OPC UA Server SDK .NET Standard

Develop OPC UA Servers with C# / VB.NET

Tutorial Workshop Server

Document Control

Version	Date	Comment
1.0.8	18-MAY-2019	Initial version based on SDK 1.0.8
1.0.9	31-MAY-2019	Updated chapter 2
1.1.0	22-JUN-2019	Updated to new structure
1.1.1	27-JUL-2019	Updated to SDK 1.1.1
1.2.0	24-NOV-2019	- Added informationen for .NET Core 2.0 on Linux, macOS - Changed to .NET 4.6.2 - Enhanced chapter *UserIdentity and UserIdentityTokens* and added usage of IsNodeAccessibleForUser() and IsReferenceAccessibleForUser() - Added chapter *Supported OPC UA Profiles* - Enhanced UA Server Design chapter - Updated user access level handling

Purpose and audience of document

Microsoft's .NET Framework is an application development environment that supports multiple languages and provides a large set of standard programming APIs. This document defines an Application Programming Interface (API) for OPC UA Client and Server development based on the .NET Standard programming model.

The OPC UA specification can be downloaded from the web site of the OPC Foundation. But only [OPC 10000-1] (Overview and Concepts) is available to the public. All other parts can only be downloaded from OPC Foundation members and may be used only if the user is an active OPC Foundation member. Because of this fact the OPC UA SDK .NET Standard API hides most of the OPC UA specifications to provide the possibility to delevlop OPC UA Clients and OPC UA Servers in the .NET Standard environment without the need to be an OPC Foundation member. The API does support OPC Unified Architecture.

This document is intended as reference material for developers of OPC UA compliant Client and Server applications. It is assumed that the reader is familiar with the Microsoft's .NET Standard and the needs of the Process Control industry.

Summary

This document gives a short overview of the functionality of the server development with the OPC UA Server SDK .NET Standard. The goal of this document is to give an introduction and can be used as base for your own implementations

Referenced OPC Documents

Documents
This document partly uses extracts taken from the OPC UA specifications to be able to give at least a short introduction into the specifications. The specifications itself are available from: http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC#Specifications
OPC Unified Architecture Textbook, written by Wolfgang Mahnke, Stefan-Helmut Leitner and Matthias Damm: http://www.amazon.com/OPC-Unified-Architecture-Wolfgang-Mahnke/dp/3540688986/ref=sr_1_1?ie=UTF8&s=books&qid=1209506074&sr=8-1
[OPC 10000-1] OPC UA Specification: Part 1 – Overview and Concepts https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/
[OPC 10000-2] OPC UA Specification: Part 2 – Security Model https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/
[OPC 10000-3] OPC UA Specification: Part 3 – Address Space Model https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model/
[OPC 10000-4] OPC UA Specification: Part 4 – Services https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/
[OPC 10000-5] OPC UA Specification: Part 5 – Information Model https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model/
[OPC 10000-6] OPC UA Specification: Part 6 – Mappings https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-6-mappings/
[OPC 10000-7] OPC UA Specification: Part 7 – Profiles https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-7-profiles/
[OPC 10000-8] OPC UA Specification: Part 8 – Data Access https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-8-data-access/
[OPC 10000-9] OPC UA Specification: Part 9 – Alarm & Conditions https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-9-alarms-and-conditions/
[OPC 10000-10] OPC UA Specification: Part 10 – Programs https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-10-programs/
[OPC 10000-11] OPC UA Specification: Part 11 – Historical Access https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-11-historical-access/
[OPC 10000-12] OPC UA Specification: Part 12 – Discovery and Global Services https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-12-discovery-and-global-services/
[OPC 10000-13] OPC UA Specification: Part 13 – Aggregates https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-13-aggregates/
[OPC 10000-14] OPC UA Specification: Part 14 – PubSub https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub/
[OPC 10000-100] OPC UA Specification Part 100 - Devices https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-100-device-information-model/

Other Referenced Documents

SOAP Part 1: SOAP Version 1.2 Part 1: Messaging Framework

http://www.w3.org/TR/soap12-part1/

SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts

http://www.w3.org/TR/soap12-part2/

XML Encryption: XML Encryption Syntax and Processing

http://www.w3.org/TR/xmlenc-core/

XML Signature:: XML-Signature Syntax and Processing

http://www.w3.org/TR/xmldsig-core/

WS Security: SOAP Message Security 1.1

http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

WS Addressing: Web Services Addressing (WS-Addressing)

http://www.w3.org/Submission/ws-addressing/

WS Trust: Web Services Trust Language (WS-Trust)

http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

WS Secure Conversation: Web Services Secure Conversation Language (WS-SecureConversation)

http://specs.xmlsoap.org/ws/2005/02/sc/WS-SecureConversation.pdf

SSL/TLS: RFC 2246: The TLS Protocol Version 1.0

http://www.ietf.org/rfc/rfc2246.txt

X200 : ITU-T X.200 – Open Systems Interconnection – Basic Reference Model

http://www.itu.int/rec/T-REC-X.200-199407-I/en

:X509: X.509 Public Key Certificate Infrastructure

http://www.itu.int/rec/T-REC-X.509-200003-I/e

HTTP: RFC 2616: Hypertext Transfer Protocol - HTTP/1.1

http://www.ietf.org/rfc/rfc2616.txt

HTTPS: RFC 2818: HTTP Over TLS

http://www.ietf.org/rfc/rfc2818.txt

IS Glossary: Internet Security Glossary

http://www.ietf.org/rfc/rfc2828.txt

NIST 800-12: Introduction to Computer Security

http://csrc.nist.gov/publications/nistpubs/800-12/

NIST 800-57: Part 3: Application-Specific Key Management Guidance

http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_PART3_key-management_Dec2009.pdf

NERC CIP: CIP 002-1 through CIP 009-1, by North-American Electric Reliability Council

http://www.nerc.com/page.php?cid=2|20

IEC 62351: Data and Communications Security

http://www.iec.ch/heb/d_mdoc-e050507.htm

SPP-ICS: System Protection Profile
Industrial Control System, by Process Control Security Requirements Forum (PCSRF)

http://www.isd.mel.nist.gov/projects/processcontrol/SPP-ICSv1.0.pdf

SHA-1: Secure Hash Algorithm RFC

http://tools.ietf.org/html/rfc3174

PKI: Public Key Infrastructure article in Wikipedia

http://en.wikipedia.org/wiki/Public_key_infrastructure

X509 PKI: Internet X.509 Public Key Infrastructure

http://www.ietf.org/rfc/rfc3280.txt

EEMUA : 2nd Edition EEMUA 191 - Alarm System - A guide to design, management and procurement
(Appendixes 6, 7, 8, 9).

http://www.eemua.co.uk/

Table of Contents

Disclaimer

This document was created strictly for information purposes. No guarantee, contractual specification or condition shall be derived from this document unless agreed to in writing. Technosoftware GmbH reserves the right to make changes in the products and services described in this document at any time without notice and this document does not represent a commitment on the part of Technosoftware GmbH in the future.

While Technosoftware GmbH uses reasonable efforts to ensure that the information and materials contained in this document are current and accurate, Technosoftware GmbH makes no representations or warranties as to the accuracy, reliability or completeness of the information, text, graphics, or other items contained in the document. Technosoftware GmbH expressly disclaims liability for any errors or omissions in the materials contained in the document and would welcome feedback as to any possible errors or inaccuracies contained herein.

Technosoftware GmbH shall not be liable for any special, indirect, incidental, or consequential damages, including without limitation, lost revenues or lost profits, which may result from the use of these materials. All offers are non-binding and without obligation unless agreed to in writing.

Trademark Notice

Microsoft, MSN, Windows and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

1 Installation

You should download the following SDK from either https://technosoftware.com/ or the license paper provided to be able to follow this tutorial:

1. **OPC UA Bundle SDK .NET Standard**
   The OPC UA Bundle SDK .NET Standard offers a fast and easy access to the OPC UA Client & Server technology. Develop OPC compliant UA Clients and Servers with C#/VB.NET targeting the .NET Standard.
   The download includes examples for .NET 4.6.2, .NET 4.7.2 and for .NET Standard 2.0.
   You can download it from https://technosoftware.com/evaluations/

Important:

An installation guide is available with the SDK or from https://technosoftware.com/download/opc-ua-net-installation/. Please read that one first and then follow this guide.

2 Supported OPC UA Profiles

The following table shows the different OPC UA profiles and if they are supported by the OPC UA Server SDK .NET:

2.1 Core Characteristics

Profile	Description	Supported
Core 2017 Server Facet	This Facet defines the core functionality required for any UA Server implementation. The core functionality includes the ability to discover endpoints, establish secure communication channels, create Sessions, browse the AddressSpace and read and/or write to Attributes of Nodes. The key requirements are support for a single Session, support for the Server and Server Capabilities Object, all mandatory Attributes for Nodes in the AddressSpace, and authentication with UserName and Password. For broad applicability, it is recommended that Servers support multiple transport and security Profiles. This Facet supersedes the "Core Server Facet".	✓
Sessionless Server Facet	Defines the use of Sessionless Service invocation in a Server.	✗
Reverse Connect Server Facet	This Facet defines support of reverse connectivity in a Server. Usually, a connection is opened by the Client before starting the UA-specific handshake. This will fail, however, when Servers are behind firewalls with no open ports to connect to. In the reverse connectivity scenario, the Server opens the connection and starts with a ReverseHello message requesting that the Client establish a Secure Channel using this connection.	✗
Reverse Connect Server Facet	This Facet defines support of reverse connectivity in a Server. Usually, a connection is opened by the Client before starting the UA-specific handshake. This will fail, however, when Servers are behind firewalls with no open ports to connect to. In the reverse connectivity scenario, the Server opens the connection and starts with a ReverseHello message requesting that the Client establish a Secure Channel using this connection.	✗
Base Server Behaviour Facet	This Facet defines best practices for the configuration and management of Servers when they are deployed in a production environment. It provides the ability to enable or disable certain protocols and to configure the Discovery Server and specify where this Server shall be registered.	✓
Request State Change Server Facet	This Facet specifies the support of the RequestServerStateChange Method.	✗
Subnet Discovery Server Facet	Support of this Facet enables discovery of the Server on a subnet using mDNS. This functionality is only applicable when Servers do not register with an LDS.	✗
Global Certificate Management Server Facet	This Facet defines the capability to interact with a Global Certificate Management Server to obtain an initial or renewed Certificate and Trust Lists.	✗

Authorization Service Server Facet	This Facet defines the support for configuring the necessary information to validate access tokens when presented by a Client during session establishment. Access Tokens are issued by Authorization Services.	⊗
KeyCredential Service Server Facet	This Facet defines the capability to interact with a KeyCredential Service to obtain KeyCredentials. For example, KeyCredentials are needed to access an Authorization Service or a Broker. The KeyCredential Service is typically part of a system-wide tool, like a GDS that also manages Applications, Access Tokens, and Certificates.	⊗
Attribute WriteMask Server Facet	This Facet defines the capability to update characteristics of individual Nodes in the AddressSpace by allowing writing to Node Attributes. It requires support for authenticating user access as well as providing information related to access rights in the AddressSpace and restricting the access rights as described.	✓
File Access Server Facet	This Facet specifies the support of exposing File information via the defined FileType. This includes reading of file as well as optionally writing of file data.	⊗
Documentation Server Facet	This Facet defines a list of user documentation that a server application should provide.	⊗

2.2 Data Access

Profile	Description	Supported
Embedded DataChange Subscription Server Facet	This Facet specifies the minimum level of support for data change notifications within subscriptions. It includes limits which minimize memory and processing overhead required to implement the Facet. This Facet includes functionality to create, modify and delete Subscriptions and to add, modify and remove Monitored Items. As a minimum for each Session, Servers shall support one Subscription with up to two items. In addition, support for two parallel Publish requests is required. This Facet is geared for a platform such as the one provided by the Micro Embedded Device Server Profile in which memory is limited and needs to be managed.	⊗
Standard DataChange Subscription 2017 Server Facet	This Facet specifies the standard support of subscribing to data changes and extends features and limits defined by the Embedded Data Change Subscription Facet. See ConformanceUnits for these limits. Note that the Method Call Service is only required for the Methods defined in this Facet. This Facet supersedes the "Standard DataChange Subscription Server Facet".	⊗
Enhanced DataChange Subscription 2017 Server Facet	This Facet specifies an enhanced support of subscribing to data changes. It is part of the Standard UA Server 2017 Profile. This Facet increases the limits defined by the Standard Data Change Subscription 2017 Server Facet.	⊗
Durable Subscription Server Facet	This Facet specifies support of durable storage of data and events even when Clients are disconnected. This Facet implies support of any of the DataChange or Event Subscription Facets.	⊗

