Session	On the session level a user authentication is performed.

Configuration options for Security

Table 3-11 describes the different configuration options for the security mechanisms.

Table 3-11

Option	Description
Security Policy	**None** – In the secure channel no security is used. **Basic128Rsa15** – Set of algorithms for the security. **Basic256** – Set of algorithms for the security with longer keys.
Message Security Mode	**None** – The messages are not secured. **Sign** – The messages are signed. **Sign&Encrypt** – The messages are signed and encrypted.
User Authentication	**Anonymous** – User authentication is not necessary. **User Password** – The user authentication is performed using user names and password. **Certificate** – The user authentication is performed using a certificate.

Exchange of certificates

The exchange of certificates between client and server and the accepting of the certificates is explained in Figure 3-7 and Table 3-12.

When all applications involved implement the guidelines of the OPC UA regarding the security configuration, then only one manual step at the server is necessary for the exchange of certificates, since the certificates are automatically exchanged between the applications and the certificates only have to be accepted by an administrator.

The manual exchange of certificates is explained in chapter 5.3.2 since not all applications implement the automatic steps yet.

Figure 3-7

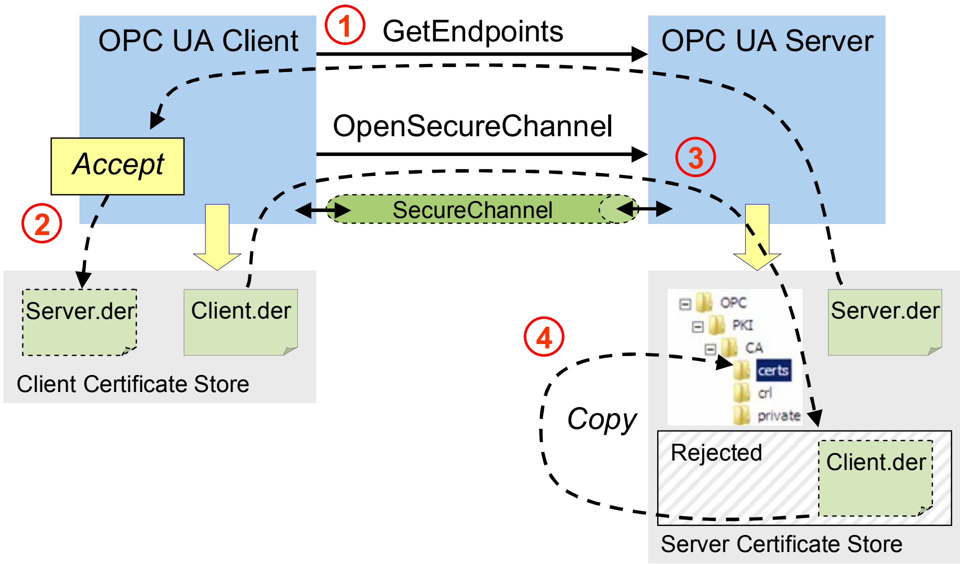


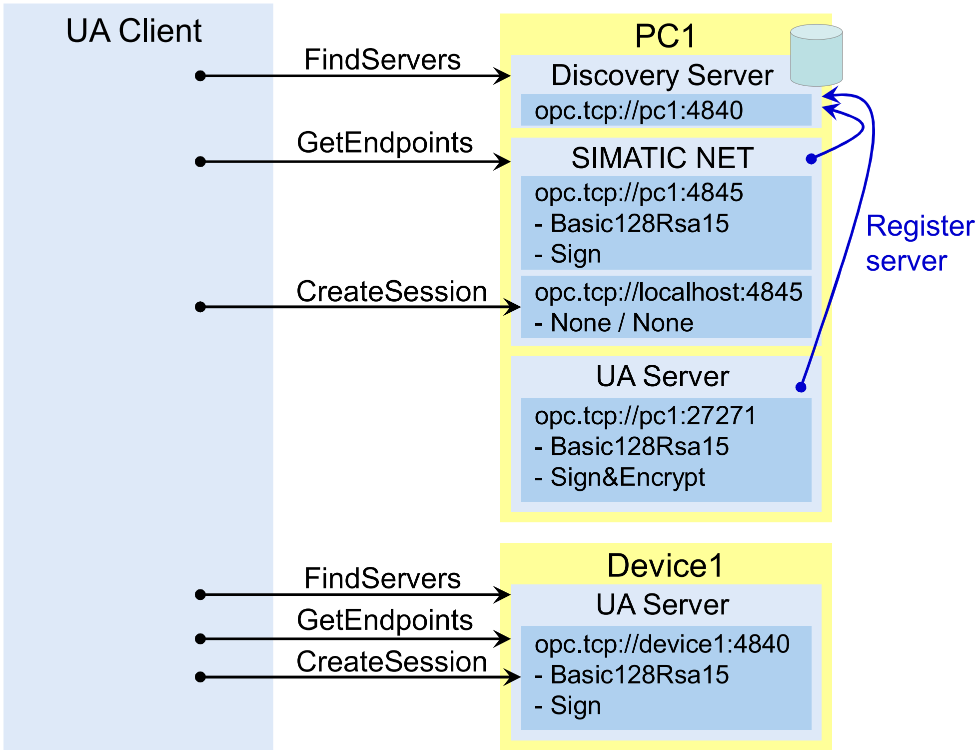
Table 3-12

Step	Description
1.	Before the client can connect itself with the server, it needs the necessary information such as the security mechanisms, protocols and the address for connection demanded by the server. This information indicates a so called endpoint. The available endpoints of a server are delivered with the GetEndpoints call. With the description of the endpoints the server also delivers its certificate.
2.	Once the endpoints have been selected with the security settings, the user is asked whether he/she wants to accept the certificate. If yes, this is stored in the certificate storage of the client.
3.	When calling the OpenSecureChannel the client certificate is transferred to the server. If the certificate is not known in the server, it will be stored in a Rejected directory.
4.	With a configuration tool of the server, certificates from the Rejected directory can be accepted. They are moved to the certificate storage of the server.

Server discovery

So far, a Local Discovery Service (LDS) has been defined for OPC UA Discovery which from its basics functionality is comparable with the OPC Enum with the classic OPC.

Figure 3-8



A LDS supplies a list of the local network nodes available on OPC UA servers. By default, the LDS is registered on port 4840 of the OPC UA. Therefore the LDS is always addressable as URL via **opc.tcp://<Node>:4840**. The servers on a PC are registered with the LDS.

A client can select a server and establish a connection with the following steps:

- Establishing a connection without security with port 4840 and calling **FindServers**. This call supplies a list of available servers and their discovery URL.
- Establishing a connection without security to the discovery URL of the desired server and calling the **GetEndpoints**. This call supplies the list of endpoints with the endpoint URLs and the security settings of the endpoints.
- Establishing a connection with the endpoint URL and the demanded security settings. Subsequently an application session can be opened with **CreateSession**.

If only one OPC UA server is available on a system, it can run on the standard port 4840, since all servers also have to implement FindServers and as a result only supply themselves. In this case, the endpoints also use port 4840.

3.2.5 Delimitation and comparison with OPC data access

Overview

The OPC Unified Architecture draws on all the features of the classic OPC interfaces and simply implements them in a joint approach for different classic OPC interfaces.

This application deals with the data access functionality in OPC UA. This is why the implementation of the OPC data access on an OPC UA is explained in this chapter.

Structure of the address space

Table 3-13 explains the implementation of the OPC DA address space on the concepts of OPC UA.

Table 3-13

OPC Data Access	OPC unified architecture
Node in the address space	
Directories are used to hierarchically structure the address space.	Directories can be realized with Object folders. The hierarchy is set up with Organizes references.
OPC Items represents data points in the address space. They are the sheets of the directories.	Tag nodes are used to depict OPC items.
Properties and attributes	
ItemID for OPC items	NodeID for all nodes in the address space
Property Item Canonical Data Type	Attribute DataType, ValueRank and ArrayDimension
Properties Item Value, Quality and Timestamp	The value attribute supplies value, status and timestamp.
Property Item Access Rights	Attribute AccessLevel and UserAccessLevel

Access to data

Table 3-14 explains the implementation of the OPC DA access to data on the concepts of OPC UA.

Table 3-14

OPC Data Access	OPC unified architecture
Context	
COM object OPCServer	OPC UA Session
COM object OPCGroup	OPC UA Subscription
OPCItem in a group	Data Monitored Item in a subscription
Creating a context	
CoInitializeEx CoInitializeSecurity CoCreateInstanceEx creates OPCServer	OpenSecureChannel CreateSession ActivateSession
AddGroup RemoveGroup IOPCGroupStateMgt::SetState	CreateSubscription DeleteSubscriptions ModifySubscription
AddItems RemoveItems	CreateMonitoredItems DeleteMonitoredItems

OPC Data Access	OPC unified architecture
Access to information	
ChangeBrowsePosition / BrowseOPCItemIDs GetItemID / QueryAvailableProperties	Browse
IOPCItemIO::Read IOPCSyncIO::Read IOPCSyncIO2::ReadMaxAge IOPCAsyncIO2::Read IOPCAsyncIO3::ReadMaxAge IOPCItemProperties::GetItemProperties	Read
IOPCItemIO::WriteVQT IOPCSyncIO::Write IOPCSyncIO2::WriteVQT IOPCAsyncIO2::Write IOPCAsyncIO3::WriteVQT	Write
OnDataChange	Publish
GetStatus ShutdownEvent	Reading or monitoring the ServerState and ServerStatus tags

3.3 Basics on S7 communication

3.3.1 General

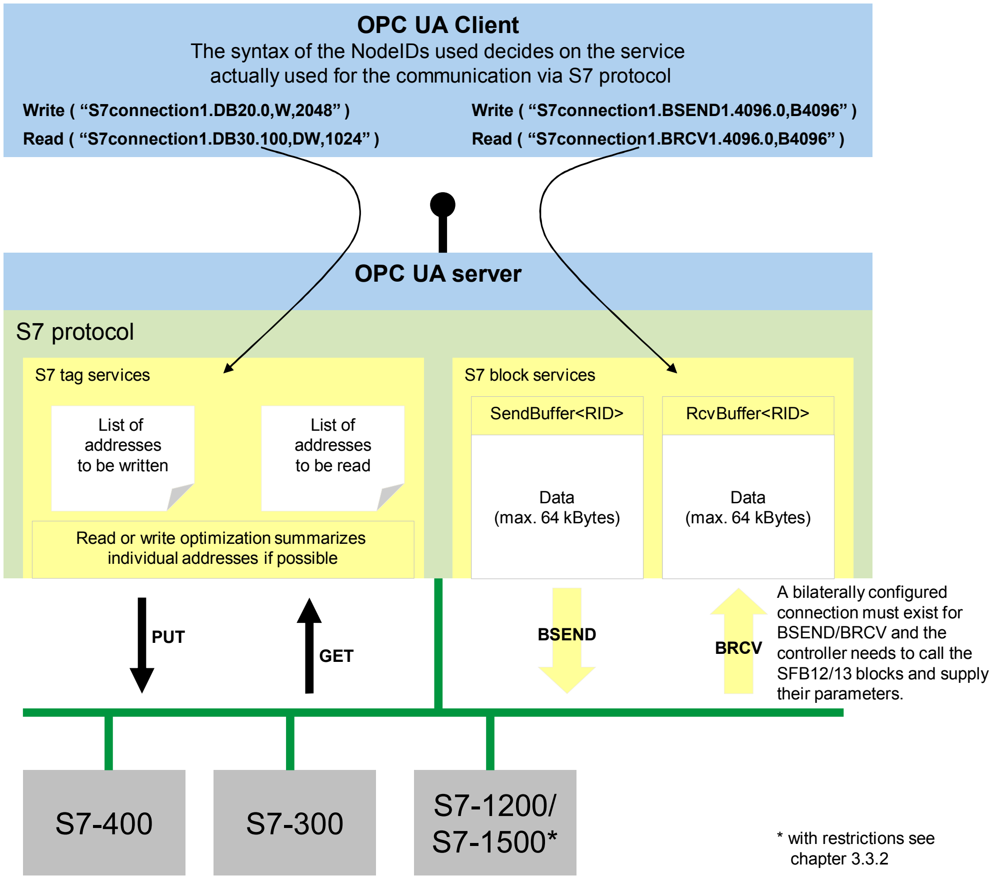
This section describes how to access S7 controllers via UA server using the SIMATIC S7 protocol.

Functional chain of the communication

The S7 communication is divided into two very different communication services, into tag services and block services. On the level of the OPC UA server they are almost completely covered. The communication service used for the controller can only be detected on the basis of the NodeID. The “S7:” name space specifies that it is a new addressing type based on the syntax of anypointers. Especially in the case of a large number of tags it offers a clearly more performant access. To achieve compatibility with the earlier syntax of ItemIDs for the classic OPC Data Access, the old structure of the identifier under the “S7COM:” namespace remains present.

Internally, the OPC UA server separates the NodeID into its components and, based on its structure, detects via which communication service communication to the S7 controller is to take place. Here, the connection name identifies the communication partner (this name represents an IP address for example) and the key word “BRCV” or “BSEND” causes the use of the block services instead of the tag services. The S7 type-identifier and the offset address indicate the position of the data within the controller, and the data type specifies the interpretation of this data.

Figure 3-9



Tag services

An S7 controller replies to requests via tag services; for this purpose only a unilaterally configured connection is necessary. Each S7 controller is a so called "S7 server" and answers PUT/GET requests without the need of any implementation in the control program of the PLC. All data areas of the controller can be directly accessed (I, Q, M, DB, etc.). This communication service is very flexible and, above all, easy to use.

NodeIDs for tag services

Syntax in namespace **S7**:

<connectionname>.<S7object>.<address>{,{<S7type>}{,<quantity>}}

Example: S7-connection_3.DB10.20,W

A tag of the word type (16bit no signs), which is located in data block 10 and which starts at the byte offset address 20 (meaning it consists of bytes 20 and 21). This tag is retrieved with PUT/GET via the connection called "S7connection_3", meaning by the S7 controller which is hidden behind this connection.

Symbolic NodeID

Apart from direct addressing via the new syntax (“S7:”Namespace) as well as the old compatible address (S7COM:-Namespace), there is the option of symbolic addressing. For this purpose, the address space is generated from STEP 7. For all symbolic identifiers of the data points in the S7 controllers which are connected with an OPC server via a S7 connection, a symbol export can be triggered. The thus generated symbols file with the ending ATI is introduced to the OPC UA server via download from STEP 7 or via XDB import. The ATI file (Advanced Tag Information) contains an image of the symbolic name for the direct addresses.

Note Symbolic addressing in the fast, highly optimized ATI variant is only available for the tag services of the S7 protocol. In different words: all symbols are eventually retrieved from the controllers via PUT/GET. Symbols which represent a BSEND or BRCV tag are not possible.

Block services

For the exchange of large data volumes, the more effective block service is available. On a bilaterally configured connection, large data volumes (up to 64kbytes) can be exchanged. Communication is based on the exchange of data buffers. However, the respective system function blocks (BSEND/BRECV) have to be called in the control program for this purpose. The OPC UA server provides the respective counterparts on the PC when the corresponding NodeIDs (former OPC items) are created.

Structure of the NodeIDs for block services

Syntax BRCV in namespace **S7**:

<connectionname>.brcv<rid>.<address>{,{<S7type>}{,<quantity>}}

Example: S7-connection_5.brcv3

The complete receive buffer for the BSEND/BRECV pair with ID 3, which is connected via the connection named “S7connection_5”, is represented in a ByteString for OPC UA. This ByteString always contains the data last sent from the communication partner with BSEND (on the other side of “S7connection_5”). On a S7 connection, several BSEND/BRECV pairs belonging together and connected via their RID can exist. Here, it is the BRECV which belongs to BSEND with ID 3.

Syntax BSEND in namespace **S7**:

<connectionname>.bsend<rid>.<bufferlength>.<address>{,{<S7type>}{,<quantity>}}

Example: S7-connection_2.bsend1.1024.100,W,20

When writing on this NodeID, an array of words (unsigned integer 16 bit) with 20 elements from the byte offset address 100 is written to the send buffer of 1024 byte length. The range of 100 to 140 is overwritten in the 1024 byte size buffer. The entire block is sent with ID 1 to the communication partner who has to provide a BRECV with ID 1 and a minimum length of 1024 bytes to be able to receive the data.

Note To be able to use the BSEND/BRCV block services, a bilaterally configured connection has to exist and the controller has to independently call the SFB12/13 blocks and supply their parameters.

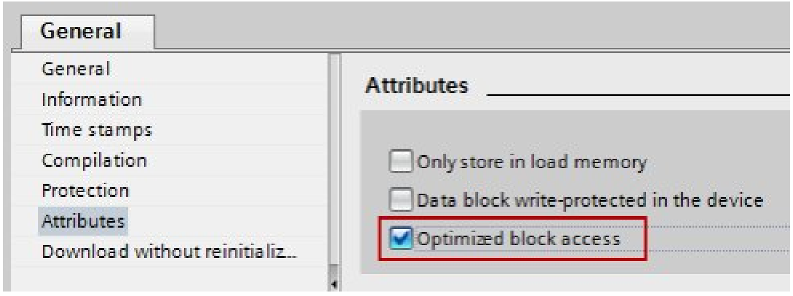
Also read the notes in the SIMATIC NET manual “Industrial Communication – Volume 2 – Interfaces” regarding the subject of block-oriented services.

3.3.2 Optimized S7 communication

Background

With an S7 connection, access to optimized data blocks in the S7-1200/S7-1500 CPUs is no longer possible. In order to fetch data from an S7-1200/S7-1500 via an S7 connection using an OPC server, the data blocks to be read from must not be optimized.

Figure 3-10 Properties of a data block

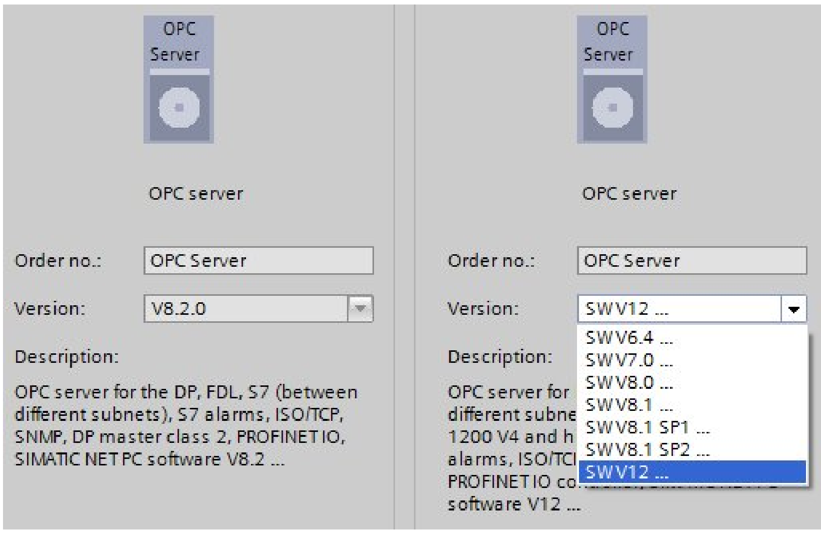


Disadvantage: the performance of the innovated S7-1200/S7-1500 controllers is affected by using non-access-optimized data blocks.

Remedy

In order to avoid performance loss in the S7-1200/S7-1500 controllers due to non-access-optimized data blocks, the so-called “optimized S7 communication” applies as of SIMATIC NET OPC V12.

