

OPC UA Server .NET

Develop OPC UA Servers with C# for of .NET 5.0 and .NET Core 3.1

First Steps implementing an OPC UA Server





Document Control

Version	Date	Comment
2.1	01-FEB-2021	Completely rewritten

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 .NET API hides most of the OPC UA specifications to provide the possibility to develop 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 .NET. 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

1	Installation	11
2	Supported OPC UA Profiles	12
2.1	Core Characteristics	12
2.2	Data Access	14
2.3	Event Access	15
2.4	Alarm & Condition	16
2.5	Generic Features	18
2.6	Redundancy	19
2.7	Historical Access	19
2.7.1	Historical Data	19
2.7.2	Historical Events	20
2.8	Aggregates	20
3	Ready to use Sample Applications.....	21
3.1	Required NuGet Packages.....	21
3.2	Directory Structure	22
3.3	OPC UA Server Solution for .NET Core.....	23
3.3.1	Prerequisites	23
3.3.2	Start the server	23
3.3.3	Start the client	23
3.3.4	Check the output	24
4	Implementing your first OPC UA Server	25
4.1	Overview	25
4.2	Tutorial “Base OPC UA Server”	27
4.2.1	Goals of this tutorial	27
4.2.2	Overview	27
4.2.3	Start developing your own OPC UA Server	28
4.2.3.1	Renaming files.....	28
4.2.3.2	Changes to project and building options	28
4.2.3.3	Changes in NameSpaces.cs	29
4.2.3.4	Changes in SampleCompany.SampleServer.Config.xml	29
4.2.3.5	Changes of class names	29
4.2.3.6	Program Customization	30
4.2.3.7	UaServerPlugin Customization.....	31
4.2.3.8	SampleCompany.SampleServerNodeManager Customization .	32

T

4.2.4	Testing your OPC UA server	35
4.2.5	SampleServer project	35
4.3	Tutorial “DataTypes OPC UA Server”	36
4.3.1	Goals of this tutorial	36
4.3.2	Overview	36
4.3.3	Start adding a model with data types.....	36
4.3.3.1	BuildDesign.bat	36
4.3.3.2	Create an empty model.....	37
4.3.3.3	Add some types to the model.....	38
4.3.3.4	Build the model	38
4.3.3.5	Embedded Resource Files	39
4.3.3.6	Extend the SampleServer.cs	39
4.3.3.7	Add the used namespace	39
4.3.3.8	Loading the generated types to the address space	40
4.3.3.9	Using the generated types.....	41
4.3.3.10	Adding a method event handler	41
4.3.4	Testing your OPC UA server	42
4.3.5	SampleServer project	42
4.4	Tutorial “User Authentication OPC UA Server”	43
4.4.1	Goals of this tutorial	43
4.4.2	Start adding user authentication	43
4.4.2.1	ImpersonateUserEvent	43
4.4.3	Address Space creation	45
4.4.3.1	CreateAddressSpace() method	45
4.4.4	User Access Handling.....	46
4.4.5	User specific browsing	48
4.4.5.1	IsReferenceAccessibleForUser	48
4.4.5.2	IsNodeAccessibleForUser	48
4.4.6	Testing your OPC UA server	49
4.4.6.1	Anonymous user	49
4.4.6.2	Operator or Administrator user	49
4.4.7	SampleServer project	49
4.5	Tutorial “Reverse Connect enabled OPC UA Server”	50
4.5.1	Goals of this tutorial	50
4.5.2	Start adding reverse connect functionality.....	50



	4.5.3	UaServerPlugin.cs.....	50
	4.5.4	Program.cs	51
	4.5.5	Testing your OPC UA server	53
	4.5.6	SampleServer project	54
5		Use of the different Methods.....	55
	5.1.1	Server Startup / Shutdown	55
	5.1.2	OnStartup.....	55
	5.1.3	OnGetLicenseInformation	55
	5.1.4	OnGetServer	56
	5.1.5	OnGetServerProperties.....	56
	5.1.6	OnGetNamespaceUris	56
	5.1.7	OnGetNodeManager	57
	5.1.8	OnInitialized.....	57
	5.1.9	OnRunning.....	57
	5.1.10	OnShutdown	57
	5.2	Address Space Creation.....	58
	5.2.1	Creating variables in the address space	58
6		Configuration.....	59
	6.1	Application Configuration	59
	6.1.1	Extensions.....	60
	6.1.2	Tracing Output	60
7		Certificate Management and Validation.....	61
8		UA Model Compiler.....	63
	8.1	Command Line Usage	63
	8.2	XSD Specification.....	64
	8.2.1	Namespaces	64
	8.2.2	ObjectTypes	64
	8.2.3	Object	65
	8.2.4	VariableType.....	66
	8.2.5	Variable	66
	8.2.6	ReferenceType	66
	8.2.7	DataType.....	66
	8.2.8	Property	66





Disclaimer

© Technosoftware GmbH. All rights reserved. No part of this document may be altered, reproduced or distributed in any form without the expressed written permission of Technosoftware GmbH.

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

For starting with OPC UA Development you can download the **OPC UA Solution .NET** from:

OPC UA Solution .NET

The OPC UA Solution .NET offers a fast and easy access to the OPC UA Client & Server technology. Develop OPC compliant UA Clients and Servers with C# targeting .NET 5.0, .NET Core 3.1 or .NET Standard 2.1. For backward compatibility we also provide .NET 4.8, .NET 4.7.2 and .NET 4.6.2 support.