Data Access Server Facet	This Facet specifies the support for an Information Model used to provide industrial automation data. This model defines standard structures for analog and discrete data items and their quality of service. This Facet extends the Core Server Facet which includes support of the basic AddressSpace behaviour.	✓
ComplexType 2017 Server Facet	This Facet extends the Core Server Facet to include Variables with structured data, i.e. data that are composed of multiple elements such as a structure and where the individual elements are exposed as component variables. Support of this Facet requires the implementation of structured DataTypes and Variables that make use of these DataTypes. The Read, Write and Subscriptions service set shall support the encoding and decoding of these structured DataTypes. As an option the Server can also support alternate encodings, such as an XML encoding when the binary protocol is currently used and vice-versa.	✓

2.3 Event Access

Profile	Description	Supported
Standard Event Subscription Server Facet	This Facet specifies the standard support for subscribing to events and is intended to supplement any of the FullFeatured Profiles. Support of this Facet requires the implementation of Event Types representing the Events that the Server can report and their specific fields. It also requires at least the Server Object to have the EventNotifier Attribute set. It includes the Services to Create, Modify and Delete Subscriptions and to Add, Modify and Remove Monitored Items for Object Nodes with an “EventNotifier Attribute”. Creating a monitoring item may include a filter that includes SimpleAttribute FilterOperands and a select list of Operators. The operators include: Equals, IsNull, GreaterThan, LessThan, GreaterThanOrEqual, LessThanOrEqual, Like, Not, Between, InList, And, Or, Cast, BitwiseAnd, BitwiseOr and TypeOf. Support of more complex filters is optional. This Facet has been updated to include several optional Base Information ConformanceUnits. These ConformanceUnits are optional to allow for backward compatibility, in the future these optional ConformanceUnits will become required, and so it is highly recommended that all servers support them.	✓
Address Space Notifier Server Facet	This Facet requires the support of a hierarchy of Object Nodes that are notifiers and Nodes that are event sources. The hierarchy is commonly used as a way to organize a plant into areas that can be managed by different operators.	✓

2.4 Alarm & Condition

Profile	Description	Supported
A & C Base Condition Server Facet	This Facet requires basic support for Conditions. Information about Conditions is provided through Event notifications and thus this Facet builds upon the Standard Event Subscription Server Facet. Conditions that are in an “interesting” state (as defined by the Server) can be refreshed using the Refresh Method, which requires support for the Method Server Facet. Optionally the server may also provide support for Condition classes	✓
A & C Refresh2 Server Facet	This Facet enhances the A & C Base Condition Server Facet with support of the ConditionRefresh2 Method.	✗
A & C Address Space Instance Server Facet	This Facet specifies the support required for a Server to expose Alarms and Conditions in its AddressSpace. This includes the A & C AddressSpace information model.	✓
A & C Enable Server Facet	This Facet requires the enabling and disabling of Conditions. This Facet builds upon the A&C Base Condition Server Facet. Enabling and disabling also requires that instances of these ConditionTypes exist in the AddressSpace since the enable Method can only be invoked on an instance of the Condition	✗
A & C Alarm Server Facet	This Facet requires support for Alarms. Alarms extend the ConditionType by adding an Active state which indicates when something in the system requires attention by an Operator. This Facet builds upon the A&C Base Condition Server Facet. This facet requires that discrete AlarmTypes be supported, it also allows for optional support of shelving, alarm comments and other discrete AlarmTypes such as Trip or Off-Normal.	✗
A & C AlarmMetrics Server Facet	This Facet requires support for AlarmMetrics. AlarmMetrics expose status and potential issues in the alarm system. A Server can provide these metrics at various levels (operator station, plant area, overall system etc.).	✗
A & C Acknowledgeable Alarm Server Facet	This Facet requires support for Acknowledgement of active Alarms. This Facet builds upon the A & C Alarm Server Facet. Acknowledgement requires support of the Acknowledge Method and the Acknowledged state. Support of the Confirmed state and the Confirm Method is optional.	✓
A & C Exclusive Alarming Server Facet	This Facet requires support for Alarms with multiple sub-states that identify different limit Conditions. This facet builds upon the A&C Alarm Server Facet. The term exclusive means only one sub-state can be active at a time. For example, a temperature exceeds the HighHigh limit the associated exclusive LevelAlarm will be in the HighHigh sub-state and not in the High sub-state. This Facet requires that a Server support at least one of the optional Alarm models: Limit, RateOfChange or Deviation.	✗
A & C Non-Exclusive Alarming Server Facet	This Facet requires support for Alarms with multiple sub-states that identify different limit Conditions. This Facet builds upon the A&C	✗

	Alarm Server Facet. The term non-exclusive means more than one sub-state can be active at a time. For example, if a temperature exceeds the HighHigh limit the associated non-exclusive LevelAlarm will be in both the High and the HighHigh sub-state. This Facet requires that a server support at least one of the optional alarm models: Limit, RateOfChange or Deviation.	
A & C Previous Instances Server Facet	This Facet requires support for Conditions with previous states that still require action on the part of the operator. This Facet builds upon the A&C Base Condition Server Facet. A common use case for this Facet is a safety critical system that requires that all Alarms be acknowledged even if it the original problem goes away and the Alarm returns to the inactive state. In these cases, the previous state with active Alarm is still reported by the Server until the Operator acknowledges it. When a Condition has previous states, it will produce events with different Branch identifiers. When previous state no longer needs attention, the branch will disappear.	⊗
A & C Dialog Server Facet	This Facet requires support of Dialog Conditions. This Facet builds upon the A & C BaseCondition Server Facet Dialogs are ConditionTypes used to request user input. They are typically used when a Server has entered some state that requires intervention by a Client. For example, a Server monitoring a paper machine indicates that a roll of paper has been wound and is ready for inspection. The Server would activate a Dialog Condition indicating to the user that an inspection is required. Once the inspection has taken place the user responds by informing the Server of an accepted or unaccepted inspection allowing the process to continue.	⊗
A & C CertificateExpiration Server Facet	This Facet requires support of the CertificateExpirationAlarmType. It is used to inform Clients when the Server's Certificate is within the defined expiration period.	⊗
A & E Wrapper Facet	This Facet specifies the requirements for a UA Server that wraps an OPC Alarm & Event (AE) Server (COM). This Profile identifies the sub-set of the UA Alarm & Condition model which is provided by the COM OPC AE specification. It is intended to provide guidance to developers who are creating servers that front-end existing applications. It is important to note that some OPC A&E COM Servers may not support all of the functionality provided by an OPC UA A&C server, in these cases similar functionality maybe available via some non-OPC interface. For example, if an A&E COM server does not support sending Alarm Acknowledgement messages to the system that it is obtaining alarm information from, this functionality may be available via some out of scope features in the underlying Alarm system. Another possibility is that the underlying system does not require acknowledgements or automatically acknowledges the alarm.	⊗

2.5 Generic Features

Profile	Description	Supported
Method Server Facet	This Facet specifies the support of Method invocation via the Call service. Methods are “lightweight” functions which are similar to the methods of a class found in any object-oriented programming language. A Method can have its scope bounded by an owning Object or an owning ObjectType. Methods with an ObjectType as their scope are similar to static methods in a class.	✓
Auditing Server Facet	This Facet requires the support of Auditing which includes the Standard Event Subscription Server Facet. Support of this Facet requires that Audit Events be produced when a client performs some action to change the state of the server, such as changing the AddressSpace, inserting or updating a value etc. The auditEntryId passed by the Client is a field contained in every Audit Event and allows actions to be traced across multiple systems. The Audit Event Types and their fields must be exposed in the Server’s AddressSpace	✗
Node Management Server Facet	This Facet requires the support of the Services that allow the Client to add, modify and delete Nodes in the AddressSpace. These Services provide an interface which can be used to configure Servers. This means all changes to the AddressSpace are expected to persist even after the Client has disconnected from the Server	✗
User Role Base Server Facet	This Facet defines support of the OPC UA Information Model to expose configured user roles and permissions.	✗
User Role Management Server Facet	This Facet defines support of the OPC UA approach to manage user roles and permissions and to grant access to Nodes and Services based on the assigned roles and permissions.	✗
State Machine Server Facet	This Facet defines support of StateMachines based on the types in UA Part 5.	✗

2.6 Redundancy

Profile	Description	Supported
Client Redundancy Server Facet	This Facet defines the Server actions that are required for support of redundant Clients. Support of this Facet requires the implementation of the TransferSubscriptions Service which allows the transfer of Subscriptions from one Client's Session to another Client's Session.	⊗
Redundancy Transparent Server Facet	This Facet requires support for transparent redundancy. If Servers implement transparent redundancy, then the failover from one Server to another is transparent to the Client such that the Client is unaware that a failover has occurred; the Client does not need to do anything at all to keep data flowing. This type of redundancy is usually a hardware solution.	⊗
Redundancy Visible Server Facet	This Facet specifies the support for non-transparent redundancy. Failover for this type of redundancy requires the Client to monitor Server status and to switch to a backup Server if it detects a failure. The Server shall expose the methods of failover it supports (cold, warm or hot). The failover method tells the Client what it must do when connecting to a Server and when a failure occurs. Cold redundancy requires a Client to reconnect to a backup Server after the initial Server has failed. Warm redundancy allows a Client to connect to multiple Servers, but only one Server will be providing values. In hot redundancy multiple Servers are able to provide data and a Client can connect to multiple Servers for the data.	⊗

2.7 Historical Access

2.7.1 Historical Data

Profile	Description	Supported
Historical Raw Data Server Facet	This Facet defines the basic functionality when supporting historical data access for raw data.	✓
Historical Aggregate Server Facet	This Facet indicates that the server supports aggregate processing to produce derived values from raw historical data.	✓
Historical Data AtTime Server Facet	This Facet indicates that the historical Server supports reading data by specifying specific timestamps.	✓
Historical Access Modified Data Server Facet	This Facet defines support of reading modified historical values (values that where modified or inserted).	⊗
Historical Annotation Server Facet	This Facet defines support for the storage and retrieval of annotations for historical data.	⊗
Historical Data Insert Server Facet	This Facet includes Historical Data Insert functionality.	⊗
Historical Data Update Server Facet	This Facet includes Historical Data Update functionality.	⊗

Historical Data Replace Server Facet	This Facet includes Historical Data Replace functionality.	⊗
Historical Data Delete Server Facet	This Facet includes Historical Data Delete functionality.	⊗
Historical Access Structured Data Server Facet	This Facet indicates that the Server supports storage and retrieval of structured values for all supported access types. If a listed access type is supported, then the corresponding optional ConformanceUnit shall be supported.	⊗

2.7.2 Historical Events

Profile	Description	Supported
Base Historical Event Server Facet	This Facet defines the server requirements to support basic Historical Event functionality, including simple filtering and general access.	✓
Historical Event Update Server Facet	This Facet includes Historical Event update access functionality.	⊗
Historical Event Replace Server Facet	This Facet includes Historical Event replace access functionality.	⊗
Historical Event Insert Server Facet	This Facet includes Historical Event insert access functionality.	⊗
Historical Event Delete Server Facet	This Facet includes Historical Event delete access functionality	⊗