Figure 3-11



This connection type is created automatically when using SIMATIC NET OPC Server V12 and an innovated controller. Access to access-optimized data blocks is then also possible using the optimized S7 connection.

Access with OPC Client

The point of access to the optimized S7 connections is realized via the SimaticNET.S7OPT server on port 4850.

4 Functional Mechanisms of this Application

General overview

Figure 4-1

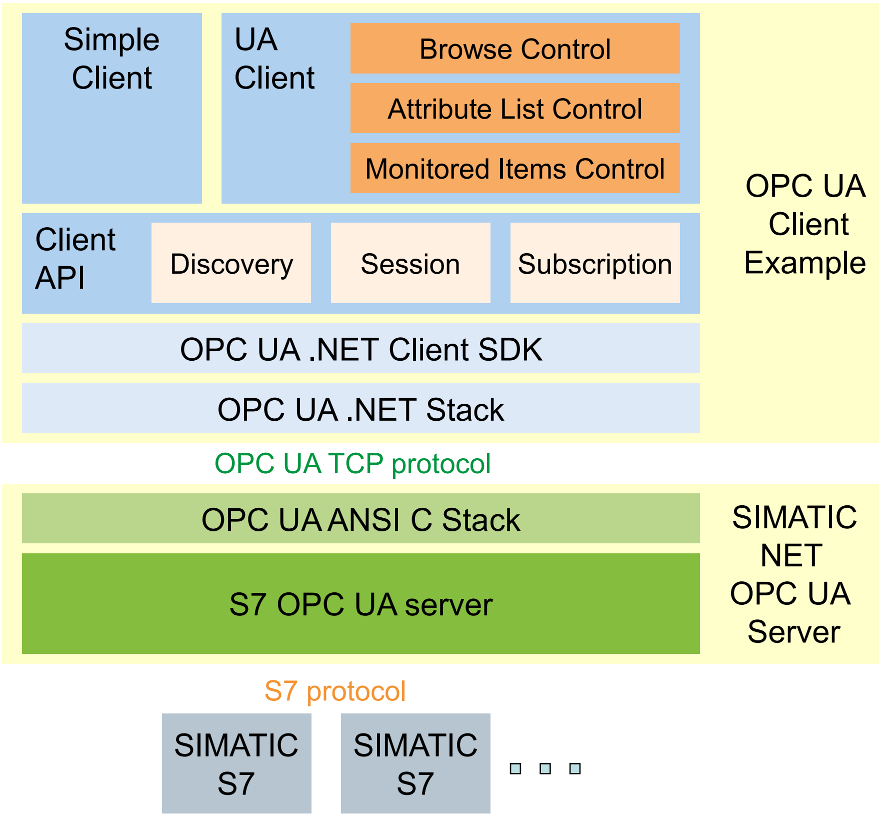


Table 4-1

Module	Description
OPC UA .NET Stack	The .NET based OPC UA communication stack of the OPC foundation.
.NET Client SDK	The .NET based OPC UA client SDK of the OPC foundation.
Client API	Reusable, simplified and tailored to this .NET Client API task. It offers reusable C# classes for discovery, session and subscription handling.
Simple Client	Simple user interface for the use of the Client API with the functions Connect, Disconnect, Read, Write and Data Monitoring. This example also shows direct addressing and the handling of namespaces.
UA Client	Convenient OPC UA client with the functions: discovery, connect, disconnect, browse, read of all attributes, write and data monitoring. General functions such as browse, listing attributes and monitoring of data tags are encapsulated in reusable controls. In this example the symbolic tags can be browsed and can be used directly from the browser.
ANSI C UA Stack	The SIMATIC NET OPC UA server uses the optimized and portable OPC UA ANSI C stack of the OPC foundation.
S7 OPC UA server	The SIMATIC NET OPC UA server implements the necessary server logic for sessions and subscriptions and the data connection to the S7 stations.

.Net Client SDK

The used SDK of the OPC foundation is not anymore maintained or supported. That does not affect the functionality of the application example.

It is recommended to use for own applications the SDKs of commercial supplier like the SDKs from Unified Automation (refer to \7\).

Program overview

The figure below shows the function blocks in the OPC UA Client and the interaction with the OPC UA Server.

Figure 4-2

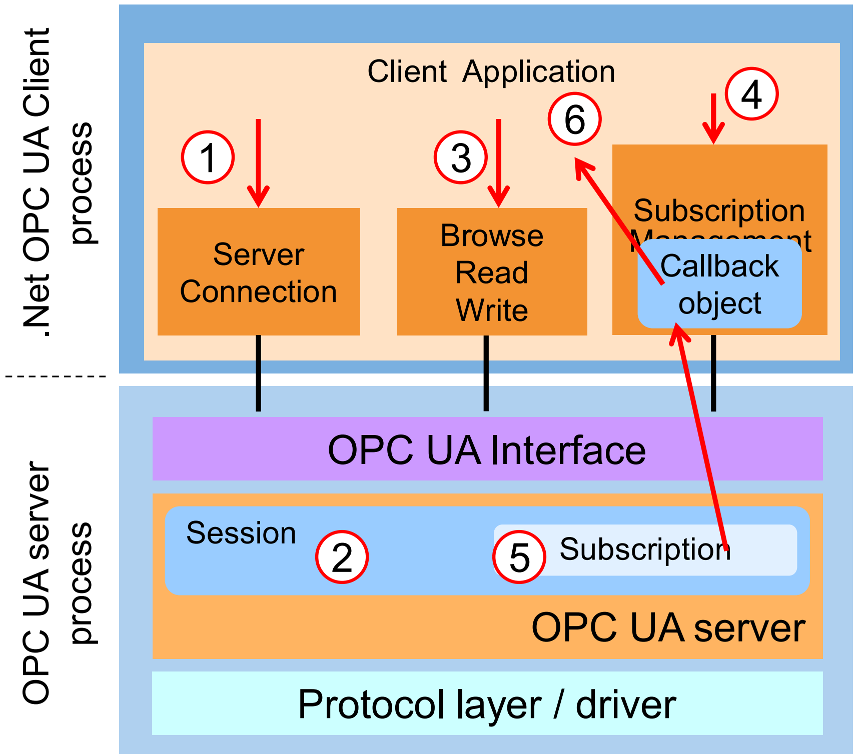


Table 4-2

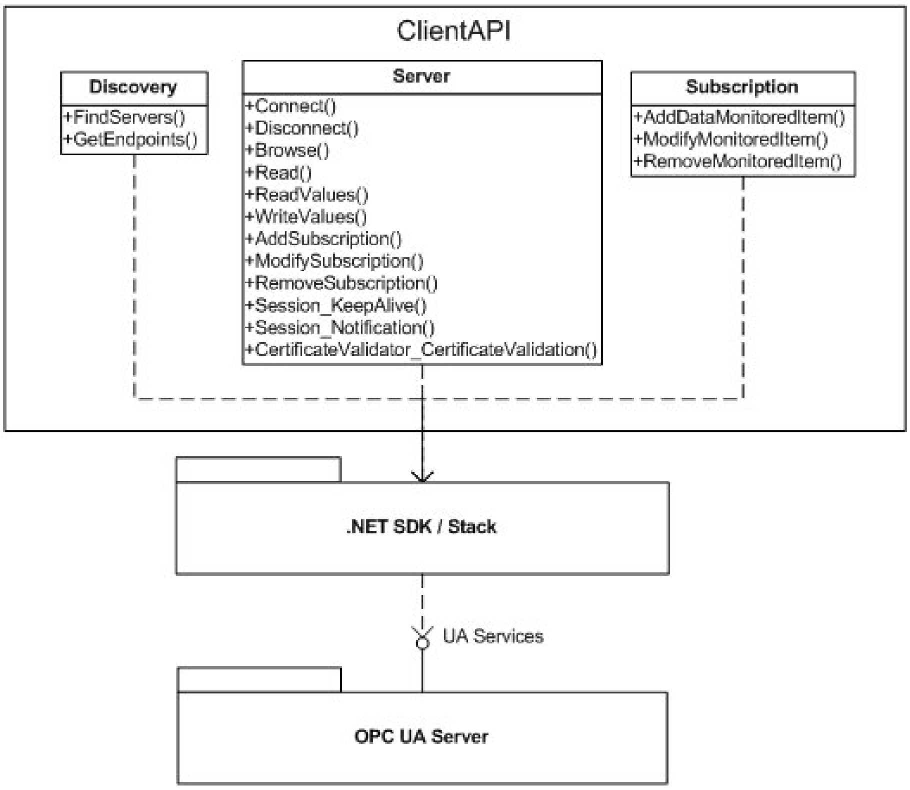
No.	Description
1	When establishing the connection between user interface and the OPC UA server, a client API object is generated on the client side. This object manages the connection with the server (2). It furthermore provides all OPC UA services with the exception of services that are related to a subscription.
2	The session object is generated in the server via the OPC UA interface.
3	When establishing a connection between user interface and the OPC UA server, the first level of the address space of the OPC UA servers is also represented. In the process, the browse service of the OPC UA interface is used. If a node is selected in the tree view of the browse control in the client, the attribute values of the node are displayed in another window via read services.
4	When registering tags to monitor value changes, a subscription object is created which supplies all OPC UA services which relating to a subscription.
5	A subscription object which manages all subscription relevant settings is generated in the server via the OPC UA interface.
6	To be able to receive value changes from the server, a callback connection is established. A SubscriptionCallback object is created in the client and connected to the subscription in the server. If changes are sent from the server to the client, it enters the changes into the monitoring window.

4.1 OPC UA Client API

The class diagram in Figure 4-3 shows the classes of the OPC UA ClientAPI. These classes encapsulate the accesses to the OPC UA server in a simple and reusable .NET API.

The classes are summarized in the .NET assembly Siemens.OpcUA.dll. It has dependencies to the .NET Client SDK Assembly Opc.Ua.Client.dll and to the .NET Stack Assembly Opc.Ua.Core.dll.

Figure 4-3



Class discovery

The **Discovery** wrapper class described in the table below encapsulates the required methods for the server discovery.

The class is implemented in the ClientDiscovery.cs file in the ClientApi project.

Table 4-3

Method	Functionality
FindServers	Detects the OPC UA servers on a computer.
GetEndpoints	Detects the available endpoints for one or several servers.

Server class

The **Server** wrapper class described in the table below encapsulates the functionality for the access to the OPC UA server. Moreover, it simplifies the use of those OPC UA services which are required by the client application, with the exception of the services for the subscription.

The class is implemented in the ClientAPI.cs file in the ClientApi project.

Table 4-4

Method	Functionality
Connect	Creates a secure channel as communication channel and a session in the OPC UA server.
Disconnect	Deletes the session in the server and disconnects the secure channel connection.
Browse	Supplies the list of nodes which are obtainable from a transferred start node via a reference. The list of results can be influenced via filter settings.
Read	Supplies the values to a list of attributes of a node.
ReadValues	Supplies the values of the attribute value of a list of nodes.
WriteValues	Writes the value of the attribute value of one or several tags.
AddSubscription	Creates a subscription and links it to the session.
ModifySubscription	Changes the settings of a subscription.
RemoveSubscription	Removes an existing subscription.
Session_KeepAlive	Keep-alive Callback.
Session_Notification	Called when the OPC UA server sends a reply (publish message).
CertificateValidator_ CertificateValidation	Called when the certificate of the servers is considered untrusted.

Subscription class

The **Subscription** wrapper class described in the table below, encapsulates the use of a subscription for the value exchange between server and client.

The class is implemented in the ClientSubscription.cs file in the ClientApi project.

Table 4-5

Method	Functionality
AddDataMonitoredItem	Creates a monitored item to monitor value changes and link them with the subscription.
ModifyMonitoredItem	Changes the settings of the monitored item.
RemoveMonitoredItem	Removes a monitored item from the subscription.

4.2 Simple OPC UA Client

The simple client provides a simple example for the use of the Client API. The most important function such as connect, disconnect, read, write and monitoring of data is displayed in a file or class with a dialog. The code for the example can be found in the SimpleClient project in the MainForm.cs file.

User interface of the simple example

The user interface is operated via buttons for the individual functions.

Figure 4-4

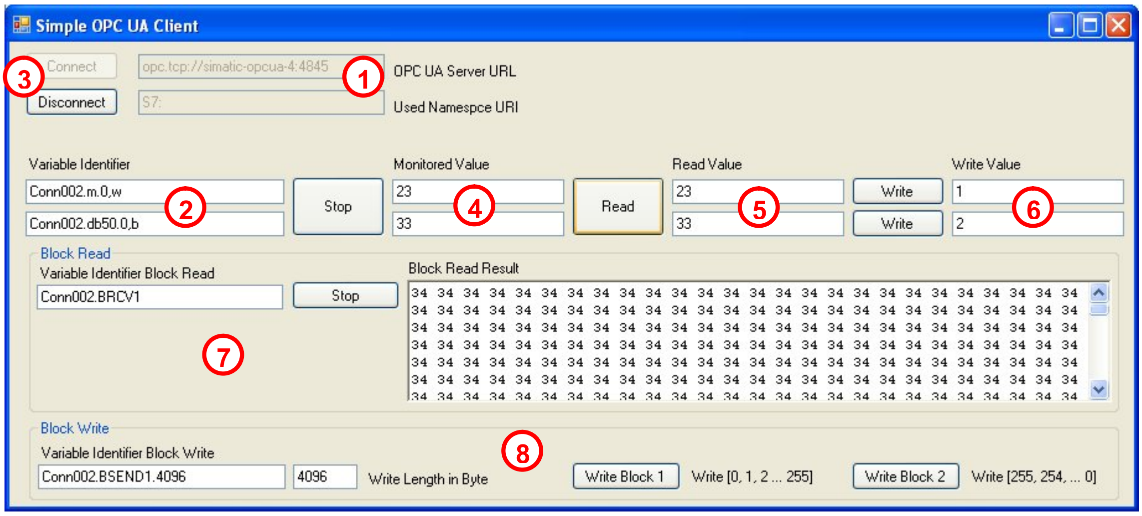


Table 4-6

No.	Description
1.	The server URL can be specified in the text box for the Server URL. For the SIMATIC NET OPC Server this is composed of **opc.tcp://<computername>:4845**. In the Namespace URI text box the namespace used is indicated. This is **S7:** for direct addressing, **S7COM:** for direct addressing via the OPC DA compatible Syntax and **SYM:** for symbolic addressing.
2.	In the text boxes for the Tag Identifier the identification code of the NodeID is indicated. For namespace S7:, for example, it is composed of**<S7connection>.<dataarea>.<offset>,<datatype>** The NodeID for reading and writing is made up of identification and namespace index. The namespace index results from the position of the namespaces in the namespace table of the server. This table can be read with Read from the server.
3.	The connection to the OPC UA server can be established or terminated via the Connect and Disconnect buttons. The connection is only established without security. Secure connection establishment is explained in the next example.
4.	A subscription is created via the Monitor button and two Monitored Items are created in the Subscription with both NodeIDs. The data changes are displayed in the text boxes next to the button. Errors are each shown instead of values.
5.	The Read button reads the values (Attribute value) of both tags with the specified NodeIDs and displays them in the text boxes next to the button.
6.	The Write button writes the value from the text box next to the button onto the tag identified by the NodeID. In order to write, “read” has to be called first since the text from the text box has to be converted in the data type suitable for the tag. The conversion is on the basis of the data type which is supplied at “read”.

No.	Description
7.	In the “Block Read” group, data can be received which is actively sent by the S7 with the BSEND block service. This can be, for example, used for the sending of result data from the S7 to a PC application.
8.	In the “Block Write” group, data blocks can be sent to the S7 which are there received by the BRECV block service. Two blocks with different contents can be sent. This can be used, for example, for the download of recipe data for the S7.

Functions of the simple example

The functions can be found in the MainForm class in the MainForm.cs file. Simple error handling is implemented in the functions. If an exception occurs for OPC UA calls, a dialog with the error message will appear. If an error occurs for one or several tags with the tag related calls, the error is displayed in the respective text boxes.

Table 4-7

Function	Description
btnConnect_Click	In this function the connection to the OPC UA server is established via the Server::Connect() function of the Client API. The URL string from the corresponding text box is transferred. If the connection has been successfully established, the namespace table will be read via the Server::ReadValues() function. In the returned table the namespace from the Namespace URI text box is searched. The index in the table is stored in a class tag.
btnDisconnect_Click	In this function the connection to the OPC UA server is established via the Server::Disconnect() function of the Client API.
btnRead_Click	In this function, first of all, the two NodeIDs from the namespace index and the identifier texts are formed. Subsequently, both values are read via the Server::ReadValues() function. The result is written into the respective text box. The result can either be the written value or an error code.
btnMonitor_Click	If no subscription has been created, the subscription will be created first in this function with Server::AddSubscription(). Subsequently, the two NodeIDs from the namespace index and the identifier texts are formed. Afterwards, both monitored items are created with Subscription::AddDataMonitoredItem(). The respective text box for the display of the values is transferred to Client Handle. If a subscription has already been created it is deleted by Server::RemoveSubscription().
ClientApi_ValueChanged	The function is indicated as callback function at Subscription::AddDataMonitoredItem(). In the function it is first of all checked whether the call arrives in the main thread of the dialog. If this is not the case, the call is transferred to the main thread of the dialog via BeginInvoke. Otherwise access to the dialog is not possible. Afterwards it is checked whether the Client Handle is a text box. If not, this is a callback for the Read block. If it is the value of a normal tag, the client handle is cast back to the text box and after checking the status, either the value or an error code is entered in the text box. However, if it is the value of a block tag, the byte array is extracted and displayed as a sequence of HEX values for the individual elements of the byte arrays.
btnWrite1_Click	The function composes the NodeID for tag 1 and calls the writeNewValue function with the NodeID, the new value as text from the text box and with the value last read.

Function	Description
btnWrite2_Click	The function composes the NodeID for tag 2 and calls the writeNewValue function by means of the NodeID, the new value as text from the text box and with the value last read.
writeNewValue	In this function the new value from a text is first of all converted to the data type which was delivered during the last read performed. Afterwards the value is written via the Server::WriteValues() function and the result is checked.
btnMonitorBlock_Click	If no block subscription has been created, the subscription is created first in this function with Server::AddSubscription(). Subsequently, the NodeID is formed from the namespace index and the identifier text. Afterwards the monitored item is created with Subscription::AddDataMonitoredItem(). If a subscription has already been created it is deleted by Server::RemoveSubscription().
btnWriteBlock1_Click	The function composes the NodeID for the block tag and simulates data record 1. The simulation creates a byte array with the length indicated in the user interface and fills the values with a tag value which starts at 0 and is incremented after each assignment. Afterwards the writeNewBlockValue function is called with the NodeID and the new value.
btnWriteBlock2_Click	The function is identical to btnWriteBlock1_Click; however, the tag value assigned here starts with 255 and is decremented after each assignment.
writeNewBlockValue	In this function the value is written via the Server::WriteValues() function and then the result is checked.

4.3 Comfortable OPC UA Client

4.3.1 User interface

The figure and table below describe the user interface of the generic OPC UA client example with which the information of the namespace of an OPC UA server can be conveniently accessed.