You can download it from <https://github.com/technosoftware-gmbh/opcua-solution-net>

This GitHub repository is automatically tested with the following environments:

- a. Linux Ubuntu 16.04
 - i. .NET 5.0.101
 - ii. Mono 6.10.0
- b. Mac OS X 10.13
 - i. .NET 5.0.101
 - ii. Mono 6.10.0
- c. Windows Server 2019
 - i. .NET 5.0.101

Important:





An installation guide is available with the solution. 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 .NET:

2.1 Core Characteristics

Profile	Description	Supported
Core 2017 Server Facet	<p>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".</p> <p>Also supported are:</p> <ul style="list-style-type: none">• User Token - User Name Password Server Facet This Facet indicates that a user token that is comprised of a username and password is supported. This user token can affect the behaviour of the ActivateSession Service.• SecurityPolicy - None This security Facet defines a security policy used for configurations with the lowest security needs. This security policy can affect the behaviour of the CreateSession and ActivateSession Services. It also results in a SecureChannel which has no channel security. By default, this security policy shall be disabled if any other security policy is available.	
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.	

T

Base Server Behaviour Facet	<p>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.</p> <p>Note: Base Server Facet is supported but Compliance Test Tool doesn't support any test cases for this.</p>	✓
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	<p>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.</p> <p>Also supported is:</p> <ul style="list-style-type: none"> • Security User Access Control Base A Server that supports this profile supports restricting some level of access to some Nodes in the AddressSpace based on the validated user. 	✓
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. Note: Enhanced DataChange Subscription 2017 Server Facet is supported in a basic form.	✓
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. Note: ComplexType 2017 Server Facet is supported in a basic form, but Compliance Test Tool doesn't support any test cases for this.	✓

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 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	✗

T

	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 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 subset 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 like 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 like static methods in a class.	✓
Auditing Server Facet	<p>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</p> <p>Note: Auditing Server Facet is supported but Compliance Test Tool doesn't support any test cases for this.</p>	✓
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 were 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.	⊗

T

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

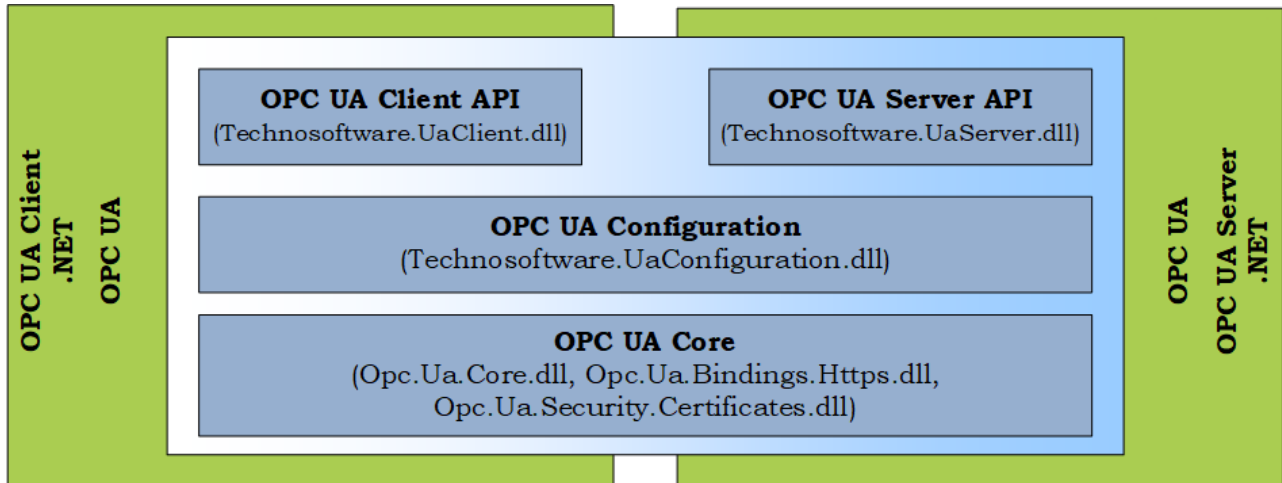
Profile	Description	Supported
Aggregate Subscription Server Facet	This Facet defines the handling of the aggregate filter when subscribing for Attribute values.	⊗

3 Ready to use Sample Applications

The OPC UA Server .NET contains several ready to use sample server applications which you can use to check your environment.

3.1 Required NuGet Packages

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



The DLL's are delivered as NuGet Packages. The OPC UA Server .NET uses the following packages:

Name	Description
Technosoftware.UaSolution.Opc.Ua.Core	The OPC UA Core Class Library.
Technosoftware.UaSolution.Opc.Ua.Security.Certificates	The OPC UA Certificate Library with all ASN.1 encode/decode related code. This is a dependency of the OPC UA Core Class Library.
Technosoftware.UaSolution.Opc.Ua.Bindings.Https	The OPC UA Https Binding Library.
Technosoftware.UaSolution.UaConfiguration	Contains configuration related classes like, e.g. ApplicationInstance .
Technosoftware.UaSolution.UaServer	The OPC UA Server Class library containing the classes and methods usable for server development.



3.2 Directory Structure

The repository contains the following basic directory layout:

1. **bin/**
 - **net462/**
Model Compiler based on .NET 4.6.2
2. **documentation/**
Additional documentation like:
 - **OPC_UA_Solution_NET_Installation_Guide.pdf**
Installation of development and run-time system
 - **OPC_UA_Solution_NET_Introduction.pdf**
Introduction in Developing OPC UA Clients and OPC UA Servers with C# / VB.NET
 - **OPC_UA_Client_Development_with_NET.pdf**
Tutorial for Developing OPC UA Clients with C# for of .NET 5.0 and .NET Core 3.1
 - **OPC_UA_Server_Development_with_NET.pdf**
Tutorial for Developing OPC UA Servers with C# for of .NET 5.0 and .NET Core 3.1
3. **examples/**
Sample applications
 - **Empty/**
 - **EmptyServer/**
Empty Server as Console application used by this tutorial as starting point.
 - **Base/**
 - **SampleServer/**
Sample Server created with the first tutorial.
 - **Workshop/**
 - **ModelDesignClient/**
Client as Console application which use the ModelDesignServer as OPC UA server.
 - **ModelDesignServer/**
Simple Server as Console application used by the ModelDesignClient. It uses a model design for the OPC Foundation Model Compiler
 - **SimpleClient/**
Simple Client as Console application which use the EmptyServer as OPC UA server.
 - **EmptyServer/**
Simple Server as Console application used by the SimpleClient. All nodes are manually created within the code.
4. **schema/**
XSD files like the UAModelDesign.xsd used for the Model Designer
5. **Workshop/**
OPC UA Workshop content as PDFs



3.3 OPC UA Server Solution for .NET Core

The main OPC UA Solution can be found in the root of the repository and is named

- NetCoreSamples.sln

The solution contains two sample clients, as well as two sample server examples used by these clients.