2.8 Aggregates

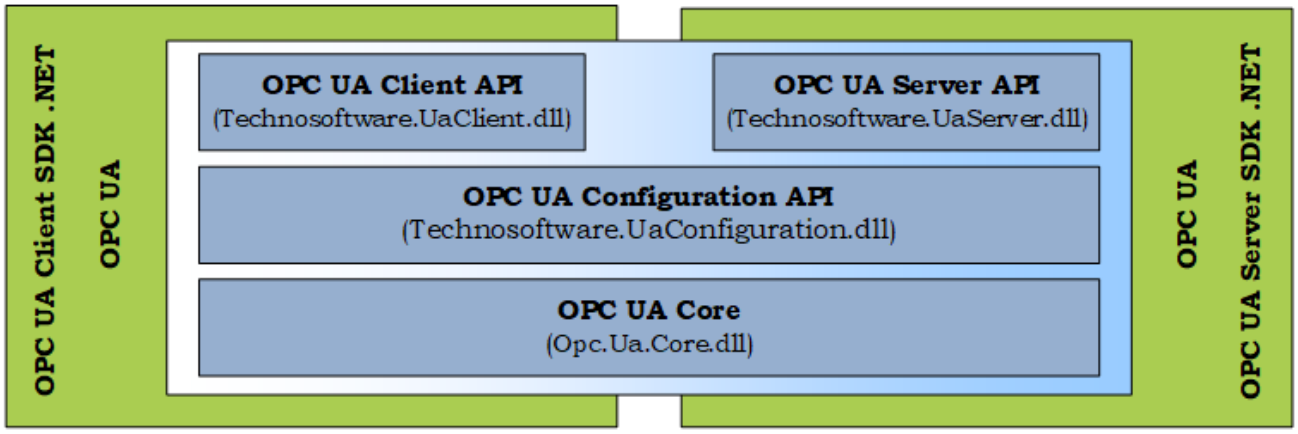
Profile	Description	Supported
Aggregate Subscription Server Facet	This Facet defines the handling of the aggregate filter when subscribing for Attribute values.	⊗

3 Sample Applications

The OPC UA Server SDK .NET Standard contains several sample server applications, but we concentrate in this tutorial on the *WorkshopServerForms*, a console base application for testing the server specific features, and the *WorkshopServerConsole*. This tutorial will refer to that code while explaining the different steps to take to accomplish the main tasks of an OPC UA server.

3.1 Required SDK DLLs

The SDK is splitted into several DLL's as shown in the picture below:



The OPC UA Server SDK .NET Standard uses the following DLL's:

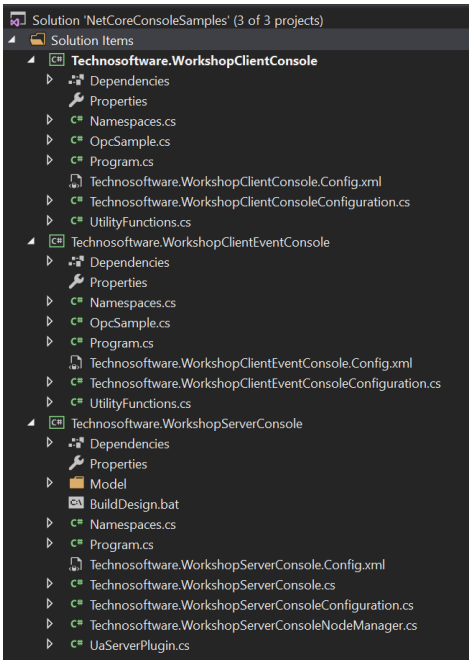
Name	Description
Opc.Ua.Core.dll	The OPC UA Stack. Based on the OPC Unified Architecture .NET Standard. We deliver three versions, one for .NET 4.6.2, one for .NET 4.7.2 and one for .NET Standard 2.0.
Technosoftware.UaConfiguration.dll	Contains configuration related classes like, e.g. `ApplicationInstance`. We deliver three versions, one for .NET 4.6.2, one for .NET 4.7.2 and one for .NET Standard 2.0.
Technosoftware.UaServer.dll	The DLL containing the classes and methods usable for OPC UA Server development. We deliver three versions, one for .NET 4.6.2, one for .NET 4.7.2 and one for .NET Standard 2.0.

3.2 Directory Structure

The basic directory layout is as follows:

- **bin/**
  - **net462/**
    Standard SDK Executables and DLL's for the .NET 4.6.2 Framework
  - **net472/**
    Standard SDK Executables and DLL's for the .NET 4.7.2 Framework
  - **netstandard2.0/**
    Standard SDK Executables and DLL's for the .NET Standard 2.0 and .NET Core 2.0 Framework
  - **redist/**
    - **OPC UA Local Discovery Server 1.03/**
      The installer and Merge-Module for the OPC UA Local Discovery Server
- **doc/**
  Additional documentation like the compiled HTML Help files for Server and Client SDK.
  - **pdf/**
    Several documentation files. The more important ones here are:
    - **OPC_UA_NET_Standard_Installation_Guide.pdf**
      This document
    - **OPC_UA_SDKs_NET_Standard_Introduction.pdf**
      Introduction in Developing OPC UA Clients and OPC UA Servers with C# / VB.NET
    - **OPC_UA_Client_Development_with_NET_Standard.pdf**
      Tutorial for Developing OPC UA Clients with C# / VB.NET based on the Workshop Client
    - **OPC_UA_Server_Development_with_NET_Standard.pdf**
      Tutorial for Developing OPC UA Servers with C# / VB.NET based on the Workshop Server
- **examples/**
  Sample applications
  - **Workshop/**
    - **ServerConsole/**
      Server as Console application used for this introduction. Features .NET 4.6.2, .NET 4.7.2 and .NET Core 2.0 compilation within one solution.
    - **ServerForms/**
      Server with Windows Forms used for this introduction
- **keys/**
  The dummy Key for signing the executables and DLL's
- **schema/**
  XSD files like the UAModelDesign.xsd used for the Model Designer
- **scripts/**
  Scripts and executables used for building the applications

3.3 OPC UA Server Solution for .NET Core 2.0



The main OPC UA Server Solution can be found at `examples\Workshop\` and are named

- NetCoreConsoleSamples.sln

The solution contains two sample clients, as well as the server example used required by these clients.

These examples build for .NET Core 2.0 and can be used on Windows, Linux, macOS.

Please follow instructions in this article to setup the dotnet command line environment for your platform. As of today, .Net Standard 2.0 is required. The article describes the installation of .NET SDK 2.2.401 for Windows, Linux and macOS. This version also works with the OPC UA Client and Server Solutions.

Please follow at least the sections

- Intro
- Download and Install

to install the .NET Core.

3.3.1 Prerequisites

Once the ***`dotnet`*** command is available, navigate to the following folder:

```
examples\Workshop
```

and execute

```
dotnet restore NetCoreConsoleSamples.sln
```

This command restores the tree of dependencies.

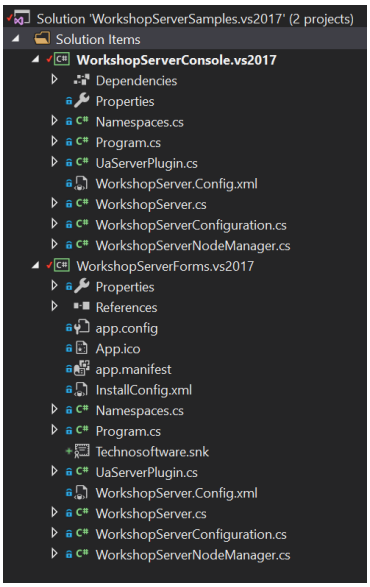
3.3.2 Start the server

1. Open a command prompt.
2. Navigate to the folder `examples\Workshop\ServerConsole`.
3. To run the server sample type
   `dotnet run --project Technosoftware.WorkshopServerConsole.csproj -a`.
   - The server is now running and waiting for connections.
   - The `-a` flag allows to auto accept unknown certificates and should only be used to simplify testing.

3.3.3 Start the client

1. Open a command prompt
2. Navigate to the folder `examples\Workshop\ClientConsole`.
3. To run the client sample type
   `dotnet run --project Technosoftware.WorkshopClientConsole.csproj -a`
   - The client connects to the OPC UA console sample server running on the same host.
   - The `-a` flag allows to auto accept unknown certificates and should only be used to simplify testing.
4. If not using the -a auto accept option, on first connection, or after certificates were renewed, the server may have refused the client certificate. Check the server and client folder `%LocalApplicationData%/OPC Foundation/pki/rejected` for rejected certificates. To approve a certificate copy it to the `%LocalApplicationData%/OPC Foundation/pki/trusted`.

3.4 OPC UA Server Solution .NET 4.6.2 / 4.7.2



The main OPC UA Server Solutions can be found at \examples\Workshop\ and are named

- Visual Studio 2017:
  WorkshopServerSamples.vs2017.sln
- Visual Studio 2019:
  WorkshopServerSamples.vs2019.sln

and uses in addition the output of the following solutions:

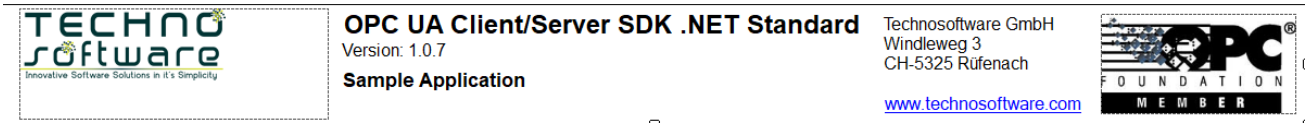
1. Technosoftware.CommonControls
   Contains the ExceptionDialog and the TitleBarControl.
2. Technosoftware.ServerControls
   Contains the ServerDiagnosticsControl and the ServerForm.

The solution contains two sample servers, one a console-based server and one a Windows Forms based server. The OPC UA functionality of both is the same.

3.4.1 CommonControls

3.4.1.1 Customizing the TitleBarControl

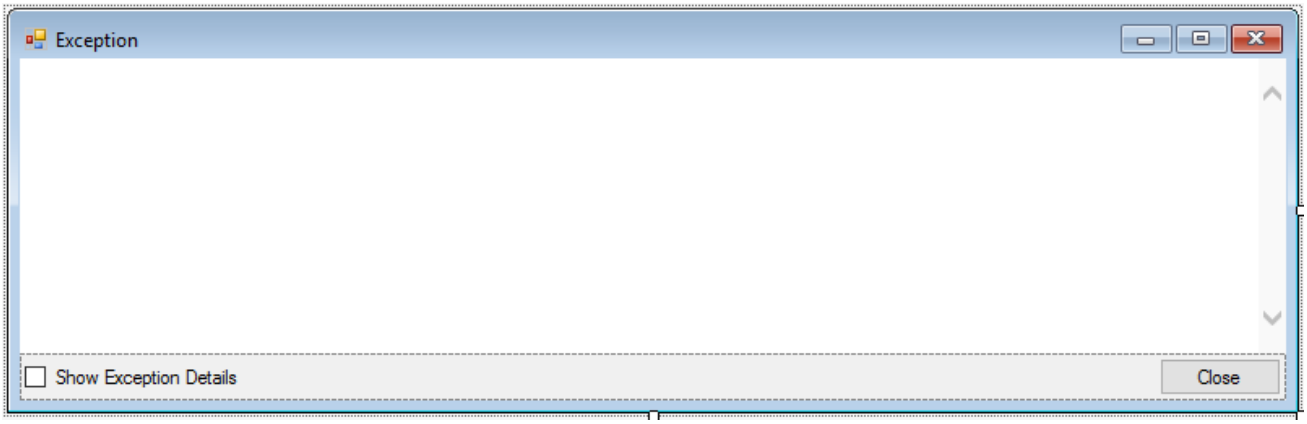
The TitleBarControl contains the header of all Windows Forms based sample server and sample client solutions provided with the SDK, e.g. WorkshopServerForms. The following picture shows how it looks like as default:



By changing this control, you can adapt the layout of the WorkshopServerForms to your needs.