Figure 4-5

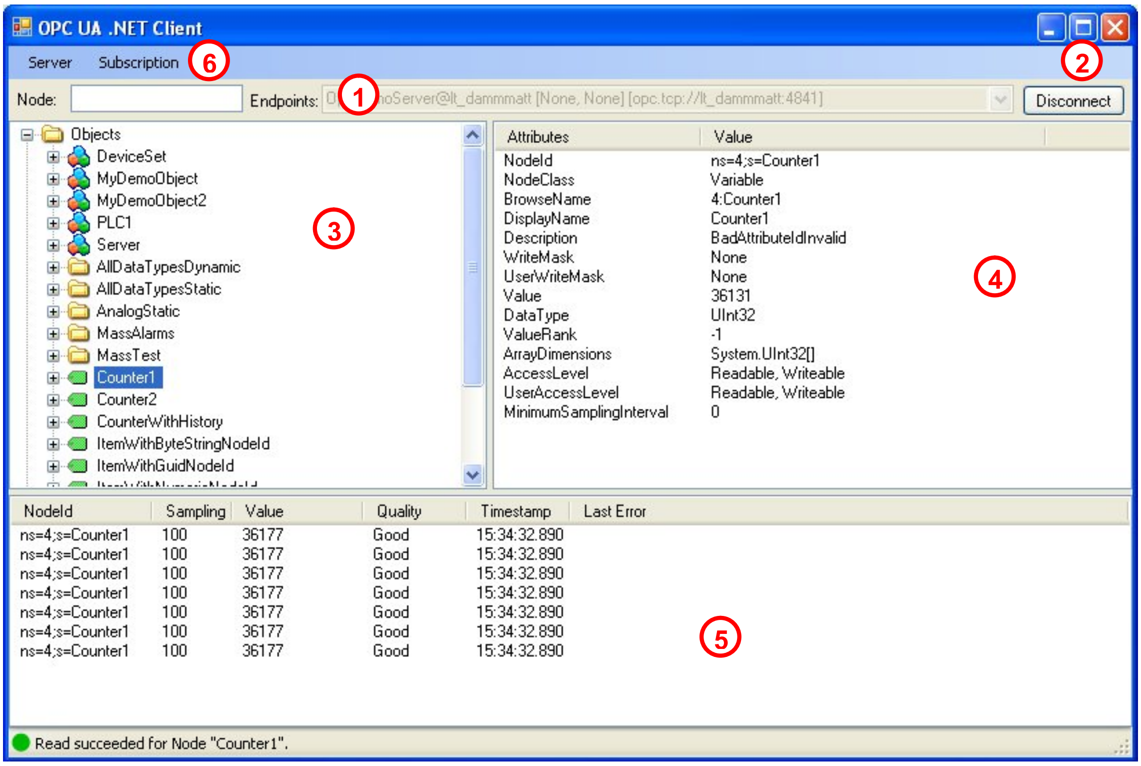


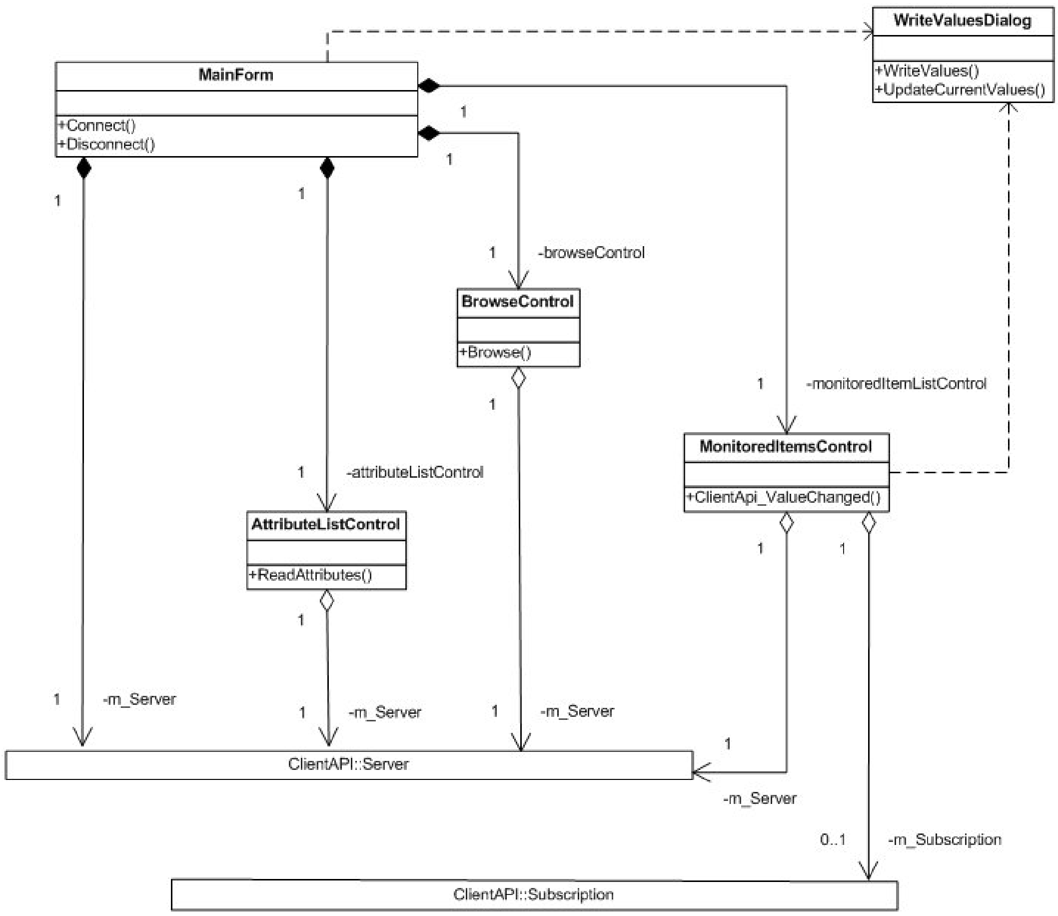
Table 4-8

No.	Description
1.	The server can be selected via the endpoints selection list. For this purpose, the list of servers and the endpoints are detected by the discovery server. The computer, from which the discovery server is to be prompted, can be entered in the node text field. If the field is empty, the local discovery server is addressed.
2.	The connection to the server can be established or terminated via the Connect button.
3.	In Browse Control the entire address space of the connected server is shown in a hierarchical tree view. Only hierarchical references are displayed.
4.	For the selected nodes the attributes are read in Browse Control and they are displayed in this control.
5.	With drag&drop the tags can be dragged from Browse Control to the monitoring window. For the tag, the NodeID, the sampling interval, the value, the time stamp and the status code is displayed.
6.	The properties of the subscription and monitored items can be changed via the context menu in the monitoring window or via the application menu. This is how e.g. the sampling interval can be changed. The dialog for writing can also be opened. Doing this, accepts those tags in the dialog which have been marked in the monitoring window.

4.3.2 Class diagram

The following class diagram shows the classes of the OPC UA sample client. These classes realize the functionality of the user interface and use the classes of the UA client API. The individual classes are explained in detail on the following pages.

Figure 4-6



MainForm class

The **MainForum** class described in the following table, implements the functionality of the main dialog of the client application.

The class name corresponds to the file name in the UAClient project.

Table 4-9

Method	Functionality
MainForm	Constructor of the class.
Connect	Implements the functionality to establish a connection with the server and initializes the Browse Control.
Disconnect	Terminates the connection to the server.

BrowseControl class

The **BrowseControl** class described in the table below, implements the functionality for displaying the address space of the server.

The class name corresponds to the file name in the UAClient project.

Table 4-10

Method	Functionality
BrowseControl	Constructor of the class.
Browse	Navigates through the tree view, starting from the selected node.

AttributeListControl class

The **AttributeListControl** class described in the table below, implements the value display of all attributes of a node in the address space.

The class name corresponds to the file name in the UAClient project.

Table 4-11

Method	Functionality
AttributeListControl	Constructor of the class.
ReadAttributes	Reads the attribute values of a node and displays them in a list.

MonitoredItemsControl class

The **MonitoredItemsControl** class described in the table below implements the creation of a subscription and of MonitoredItems and the display of values or the status of the Monitored Items.

The class name corresponds to the file name in the UAClient project.

Table 4-12

Method	Functionality
MonitoredItemsControl	Constructor of the class.
ClientApi_ValueChanged	Callback function for the Data Change Events of the client API. The new value, time stamp and status are entered in the respective line. For this purpose the Client Handle is casted on the object which represents the line.
MonitoredItems_DragDrop	In this function a NodeID which is dragged to the control via drag&drop, is created as Data Monitored Item. If no subscription exists yet, a subscription is created first.

WriteValuesDialog class

The **WriteValuesDialog** class described in the table below implements the display and the change of the current values of one or of several tags.

The class name corresponds to the file name in the UAClient project.

Table 4-13

Method	Functionality
WriteValuesDialog	Constructor of the class.
WriteValues	Writes the value of one or several tags.
UpdateCurrentValues	Updates the values in the dialog by calling Read.

Using the client API in the example

The table below lists the files and functions in which the client API is used.

Table 4-10

Client API	Used in
Class discovery	
FindServers	MainForm.cs in the UrlCB_DropDown function
GetEndpoints	MainForm.cs in the UrlCB_DropDown function
Server class	
Connect	MainForm.cs in the Connect function
Disconnect	MainForm.cs in the Disconnect function
Browse	BrowseControl.cs in the Browse function
Read	AttributeListControl.cs in the ReadAttributes function
ReadValues	WriteValuesDialog.cs in the UpdateCurrentValues function
WriteValues	WriteValuesDialog.cs in the WriteValues function
AddSubscription	MonitoredItemsControl.cs in the MonitoredItems_DragDrop function
ModifySubscription	MonitoredItemsControl.cs in the MonitoringMenu_PublishingInterval_Click function
RemoveSubscription	MonitoredItemsControl.cs in the RemoveSubscription function
Subscription class	
AddDataMonitoredItem	MonitoredItemsControl.cs in the MonitoredItems_DragDrop function
ModifyMonitoredItem	MonitoredItemsControl.cs in the MonitoringMenu_SamplingInterval_Click function
RemoveMonitoredItem	MonitoredItemsControl.cs in the MonitoringMenu_RemoveItems_Click function

4.3.3 Sequence diagrams

Establishing and terminating the connection to the OPC UA server - User interface

The following sequence diagram shows the procedures which are necessary to establish the connection to the OPC UA server. By clicking **Combobox Endpoints** the user selects an available endpoint first.

Establishing a connection can be started via the **Connect button** in the user interface or via the **Server menu**. Once the connection was successfully established the label **Disconnect** appears on the **Connect button**. The sequence diagram also shows the processes which are triggered through the “Disconnect Server” action via the **Disconnect button**.

Figure 4-7

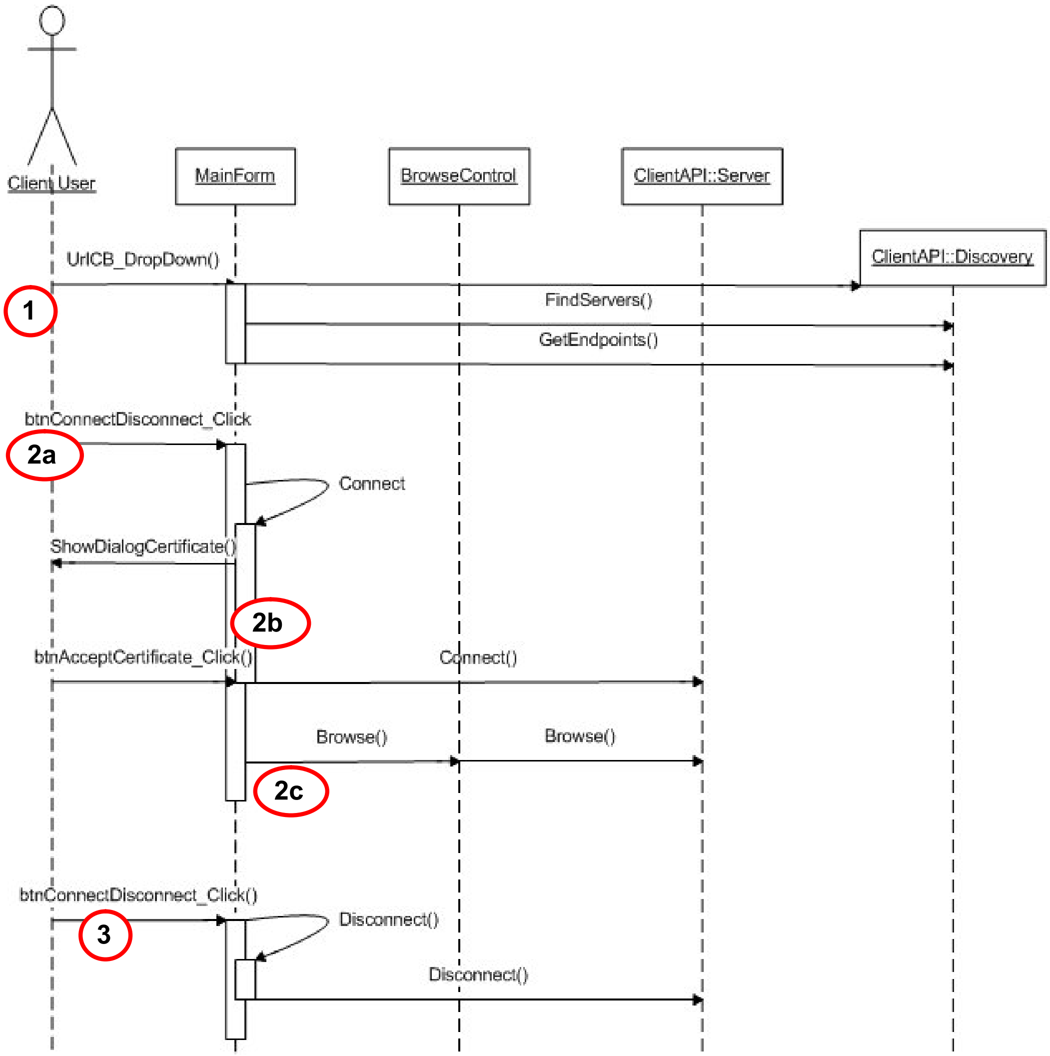


Table 4-14

No.	Description
1	When opening the selection list a ClientAPI::Discovery object is created and the methods **FindServers** and **GetEndpoints** are called.
2	The **ConnectDisconnect_Click** method is called at the MainForm object via the “Connect” user action. With this method, the actions to establish a connection to the server are performed. In the first step, the ClientAPI::Server object and there, the **Connect** method is called (2a). This ensures that the connection to the OPC server, defined by the URL, is established. If a certificate is considered untrusted, the user can still accept it and establishing the connection is continued (2b). In case of an existing connection, the **Browse** method is called in a second step on the ClientAPI::Server object (2c). This displays the first level of the address space of the OPC UA servers in BrowseControl.
3	Through the “Disconnect Server” user action, the **ConnectDisconnect_Click** method is called on the MainForm object. In the case of an already existing connection, this method calls the **Disconnect** method at the ClientAPI::Server object. This terminates the connection with the OPC UA server.

Establishing a connection to the OPC server and closing it – Client API

The sequence diagram below shows the processes during establishing a connection to an OPC UA server in the context of the client API and when closing it.

Figure 4-8

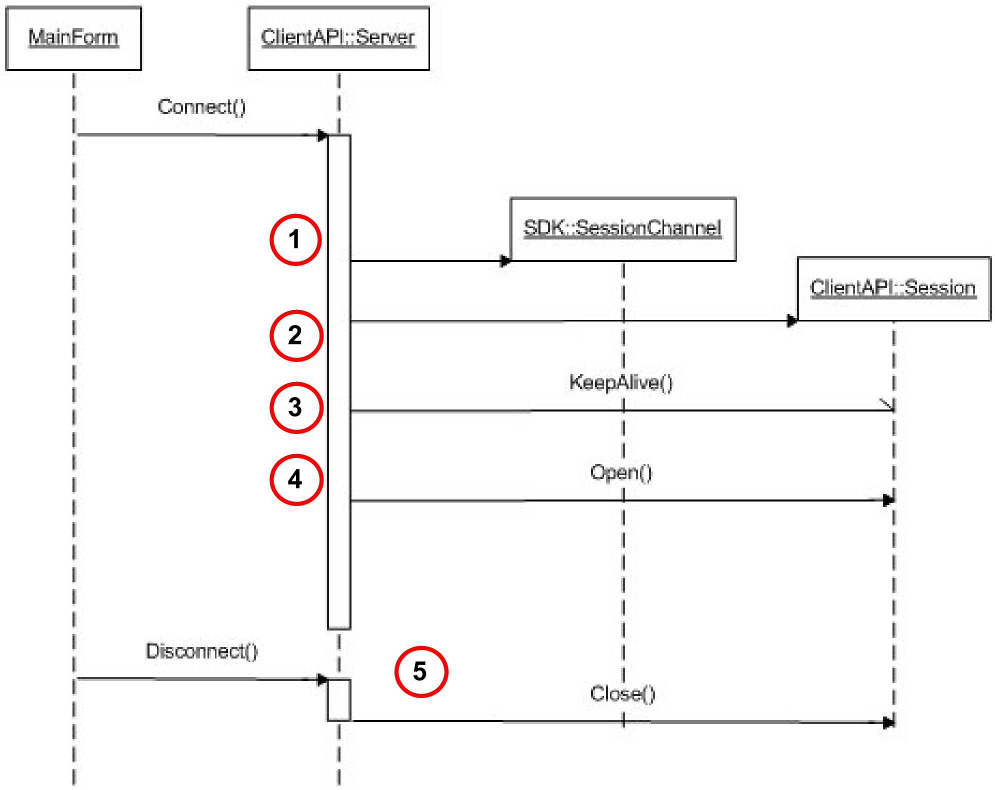


Table 4-15

No.	Description
1	The **Connect** action creates an SDK::SessionChannel object for the establishment of a secure connection to the server.
2	Subsequently a ClientAPI::Session object is generated which encapsulates the channel to the server.
3	In the next step the ClientAPI::Session object registers a **KeepAlive Callback** at the OPC UA server.
4	Finally, the **Open** call establishes the actual connection between client and server.
5	Within the framework of the **Disconnect** action, **Close** is called on the ClientAPI::Session object.

Reading attributes

The sequence diagram below shows the processes during the reading of attribute values.

Figure 4-9

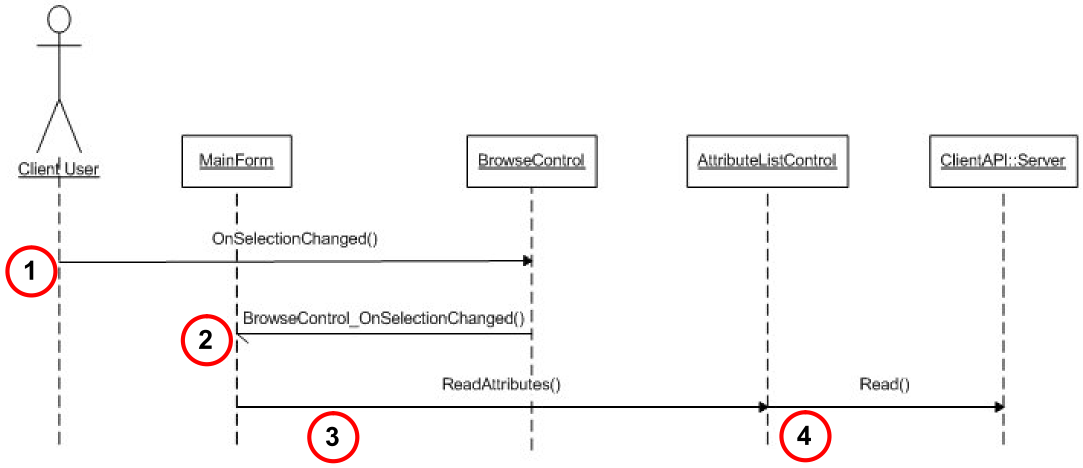


Table 4-16

No.	Description
1	If a node is selected in the BrowseControls tree, an OnSelectionChanged Event is triggered.
2	The event is passed on to the MainForm.
3	The **ReadAttributes** method of the AttributeListControl is called by the event.
4	There, Read on the ClientAPI::Server object is called and the values are displayed.