Important:

An installation guide is available with the solution or from

https://technosoftware.com/documentation/OPC_UA_Solution_NET_Installation_Guide.pdf.

Please read that one first and then follow this guide.

3.3.1 Prerequisites

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

/

and execute

```
dotnet restore NetCoreSamples.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/EmptyServer`.
3. To run the server sample type

```
dotnet run --no-restore --framework netcoreapp3.1 --project Technosoftware.EmptyServer.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/SimpleClient`.
3. To run the client sample type

```
dotnet run --no-restore --framework netcoreapp3.1 --project Technosoftware.SimpleClient.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`.



3.3.4 Check the output

If everything was done correctly the client should show the following lines:

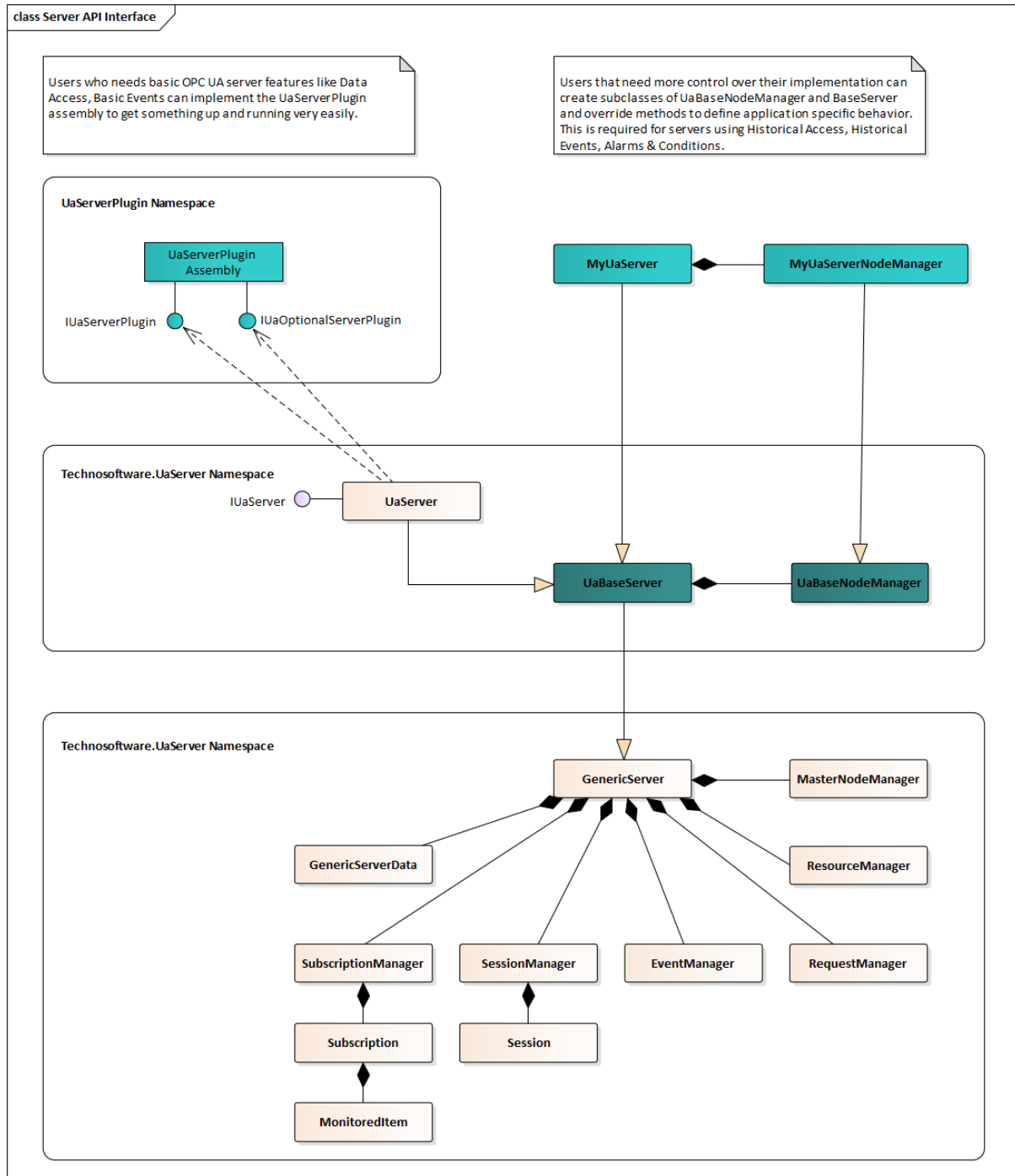
```
Technosoftware .NET Core OPC UA Simple Client
1 - Create an Application Configuration.
2 - Discover endpoints of opc.tcp://localhost:55550/TechnosoftwareEmptyServer.
   Selected endpoint uses: Aes256_Sha256_RsaPss
3 - Create a session with OPC UA server.
4 - Browse address space.
5 - Read a single value.
Node Value:1.020668067406195E-26
6 - Read multiple values.
Status of Read of Node ns=2;s=Scalar_Simulation_Number is: Good
```

You can abort the running application with Ctrl-C.

4 Implementing your first OPC UA Server

4.1 Overview

We concentrate in this tutorial on the *EmptyServer*, a console-based application for testing the server specific features. This tutorial will refer to that code while explaining the different steps to take to accomplish the main tasks of an OPC UA server.



T

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](#), [IUaOptionalServerPlugin](#) and [IUaReverseConnectServerPlugin](#) 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](#), [IUaOptionalServerPlugin](#) and optionally [IUaReverseConnectServerPlugin](#).



4.2 Tutorial “Base OPC UA Server”

This chapter describes how to create your first OPC UA server based on the EmptyServer project. It shows the basic steps required to adapt one of the examples to your needs. It is highly recommended to follow this tutorial. More detailed topics will be later discussed and base on the sample server created in this chapter.

4.2.1 Goals of this tutorial

You will learn the following topics:

1. Minimum required changes to start a new OPC UA server development.
2. Create a basic address space with a static base data variable as well as a simulated data variable.
3. Using of OnSimpleWriteValue to be able to handle changes coming from an OPC UA client.
4. Changing values of the simulated variable (OnSimulation).

4.2.2 Overview

The example OPC UA Servers uses all a common layout and consists at least of the following C# files:

Name	Description
Directory.Build.props	A user-defined file that provides customizations to projects under a directory. In our case just imports the targets.props file.
targets.props	A user-defined file that defines for which target the server should be build for. Adapt the following parameter to your needs: <code><AppTargetFramework>netcoreapp3.1</AppTargetFramework></code> Possible values are <code>net462</code> , <code>net472</code> , <code>net48</code> , <code>netcoreapp3.1</code> or <code>net5.0</code>
Namespace.cs	Constant(s) defining the used namespaces for the application, e.g. <code>public const string EmptyServer = "http://emptycompany.com/EmptyServer";</code>
Program.cs	Contains the code to startup the base OPC UA server (<code>UaServer</code>).
UaServerPlugin.cs	Contains the implementation of the interfaces <code>IUaServerPlugin</code> , <code>IUaOptionalServerPlugin</code> and optionally <code>IUaReverseConnectServerPlugin</code> This file consists because of historic reasons where a real simple server with just a plugin DLL was possible.
*.Config.xml	Contains the configuration of the OPC UA Server, e.g. ApplicationName. For the empty server it is named: <code>Technosoftware.EmptyServer.Config.xml</code>
*Server.cs	Implements the basic OPC UA Server and overrides the <code>UaBaseServer</code> class. For the empty server it is named: <code>Technosoftware.EmptyServer.cs</code>
*ServerNodeManager.cs	Implements the OPC UA NodeManager and overrides the <code>UaBaseNodeManager</code> class. For the empty server it is named: <code>Technosoftware.EmptyServerNodeManager.cs</code>



In this tutorial we use a general naming of files and namespaces:

Company: [EmptyCompany](#) or [SampleCompany](#) or for the ready to use servers [Technosoftware](#)
Server Namespace: [EmptyServer](#) or [SampleServer](#)
Namespace: [EmptyCompany.EmptyServer](#) or [SampleCompany.SampleServer](#)
File Names: Starts with [EmptyCompany.EmptyServer](#) or [SampleCompany.SampleServer](#)

4.2.3 Start developing your own OPC UA Server

You need the following files to be able to develop your own OPC UA Server:

1. Copy the directory with the empty OPC UA Server at `\examples\Empty\EmptyServer`, to your development directory, e.g. `c:\solutions`
2. Rename the folder to your needs, e.g. `c:\solutions\SampleServer`.

In the following chapters you will find some required and recommended changes necessary before you start changing the source code. For a first OPC UA Server we recommend changing only those values and keep everything else untouched. As you proceed further with this tutorial more and more changes and enhancements are done.

4.2.3.1 Renaming files

It is a good idea to rename several files to fit your project. As mentioned in the last chapter we use a general naming of files and namespaces and for this tutorial we defined it as:

Company: [SampleCompany](#)
Server Namespace: [SampleServer](#)
Namespace: [SampleCompany.SampleServer](#)
File Names: Starts with [SampleCompany.SampleServer](#)

Of course, you can replace the company name with your company name and the server namespace with a more usable one for your project.

Please change the names of the following files:

1. `EmptyCompany.EmptyServer.Config.xml` to [SampleCompany.SampleServer.Config.xml](#)
2. `EmptyCompany.EmptyServer.cs` to [SampleCompany.SampleServer.cs](#)
3. `EmptyCompany.EmptyServer.csproj` to [SampleCompany.SampleServer.csproj](#)
4. `EmptyCompany.EmptyServerNodeManager.cs` to [SampleCompany.SampleServerNodeManager.cs](#)

4.2.3.2 Changes to project and building options

1. Please open the [SampleCompany.SampleServer.csproj](#) file with Visual Studio 2019.
2. Replace `EmptyCompany.EmptyServer` with [SampleCompany.SampleServer](#) in all your project files.
3. Check the project properties Package dialog or edit the [SampleCompany.SampleServer.csproj](#) file and adapt the following entries: `<Company>`, `<Product>`, `<Description>` and `<Copyright>`
4. The [SampleCompany.SampleServer.csproj](#) file use `$(AppTargetFramework)` for the target it should build for. That one is defined in the `targets.props` file and can be changed to [net462](#), [net472](#), [net48](#), [netcoreapp3.1](#) or [net5.0](#).



4.2.3.3 Changes in NameSpaces.cs

1. Open the `Namespaces.cs` file with Visual Studio 2019.
2. Rename variable `EmptyServer` to `SampleServer` for the whole project.
3. Change the value of `SampleServer` to <http://samplecompany.com/SampleServer>.

4.2.3.4 Changes in SampleCompany.SampleServer.Config.xml

1. Open the `SampleCompany.SampleServer.Config.xml` file with Visual Studio 2019.
2. The values of the following global parameters changes must at least be changed:
 - `<ApplicationName>SampleCompany OPC UA Sample Server</ApplicationName>`
 - `<ApplicationUri>urn:localhost:SampleCompany:SampleServer</ApplicationUri>`
 - `<ProductUri>uri:samplecompany.com:SampleServer</ProductUri>`
3. In the section `<SecurityConfiguration><ApplicationCertificate>` you need to define the Subject name for the application certificate.
 - `<SubjectName>CN=SampleCompany OPC UA Sample Server, C=CH, S=Aargau, O=SampleCompany, DC=localhost</SubjectName>`
4. In the section `<ServerConfiguration>` you need to define the URL the server should be reachable. One entry is for the opc.tcp protocol and one for the https protocol:
 - `<ua:String>opc.tcp://localhost:55555/SampleServer</ua:String>`
 - `<ua:String>https://localhost:55556/SampleServer</ua:String>`

4.2.3.5 Changes of class names

1. Open the `SampleCompany.SampleServer.cs` file with Visual Studio 2019.
2. Rename the `EmptyServer` class to `SampleServer` for the whole project.
3. Open the `SampleCompany.SampleServerNodeManager.cs` file with Visual Studio 2019.
4. Rename the `EmptyServerNodeManager` class to `SampleServerNodeManager` for the whole project.
5. Open the `Program.cs` file with Visual Studio 2019.
6. Rename the `MyEmptyServer` class to `MySampleServer` for the whole project.
7. Rename the Task `EmptyServer` method to `SampleServer` for the whole project.



4.2.3.6 Program Customization

The `Program.cs` file contains the startup of the application and the OPC UA Server. The important part related to OPC UA are:

Using statements:

```
using Opc.Ua;  
  
using Technosoftware.UaServer;  
using Technosoftware.UaServer.Sessions;
```

Definitions of variables:

Two variables in the `class MySampleServer`:

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

Console output in Main:

Change the console output:

```
Console.WriteLine("SampleCompany {0} OPC UA Sample Server", Utils.IsRunningOnMono() ?  
"Mono" : ".NET Core");
```

Starting the server:

Within the Task `SampleServer()`:

```
// start the server.  
await uaServer_.StartAsync(uaServerPlugin_, "SampleCompany.SampleServer", null);
```

Please ensure that the second parameter `"SampleCompany.SampleServer"` fits your configuration file name `SampleCompany.SampleServer.Config.xml`.



4.2.3.7 UaServerPlugin Customization

The `UaServerPlugin` class can return a custom `UaBaseNodeManager` and `UaBaseServer` as shown in the sample below.

Open the `UaServerPlugin.cs` file with Visual Studio 2019 and verify the following shown methods:

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

    public UaBaseNodeManager OnGetNodeManager(IUaServer opcServer, IUaServerData uaServer,
                                              ApplicationConfiguration configuration, params string[] namespaceUris)
    {
        return new SampleServerNodeManager(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 `UaBaseNodeManager` must override the `CreateAddressSpace` method as shown in the next chapter.

Another changes/checks are required in the method:

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

    return properties;
}

/// <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 namespaceUris = new string[1];
    namespaceUris[0] = Namespaces.SampleServer;
    return namespaceUris;
}
```



4.2.3.8 SampleCompany.SampleServerNodeManager Customization

Open the [SampleCompany.SampleServerNodeManager.cs](#) file with Visual Studio 2019 and do the following additions and changes:

Add using directives:

```
using System.Threading;
```

Add private variables under:

```
#region Private Fields
private Opc.Ua.Test.DataGenerator generator_;
private Timer simulationTimer_;
private UInt16 simulationInterval_ = 1000;
private bool simulationEnabled_ = true;
private List<BaseDataVariableState> dynamicNodes_;
```

Add the following event handlers:

```
#region Event Handlers
private ServiceResult OnWriteInterval(ISystemContext context, NodeState node, ref object value)
{
    try
    {
        simulationInterval_ = (ushort)value;

        if (simulationEnabled_)
        {
            simulationTimer_.Change(100, simulationInterval_);
        }

        return ServiceResult.Good;
    }
    catch (Exception e)
    {
        Utils.Trace(e, "Error writing Interval variable.");
        return ServiceResult.Create(e, StatusCodes.Bad, "Error writing Interval variable.");
    }
}

private ServiceResult OnWriteEnabled(ISystemContext context, NodeState node, ref object value)
{
    try
    {
        simulationEnabled_ = (bool)value;

        if (simulationEnabled_)
        {
            simulationTimer_.Change(100, simulationInterval_);
        }
        else
        {
            simulationTimer_.Change(100, 0);
        }

        return ServiceResult.Good;
    }
    catch (Exception e)
    {
        Utils.Trace(e, "Error writing Enabled variable.");
        return ServiceResult.Create(e, StatusCodes.Bad, "Error writing Enabled variable.");
    }
}
#endregion
```


T

These event handlers are called if an OPC UA client changes the value of the interval variable (`OnWriteInterval`) or the enabled variable (`OnWriteEnabled`). Within the address space creation these event handlers are assigned with

```
intervalVariable.OnSimpleWriteValue = OnWriteInterval;  
intervalVariable.OnSimpleWriteValue = OnWriteInterval;
```