3.4.1.2 Customizing the ExceptionDlg

The ExceptionDlg is used for displaying exceptions. The following picture shows how it looks like as default:

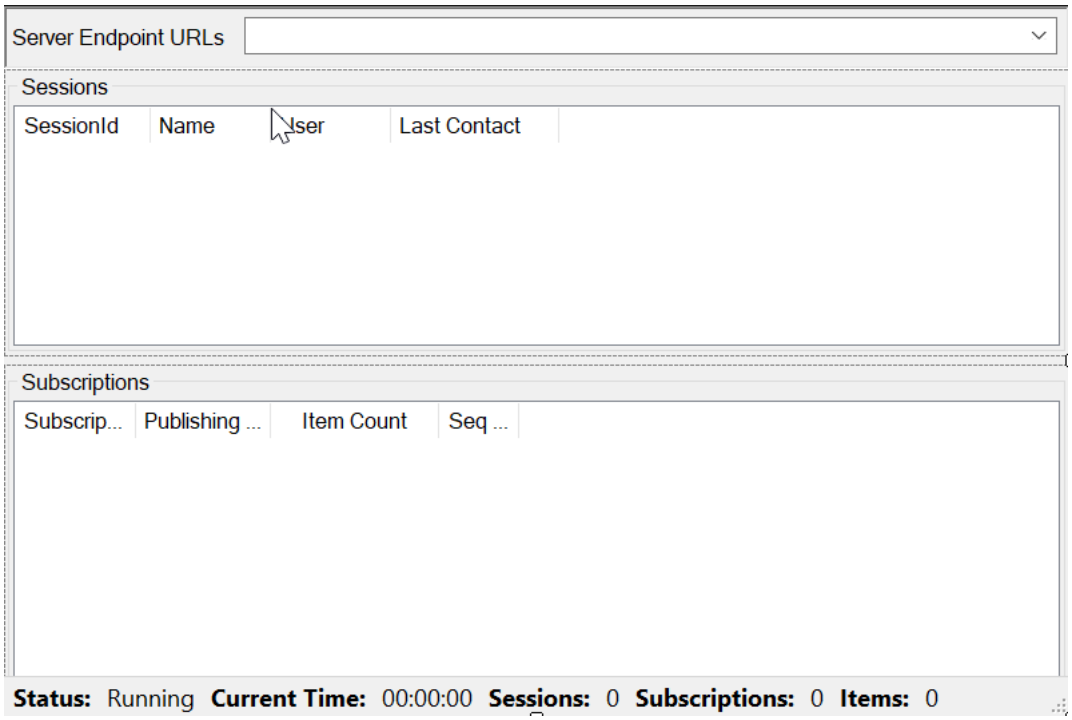


By changing this dialog, you can adapt the layout of the exception dialog for the WorkshopServerForms to your needs.

3.4.2 ServerControls

3.4.2.1 Customizing the ServerDiagnosticControl

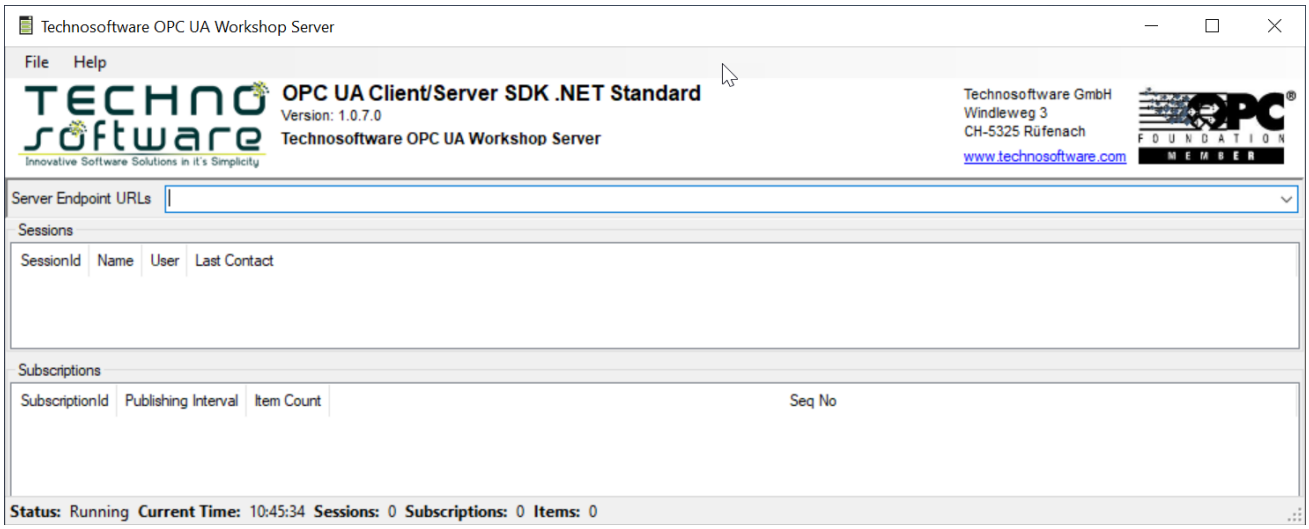
The ServerDiagnosticControl display information about the active sessions and subscriptions made by one or more OPC UA Clients. The following picture shows how it looks like as default:



By changing this dialog, you can adapt the layout of this control for the WorkshopServerForms to your needs.

3.4.2.2 Customizing the ServerForm

The ServerForm combines the different controls to the main dialog used by the WorkshopServerSample. The following picture shows how it looks like as default:



By changing this dialog, you can adapt the layout of this form for the WorkshopServerForms to your needs.

4 Configuration

4.1 Application Configuration

The SDK provides an extensible mechanism for storing the application configuration in an XML file. The class is extensible so developers can add their own configuration information to it. The table below describes primary elements of the ApplicationConfiguration class.

Name	Type	Description
ApplicationName	String	A human readable name for the application.
ApplicationUri	String	A globally unique name for the application. This should be a URL with which the machine domain name or IP address as the hostname followed by the vendor/product name followed by an instance identifier. For example: http://machine1/OPC/UASampleServer/4853DB1C-776D-4ADA-9188-00CAA737B780
ProductUri	String	A human readable name for the product.
ApplicationType	ApplicationType	The type of application. Possible values: Server_0, Client_1, ClientAndServer_2 or DiscoveryServer_3
SecurityConfiguration	SecurityConfiguration	The security configuration for the application. Specifies the application instance certificate, list of trusted peers and trusted certificate authorities.
TransportConfigurations	TransportConfiguration Collection	Specifies the Bindings to use for each transport protocol used by the application.
TransportQuotas	TransportQuotas	Specifies the default limits to use when initializing WCF channels and endpoints.
ServerConfiguration	ServerConfiguration	Specifies the configuration for Servers
ClientConfiguration	ClientConfiguration	Specifies the configuration for Clients
TraceConfiguration	TraceConfiguration	Specifies the location of the Trace file. Unexpected exceptions that are silently handled are written to the trace file. Developers can add their own trace output with the Utils.Trace(…) functions.
Extensions	XmlElementCollection	Allows developers to add additional information to the file.
MessageContext	ServiceMessageContext	The context to use when serializing/deserializing messages.
CertificateValidator	CertificateValidator	This is the custom certificate validator used by the application.

The ApplicationConfiguration can be persisted anywhere but the class provides functions that load/save the configuration as an XML file on disk. The location of the XML file can be specified in the app.config file for the application if the ConfigurationLocation is specified as a configuration section.

The declaration for the configuration section in the app.config looks like this:

```
<configSections>
       <section name="WorkshopServer" type="Opc.Ua.ApplicationConfigurationSection,Opc.Ua.Core"/>
</configSections>
```

The name may be any text that is unique within the app.config file. The ConfigurationLocation would look like this:

```
<WorkshopServer>
       <ConfigurationLocation xmlns="http://opcfoundation.org/UA/SDK/Configuration.xsd">
              <FilePath>WorkshopServer.Config.xml</FilePath>
       </ConfigurationLocation>
</WorkshopServer>
```

The FilePath can be an absolute path or a relative path. If it is a relative path the current directory is searched followed by the directory where the executable resides. The SDK also supports prefixes which can be replaced with environment variables. The latter functionality requires a token enclosed by '%' symbols at the start of the message. The SDK will first check for a symbol that matches one of the values from the Environment.SpecialFolder enumeration. If not found it will use the environment variable of the same name.

Note that the same feature exists for all fields that represent file directory paths in the ApplicationConfiguration object.

The Application Configuration file of the WorkshopServerSample can be found in the file WorkshopServerSample.Config.xml.

4.1.1 Extensions

The Application Configuration file of the WorkshopServerForms uses the Extensions feature to make the Excel Configuration configurable.

Name	Type	Description
ConfigurationFile	String	The full path including file name of the Excel file used for the configuration of the address space.

The Extension looks like:

```
<Extensions>
  <ua:XmlElement>
    <WorkshopServerConfiguration xmlns="http://technosoftware.com/WorkshopServer">
      <ConfigurationFile>.\WorkshopServerConfiguration.xlsx</ConfigurationFile>
    </WorkshopServerConfiguration>
  </ua:XmlElement>
</Extensions>
```

To get the configuration value the WorkshopServerSample uses the following calls:

```
// get the configuration for the node manager. In case no configuration exists
// use suitable defaults.
configuration_ = configuration.ParseExtension<WorkshopServerConfiguration>() ??
                 new WorkshopServerConfiguration();

string configurationFile = configuration_.ConfigurationFile;
```

Important:
This only shows how to use the Extension feature. The Excel based configuration is not implemented at all.

4.1.2 Tracing Output

With the TraceConfiguration UA client and server applications can activate trace information.
TechnosoftwareUaClient and TechnosoftwareUaClientSample creates the following logfiles:

WorkshopServerSample:
%CommonApplicationData%\Technosoftware\Logs\WorkshopServer.log.txt

where

%CommonApplicationData% typically points to C:\ProgramData

4.2 Installed Application

Important:
The following feature is only available with .NET 4.6.2 and .NET 4.7.2. .NET Standard 2.0 doesn't support this.

The SDK provides an installation configuration mechanism for installing/uninstalling an application. For this an InstallConfig.xml file containing the InstalledApplication class definition should be added to your project as embedded resource. The table below describes some of the primary elements of the InstalledApplication class.

Name	Type	Description
ConfigureFirewall	Boolean	Specifies whether the firewall should be configured. True if the firewall should be configured; false otherwise.
DeleteCertificatesOnUninstall	Boolean	Specifies whether the certificates should be deleted if the application gets uninstalled. True if the certificates should be deleted; false otherwise.
InstallAsService	Boolean	Specifies whether the application should be installed as service. True if the application should be installed as service; false otherwise.
ServiceStartMode	StartMode	Specifies how the service start mode should be configured. Possible values: Boot, System, Auto, Manual or Disabled
ServiceUserName	String	Specifies the username of the user used for running the application as service.
ServicePassword	String	Specifies the password of the user used for running the application as service.
ServiceDescription	String	Specifies the description for the service.

The Installed Application file of the WorkshopServerForms example can be found in the file InstallConfig.xml and looks like:

```
<?xml version="1.0" encoding="utf-8" ?>
<s0:InstalledApplication
  xmlns:s0="http://opcfoundation.org/UA/SDK/Installation.xsd"
  xmlns:s1="http://opcfoundation.org/UA/SDK/Configuration.xsd"
  xmlns="http://opcfoundation.org/UA/2011/03/SecuredApplication.xsd"
  xmlns:ua="http://opcfoundation.org/UA/2008/02/Types.xsd">

  <ApplicationName>Technosoftware OPC UA Workshop Server Forms</ApplicationName>
  <ApplicationUri></ApplicationUri>
  <ApplicationType>Server_0</ApplicationType>
  <ConfigurationFile>Technosoftware.WorkshopServerForms.Config.xml</ConfigurationFile>

  <s0:DeleteCertificatesOnUninstall>true</s0:DeleteCertificatesOnUninstall>
  <s0:ConfigureFirewall>true</s0:ConfigureFirewall>
  <s0:SetConfigurationFilePermisions>false</s0:SetConfigurationFilePermisions>
  <s0:SetExecutableFilePermisions>false</s0:SetExecutableFilePermisions>
  <s0:InstallAsService>false</s0:InstallAsService>
  <s0:ServiceStartMode>Auto</s0:ServiceStartMode>
  <s0:ServiceDescription>Technosoftware OPC UA Workshop Server Forms</s0:ServiceDescription>

  <s0:TraceConfiguration>

<s1:OutputFilePath>%CommonApplicationData%\Technosoftware\Logs\Technosoftware.WorkshopServerForms.
InstallLog.log</s1:OutputFilePath>
    <s1:DeleteOnLoad>true</s1:DeleteOnLoad>
    <!-- Show Only Errors -->
    <!-- <s1:TraceMasks>1</s1:TraceMasks> -->
    <!-- Show Only Security and Errors -->
    <!-- <s1:TraceMasks>513</s1:TraceMasks> -->
    <!-- Show Only Security, Errors and Trace -->
    <s1:TraceMasks>515</s1:TraceMasks>
    <!-- Show Only Security, COM Calls, Errors and Trace -->
    <!-- <s1:TraceMasks>771</s1:TraceMasks> -->
    <!-- Show Only Security, Service Calls, Errors and Trace -->
    <!-- <s1:TraceMasks>523</s1:TraceMasks> -->
    <!-- Show Only Security, ServiceResultExceptions, Errors and Trace -->
    <!-- <s1:TraceMasks>519</s1:TraceMasks> -->
  </s0:TraceConfiguration>

</s0:InstalledApplication>
```

5 Certificate Management and Validation

The stack provides several certificate management functions including a custom `CertificateValidator` that implements the validation rules required by the specification. The `CertificateValidator` is created automatically when the ApplicationConfiguration is loaded. Any WCF channels or endpoints that are created with that ApplicationConfiguration will use it.