Writing of values

The sequence diagram below shows the processes during the writing of tag values.

Figure 4-10

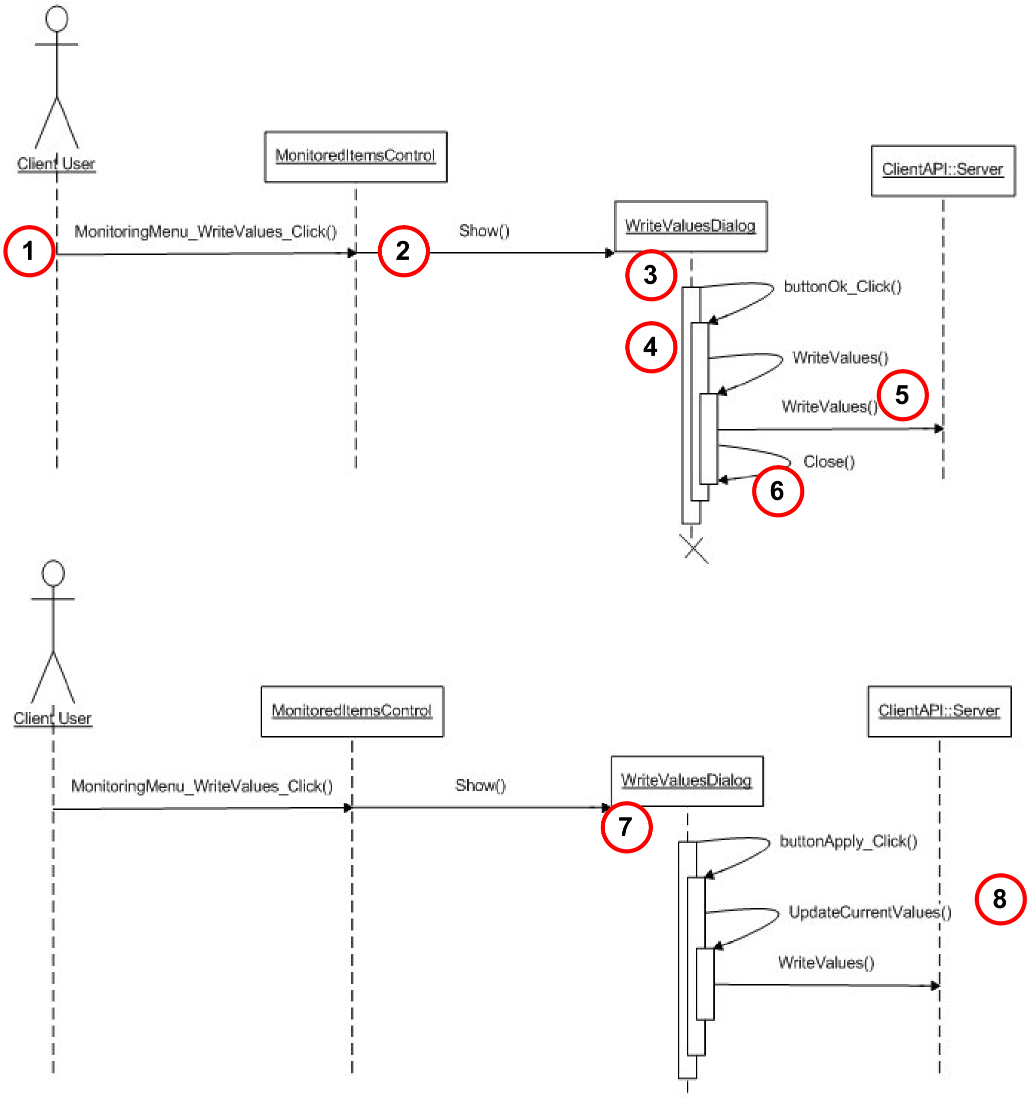


Table 4-17

No.	Description
1+2	When clicking the “WriteValues” entry in the **MonitoredItemsControl** (1) context menu, the “Write Values” dialog is opened (2). This is where the current values of the tags previously selected in **MonitoredItemsControl** will appear.
3+4	If the user clicks the **Ok button** (**buttonOk_Click** action) after entering the new values, **WriteValues** is called on the WriteValuesDialog object.
5	In turn **WriteValues** is called on the ClientAPI::Server object.
6	The dialog is eventually closed by calling **Close** on the WriteValuesDialog object.
7	Alternatively, the user can click the **Apply button** (**buttonApply_Click** action).
8	The UpdateCurrentValues is then called and the dialog is not closed.

Registering tags for monitoring

The sequence diagram below shows the processes when registering tags for the monitoring of value changes.

Figure 4-11

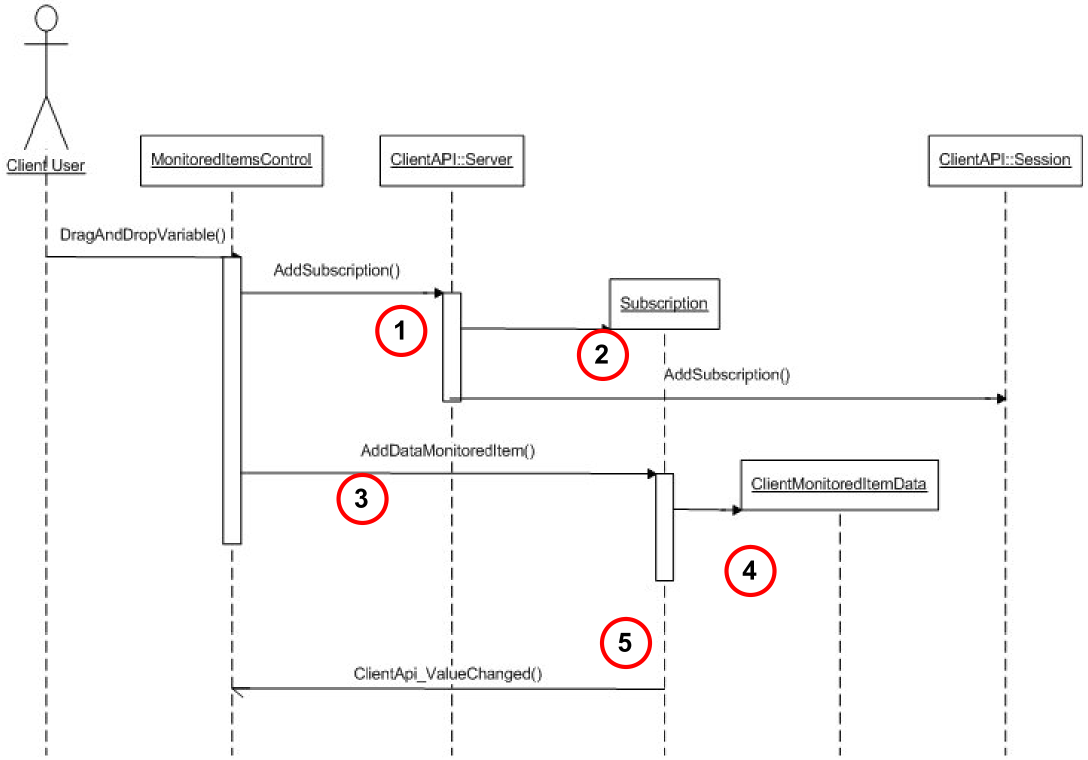


Table 4-18

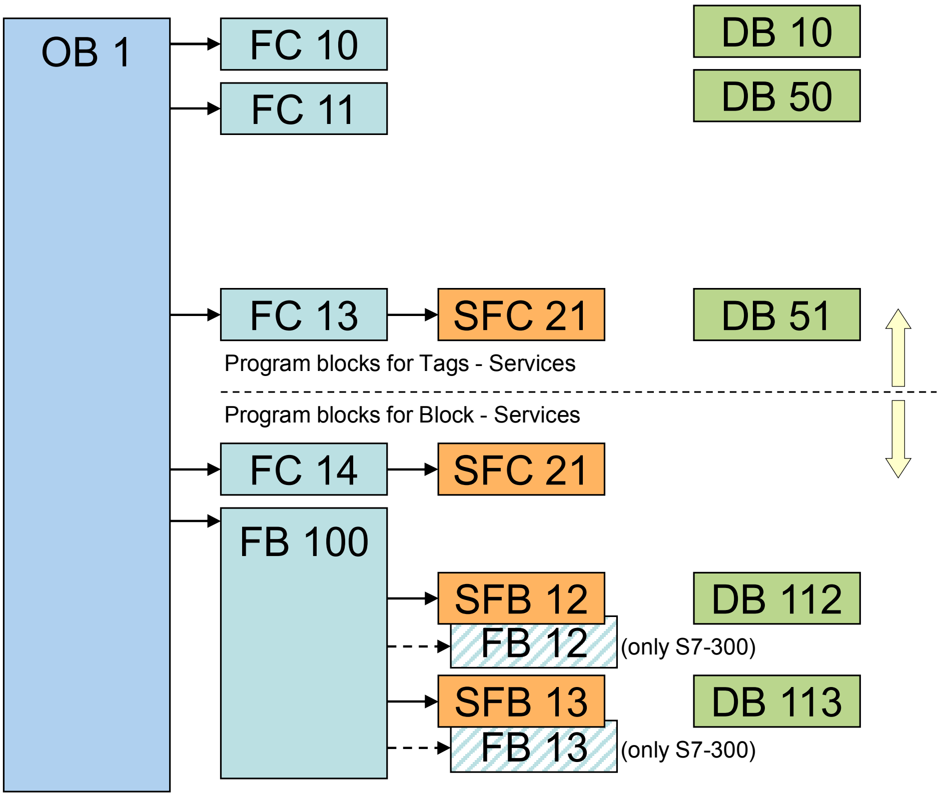
No.	Description
1	First of all, a drag&drop action with the MonitoredItemsControl as target checks whether a subscription exists. If this is not the case, **AddSubscription** is called on the client API.
2	In the process, a subscription object is generated and subsequently the **AddSubscription** is called on the ClientAPI::Session object.
3	In the next step **AddDataMonitoredItem** is called on the subscription object.
4	This is where a ClientMonitoredItemData object is generated.
5	From now on, value changes are received by the server and passed on to MonitoredItemsControl, to be displayed there.

4.4 S7 program

Overview

The S7 program is essentially divided in two parts. First of all, the dynamic data for the tag services is simulated; afterwards the send data is simulated and the block services are called in FB 100.

Figure 4-12



NOTICE	**The S7 program cannot be used together with optimized blocks due to the used absolute address!**

Simulation of dynamic data

The table below gives a brief overview of program parts and their function for data simulation. Details were deliberately excluded here; further comments are contained in the STL code.

Table 4-19

Block	Remarks
OB1	Cyclic Main initially, a variable timer is set here whose interval is used to call the other program functions. The rate of change of data can be set via DB10 byte 0.
FC10	ChangeDateAndTime increments a date value as well as the time in DB51.
FC11	ChangeSimpleTypes increments the data of DB51. 8bit types are incremented with +1, 16bit types with +100 and 32bit types with +1000.
FC13	ChangeString increments a string of length 10 in DB51.
DB10	SimulationConfiguration contains global tags for the configuration of data simulation.
DB50	StaticDataTypes contains simple data types which were given symbolic names. The values are pre-initialized with maximum end value range.
DB51	DynamicDataTypes contains simple data types which were given symbolic names. The values are incremented with the functions FC10 to FC13 according to their value ranges.
SFC21	FILL auxiliary function to fill data areas with values, storage initialization.

Block-oriented data

The S7 communication for the S7-300 is realized via FBs (loadable function blocks) and not via SFBs (integrated system function blocks) as is the case for the S7-400. However, if you call a SFB instead of an FB in the S7 program of the S7-300, the block delivers an "ERROR" and displays "STATUS = 27". This status indicates that the function block for the S7 communication does not exist on the S7-300. The communication FBs for S7-300 are located in the Instructions tab of the Task Card at Communication > S7 communication.

The table below gives a brief overview of the program parts and their function regarding BSEND/BRCV. Details were deliberately excluded here; further comments are contained in the STL code.

Table 4-20

Block	Remarks
OB1	Cyclic Main Call of the send block (SFB12) for BSEND and of the receive block (SFB13) for BRCV via function block 100. For S7-300, FB12 and FB13 from the library are used since S7-300 communicates via loadable FBs and not via system functions.
FC14	ChangeSendData Increments a byte which will then be copied to the entire send buffer
FB100 + DB100 (Instance DB)	InvokeBSENDandBRCV Calls the real communication blocks and supplies its parameters.
DB112	SendData Data block with 4096 bytes length which will be transferred to the send block.
DB113	ReceiveData Data block with 4096 bytes length which will be transferred to the receive block.
SFB12 + DB12 (Instance DB)	BSEND The send block transfers the send buffer to the communication processor (CP), which will then send it according to RID and connectionID to the communication partner.
FB12	BSEND (only S7-300)
SFB13 + DB13 (Instance DB)	BRCV The receive block picks up the data package last received from the communication processor (CP) according to RID and connectionID and files it in the receive buffer.
FB13	BRCV (only S7-300)
SFC21	FILL Auxiliary function to fill data areas with values, storage initialization.

The program logic sends or receives data blocks and supplies the respective parameters via the FB100. When larger data packages are sent, a multiple call of the BSEND or BRCV block is necessary. This multiple call is performed by FB100.

Note If the send and receive block is called cyclically, a high communication load may be generated and there is furthermore the danger that the data buffers are overwritten before they are processed by the counterpart. This is why the program logic should contain a flow control to ensure data consistency. For this purpose the parameters DONE (ready) and NDR (new data received) are to be used.

In order to be able to exchange data between two S7-300 stations via a configured S7 connection, communication functions need to be called in the S7 program. The FB12 "BSEND" block is used for sending data and the FB13 "BRCV" block for receiving data.

Here, the S7 connection has to be bilaterally configured since the S7 communication via FB12 "BSEND" and FB13 "BRCV" is based on the client-client principle.

5 Configuration and Settings

5.1 Configuring the SIMATIC S7 stations

General

It is assumed that all hardware and software components have been successfully installed and cabled.

Here, you will find an overview of the IP addresses used in the sample project:

Table 5-1

Address	Station	Remarks
192.168.172.1	PC OPC Server station	
192.168.172.2	S7-300, CP 343-1	optional
192.168.172.3	S7-400, CP 443-1 EX/GX 20	
192.168.172.4	S7-1500, CPU 1516-3 PN/DP	optional
192.168.172.5	Laptop OPC Client station	

Specific characteristic of S7-1200 as of FW V4.0 and S7-1500

When using optimized data blocks in an S7-1200 station as of FW V4.0 or an S7-1500 station, you need to use OPC Server V12 in order to access the data blocks of the controller.

STEP 7 then automatically creates “optimized S7 connections”, since the standard access to optimized data blocks is not possible via OPC.

The OPC client must then access the SimaticNET.S7OPT server via port 4850.

If you do not wish to use optimized S7 connections, then the following conditions must be fulfilled:

1. Using the OPC server in version 8.2 at the most.
2. Using non-optimized data blocks.

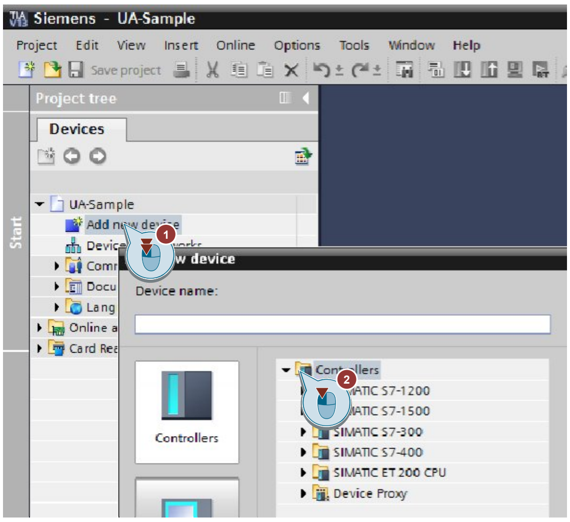
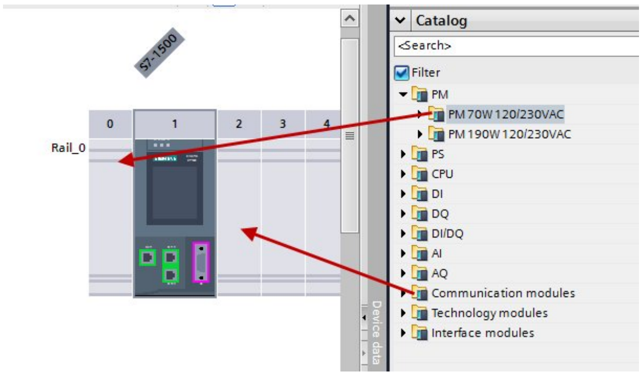
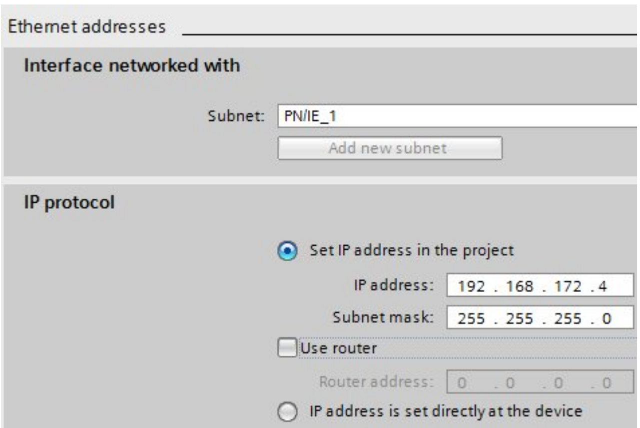
Instruction

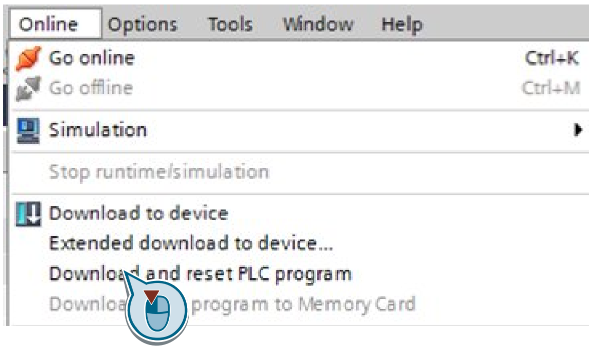
The following configuration steps of the SIMATIC S7 stations exemplify the procedure. Adjust the configuration independently, as required for your hardware.

Note After saving and compiling, all configuration information is overwritten.

The following table shows the configuration and settings of the SIMATIC S7 stations.