Add the following helper methods:

```
#region Helper Methods  
/// <summary>  
/// Creates a new variable.  
/// </summary>  
private BaseDataVariableState CreateDynamicVariable(NodeState parent, string path, string name, string  
description, NodeId dataType, int valueRank, byte accessLevel, object initialValue)  
{  
    var variable = CreateBaseDataVariableState(parent, path, name, description, dataType, valueRank,  
accessLevel, initialValue);  
    dynamicNodes_.Add(variable);  
    return variable;  
}  
  
private object GetNewValue(BaseVariableState variable)  
{  
    if (generator_ == null)  
    {  
        generator_ = new Opc.Ua.Test.DataGenerator(null) {BoundaryValueFrequency = 0};  
    }  
  
    object value = null;  
    var retryCount = 0;  
  
    while (value == null && retryCount < 10)  
    {  
        value = generator_.GetRandom(variable.DataType, variable.ValueRank, new uint[] { 10 },  
opcServer_.NodeManager.ServerData.TypeTree);  
        retryCount++;  
    }  
  
    return value;  
}  
  
private void DoSimulation(object state)  
{  
    try  
    {  
        lock (Lock)  
        {  
            foreach (var variable in dynamicNodes_)  
            {  
                opcServer_.WriteBaseVariable(variable, GetNewValue(variable), StatusCodes.Good,  
DateTime.UtcNow);  
            }  
        }  
    }  
    catch (Exception e)  
    {  
        Utils.Trace(e, "Unexpected error doing simulation.");  
    }  
}  
#endregion
```

The `DoSimulation` method loops through all simulated values (in this tutorial just one) and uses `WriteBaseVariable` of the base OPC UA server implementation (`opcServer_`) to write a new value to it.



Replace the CreateAddressSpace with the one shown below:

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

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

        var root = CreateFolderState(null, "My Data", new LocalizedText("en", "My Data"),
            new LocalizedText("en", "Root folder of the Sample Server"));
        References.Add(new NodeStateReference(ReferenceTypes.Organizes, false, root.NodeId));
        root.EventNotifier = EventNotifiers.SubscribeToEvents;
        opcServer_.AddRootNotifier(root);

        try
        {
            #region Static
            var staticFolder = CreateFolderState(root, "Static", "Static", "A folder with a sample
static variable.");
            const string scalarStatic = "Static_";
            CreateBaseDataVariableState(staticFolder, scalarStatic + "String", "String", null,
DataTypeIds.String, ValueRanks.Scalar, AccessLevels.CurrentReadOrWrite, null);
            #endregion

            #region Simulation
            var simulationFolder = CreateFolderState(root, "Simulation", "Simulation", "A folder with
simulated variables.");
            const string simulation = "Simulation_";

            var simulatedVariable = CreateDynamicVariable(simulationFolder, simulation + "Double",
"Double", "A simulated variable of type Double. If Enabled is true this value changes based on the defined
Interval.", DataTypeIds.Double, ValueRanks.Scalar, AccessLevels.CurrentReadOrWrite, null);

            var intervalVariable = CreateBaseDataVariableState(simulationFolder, simulation +
"Interval", "Interval", "The Interval used for changing the simulated values.", DataTypeIds.UInt16,
ValueRanks.Scalar, AccessLevels.CurrentReadOrWrite, simulationInterval_);
            intervalVariable.OnSimpleWriteValue = OnWriteInterval;

            var enabledVariable = CreateBaseDataVariableState(simulationFolder, simulation +
"Enabled", "Enabled", "Specifies whether the simulation is enabled (true) or disabled (false).",
DataTypeIds.Boolean, ValueRanks.Scalar, AccessLevels.CurrentReadOrWrite, simulationEnabled_);
            enabledVariable.OnSimpleWriteValue = OnWriteEnabled;
            #endregion

        }
        catch (Exception e)
        {
            Utils.Trace(e, "Error creating the address space.");
        }
        AddPredefinedNode(SystemContext, root);
        simulationTimer_ = new Timer(DoSimulation, null, 1000, 1000);
    }
}
```



4.2.4 Testing your OPC UA server

You should now be able to build and start your first OPC UA server. Using the Unified Automation UaExpert you can use to connect to the OPC UA server and should see the following address space:

- 📁 Root
 - ▼ 📁 Objects
 - > 📁 Aliases
 - ▼ 📁 My Data
 - ▼ 📁 Simulation
 - > 🟢 Double
 - > 🟢 Enabled
 - > 🟢 Interval
 - ▼ 📁 Static
 - > 🟢 String
 - > 🌐 Server
 - > 📁 Types
 - > 📁 Views

You can drag&drop the variables “Double”, “Enabled” and “Interval” to the Data Access View and see the value of the “Double” variable changing. By changing the “Interval” the update interval of the “Double” should change and with setting “Enabled” to false no more changes should happen to the “Double” variable.

4.2.5 SampleServer project

For your convenience the resulting sample server is also available in the distribution at
`\examples\Base\SampleServer`



4.3 Tutorial “DataTypes OPC UA Server”

This chapter describes how to create your own data types and use it in your OPC UA server. This tutorial is based on the base SampleServer project. It shows the basic steps required to use the model compiler and a XML based model to create some data types.

4.3.1 Goals of this tutorial

You will learn the following topics:

1. Minimum required changes to use your own data types.
2. Introduction in how to use the model compiler.
3. Adding different types like MethodType, ObjectType

4.3.2 Overview

Please check chapter 8 to get an introduction and a basic overview of the model compiler.

4.3.3 Start adding a model with data types

You need the following files to be able to develop your own OPC UA Server:

1. Copy the directory with the base OPC UA Server at `\examples\Base\SampleServer`, to your development directory, e.g. `c:\solutions\examples\DataTypes\SampleServer`

In the following chapters you will find some required and recommended changes necessary before you start changing the source code. For a first OPC UA Server we recommend changing only those values and keep everything else untouched. As you proceed further with this tutorial more and more changes and enhancements are done.

4.3.3.1 BuildDesign.bat

Create a batch file called `BuildDesign.bat` the following content:

```
rem @echo off
setlocal

SET PATH=..\..\..\bin\net462;%PATH%;

echo Building ModelDesign
Technosoftware.UaModelCompiler.exe -version v104 -d2 ".\Model\ModelDesign.xml" -cg ".\Model\ModelDesign.csv" -o2 ".\Model\"
echo Success!