The `CertificateValidator` uses the trust lists in the ApplicationConfiguration to determine whether a certificate is trusted. A certificate that fails validation is always placed in the Rejected Certificates store. Applications can receive notifications when an invalid certificate is encountered by using the event defined on the `CertificateValidator` class.

The Stack also provides the `CertificateIdentifier` class which can be used to specify the location of a certificate. The Find() method will look up the certificate based on the criteria specified (SubjectName, Thumbprint or DER Encoded Blob).

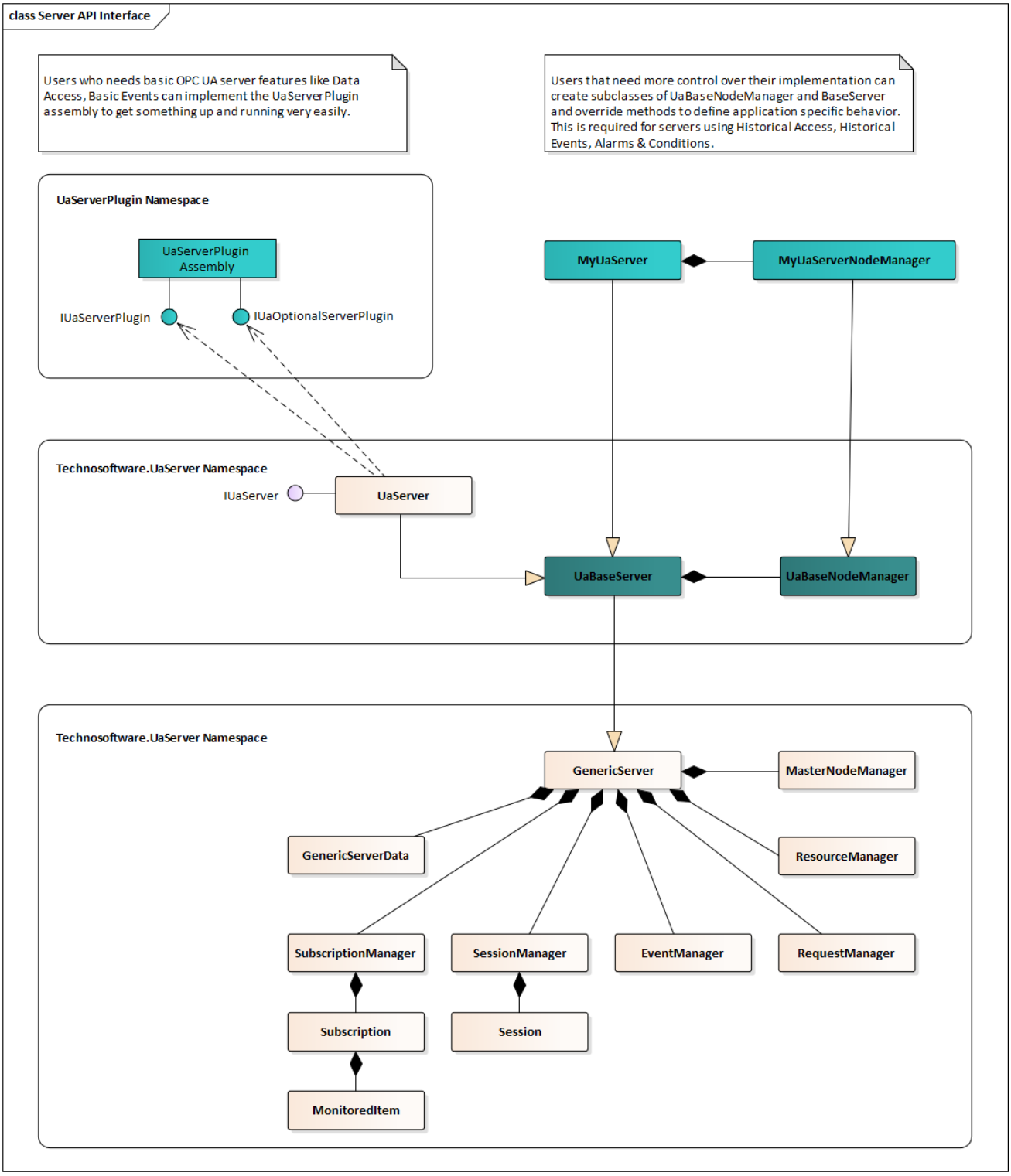
Each application has a SecurityConfiguration which must be managed carefully by the Administrator since making a mistake could prevent applications from communicating or create security risks. The elements of the SecurityConfiguration are described in the table below:

Name	**Description**
ApplicationCertificate	Specifies where the private key for the Application Instance Certificate is located. Private keys should be in the Personal (My) store for the LocalMachine or the CurrentUser. Private keys installed in the LocalMachine store are only accessible to users that have been explicitly granted permissions.
TrustedIssuerCertificates	Specifies the Certificate Authorities that issue certificates which the application can trust. The structure includes the location of a Certificate Store and a list of individual Certificates.
TrustedPeerCertificates	Specifies the certificates for other applications which the application can trust. The structure includes the location of a Certificate Store and a list of individual Certificates.
InvalidCertificateDirectory	Specifies where rejected Certificates can be placed for later review by the Admistrator (a.k.a. Rejected Certificates Store)

The Administrator needs to create an application instance certificate when applications are installed, when the ApplicationUri or when the hostname changes. The Administrator can use the OPC UA Configuration Tool included in the SDK or use the tools provided by their Public Key Infrastructure (PKI). If the certificate is changed the Application Configuration needs to be updated.

Once the certificate is installed the Administrator needs to ensure that all users who can access the application have permission to access the Certificate's private key.

6 UA Server Design



The Server API is designed to ease the server development by handling the standard tasks which all servers need to do and provides APIs that allow users to add their own functionality. These APIs fall into the following categories:

- The first level, the Core Layer, implements all common code necessary for an OPC UA Server and manages communication connection infrastructure like `GenericServer`, `GenericServerData`, `MasterNodeManager`, `ResourceManager`, `SubscriptionManager`, `SessionManager`, `EventManager` and `RequestManager`.
- The second level, the Base Layer are interfaces and implementations like `UaBaseServer` and `UaBaseNodeManager` for information integration and mapping of the OPC UA defined services. It includes a standard implementation for the Base Layer and require that the user creates subclasses of the classes defined here.
- The third level, the Plugin Layer are interfaces and implementations like `IUaServerPlugin` and `IUaOptionalServerPlugin` which allows a very quick and easy implementation of an OPC UA Server. The user can start implementing a base OPC UA server supporting Data Access and Simple Events and later enhance it through adding subclasses of the classes defined in the Base Layer.

The Core Layer classes are used in user applications and tools and should not be changed or subclassed. The Base Layer classes can be subclassed and extended by your application. The Plugin Layer can be used by implementing the interfaces `IUaServerPlugin` and optionally `IUaOptionalServerPlugin`.

6.1 Simple OPC UA Servers

Important: The generic server executable feature is not yet supported and will be available later for .NET 4.6.2 and .NET 4.7.2 only.

6.1.1 Overview

The server consists of two main interfaces:

- The generic OPC UA server (`IUaServer` interface, generic server executable)
- The customization (`IUaServerPlugin`, UaServerPlugin.dll)

These two interfaces can be used for simple OPC UA servers using Data Access or Simple Events features.

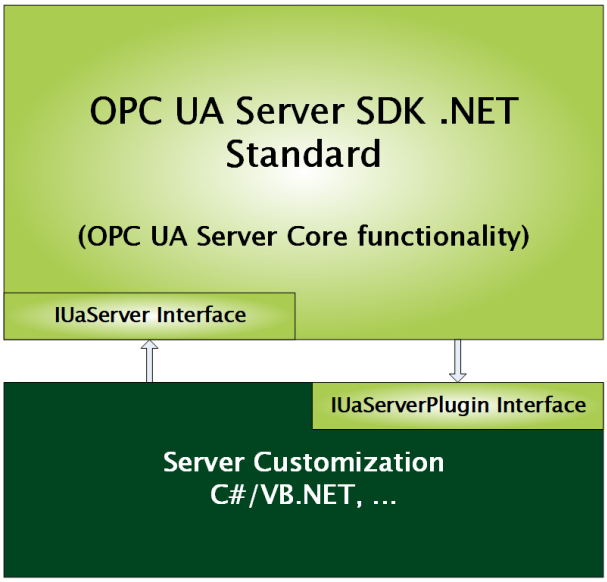
The generic server manages the OPC UA client interfaces and the internal data cache. The main class is the `UaServer` class which handles the interaction with the customization DLL and instances a `UaBAseServer` object.

The customization part handles the device interface. The 'device' doesn't need to be a hardware component, it can also be a database or another software application.

A set of customization interface methods handles:

- configuration
- the data exchange
- item update coordination

between the generic and the customization part.

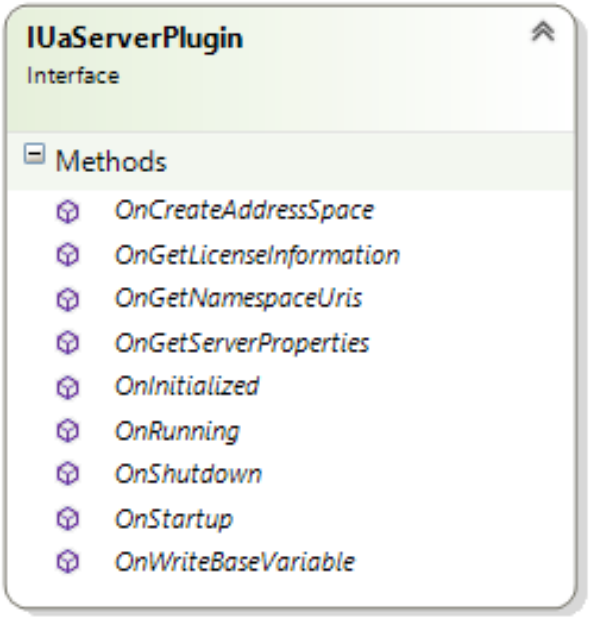


6.1.2 Customization

The `UaServer` loads a specific Customization Assembly which implements the `IUaServerPlugin` interface. The Customization Assembly can call methods of the generic server executable via the `IUaServer` interface.

The `UaServer` calls some methods at startup:

1. OnStartup
2. OnGetLicenseInformation
3. OnGetNamespaceUris
4. OnCreateAddressSpace
5. OnInitialized
6. OnGetServerProperties
7. OnRunning



and some when a client calls certain functions:

1. OnWriteBaseVariable

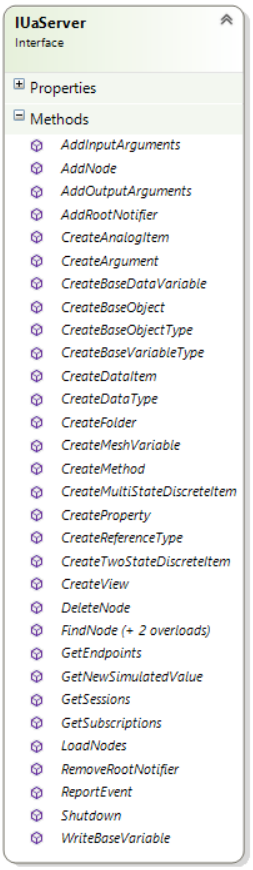
Whenever the server shutdowns the

1. OnShutdown

is called.