Table 5-2

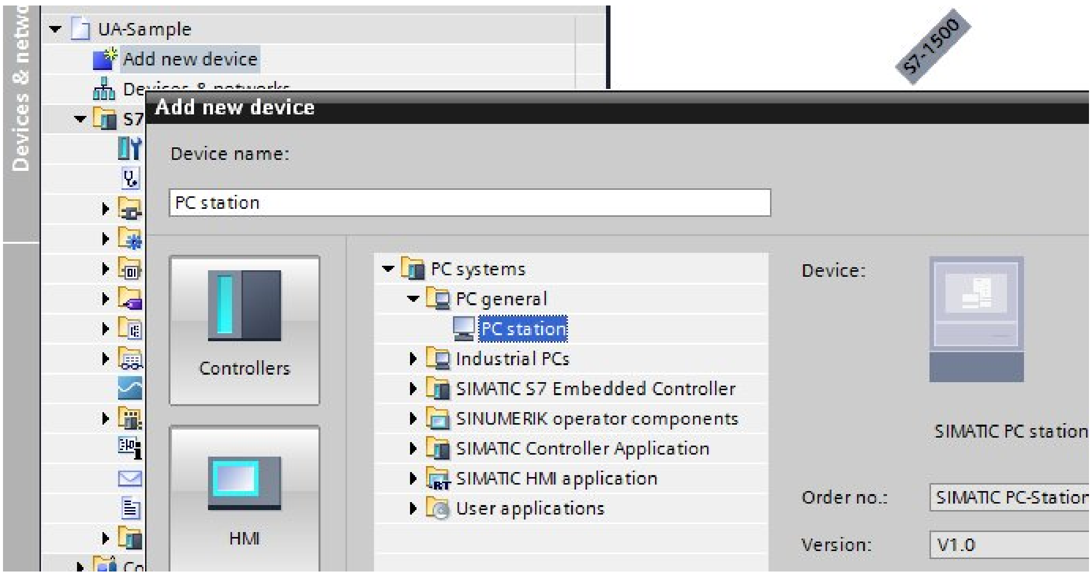
No.	Action	Remarks
1.	Open STEP 7 V13 and create a new project.	Here, the name “UA-Sample” was used.
2.	Insert a SIMATIC 400 station and assign a name (here “S7-400”). Insert a SIMATIC 300 station and assign a name (here “S7-300”). Insert a SIMATIC 1500 station and assign a name (here “S7-1500”).	
3.	Open SIMATIC stations in the device view and insert possibly used CPs and other components according to the hardware actually inserted.	
4.	For changing the IP address in the inspector window you go to the Properties tab. There you select the Ethernet addresses. Set the IP address and subnet mask. Creating an Ethernet network. A MAC address is entered only if the station is to communicate via ISO transport layer 4.	
5.	Repeat the steps for all SIMATIC stations.	
6.	Create an S7 connection	The connection configuration is described together with the configuration of the PC station (in chapter 6.2)

No.	Action	Remarks
7.	If all of the settings have been confirmed, load the configuration into the S7 CPUs by restarting.	

5.2 Configuration of the OPC server station

The configuration and settings of the SIMATIC PC station are made via STEP 7 and are described step by step. Alternatively, a configuration can also be made using the NCM-PC software package. The procedure is identical, however, unilaterally configured connections are used.

Table 5-3

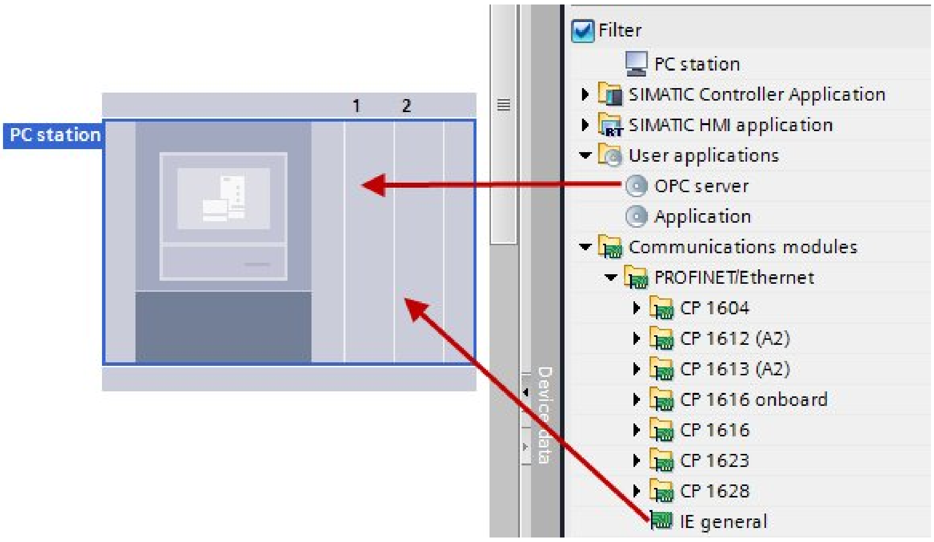
No.	Action Remarks
1.	Starting STEP 7 V13. Opening the previously created "UA-Sample" project.
2.	Inserting a SIMATIC PC station and assigning a name. The name of the PC station must be identical with the "Windows name" of the PC (see `Computer` → `Properties` → `Computer name`). 

3. Open the Device view of the PC station. There, you enter the Ethernet card and the OPC server V8.2.

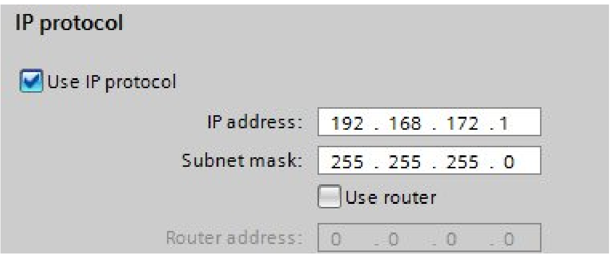
Note

If you wish to use an S7-1200 or an S7-1500 CPU with optimized data blocks, you need to use OPC Server V12.

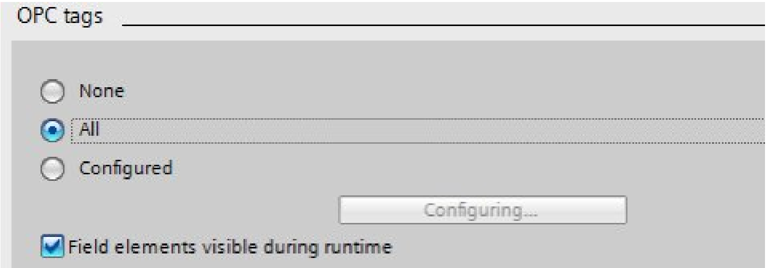
The slot must be identical with the index assigned in the configuration console, here index "2" for Ethernet card. The "OPC server" application was plugged into slot "1".



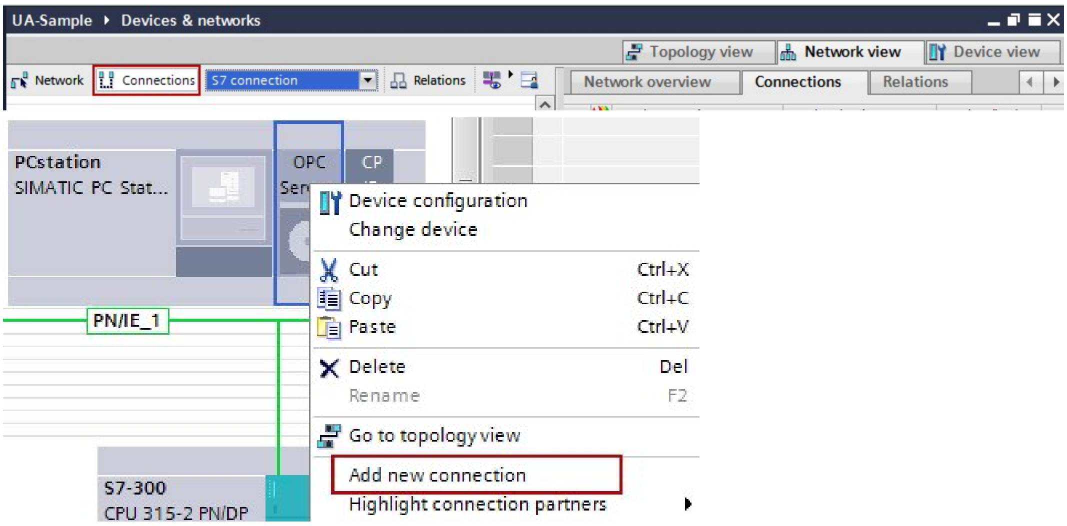
4. For the Ethernet card, an IP address (here "192.168.172.1") is assigned in the Properties and the card is connected with the Ethernet network.

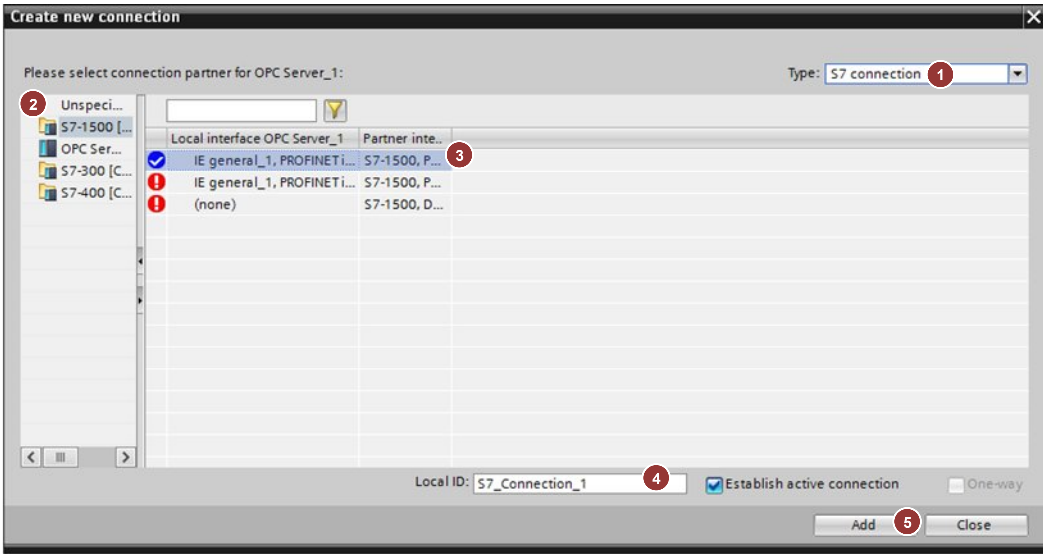


5. In the Properties pages of the OPC server, in the "S7" tab, the usage of symbolic addressing is activated.

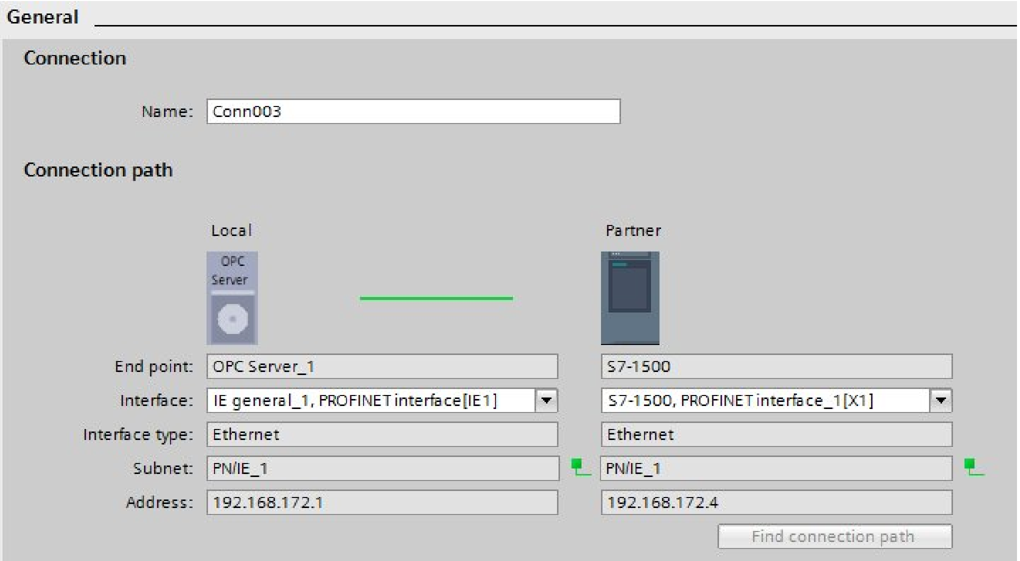


6.	In the Network view, three S7 connections from the OPC server to the S7 controllers are created. The connections are created via Ethernet.

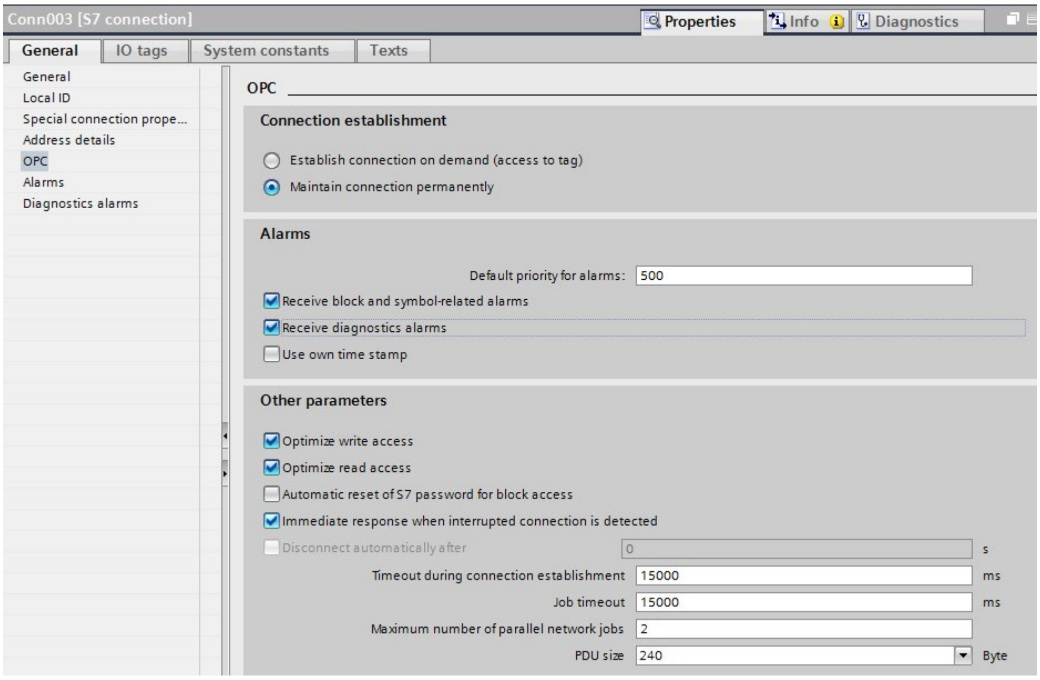




7. S7 connection via Ethernet:
After the S7 connections have been created, further settings are made by clicking on the connection and on the Properties tab in the inspector window.
After the connection path has been selected, the connection name can be changed (here “Conn001” for the connection to S7-300, “Conn002” for the connection to S7-400 and “Conn003”).
The connection partners and the parameters of the connection are displayed.



8. In the Properties of the S7 connections, the “OPC” menu is selected where connection-specific settings are made.
The connection establishment is set to permanent in order to maintain the connection even while no communication is taking place.
The connection is configured for the transmission of alarms and diagnostic events.
Furthermore, the immediate reaction to an interrupted connection is activated in order to avoid unnecessary wait times for timeouts.



9. After the connections have been created, the new configuration must be loaded into the stations.
The PC station can also be downloaded with the XDB-file, as described in chapter 6.

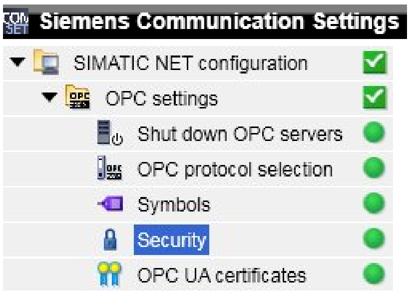
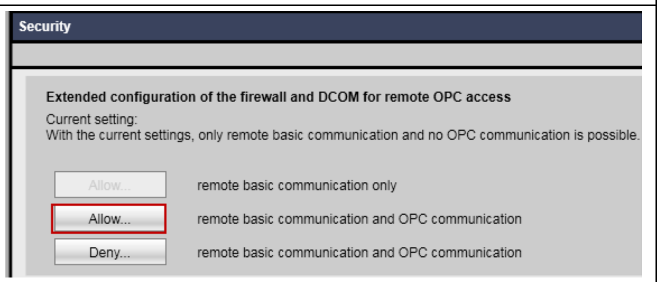
5.3 Configuration of the OPC UA Security

The security mechanisms of the OPC Unified Architecture are set at different levels. Encryption and signature of the transmission as well as authentication for the connection establishment can be set separately from each other. After installing the SIMATIC NET OPC UA Server, secure connections are principally possible. Apart from this encrypted communication, non-encrypted connection is also possible. The server accepts authentication with user and password or also the anonymous connection establishment. These settings are “insecure” and are only used to simplify commissioning. The OPC UA server can be configured in a way that it only accepts an encrypted transfer with user authentication.

5.3.1 OPC UA remote communication

All settings necessary on the server side, regarding the Windows firewall can simply be set and can also be removed again with the “Set PC Station” (Configuration Console) configuration tool.

Table 5-4

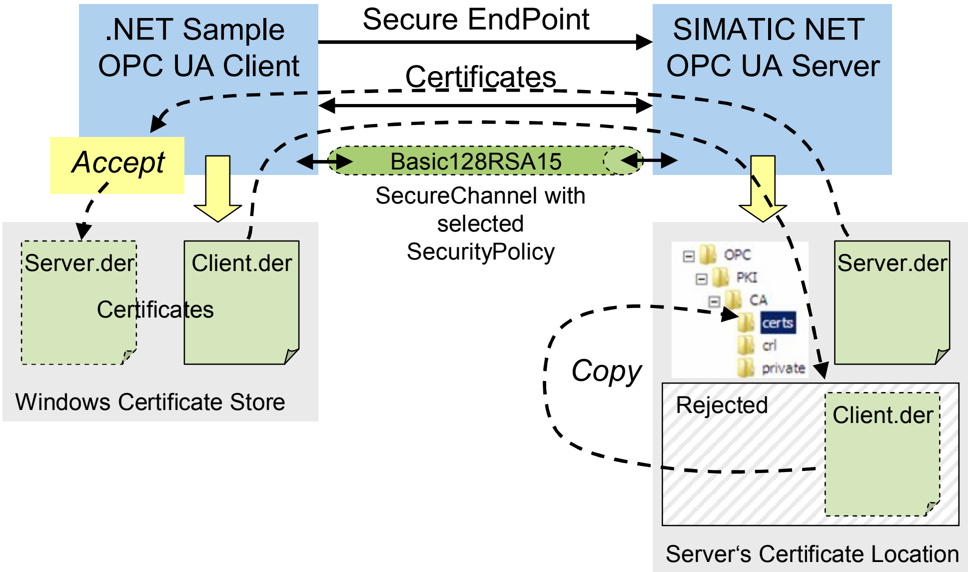
No.	Action	Remarks
1.	Start the configuration tool: “`Communication Settings`” and select the “SIMATIC Net configuration > OPC settings > `Security`” menu	
2.	With a single push of a button all necessary settings are performed in the firewall to permit remote communication or to block it again.	

Note Please note, that an exception for the application and for the TCP port (4845) also has to be entered at the firewall of the “OPC Client Station” PC.

5.3.2 Certificate storage

Certificates are exchanged when establishing a secure connection between OPC UA client and OPC UA server. Both applications have to check and accept the corresponding certificate of the counterpart so that a connection can be established.

Figure 5-1



The OPC UA client of this example uses the **Windows Certificate Store**. This is where the public certificate of the client is located.