Callback functions allow the customization to get status information from the generic server or change data in the generic server. The most important callback methods ar the different Create item methods, e.g.:

1. CreateDataItem
2. CreateFolder
3. CreateAnalogItem
4. CreateDataType
5. CreateReferenceType
6. CreateProperty



Node specific handling is done via:

1. AddNode allows adding of a node to the address space
2. DeleteNode allows deleting a previously adsed node from the address space
3. FindNode allows searching a node in the address space

Updating node information like value and timestamp is done via:

1. WriteBasevariable allows updating the value of a node

6.2 Full featured OPC UA Servers

6.2.1 Overview

The Base Layer consists of interfaces and implementations like `UaBaseServer` and `UaBaseNodeManager` for information integration and mapping of the OPC UA defined services. It includes a standard implementation for the Base Layer and require that the user creates subclasses of the classes defined here. It gets used if the `IUaOptionalServerPlugin` is implemented in the Customization assembly and includes subclasses of

- UaBaseNodeManager and/or
- UaBaseServer

6.2.2 Customization

The `UaServerPlugin` class can return a custom `UaBaseNodeManager` and/or `UaBaseServer` as shown in the sample below. Most of the time using a custom `UaBaseNodeManager` will be enough.

UaServerPlugin.cs:

```
public class UaServerPlugin : IUaServerPlugin, IUaOptionalServerPlugin, IDisposable
{
    #region Optional Server Plugin methods
    public UaBaseServer OnGetServer()
    {
        return null;
    }

    public UaBaseNodeManager OnGetNodeManager(IUaServer opcServer,
                               IUaServerData uaServer,
                               ApplicationConfiguration configuration,
                               params string[] namespaceUris)
    {
        return new WorkshopServerFormsNodeManager(opcServer, this, uaServer,
                               configuration, namespaceUris);
    }
    #endregion
}
```

Using a custom `UaBaseNodeManager` means that the `OnCreateAddressSpace` method of the UaServerPlugin class is no longer called. Instead the custom must `UaBaseNodeManager` override the `CreateAddressSpace` method as shown below.

6.2.2.1 Using a Custom UaBaseNodeManager

Technosoftware.WorkshopServerFormsNodeManager.cs:

```
public class WorkshopServerFormsNodeManager : UaBaseNodeManager
{
    public override void CreateAddressSpace(IDictionary<NodeId,
                                            IList<IReference>> externalReferences)
    {
        lock (Lock)
        {
            dynamicNodes_ = new List<BaseDataVariableState>();

            IList<IReference> references;

            if (!externalReferences.TryGetValue(ObjectIds.ObjectsFolder, out references))
            {
                externalReferences[ObjectIds.ObjectsFolder] =
                                            References = new List<IReference>();
            }
            else
            {
                References = references;
            }

            LoadPredefinedNodes(SystemContext, externalReferences);

            FolderState root = opcServer_.CreateFolder(null, "My Data", "My Data");

        }
    }
}
```

The `UaBaseNodeManager` offers several methods to be overriden which can be grouped as follows:

AddressSpace related

1. CreateAddressSpace
2. DeleteAddressSpace
3. GetManagerHandle

Node related

1. Write
2. ValidateNode
3. Read

Method related

1. Call

Events related

1. SubscribeToEvents
2. SubscribeToAllEvents

HistoricalAccess related

1. HistoryUpdateData
2. HistoryUpdateStructureData
3. HistoryDeleteRawModified
4. HistoryDeleteAtTime
5. HistoryReleaseContinuationPoints
6. HistoryReadRawModified
7. HistoryReadProcessed
8. HistoryReadAtTime

HistoricalEvents related

1. HistoryUpdateEvents
2. HistoryDeleteEvents
3. HistoryReadEvents

6.2.2.2 Using a Custom UaBaseServer

In some cases, it might be usefull to use your own custom UaBaseServer implementation. For example, for usage of the authentication features.

UaServerPlugin.cs:

```
public class UaServerPlugin : IUaServerPlugin, IUaOptionalServerPlugin, IDisposable
{
    #region Optional Server Plugin methods
    public UaBaseServer OnGetServer()
    {
        return new WorkshopServerForms();
    }

    public UaBaseNodeManager OnGetNodeManager(IUaServer opcServer,
                                IUaServerData uaServer,
                                ApplicationConfiguration configuration,
                                params string[] namespaceUris)
    {
        return new WorkshopServerFormsNodeManager(opcServer, this, uaServer,
                                configuration, namespaceUris);
    }
    #endregion
}
```

Technosoftware.WorkshopServerForms.cs:

```
public class WorkshopServerForms : UaBaseServer
{
    #region User Validation Functions
    /// <summary>
    /// Called when a client tries to change its user identity.
    /// </summary>
    private void OnImpersonateUser(object sender, UaImpersonateUserEventArgs args)
    {
        Session session = (Session)sender;
        // check for a user name token.
        UserNameIdentityToken userNameToken = args.NewIdentity as UserNameIdentityToken;
        if (userNameToken != null)
        {
            args.Identity = VerifyPassword(userNameToken);
            return;
        }
    }
```

```
        /// <summary>
        /// Validates the password for a username token.
        /// </summary>
        private IUserIdentity VerifyPassword(UserNameIdentityToken userNameToken)
        {
            var userName = userNameToken.UserName;
            var password = userNameToken.DecryptedPassword;
            if (String.IsNullOrEmpty(userName))
            {
                // an empty username is not accepted.
                throw ServiceResultException.Create(StatusCodes.BadIdentityTokenInvalid,
                    "Security token is not a valid username token. An empty username is not
accepted.");
            }

            if (String.IsNullOrEmpty(password))
            {
                // an empty password is not accepted.
                throw ServiceResultException.Create(StatusCodes.BadIdentityTokenRejected,
                    "Security token is not a valid username token. An empty password is not
accepted.");
            }

            // verify operator and administrator users
            if (!((userName == "operator" && password == "password1") ||
                 (userName == "administrator" && password == "password2")))
            {
                // construct translation object with default text.
                TranslationInfo info = new TranslationInfo(
                    "InvalidPassword",
                    "en-US",
                    "Invalid username or password.",
                    userName);

                // create an exception with a vendor defined sub-code.
                throw new ServiceResultException(new ServiceResult(
                    StatusCodes.BadUserAccessDenied,
                    "InvalidPassword",
                    LoadServerProperties().ProductUri,
                    new LocalizedText(info)));
            }

            return new UserIdentity(userNameToken);
        }
        #endregion

    }
```

The `UaBaseServer` offers several methods to be overriden, like:

1. OnValidateOperationRequest
2. OnOperationRequestComplete

6.3 Use of the different Methods

6.3.1 Server Startup / Shutdown

The main instance of an OPC UA Server is provided by the UaServer class and the main customization methods called by the UaServer are in the UaServerPlugin class. Within the main entry point of your server, e.g. Program.cs, you have to declare above classes, e.g.:

```
private static readonly UaServer uaServer_ = new UaServer();
private static readonly UaServerPlugin uaServerPlugin_ = new UaServerPlugin();
```

To start the OPC UA Server you only need to call the Start() method like shown below:

```
try
{
    uaServer_.Start(uaServerPlugin_, "WorkshopServer", args);
}
catch (Exception e)
{
    ExceptionDlg.Show("WorkshopServer", e);
}
```

The UaServer now starts up and calls the following methods defined in the UaServerPlugin.cs:

6.3.2 OnStartup

This method is the first method called from the generic server at the startup. It provides the possibility to use a custom specific argument handling or even inform the UaServer that it should stop starting by returning an error code. The base implementation looks like:

```
public StatusCode OnStartup(string[] args)
{
    return StatusCodes.Good;
}
```

6.3.3 OnGetLicenseInformation

This method is called from the generic server to get the license information. For the WorkshopServer it looks like:

```
public void OnGetLicenseInformation(out string licenseOwner, out string serialNumber)
{
    licenseOwner = "";
    serialNumber = "";
}
```

This turns the SDK into a full product version. Returning empty strings here results in an evaluation version of the server.

6.3.4 OnGetServer

To be able to use advanced features of the SDK the WorkshopServerSample returns here an instant of the `WorkshopServerSample` which then can override some methods of the `UaBaseServer` class and looks like:

```
public UaBaseServer OnGetServer()
{
    return new WorkshopServer();
}
```

One sample for overriding is the implementation os user authentication.

6.3.5 OnGetServerProperties

This method returns some standard properties of the OPC UA Server and looks like:

```
public ServerProperties OnGetServerProperties()
{
    var properties = new ServerProperties
                         {
                             ManufacturerName = "Technosoftware GmbH",
                             ProductName = "Technosoftware OPC UA Workshop Server Sample",
                             ProductUri = "http://technosoftware.com/WorkshopServer/v1.0",
                             SoftwareVersion = GetAssemblySoftwareVersion(),
                             BuildNumber = GetAssemblyBuildNumber(),
                             BuildDate = GetAssemblyTimestamp()
                         };

    return properties;
}
```

6.3.6 OnGetNamespaceUris

This method returns the namespaces used by the application. Typically, only one namespace is returned here and there is no need to change the default implementation for the WorkshopServerSample. It looks like:

```
public string[] OnGetNamespaceUris()
{
    // set one namespace for the type model.
    var namespaceUrls = new string[1];
    namespaceUrls[0] = Namespaces.WorkshopServer;
    return namespaceUrls;
}
```

6.3.7 OnGetNodeManager

To be able to use advanced features of the SDK the WorkshopServerSample returns here an instant of the `WorkshopServerSampleNodeManager` which then can override some methods of the `UaBaseNodeManager` class and looks like:

```
public UaBaseNodeManager OnGetNodeManager(IUaServer opcServer, IUaServerData uaServer,
                                          ApplicationConfiguration configuration,
                                          params string[] namespaceUris)
{
    return new WorkshopServerNodeManager(opcServer, this, uaServer, configuration,
                                         namespaceUris);
}
```

The `WorkshopServerSampleNodeManager` is the main place to change the behavior of the UA Server like creating address space, handling client writes to data points as well as handling communication to the underlying machine.

6.3.8 OnInitialized

This method is called after the node manager is initialized. So if something fails in the WorkshopServerSampleNodeManager this method may not be reached. It looks like:

```
public void OnInitialized(IUaServer opcServer, ApplicationConfiguration configuration)
{
    opcServer_ = opcServer;
}
```

6.3.9 OnRunning

This method is called from the generic server when the server was successfully started and is running. In the standard implementation this is the place where the UI is initialized and started up:

```
public StatusCode OnRunning()
{
    // Initialize the user interface.
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    // run the application interactively.
    Application.Run(new ServerForm(opcServer_));

    return StatusCodes.Good;
}
```

6.3.10 OnShutdown

This method is called from the generic server when a Shutdown is executed. To ensure proper process shutdown, any communication channels should be closed and all threads terminated before this method returns. The standard implementation looks like:

```
public StatusCode OnShutdown(ServerState serverState, string reason, Exception exception)
{
    return StatusCodes.Good;
}
```

6.4 Address Space Creation

The creation of the address space is done in the WorkshopServerNodeManager.cs in the method CreateAddressSpace. Please check the source code to see how different types of nodes are created. The next chapter gives just some hints about this topic.