At the first start, the example clients create a public certificate (UA Test Client) in this certificates administration The test client must be started with administration rights upon when first started.

When establishing an encrypted connection, server and client exchange their certificates. The client displays the certificate and the user has to trust this certificate. By accepting, the server certificate is stored in the Windows certificate store.

The OPC UA server uses its **own** certificate directory and is independent from the Windows certificate store. First of all, the OPC UA server will reject each certificate of an unknown client and will saves it in a “rejected-folder”, for reasons of security. An administrator has to copy this client certificate in the list of trusted certificates, just as with other server services, to allow the corresponding client access to the server. The location at which the OPC UA server stores and manages its own and the certificates of the OPC UA clients, is the data directory of the OPC UA server.

C:\Documents and settings\All Users\Application Data\Siemens\OPC\PKI\CA

This is where three subfolders are located with the following content:

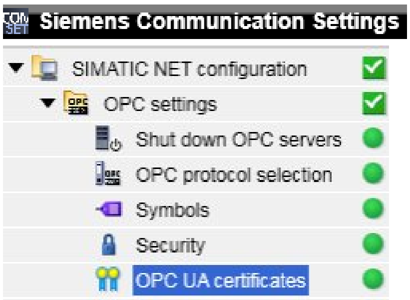
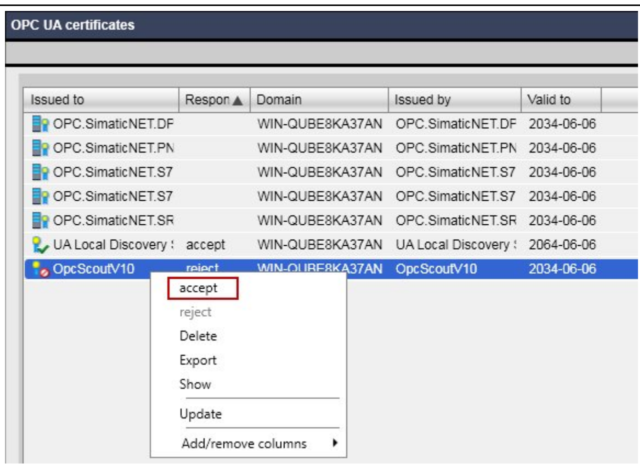
- **\certs**
  contains the public certificate of the server as well as all trusted certificates of clients. Public certificates from OPC UA clients have to be copied in this folder so that the server accepts them.
- **\crl**
  contains a file with a list of untrusted certificates, the so called "RevocationList"
- **\private**
  contains the private certificate of the OPC UA server. This certificate must not be accessible to anybody.

The server independently creates the \reject\ folder underneath of \certs\ and first of all saves all unknown client certificates in this "rejected-folder". By simply "moving" the file, the certificate can be made trusted.

Configuration server with SIMATIC NET DVD V8.2

From SIMATIC NET CD V8.0 on this is possible with the "Siemens Communication Settings" configuration tool.

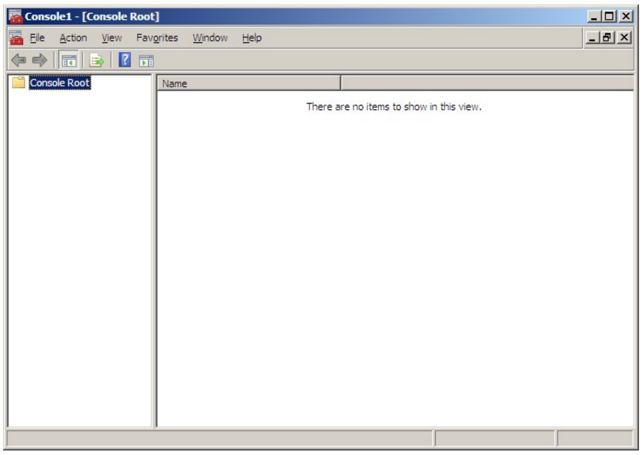
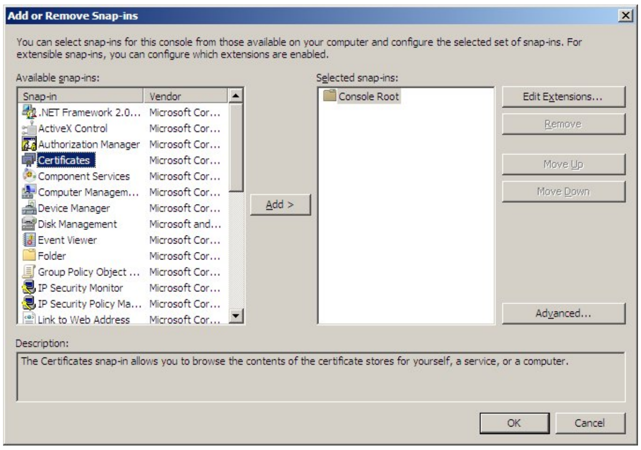
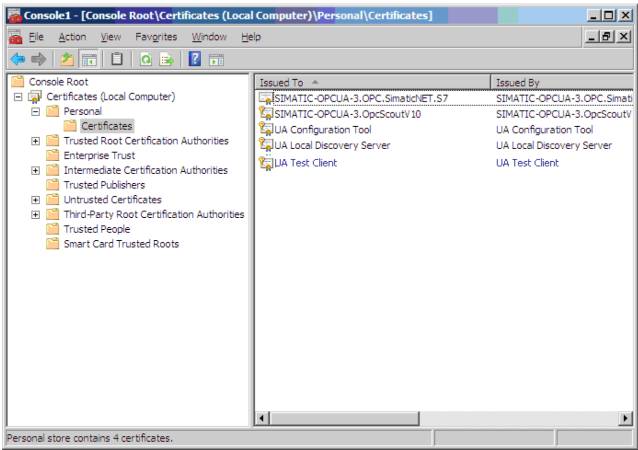
Table 5-5

No.	Action	Remarks
1.	Start the configuration tool: "`Siemens Communication Settings`" and select the "SIMATIC Net configuration > OPC settings > `OPC-UA-certificates`" menu	On the server PC. 
2.	All public UA certificates known to the server are found here. With a right-click on a selected certificate you can accept it.	

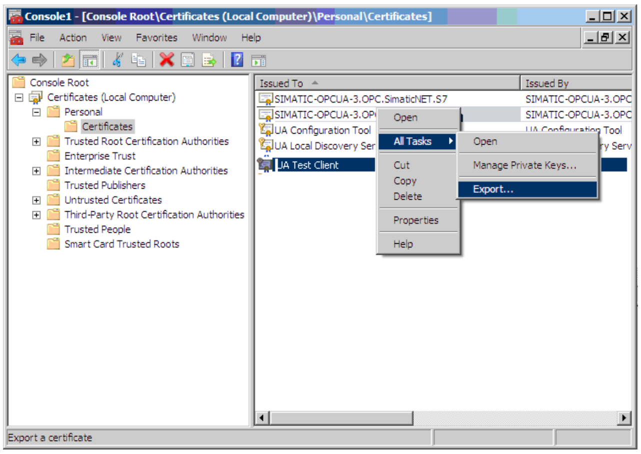
Configuration server with SIMATIC NET CD 2008 (V7.1)

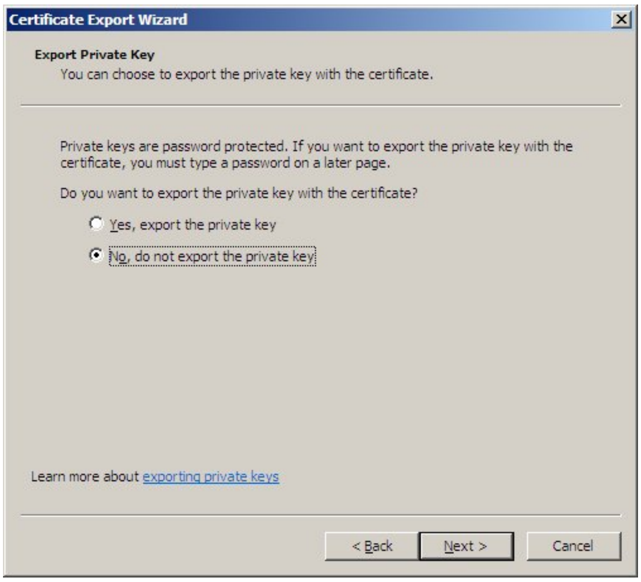
In the SIMATIC NET CD 2008 (V7.1) the untrusted client certificates have to be copied manually to the certificate directory of the server. They are first exported out of the Windows certificate store from the client PC.

Table 5-6

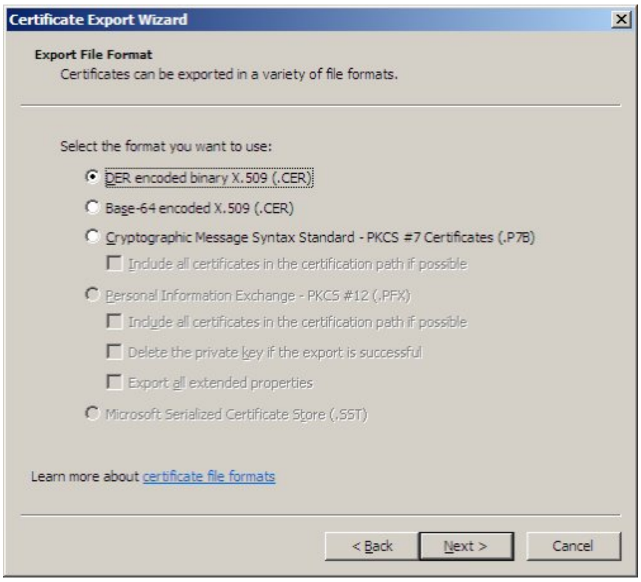
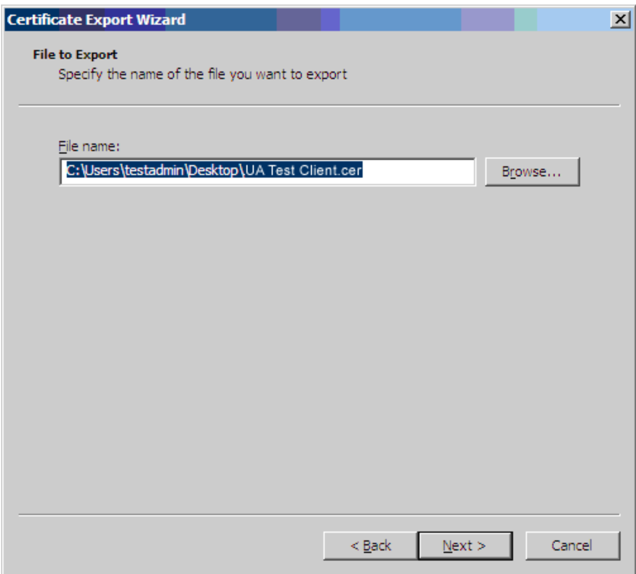
No.	Action	Remarks
3.	Open the Windows certificate store in the Management Console.	On the “OPC Client Station” client PC Start→ Execute → “mmc” 
4.	In the file menu select “Add or Remove SnapIn” and afterwards select “Add>”.	
5.	Select the certificates for the “Computer account” of the local computer.	

<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td>6.</td><td>Select the certificate to be exported (right-click → All Tasks → Export…).</td><td>In this example, the UA Test Client certificate created by the OPC UA Example Clients at the first START.
The Export wizard is started



</td></tr>
<tr><td>7.</td><td>The private key is NOT exported but only the “public” key</td><td>


</td></tr>
</table>

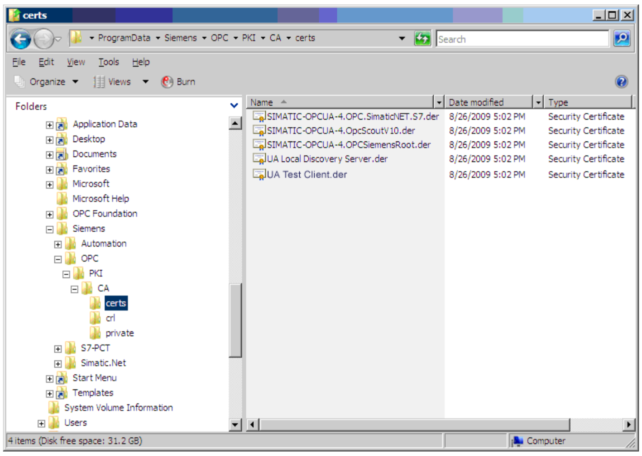
No.	Action	Remarks
8.	Select **DER** coding	
9.	Enter a file name and store it in the certificate file.	

Manual import of client certificate at server

In the SIMATIC NET CD 2008 (V7.1) trusted client certificates have to be copied directly to the certificate directory of the server. The certificate file from the client PC, exported from the Windows certificate store is renamed and copied on the server PC (in the certificate directory of the server).

Table 5-7

No.	Action	Remarks
1.	Open the directory in the Windows Explorer.	On the PC “OPC Server Station”
2.	Copy the file on the “OPC Server Station” server PC and change the file ending to “**DER**”	In this example here: “UA Test Client.der”

No.	Action	Remarks
3.	In the certificate directory of the servers you will find all public OPC UA certificates trusted by the server. This is where the client certificate has to be copied to be accepted by the server.	The directory is located in the SIMATIC NET software data directory. C:\Documents and settings\All Users\Application Data \Siemens\OPC\PKI\CA\ 

5.3.3 Authentication, SecurityPolicy and MessageSecurityMode

The OPC UA server supports the authentication of clients during connection establishment. Two types of authentication are supported:

- Anonymous
- UserName / Password

After the installation both modes are active, to make commissioning easier. The server can be reconfigured so that anonymous logons are no longer possible. User name and password have to be indicated by the client and the server checks it against the Windows user administration. Thus, only clients which have a Windows account on the server machine can connect.

The server has two connection endpoint configurations which can be used by the clients. Each of these endpoints represents another encryption mechanism for data transmission

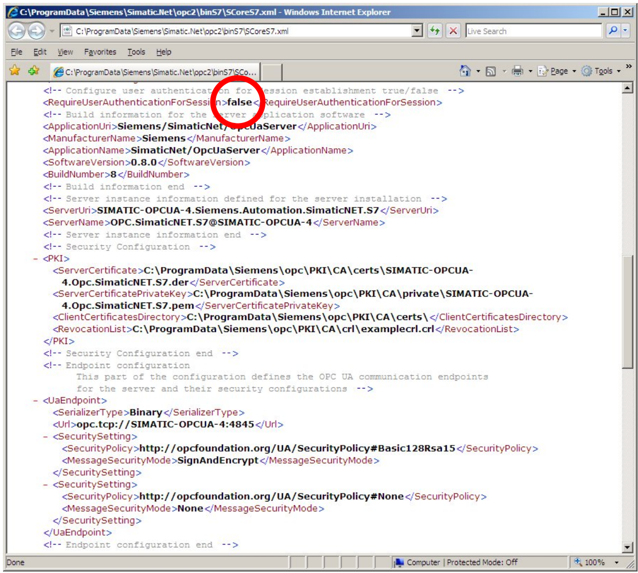
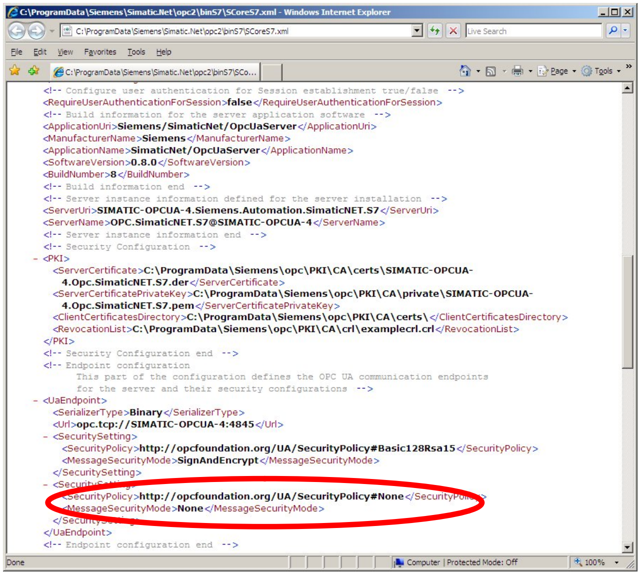
- None
- Basic128RSA15

After the installation both SecurityPolicies are active to make commissioning easier. The server can be reconfigured to only use secure encrypted connections. Thus, only clients can connect which know how to handle the Basic128RSA15 encryption.

If you want to change the security mechanisms of the OPC UA server, edit the configuration file of the OPC UA server and afterwards restart the server.

C:\ Documents and settings\All Users\Application Data \Siemens\Simatic.Net\opc2\bins7\ SCoreS7.xml

Table 5-8

No.	Action	Remarks
1.	Open the OPC UA “SCoreS7.xml” server configuration file in an editor (e.g. notepad)	The file is located in the SIMATIC NET software data directory. C:\Documents and Settings\All Users\Application Data\Siemens\Simatic.Net\opc2\bins7
2.	Set the RequireUserAuthenticationForSession to “true” if each OPC UA client is to authenticate itself by username and password to be able to establish a session.	
3.	Delete the complete SecuritySetting entry for SecurityPolicy “none” and MessageSecurityMode “none” from the UAEndpoint configuration. Afterwards the server will only allow encrypted transmissions of the “Basic128RSA15” type with signature and encryption. Only clients which also support this encryption type can connect themselves (if their certificate is trusted).	

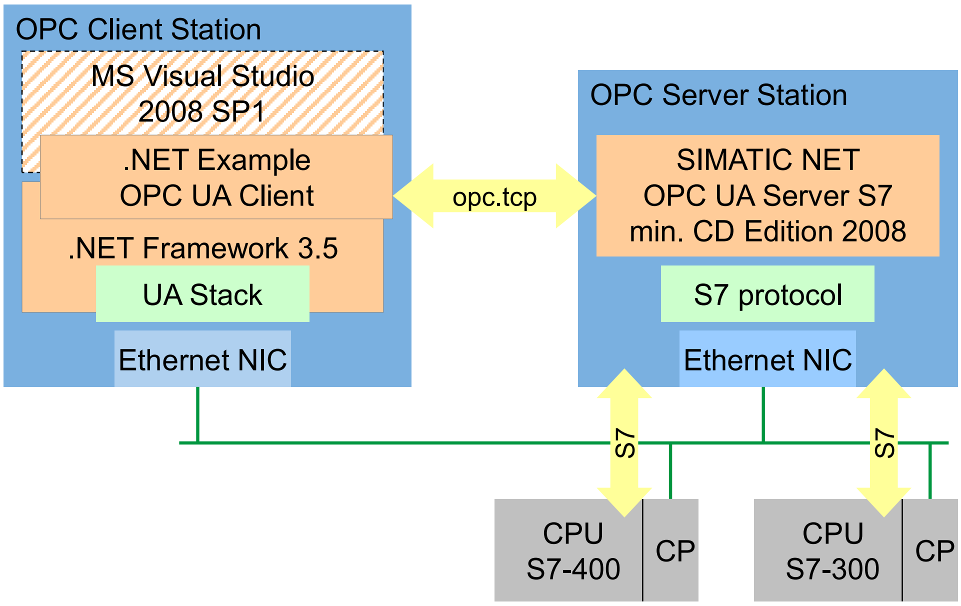
6 Installation

This chapter describes which hardware and software components have to be installed. It is also important to read the descriptions, manuals and any delivery information supplied with the products.