6.4.1 Creating variables in the address space

Variables are created always in the following sequence:

1) Create the root folder

```
FolderState root = opcServer_.CreateFolder(null, "My Data", "My Data");
References.Add(new NodeStateReference(ReferenceTypes.Organizes, false, root.NodeId));
root.EventNotifier = EventNotifiers.SubscribeToEvents;
opcServer_.AddRootNotifier(root);
```

2) Create the sub folder

```
// create the sub folder
excelDataFolder = opcServer_.CreateFolder(rootFolder, folderStructure,
                                          excelDataVariable.GetValue(strSubFolder));
```

3) Create the variable

```
BaseDataVariableState variable =
    CreateVariable(excelDataFolder,
                           excelDataPrefix + excelDataVariable.GetValue(strBrowseName),
                           excelDataVariable.GetValue(strBrowseName),
                           excelDataVariable.GetValue(strDataType),
                           ValueRanks.Scalar);
```

or

```
AnalogItemState analogVariable =
    (AnalogItemState)CreateAnalogItemVariable(excelDataFolder,
                         excelDataPrefix + excelDataVariable.GetValue(strBrowseName),
                         excelDataVariable.GetValue(strBrowseName),
                         excelDataVariable.GetValue(strDataType),
                         ValueRanks.Scalar);
```

4) Add the variable to a list of variables

```
// add the variable to the address space
variables.Add(variable);
```

or

```
// add the variable to the address space
variables.Add(analogVariable);
```

If a root folder or a sub folder has already been created do not recreate the same root or sub folder otherwise the reference of the initially created folder is lost and variables already added to these folders might be missing in the address space.

The type of variable to be used during variable creation depends on the presence of the engineering units. Variables without engineering units must be of type "`BaseDataVariableState`". Variables with engineering units must be of type "`AnalogItemState`" since only this type allows to assign engineering units.

After all variables are added to the list of variables all nodes and their children of each root folder have to be indexed. This is done at the end of the method CreateAddressSpace:

```
    // for all root folders recursively index the nodes and its children
    foreach (FolderState[] folder in folders.Values)
    {
        // for one root folders recursively index the nodes and its children
        // folder[0] is always the root folder
        AddPredefinedNode(SystemContext, folder[0]);
    }
```

After indexing all nodes and their children of all root folders the address space of the OPC-UA server is ready.

6.5 UserIdentity and UserIdentityTokens

The SDK provides the UserIdentity class which converts UA user identity tokens to and from the SecurityTokens used by WCF. The SDK currently supports

- Anonymous
- Username

user authentication.

6.5.1 ImpersonateUserEvent

The Workshop server main (WorkshopServer.cs) implements a basic user handling based on username / password combinations. For that it adds an event handler for the ImpersonateUserEvent:

```
protected override void OnServerStarted(IUaServerData server)
{
    base.OnServerStarted(server);

    // request notifications when the user identity is changed. all valid users are accepted
    // by default.
    server.SessionManager.ImpersonateUserEvent += OnImpersonateUser;

}


private void OnImpersonateUser(object sender, UaImpersonateUserEventArgs args)
{
    Session session = (Session)sender;
    // check for a user name token.
    UserNameIdentityToken userNameToken = args.NewIdentity as UserNameIdentityToken;

    if (userNameToken != null)
    {
        args.Identity = VerifyPassword(userNameToken);
        return;
    }
}
```

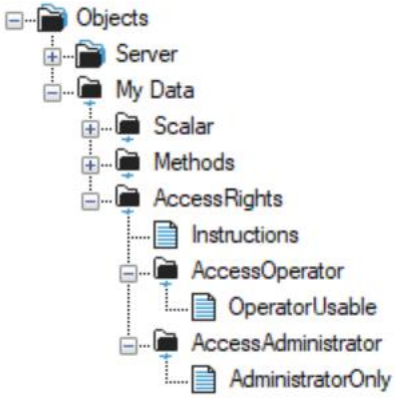
The following username/password combinations are supported:

- operator / password1
- administrator / password2

6.5.2 Address Space creation

The Workshop Server creates two variables under the AccessRights folder

- Operator or Administrator can write the following variable: ns=2;s=AccessRights_AccessOperator_OperatorUsable
- Administrator only can write the following variable: ns=2;s=AccessRights_AccessAdministrator_AdministratorOnly



The code for the user handling is in the WorkshopServerNodeManager and involves the following methods:

6.5.2.1 CreateAddressSpace() method

See region "Access Rights handling for creating the folders and nodes.

The user handling is done with the `OnReadUserAccessLevel` and in `OnSimpleWriteValue` event handling, e.g. for the administrator only this is done with:

```
#region Access Rights Administrator Handling
// sub-folder for "AccessAdministrator"
FolderState folderAccessRightsAccessAdministrator = opcServer_.CreateFolder(folderAccessRights,
"AccessRights_AccessAdministrator", "AccessAdministrator");
const string accessRightsAccessAdministrator = "AccessRights_AccessAdministrator_";

BaseDataVariableState arAdministratorRW = CreateVariable(folderAccessRightsAccessAdministrator,
accessRightsAccessAdministrator + "AdministratorOnly", "AdministratorOnly", BuiltInType.Int16,
ValueRanks.Scalar);
arAdministratorRW.AccessLevel = AccessLevels.CurrentReadOrWrite;
arAdministratorRW.UserAccessLevel = AccessLevels.CurrentReadOrWrite;
arAdministratorRW.OnReadUserAccessLevel = OnReadAdministratorUserAccessLevel;
arAdministratorRW.OnSimpleWriteValue = OnWriteAdministratorValue;
arAdministratorRW.OnReadValue = OnReadAdministratorValue;
dynamicNodes_.Add(arAdministratorRW);
#endregion
```

6.5.2.2 User Access Handling

User access handling is done in two steps. A client getting the `UserAccessLevel` attribute of a node can be handled with the following event handler, e.g. for the administrator only writable node:

During creation of the address space the following event handler was defined for getting the `UserAccessLevel` attribute:

```
arAdministratorRW.OnReadUserAccessLevel = OnReadAdministratorUserAccessLevel;
```

and is implemented like this:

```
public ServiceResult OnReadAdministratorUserAccessLevel(ISystemContext context, NodeState node,
                                                        ref byte value)
{
    // If user identity is not set default user access level handling should apply
    if (context.UserIdentity != null && context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        value = AccessLevels.None;
    }
    else
    {
        if (context.UserIdentity != null && context.UserIdentity.TokenType == UserTokenType.UserName)
        {
            UserNameIdentityToken user = context.UserIdentity.GetIdentityToken() as UserNameIdentityToken;
            if (user.UserName == "administrator")
            {
                value = AccessLevels.CurrentReadOrWrite;
            }
            else
            {
                value = AccessLevels.None;
            }
        }
    }
    return ServiceResult.Good;
}
```

Reading the values can be restricted with the following event handler:

```
private ServiceResult OnReadAdministratorValue(
                            ISystemContext context,
                            NodeState node,
                            NumericRange indexRange,
                            QualifiedName dataEncoding,
                            ref object value,
                            ref StatusCode statusCode,
                            ref DateTime timestamp)
{
    // If user identity is not set default user access level handling should apply
    if (context.UserIdentity != null && context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        return StatusCodes.BadUserAccessDenied;
    }

    if (context.UserIdentity != null && context.UserIdentity.TokenType == UserTokenType.UserName)
    {
        UserNameIdentityToken user = context.UserIdentity.GetIdentityToken() as UserNameIdentityToken;
        if (user.UserName != "administrator")
        {
            return StatusCodes.BadUserAccessDenied;
        }
    }
    return ServiceResult.Good;
}
```

Also, during creation of the address space, the following event handler was defined for writing the variable:

```
arAdministratorRW.OnSimpleWriteValue = OnWriteAdministratorValue;
```

and is implemented like this:

```
public ServiceResult OnWriteAdministratorValue(ISystemContext context, NodeState node, ref object value)
{
    // If user identity is not set default user access level handling should apply
    if (context.UserIdentity != null && context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        return StatusCodes.BadUserAccessDenied;
    }

    if (context.UserIdentity != null && context.UserIdentity.TokenType == UserTokenType.UserName)
    {
        UserNameIdentityToken user = context.UserIdentity.GetIdentityToken() as UserNameIdentityToken;
        if (user.UserName != "administrator")
        {
            return StatusCodes.BadUserAccessDenied;
        }
    }
    return ServiceResult.Good;
}
```

6.5.3 User specific browsing

User specific handling can also be used while the client browses the address space of the server. The two user specific nodes

- Operator or Administrator can write the following variable:
  ns=2;s=AccessRights_AccessOperator_OperatorUsable
- Administrator only can write the following variable:
  ns=2;s=AccessRights_AccessAdministrator_AdministratorOnly

are not visible for anonymous users. This user handling is done with implementing the `IsNodeAccessibleForUser()` and `IsReferenceAccessibleForUser()` methods.

6.5.3.1 IsReferenceAccessibleForUser

While the client is browsing the address space of the server the UaBaseNodeManager calls `IsReferenceAccessibleForUser()` for each reference it comes to. If the browse comes to the folder node *`AccessRights`* it will get the reference to the nodes *`AccessAdministrator`* and *`AccessOperator`*. Idea in the sample is to hide both nodes for an anonymous user and therefore the references to them must be hidden as well. The implementation will then looks like:

```
protected override bool IsNodeAccessibleForUser(UaServerContext context,
                                                UaContinuationPoint continuationPoint, NodeState node)
{
    if (context.UserIdentity == null || context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        if ( (node.NodeId.Identifier.ToString() == "AccessAdministrator") ||
             (node.NodeId.Identifier.ToString() == "AccessOperator") )
        {
            return false;
        }
    }
    return true;
}
```

6.5.3.2 IsNodeAccessibleForUser

To fully hide the nodes *AccessAdministrator* and *AccessOperator* we need to implement `IsNodeAccessibleForUser()` also. The implementation in the sample hides the two variables for anonymous users:

```
protected override bool IsNodeAccessibleForUser(UaServerContext context,
                                                UaContinuationPoint continuationPoint, NodeState node)
{
    if (context.UserIdentity == null || context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        if ( (node.NodeId.Identifier.ToString() == "AccessAdministrator") ||
             (node.NodeId.Identifier.ToString() == "AccessOperator") )
        {
            return false;
        }
    }

    return true;
}
```

7 UA Model Compiler

The Model Compiler (`Technosoftware.UaModelCompiler.exe`) is based on the OPC Foundation Model Compiler available at https://github.com/OPCFoundation/UA-ModelCompiler. There are no modifications made and with the delivered executable `Technosoftware.UaModelCompiler.exe` we only guarantee that the version delivered fits the SDK.

The OPC Foundation Model Compiler will generate C# and ANSI C source code from XML files which include the UA Services, datatypes, error codes, etc.; and numerous CSV files that contain NodeIds, error codes, and attributes etc.

The input format for the tool is a file that conforms to the schema defined in `UA Model Design.xsd`. For easier usage we renamed this file to `UAModelDesign.xsd`.

The output of the tool includes:

- A NodeSet which conforms to the schema defined in Part 6 Annex F;
- An XSD and BSD (defined in Part 3 Annex C) that describes any datatypes;
- Class and constant definitions suitable for use with the .NET sample libraries;
- Other data files used to load an information model into a Server built with the .NET sample libraries;
- A CSV file which contains numeric identifiers.

The UA Model Design.xsd (https://github.com/OPCFoundation/UA-ModelCompiler/blob/master/ModelCompiler/UA%20Model%20Design.xsd) has more information about the schema itself.

7.1 Command Line Usage

The Model Compiler is called with several options, e.g. like:

```
Technosoftware.UaModelCompiler.exe -d2 ".\ModelDesign.xml" -cg ".\ModelDesign.csv" -o2 ".\"
```

And uses the following command line parameters:

Parameter	Description
-?	Prints a help text.
-d	The path to the XML file which contains the UA information model.
-c	The path to the CSV file which contains the unique identifiers for the types defined in the UA information model.
-cg	Generates the unique identifier CSV file if it doesn't exist. If the file already exists, the existing one will be used.
-o	The output directory for a single file output.
-om	The output directory for a multiple file output.
-id	The start identifier
-console	The output goes to the standard error output (console) instead of error window.

7.2 XSD Specification

The UA ModelDesign Schema (`UAModelDesign.xsd`) specifies most of the usable UA classes and contains for example the following elements:

7.2.1 Namespaces

This element defines the mappings between the URIs used to identify namespaces and the symbols used in code. It is possible that a user defined design file includes other user defined design files. This allows the user to split larger models into several files. The location of included design files is specified with the FilePath attribute (absolute or relative path without the .xml suffix).

```
  <!--
  This element defines the mappings between the URIs used to identify namespaces and the symbols
used in code.
  User defined design files can include other user defined design files.
  The location of included design files is specified with the FilePath attribute (absolute or
relative path without the .xml suffix).
  -->
  <opc:Namespaces>
    <opc:Namespace Name="OpcUa" Prefix="Opc.Ua"
XmlNamespace="http://opcfoundation.org/UA/2008/02/Types.xsd">http://opcfoundation.org/UA/</opc:Nam
espace>
    <opc:Namespace Name="Engineering" Prefix="Technosoftware.WorkshopServerForms.Engineering"
InternalPrefix="Technosoftware.WorkshopServerForms.Engineering">http://technosoftware.com/Workshop
ServerForms/Engineering</opc:Namespace>
    <opc:Namespace Name="Operations" Prefix="Technosoftware.WorkshopServerForms.Operations"
InternalPrefix="Technosoftware.WorkshopServerForms.Operations">http://technosoftware.com/WorkshopS
erverForms/Operations</opc:Namespace>
    <opc:Namespace Name="WorkshopServerForms" Prefix="Technosoftware.WorkshopServerForms.Model"
InternalPrefix="Technosoftware.WorkshopServerForms.Model">http://technosoftware.com/WorkshopServer
Forms/Model</opc:Namespace>
  </opc:Namespaces>
```

7.2.2 ObjectTypes

An ObjectType is a Node that represents the type definition for an Object.An ObjectType is a Node that represents the type definition for an Object.

```
  <opc:ObjectType SymbolicName="GenericSensorType" BaseType="ua:BaseObjectType">
    <opc:Description>A generic sensor that read a process value.</opc:Description>
    <opc:Children>
      <opc:Variable SymbolicName="Output" DataType="ua:Double" ValueRank="Scalar"
TypeDefinition="ua:AnalogItemType" />
    </opc:Children>
  </opc:ObjectType>
```

7.2.3 Object

An Object is a Node that represents a physical or abstract element of a system. Objects are modelled using the OPC UA Object Model. Systems, subsystems and devices are examples of Objects. An Object may be defined as an instance of an ObjectType.

Normally instances are created by program logic instead of being part of the model. This is provided as an example of how to do it when there is a need for persistent instances. Note that the DisplayNames are overridden for the base level nodes.

```
  <!--
  Declare an instance of a Boiler.
  Normally instances are created by program logic instead of being part of the model.
  This is provided as an example of how to do it when there is a need for persistent instances.
  Note that the DisplayNames are overridden for the base level nodes.
  -->
  <opc:Object SymbolicName="Boiler1" TypeDefinition="BoilerType" SupportsEvents="true">
    <opc:BrowseName>Boiler #1</opc:BrowseName>
    <opc:Children>
      <opc:Object SymbolicName="InputPipe" TypeDefinition="BoilerInputPipeType"
SupportsEvents="true">
        <opc:DisplayName>Pipe1001</opc:DisplayName>
      </opc:Object>
      <opc:Object SymbolicName="Drum" TypeDefinition="BoilerDrumType" SupportsEvents="true">
        <opc:DisplayName>Drum1001</opc:DisplayName>
      </opc:Object>
      <opc:Object SymbolicName="OutputPipe" TypeDefinition="BoilerOutputPipeType"
SupportsEvents="true">
        <opc:DisplayName>Pipe1002</opc:DisplayName>
      </opc:Object>
      <opc:Object SymbolicName="FlowController" TypeDefinition="FlowControllerType">
        <opc:DisplayName>FC1001</opc:DisplayName>
      </opc:Object>
      <opc:Object SymbolicName="LevelController" TypeDefinition="LevelControllerType">
        <opc:DisplayName>LC1001</opc:DisplayName>
      </opc:Object>
      <opc:Object SymbolicName="CustomController" TypeDefinition="CustomControllerType">
        <opc:DisplayName>CC1001</opc:DisplayName>
      </opc:Object>
    </opc:Children>

    <!--
    Link the instance back to the ObjectsFolder
    -->
    <opc:References>
      <opc:Reference IsInverse="true">
        <opc:ReferenceType>ua:Organizes</opc:ReferenceType>
        <opc:TargetId>ua:ObjectsFolder</opc:TargetId>
      </opc:Reference>
    </opc:References>

  </opc:Object>
```

7.2.4 VariableType

```
  <VariableType SymbolicName="tns:MultiStateValueDiscreteType" BaseType="tns:DiscreteItemType"
DataType="tns:Number">
    <Children>
      <Property SymbolicName="tns:EnumValues" DataType="tns:EnumValueType" ValueRank="Array" />
      <Property SymbolicName="tns:ValueAsText" DataType="tns:String" />
    </Children>
  </VariableType>
```

7.2.5 Variable

```
  <opc:Variable SymbolicName="Output" DataType="ua:Double" ValueRank="Scalar"
TypeDefinition="ua:AnalogItemType"/>
```

7.2.6 ReferenceType

```
  <opc:ReferenceType SymbolicName="SignalTo" BaseType="ua:NonHierarchicalReferences">
    <opc:Description>A reference that indicates an electrical signal between two
variables.</opc:Description>
    <opc:InverseName>SignalFrom</opc:InverseName>
  </opc:ReferenceType>
```

7.2.7 DataType

```
  <opc:DataType SymbolicName="VehicleType" BaseType="ua:Structure">
    <opc:Fields>
      <opc:Field Name="Make" DataType="ua:String"></opc:Field>
      <opc:Field Name="Model" DataType="ua:String"></opc:Field>
    </opc:Fields>
  </opc:DataType>
```

7.2.8 Property

```
<opc:Property SymbolicName="SetPoint"/>
```

7.3 Usage

The ServerForms and ServerConsole samples can be used as sample implementation of a UaServerPlugin using an information model based on classes generated with the Model Compiler. Please follow the instructions below to be able to use the Model Compiler in your own servers.

7.3.1 Create a model design

The model of the Workshop server samples is defined in the files

- ModelDesign.xml
- EngineeringDesign.xml
- OperationsDesign.xml

They can be found in the sub-directory Model. You can use that as reference for your own model design files.

7.3.2 Using the Model Compiler

For ease of use create a batch file, e.g, BuildDesign.bat, which executes the model compiler. For the Workshop server samples the batch file looks like:

```
rem @echo off

setlocal

SET PATH=..\..\..\Scripts;..\..\..\\bin\net472;..\..\..\..\bin\net472;%PATH%;


echo Building OperationsDesign

Technosoftware.UaModelCompiler.exe -d2 ".\Model\OperationsDesign.xml" -cg
".\Model\OperationsDesign.csv" -o2 ".\Model"

echo Success!


echo Building EngineeringDesign

Technosoftware.UaModelCompiler.exe -d2 ".\Model\EngineeringDesign.xml" -cg
".\Model\EngineeringDesign.csv" -o2 ".\Model"

echo Success!


echo Building ModelDesign

Technosoftware.UaModelCompiler.exe -d2 ".\Model\ModelDesign.xml" -cg ".\Model\ModelDesign.csv" -o2
".\Model"

echo Success!

Pause
```

Execute the batch file and for the Workshop server samples the following files should be created:

- *.Classes.cs
- *.Constants.cs
- *.DataTypes.cs
- Technosoftware.WorkshopServerForms.Model.PredefinedNodes.uanodes
- *.csv (only if it not already exists)

In addition, also some more files are created but not used at the moment.

7.3.3 Add the files to your server project

1. Add the C# files *.Classes.cs, *.Constants.cs and *.DataTypes.cs to your project.
2. Add the UANODES file Technosoftware.WorkshopServerForms.Model.PredefinedNodes.uanodes as Embedded resource to your project.

7.3.4 Add the used namespaces

You need to add the model design specific namespaces by modifying the OnGetNamespaceUris() method.

```
        /// <summary>
        /// Defines namespaces used by the application.
        /// </summary>
        /// <returns>Array of namespaces that are used by the application.</returns>
        public string[] OnGetNamespaceUris()
        {
            // set one namespace for the type model.
            var namespaceUrls = new string[3];
            namespaceUrls[0] =
Technosoftware.WorkshopServerForms.Model.Namespaces.WorkshopServerForms;
            namespaceUrls[1] = Technosoftware.WorkshopServerForms.Model.Namespaces.Engineering;
            namespaceUrls[2] = Technosoftware.WorkshopServerForms.Model.Namespaces.Operations;
            return namespaceUrls;
        }
```

7.3.5 Load the predefined nodes

You need to load the UANODES file by adding this to the LoadPredefinedNodes() method of the Node Manager (Technosoftware.WorkshopServerFormsNodeManager.cs)

```
        /// <summary>
        /// Loads a node set from a file or resource and addres them to the set of predefined
nodes.
        /// </summary>
        protected override NodeStateCollection LoadPredefinedNodes(ISystemContext context)
        {
            // We know the model name to load but because this project is compiled for different
environments we don't know
            // the assembly it is in. Therefor we search for it:
            var assembly = this.GetType().GetTypeInfo().Assembly;
            var names = assembly.GetManifestResourceNames();
            string resourcePath = String.Empty;

            foreach (var module in names)
            {
                if
(module.Contains("Technosoftware.WorkshopServerForms.Model.PredefinedNodes.uanodes"))
                {
                    resourcePath = module;
                    break;
                }
            }

            if (resourcePath == String.Empty)
            {
                // No assembly found containing the nodes of the model. Behaviour here can differ
but in this case we just return null.
                return null;
            }

            NodeStateCollection predefinedNodes = new NodeStateCollection();
            predefinedNodes.LoadFromBinaryResource(context, resourcePath, assembly, true);
            return predefinedNodes;
        }
```

7.3.6 Use the classes

You can use the classes created by the model compiler in your code, e.g. to dynamically create objects based on these classes.

```
    FolderState plantFolder = opcServer_.CreateFolder(root, "Plant", "Plant");
    const string datatypes = "Plant_";
    Technosoftware.WorkshopServerForms.Model.MachineState machine1 = new
Technosoftware.WorkshopServerForms.Model.MachineState(null);
    ParsedNodeId pnd1 = new ParsedNodeId() { NamespaceIndex = NamespaceIndex, RootId = "Machine
#1" };

    machine1.Create(
        SystemContext,
        pnd1.Construct(),
        new QualifiedName("Machine #1", NamespaceIndex),
        null,
        true);

    machine1.AddReference(ReferenceTypeIds.Organizes, true, plantFolder.NodeId);
    plantFolder.AddReference(ReferenceTypeIds.Organizes, false, machine1.NodeId);

    AddPredefinedNode(SystemContext, machine1);

    Technosoftware.WorkshopServerForms.Model.MachineState machine2 = new
Technosoftware.WorkshopServerForms.Model.MachineState(null);
    ParsedNodeId pnd2 = new ParsedNodeId() { NamespaceIndex = NamespaceIndex, RootId = "Machine
#2" };

    machine2.Create(
        SystemContext,
        pnd2.Construct(),
        new QualifiedName("Machine #2", NamespaceIndex),
        null,
        true);

    machine2.AddReference(ReferenceTypeIds.Organizes, true, plantFolder.NodeId);
    plantFolder.AddReference(ReferenceTypeIds.Organizes, false, machine2.NodeId);

    AddPredefinedNode(SystemContext, machine2);
```

Why Technosoftware GmbH?...

- Professionalism
  Technosoftware GmbH is, measured by the number of employees, truly not a big company. However, when it comes to flexibility, service quality, and adherence to schedules and reliability, we are surely a great company which can compete against the so called leaders in the industry. And this is THE crucial point for our customers.

- Continuous progress
  Lifelong learning and continuing education is, especially in the information technology, essential for future success. Concerning our customers, we will constantly be accepting new challenges and exceeding their requirements again and again. We will continue to do everything to fulfill the needs of our customers and to meet our own standards.

- High Quality of Work
  We reach this by a small, competent and dynamic team of coworkers, which apart from the satisfaction of the customer; take care of a high quality of work. We concern the steps necessary for it together with consideration of the customers' requirements.

- Support
  We support you in all phases – consultation, direction of the project, analysis, architecture & design, implementation, test and maintenance. You decide on the integration of our coworkers in your project; for an entire project or for selected phases.

Technosoftware GmbH
Windleweg 3, CH-5235 Rüfenach

sales@technosoftware.com
www.technosoftware.com