Installing the hardware

The figure below shows the hardware setup of the application as well as the required software components.

Figure 6-1



Note The installation guidelines for Industrial Ethernet must generally be observed.

Software installation

This chapter describes the steps for the software installation required for the application.

Table 6-1

No.	Action	Remarks
1	Install SIMATIC NET DVD V12 onto the SIMATIC OPC Server station.	
2	Install STEP 7 V13 on the engineering station (e.g. laptop).	Only required for programming and the configuration of the connection of the S7-300/400 and for creating symbol information
3	Install the .NET Framework 3.5 SP1 on the OPC client station.	This step is only necessary when the framework has not yet been installed by any other application.
4	Install Microsoft Visual Studio 2010 on the OPC UA Client station. Install the C# development environment.	This step is only necessary if the code is verified or modified.

Installing the examples

This chapter describes the steps for the installation of the example code.

Table 6-2

No.	Action	Remarks
1	Install the application software on the OPC UA client station. Copy the Executables as well as the Assemblies (file: “bin”) and also the source code (file: “scr”) in a directory to which you have access rights.	You only need the source code if you want to make modifications or verify individual functions.
2	Install the STEP 7 program. Copy and unzip the STEP 7 project.	If necessary, adjust the configuration of your hardware.

7 Commissioning the Application

It is required that all hardware and software components have been successfully installed and cabled.

Note The project file (XDB) delivered with this example contains the completely configured PC station. This file can only be used without adjustment if the hardware is identical with the configuration.

If different hardware is used, the configuration of the PC station has to be adjusted. This is particularly necessary when the hardware releases differ or when the Ethernet addresses are not identical. The STEP 7 project included in the delivery has to be opened and adapted accordingly (see chapter 6). After saving and compiling, all configuration information is overwritten in the XDB file.

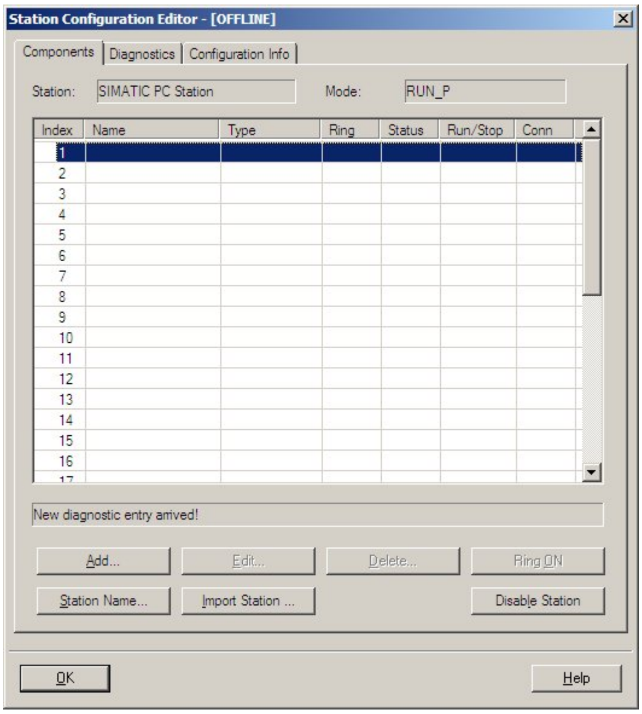
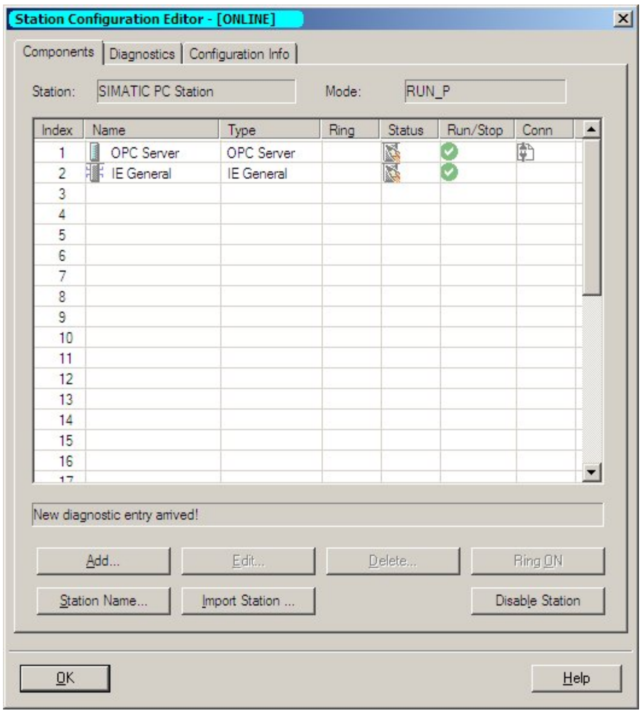
Commissioning the PC station

The following table shows the configuration and settings of the PC stations by importing an XDB file.

Table 7-1

No.	Action	Remarks
1.	Dearchiving the project	Unzip the **OPC_UA_STEP7_v10.zip** file
2.	Open project file with STEP 7, configure the CPU stations and load the configuration into the controller.	Make the corresponding adjustments if you use different hardware. The PC station can also be configured and loaded “online”. Alternative: load the PC station using the XDB file: see steps below.

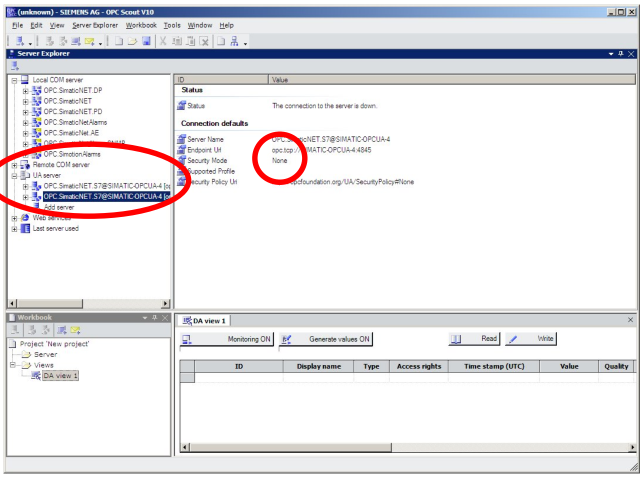
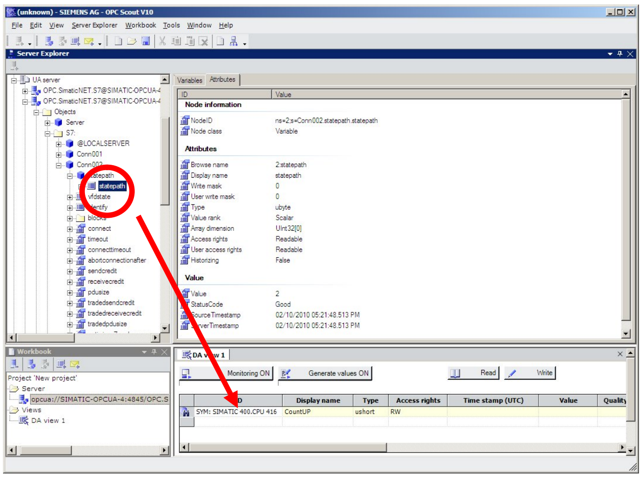
5.3 Configuration of the OPC UA Security

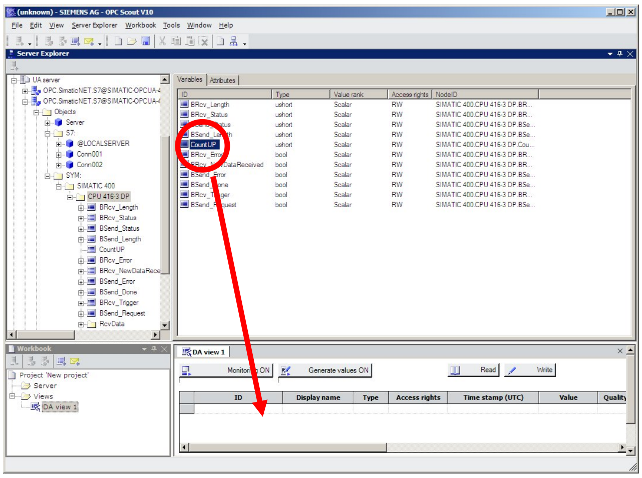
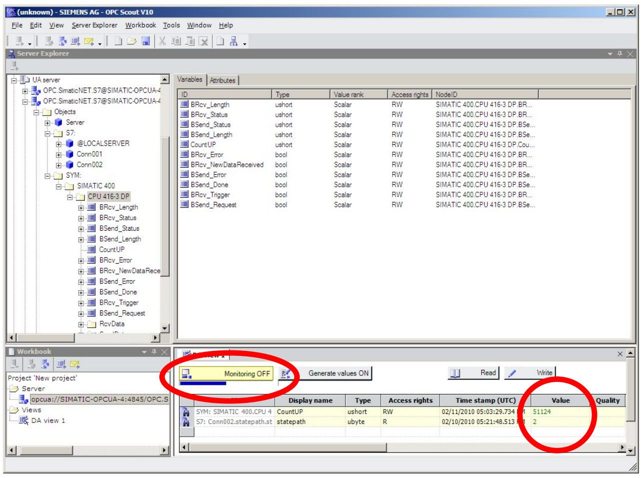
No.	Action	Remarks
3.	Open the `Station Configurator` by double-clicking on the  icon in the task bar.	
4.	Click on `Import Station`	Confirm the query with Yes.
5.	Select the XDB file	The XDB file is located in the XDB subdirectory of the extracted ZIP file in the directory tree of the STEP7 project.
6.	After importing the XDB file the PC station has been configured. In the process, not only the connection information was accepted but the symbol file was also extracted from the XDB and made available on the server.	

Testing with OPC Scout V10

The new OPC Scout V10 is a combined OPC client, which handles the classis COM OPC interfaces as well as the new OPC UA. This client is delivered and installed with the SIMATIC NET CD.

Table 7-2

No.	Action	Remarks
1.	Start the OPC Scout V10 from the start menu	On the PC “OPC Server Station”
2.	Connect with the “unsecure” endpoint “none”. The connection is established as soon as you open the corresponding entry in the tree view.	
3.	Browse to the “S7:” subfile and insert the “&statepath” tag from the “Conn002” connection via drag&drop in the bottom window.	

No.	Action	Remarks
4.	Browse to the “SYM:” subfile and insert the “CountUP” tag via drag&drop in the bottom window.	
5.	Switch on monitoring and check whether the corresponding values are displayed in the Value column. The connection status tag has to have (established) value “2” and the counter should increment.	

Starting the example application

Table 7-3

No.	Action	Remarks
1.	Start the respective application “simple” or “comfortable client” with administrator rights.	The file is located in the “bin” subfile of the unzipped example. At the first program start, you need to start the clients with administrator rights in order to create the certificates of the clients. Start the clients by right-clicking on the EXE file and then selecting the administrator account at “`Execute as`”.
2.	Starting the example application	See chapter 8

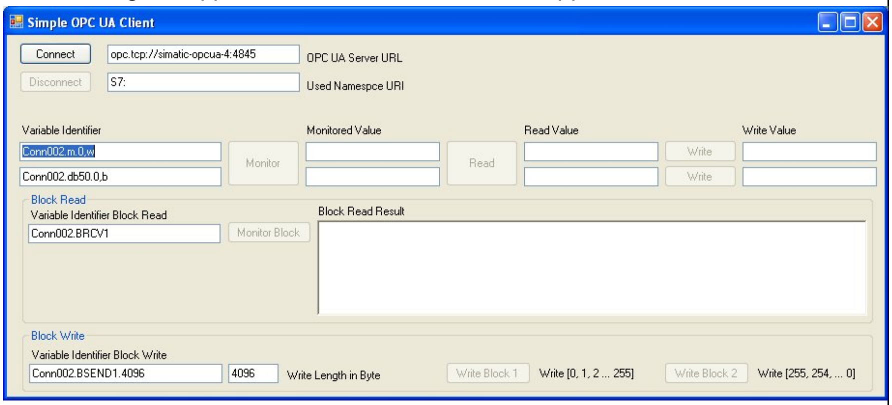
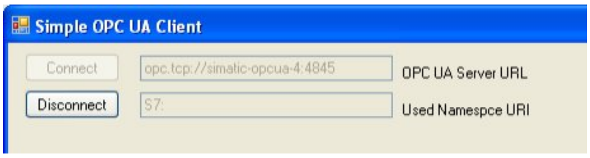
8 Operating the Application

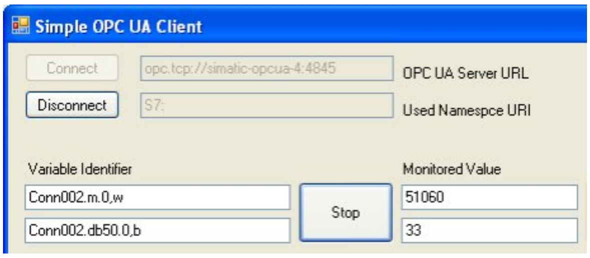
8.1 Operating the Simple Client

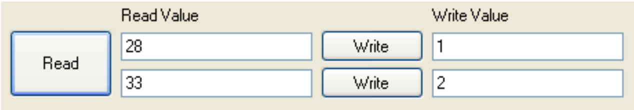
Access with indirect addressing

In the first part of the simple example, data of S7 tags is monitored, read and written via direct addressing.

Table 8-1

<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td>1</td><td>Start the simple client with administrator rights.</td><td>After starting the application this user interface will appear.

</td></tr>
<tr><td>2</td><td>Establish connection to server</td><td>The server URL has to be entered manually. All identifiers for NodeIDs in the user interface use the same namespace. The namespace S7: is specified for the direct addressing, by default
The Connect button can be used to establish the connection to the server.


Note
In order to access optimized data blocks of the S7-1200/S7-1500 CPU, at least OPC server V12 needs to be configured and the OPC server be accessed via the SimetcNET.S7OPT (Port 4850) via the client.</td></tr>
<tr><td>3</td><td>Monitoring of data</td><td>For the monitoring of data changes, reading, and writing, two tags can be indicated. The NodeID is made up of the identifier and namespace. Via the Monitor button, monitoring can be switched on. The reported data changes are displayed in the text boxes next to the button.

</td></tr>
</table>

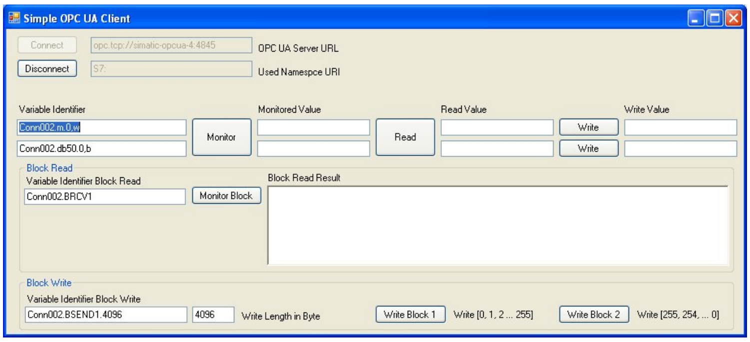
No.	Action	Remarks
4	Reading and writing	For reading and writing, the same NodeIDs are used as for monitoring. By pressing the **Read** button the two tags are read. The tags can be written individually via the two **Write** buttons. Before writing, it has to be read first, since the data conversion for writing is on the basis of the read values. 

Using the block services

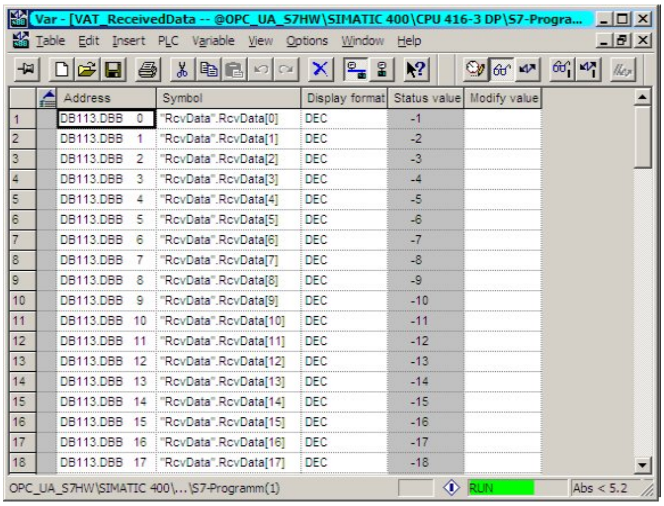
A typical application for block-oriented services is shown here as a clear, simple example: the sending of recipe data to the S7 or the receiving of result data from the S7, whilst using the BSEND and BRCV block services.

The S7 controller receives a recipe from the “OPC Client Station” PC station. Two “recipes” were simulated in the OPC UA client which are sent to the OPC UA server after clicking the corresponding button via a write job to the NodeID of the BSEND tag. The OPC UA server sends a BSEND call to the corresponding controller. In turn, the S7 has to call an appropriate BRCV to receive the data. The BRCV has to be available with the correct RID for the right connection with the required minimum length at the right time. Proceed according to the step-by-step instruction below.

Table 8-2

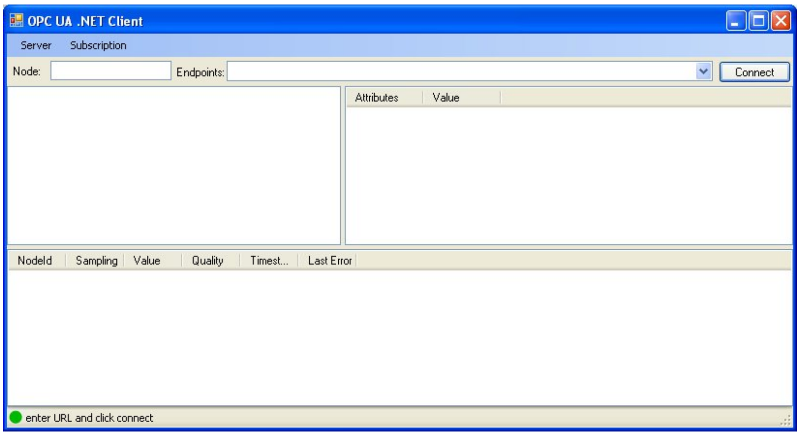
No.	Action	Remarks
1.	Start STEP 7 and open the tag table (VAT) of the receive buffer.	Go to the SIMATIC controller -> CPU ->S7 program -> blocks and open “VAT_ReceivedData”. Switch to the Online view (glasses)
2.	Start the OPC UA client and connect yourself with the OPC UA server	
3.	Receiving from the S7	For receiving from the S7, subscription with Monitored Item must be created. For this purpose, the “Monitoring Block” has to be clicked in the “Block Read” group. In the field next to the button the result data is shown as HEX code. Please consider the R_ID as part of the NodeID. The S7 has to supply the BSEND block with this ID 
4.	Sending of data to S7	Select the recipe by either pressing “Write Block 1” or “Write Block 2”. The first button writes into the data block by incrementing a value for each array entry from 0 onwards. The second button decrements the array entries from 255.

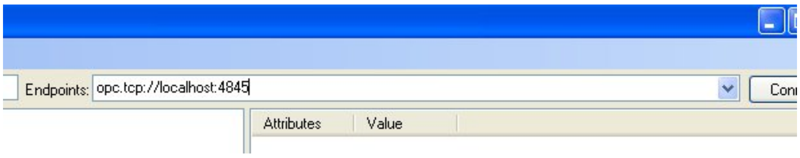
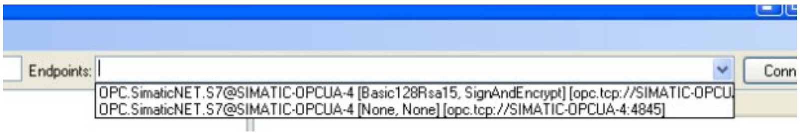
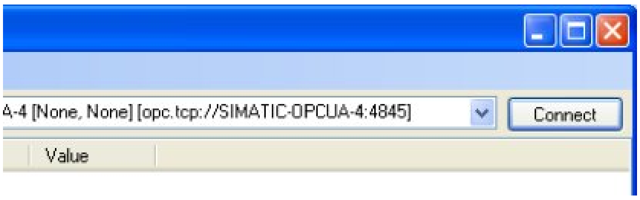
<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td></td><td></td><td>Please consider the R_ID as part of the NodeID. In the S7, the BRCV block has to be supplied with this ID.

</td></tr>
<tr><td>5.</td><td>Check with the tag table (VAT) whether the data has arrived.</td><td>
</td></tr>
<tr><td>6.</td><td>Processing in S7</td><td>Now the recipe should be processed in S7 and then, the BRCV should be switched to ready to receive.</td></tr>
<tr><td>7.</td><td>Reenabling of receiving</td><td>Once the data was processed, actively call the BRCV (in the example this happens automatically).</td></tr>
<tr><td>8.</td><td>Resending</td><td>Now send the second recipe with the OPC UA client.</td></tr>
</table>

8.2 Operating the convenient OPC UA client

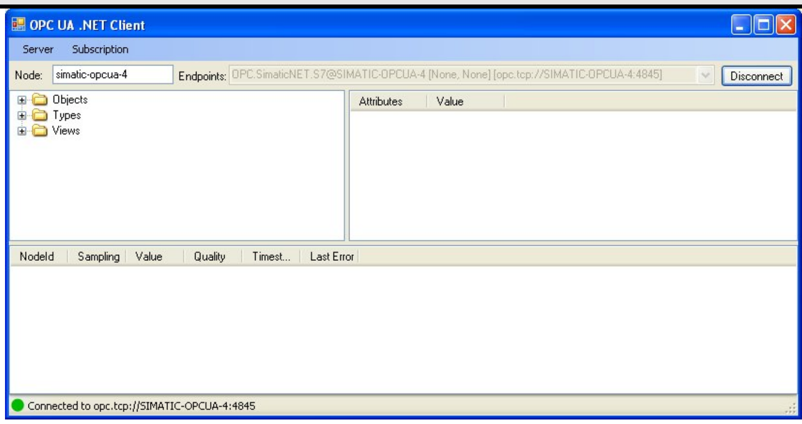
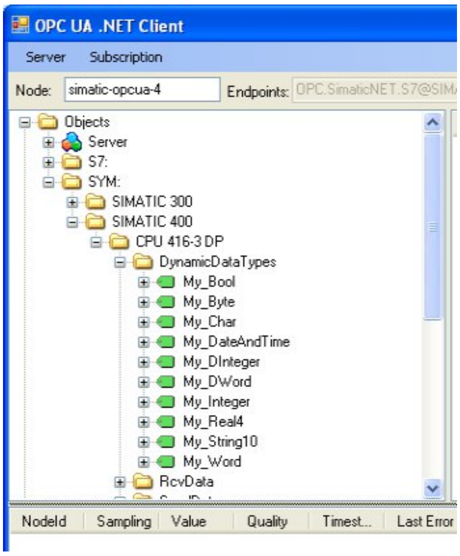
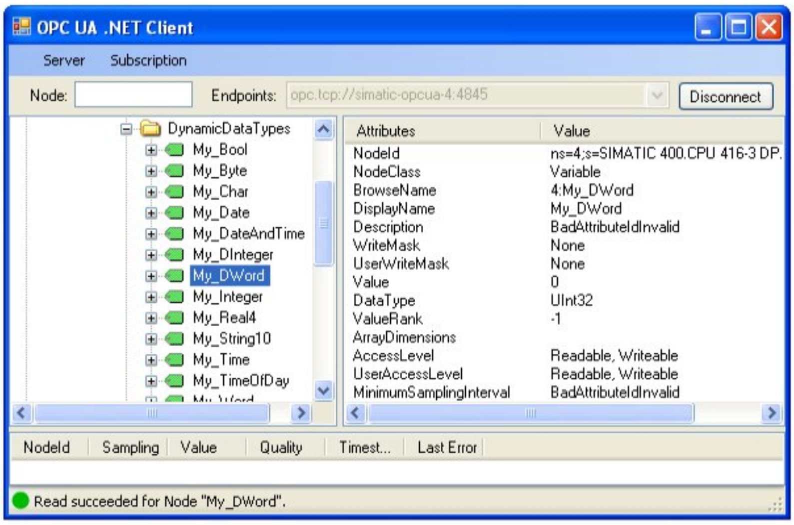
Table 8-3

<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td>1</td><td>Start the OPC UA client (this example)</td><td>After starting the application this user interface will appear.

</td></tr>
<tr><td>2</td><td>Manually enter a server URL</td><td>To establish a connection with an OPC UA server the URL of the server can be entered manually in the Endpoints text field.
If the SIMATIC NET OPC UA server is installed on the same computer, then the following URL can be used: opc.tcp://localhost:4845.
When the entry is manual, it will be tried to establish a connection without security.</td></tr>
</table>

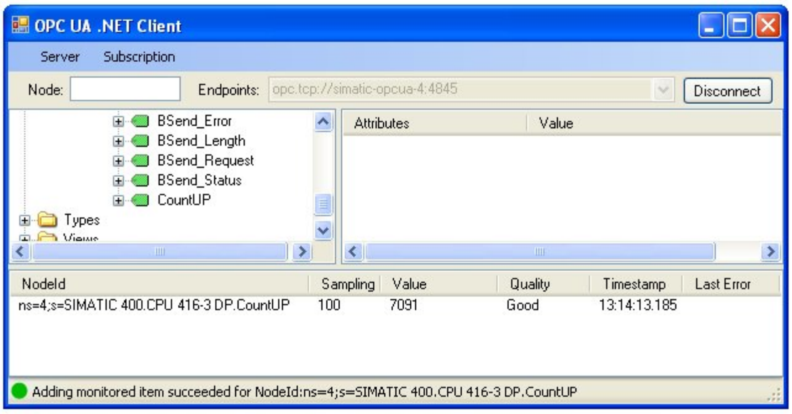
<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td></td><td></td><td>Note
In order to access optimized data blocks of the S7-1200/S7-1500 CPU, at least OPC server V12 needs to be configured and the OPC server be accessed via the SimetcNET.S7OPT (Port 4850) via the client.

</td></tr>
<tr><td>3</td><td>List of available endpoints</td><td>Via the Endpoints selection list, the list of available endpoints can be listed. If the Node text field is empty, the local servers or their endpoints will be listed.
Entering a remote computer name in the Node text field and subsequently clicking the Endpoints selection list, will display the available endpoints on the remote computer.
Apart from the server name, the security settings and the URL of the endpoint is displayed.

</td></tr>
<tr><td>4</td><td>Establishing a connection with the server</td><td>The Connect button is used to establish the connection to the server.


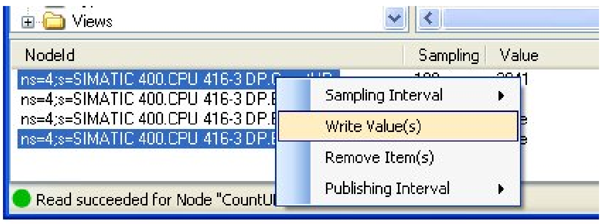
If the server certificate is considered untrusted, the user will be asked whether he/she wants to accept it nevertheless.
The necessary steps for the server to accept the client certificate are described in chapter 5.3.2.


If establishing a connection was successful, the top level of the address space of the OPC UA server is displayed in the window on the left:</td></tr>
</table>

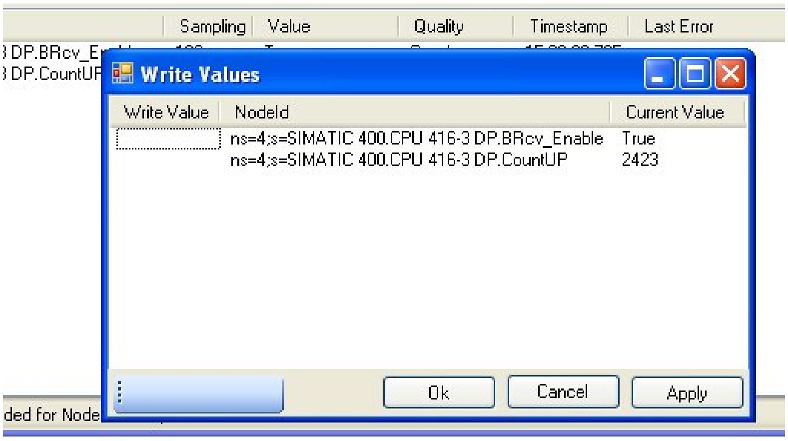
No.	Action	Remarks
		 **Disconnect** will now appear on the button.
5	Browse	The tree view makes it possible to navigate in the address space of the server. 
6	Reading the node attributes	By selecting a node in the tree view, its attributes and their values are displayed on the right, next to the list. 

<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td>7</td><td>Monitoring tags (Monitored items).</td><td>Tags present in the address space of the server, can be dragged from the tree view into the bottom monitoring window via drag&drop. This is where the value changes will be displayed.

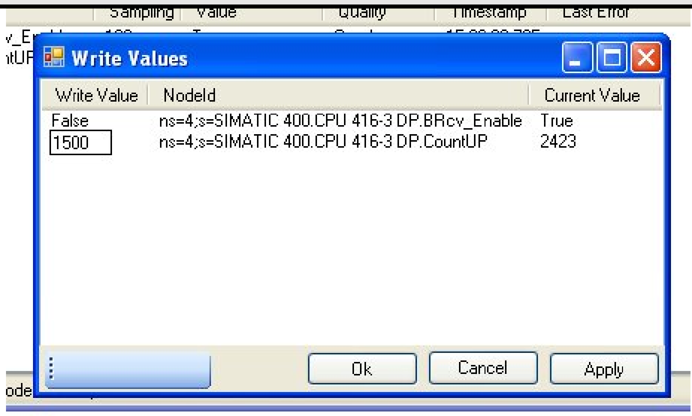

</td></tr>
<tr><td>8</td><td>Writing of OPC items</td><td>If one or several tags have been selected in a monitoring window, the user can start a write process via the context menu of the list:

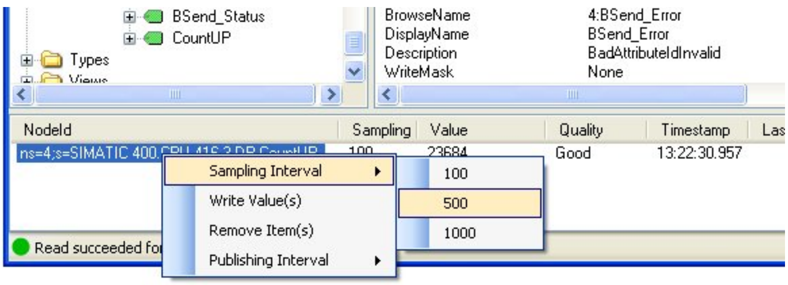
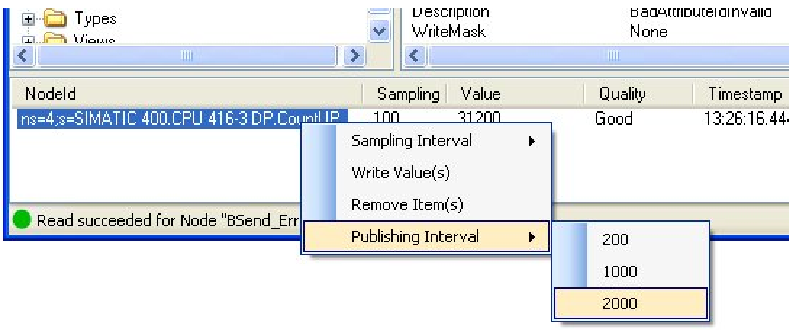


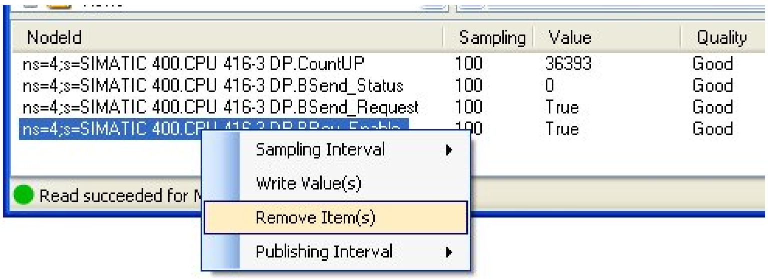
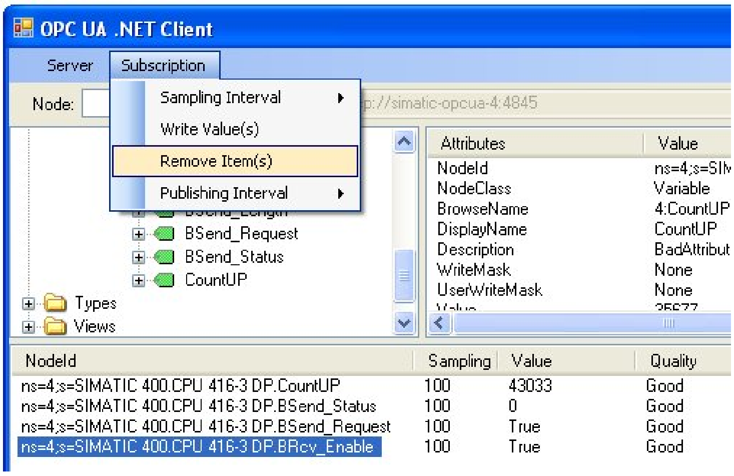
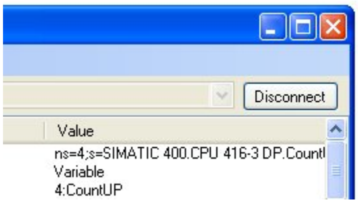
In the dialog window now opened, the tags selected from the list earlier are displayed with their current values:



In the Write Value column, the values can be entered:</td></tr>
</table>

<table>
<tr><th>No.</th><th>Action</th><th>Remarks</th></tr>
<tr><td></td><td></td><td>

By clicking the Ok button the new values are written and the dialog is closed. However, if the Apply button is clicked, the values are written and the dialog remains open for the further writing of values.</td></tr>
<tr><td>9</td><td>Change the sampling interval</td><td>Three different sampling intervals can be set by using the context menu. Several entries of the list field can be selected.

</td></tr>
<tr><td>10</td><td>Change the publishing interval</td><td>The publish interval of the subscription can also be changed via the context menu of the monitoring window.

</td></tr>
</table>

No.	Action	Remarks
11	Remove monitored items	The selection of the **Remove Item(s)** menu entry, removes all tags selected in the list. 
12	Calling of actions from the menu bar	Actions 8 to 11 can also be carried out via the menu bar: 
13	Terminate connection with the server	The **Disconnect** button can be used to terminate the connection with the OPC UA server. 
14	Connection error of OPC UA server connection with the controller	If the connection between OPC UA server and S7 is interrupted then the status of the tags in the monitoring list changes to “Bad”.
15	Error when calling	If errors occur during OPC UA calls, these errors are displayed in the status bar of the application. If the server does not respond, the client API will deliver a timeout error.

9 Further Notes, Tips & Tricks, etc.

Reusability and expansion of the client API

The client API is realized as an independent, reusable assembly DLL. It can be directly used in other applications. An expansion for additional OPC UA features such as method calls can be easily achieved.

Reusability of the GUI controls

The GUI elements for browsing, listing of attributes and for monitoring of tag values have been created as controls. For reusability it makes sense to store these controls in an independent assembly DLL.

Saving NodeIDs

NodeIDs are made up of an identifier and the namespace index. Although the namespace index does not change as long as the OPC UA server is running, it is always possible that the index changes during a restart of the OPC UA server. Although this is not the case with the SIMATIC NET OPC UA server, an OPC UA client should nevertheless be prepared for it when storing the NodeIDs.

When storing, the index must not simply be saved but the namespace URI has to be saved. This URI remains constant, even when the index changes.

There are two strategies to save the namespace URI:

- Instead of saving the index, the URI is saved with the identifier for the NodeID. This is the easiest variant but has the disadvantage that a great deal of redundant information is saved when the namespace URI is the same for all stored NodeIDs of the server.
- The index is saved with the identifier but the appropriate namespace table with the namespace URIs is stored in parallel. This variant is more efficient. However, it requires an additional storage location for the table.

For both variants the namespace table has to be read from the server, after establishing a connection with the server, and the namespace URI has to be reimplemented in the current index.

Optimizing of the NodeIDs for read and write

NodeIDs may contain long texts and are therefore not suitable to be used in cyclic calls of Read and Write since this causes an unnecessary overhead on the network and during the processing on the server.

OPC UA provides the special services RegisterNodes and UnregisterNodes, to be able to achieve an optimization. RegisterNodes supplies a list of optimized NodeIDs with numeric identifiers for a list of original NodeIDs, which can be used like Handles. These NodeIDs are only four bytes long on the network and can be used for very fast data access on the server.

Since the Handle is also a NodeID, it can be used in all services instead of the original NodeID. However, the optimized NodeID is only valid within the session.

If registered NodeIDs are no longer needed, they should be released with UnregisterNode to release resources on the server.

10 Links & Literature

10.1 Bibliographic references

This list is not complete and only represents a selection of relevant literature.

Table 10-1

	Topic	Title
/1/	STEP 7	Automating with STEP 7 in STL and SCL Hans Berger Publicis Corporate Publishing ISBN 3-89578-113-4
/2/	OPC UA	OPC Unified Architecture Mahnke, Leitner, Damm Springer Verlag ISBN 978-3-540-68898-3

10.2 Internet link specifications

This list is by no means complete and only presents a selection of suitable information.

Table 10-2

	Topic	Title
\1\	Link to this document	http://support.automation.siemens.com/WW/view/en/42014088
\2\	Siemens I IA/DT Customer Support	http://support.automation.siemens.com
\3\	OPC Data Access Custom Interface Version 3.0	Specification on the OPC Foundation website for download for OPC members www.opcfoundation.org
\4\	OPC unified architecture	Specification on the OPC Foundation website for download for OPC members www.opcfoundation.org
\5\	SIMATIC NET Commissioning SIMATIC NET PC stations – Instruction and quick start for SIMATIC NCM PC / STEP 7 from Version V5.2	Description or information on: • General information on the PC tools. • Functions of NCM PC. Installed by SIMATIC NET, see: Start → Simatic → Documentation → English. http://support.automation.siemens.com/WW/view/en/13542666
\6\	SIMATIC NET – Industrial communication with PG/PC	Manual for industrial communication on PG/PC with SIMATIC NET. Installed by SIMATIC NET, see: Start → Simatic → Documentation → English. http://support.automation.siemens.com/WW/view/en/2044387
\7\	Unified automation	http://www.unified-automation.com/downloads/opc-ua-development.html

11 History

Table 11-1

Version	Date	Modifications
V1.1	06/2014	Revised version: migration to STEP 7 V13, adjusting the documentation
V1.0	04/2010	First version