pause
```

Ensure that `..\..\..\bin\net462` points to the path the `Technosoftware.UaModelCompiler.exe` is located.



4.3.3.2 Create an empty model

1. Create a subdirectory called `Model`.
2. Create a file called `ModelDesign.xml` in the subdirectory `Model`.
3. Put the following content into that file:

```
<?xml version="1.0" encoding="utf-8" ?>
<opc:ModelDesign
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:opc="http://opcfoundation.org/UA/ModelDesign.xsd"
  xsi:schemaLocation="http://opcfoundation.org/UA/ModelDesign.xsd ../../../../schema/UAModelDesign.xsd"
  xmlns:ua="http://opcfoundation.org/UA/"
  xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
  xmlns="http://samplecompany.com/SampleServer/Model"
  TargetNamespace="http://samplecompany.com/SampleServer/Model"
>
  <!--
    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:Namespa
    ce>
    <opc:Namespace Name="SampleServer" Prefix="SampleCompany.SampleServer.Model"
    InternalPrefix="SampleCompany.SampleServer.Model">http://samplecompany.com/SampleServer/Model</opc:Nam
    espace>
  </opc:Namespaces>

  <!-- Create Types -->

</opc:ModelDesign>
```

Important:

If you ensure that `../../../../schema/UAModelDesign.xsd` points to the `UAModelDesign.xsd` Visual Studio can give you some information about the different elements.

With the following entries you define the namespace for the new model:

```
xmlns="http://samplecompany.com/SampleServer/Model"
TargetNamespace="http://samplecompany.com/SampleServer/Model"
```

and

```
<opc:Namespace Name="SampleServer" Prefix="SampleCompany.SampleServer.Model"
InternalPrefix="SampleCompany.SampleServer.Model">http://samplecompany.com/SampleServer/Model</opc:Namespace>
```



4-3-3.3 Add some types to the model

Place the following content under `<!-- Create Types -->`:

```
<!-- MethodTypes -->
<opc:Method SymbolicName="GetMachineDataMethodType">
  <opc:OutputArguments>
    <opc:Argument Name="MachineName" DataType="ua:String"></opc:Argument>
    <opc:Argument Name="Manufacturer" DataType="ua:String"></opc:Argument>
    <opc:Argument Name="SerialNumber" DataType="ua:String"></opc:Argument>
    <opc:Argument Name="IsProducing" DataType="ua:Boolean"></opc:Argument>
    <opc:Argument Name="MachineState" DataType="ua:UInt32"></opc:Argument>
  </opc:OutputArguments>
</opc:Method>

<!--ObjectTypes -->
<opc:ObjectType SymbolicName="GenericControllerType" BaseType="ua:BaseObjectType">
  <opc:Children>
    <opc:Variable SymbolicName="SetPoint" DataType="ua:Double" TypeDefinition="ua:AnalogItemType" AccessLevel="ReadWrite"></opc:Variable>
    <opc:Variable SymbolicName="Measurement" DataType="ua:Double" TypeDefinition="ua:AnalogItemType"></opc:Variable>
  </opc:Children>
</opc:ObjectType>

<opc:ObjectType SymbolicName="FlowControllerType" BaseType="GenericControllerType"></opc:ObjectType>
<opc:ObjectType SymbolicName="LevelControllerType" BaseType="GenericControllerType"></opc:ObjectType>
<opc:ObjectType SymbolicName="TemperatureControllerType" BaseType="GenericControllerType"></opc:ObjectType>

<opc:ObjectType SymbolicName="MachineType" BaseType="ua:BaseObjectType">
  <opc:Children>
    <opc:Object SymbolicName="Temperature" TypeDefinition="TemperatureControllerType"></opc:Object>
    <opc:Object SymbolicName="Flow" TypeDefinition="FlowControllerType"></opc:Object>
    <opc:Object SymbolicName="Level" TypeDefinition="LevelControllerType"></opc:Object>
    <opc:Method SymbolicName="GetMachineData" TypeDefinition="GetMachineDataMethodType"></opc:Method>
  </opc:Children>
</opc:ObjectType>
```

4-3-3.4 Build the model

Now you can start the BuildDesign.bat. If it successfully finish the subdirectory **Model** should have the following content:

Name	Date modified	Type	Size
ModelDesign.csv	2/1/2021 4:30 PM	Microsoft Excel Com...	6 KB
ModelDesign.xml	2/1/2021 4:27 PM	XML Document	4 KB
SampleCompany.SampleServer.Model.Classes.cs	2/1/2021 4:30 PM	Visual C# Source File	39 KB
SampleCompany.SampleServer.Model.Constants.cs	2/1/2021 4:30 PM	Visual C# Source File	31 KB
SampleCompany.SampleServer.Model.DataTypes.cs	2/1/2021 4:30 PM	Visual C# Source File	2 KB
SampleCompany.SampleServer.Model.NodeIds.csv	2/1/2021 4:30 PM	Microsoft Excel Com...	1 KB
SampleCompany.SampleServer.Model.NodeSet.xml	2/1/2021 4:30 PM	XML Document	72 KB
SampleCompany.SampleServer.Model.NodeSet2.xml	2/1/2021 4:30 PM	XML Document	18 KB
SampleCompany.SampleServer.Model.PredefinedNodes.uanodes	2/1/2021 4:30 PM	UANODES File	2 KB
SampleCompany.SampleServer.Model.PredefinedNodes.xml	2/1/2021 4:30 PM	XML Document	31 KB
SampleCompany.SampleServer.Model.Types.bsd	2/1/2021 4:30 PM	BSD File	1 KB
SampleCompany.SampleServer.Model.Types.xsd	2/1/2021 4:30 PM	XML Schema File	1 KB



4.3.3.5 Embedded Resource Files

Ensure that all .uanodes files are used as Embedded Resource files in your project.

4.3.3.6 Extend the SampleServer.cs

Also, if this tutorial doesn't add any encodeable types it is highly recommended to add the following method to SampleCompany.SampleServer.cs:

```
#region Overridden Methods
/// <summary>
/// Adds all encodeable types to the server.
/// </summary>
/// <param name="uaServerData">The uaServerData data implementing the IUaServerData
interface.</param>
public override void AddEncodeableTypes(IUaServerData uaServerData)
{
    // add the types defined in the information model library to the factory.
    uaServerData.Factory.AddEncodeableTypes(GetType().GetTypeInfo().Assembly);
}
#endregion
```

4.3.3.7 Add the used namespace

You need to add the model design specific namespace(s) 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 namespaceUris = new string[2];
    namespaceUris[0] = Namespaces.SampleServer;
    namespaceUris[1] = Model.Namespaces.SampleServer;
    return namespaceUris;
}
```



4.3.3.8 Loading the generated types to the address space

To be able to use the generated model the defined nodes must be loaded. The nodes generated by the model compiler are in the file `SampleCompany.SampleServer.Model.PredefinedNodes.uanodes`. Loading this file can be done by implementing the following method in `SampleCompany.SampleServerNodeManager.cs`:

```
#region Overridden Methods
/// <summary>
/// Loads a node set from a file or resource and address 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. Therefore we search for it:
    var assembly = this.GetType().GetTypeInfo().Assembly;
    var names = assembly.GetManifestResourceNames();
    var resourcePath = String.Empty;

    foreach (var module in names)
    {
        if (module.Contains("SampleCompany.SampleServer.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;
    }

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




4.3.3.9 Using the generated types

`MachineType` is the main type and creating a variable using these types can be done by adding the following to the `CreateAddressSpace()` method in `SampleCompany.SampleServerNodeManager.cs`:

```
#region Plant
var plantFolder = CreateFolderState(root, "Plant", "Plant", null);
var machine1 = new Model.MachineState(null);
var 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);

var machine2 = new Model.MachineState(null);
var 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);
#endregion
```

This will create a folder `Plant` as well as to variables `Machine #1` and `Machine #2`.

4.3.3.10 Adding a method event handler

The model also contains a `MethodType` with name `GetMachineDataMethodType` and a method definition:

```
<opc:Method SymbolicName="GetMachineData" TypeDefinition="GetMachineDataMethodType"></opc:Method>
```

You can add a event handler for this method in the `SampleCompany.SampleServerNodeManager.cs` by adding the following code:

```
machine1.GetMachineData.OnCall = OnGetMachineDataMachine1;
```

and the event handler:

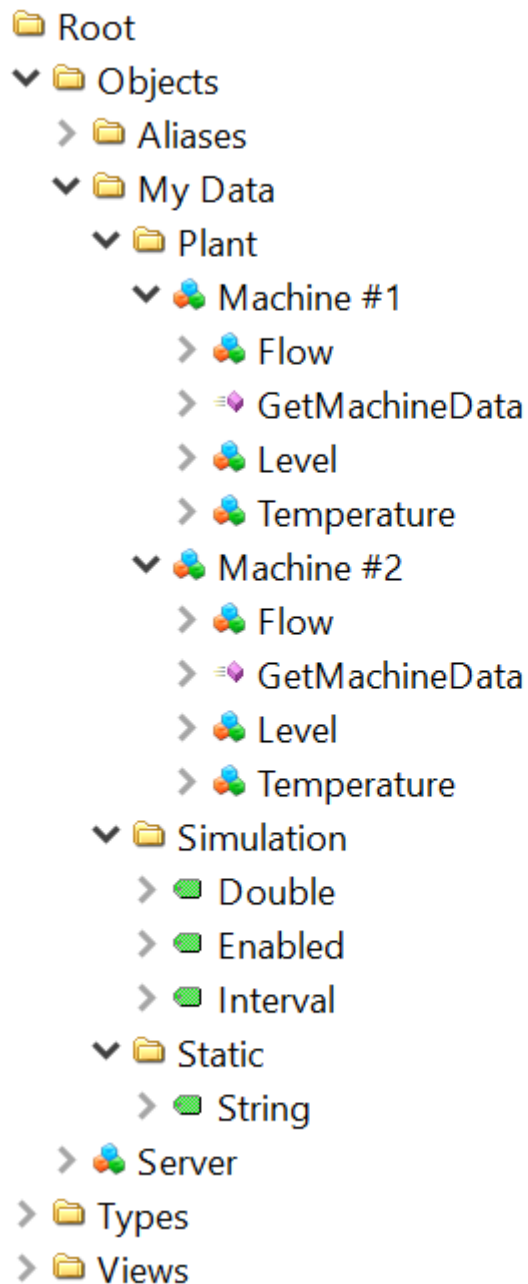
```
private ServiceResult OnGetMachineDataMachine1(ISystemContext context, MethodState method, NodeId objectid, ref string machinename, ref string manufacturer, ref string serialnumber, ref bool isproducing, ref uint machine state)
{
    machinename = "Machine #1";
    manufacturer = "SampleCompany";
    serialnumber = "SN 1079";
    isproducing = true;

    return ServiceResult.Good;
}
```

T

4.3.4 Testing your OPC UA server

You should now be able to build and start the OPC UA server. Using the Unified Automation UaExpert you can use to connect to the OPC UA server and should see the following address space:



You can drag&drop the variables “Double”, “Enabled” and “Interval” to the Data Access View and see the value of the “Double” variable changing. By changing the “Interval” the update interval of the “Double” should change and with setting “Enabled” to false no more changes should happen to the “Double” variable.

4.3.5 SampleServer project

For your convenience the resulting sample server is also available in the distribution at
`\examples\DataTypes\SampleServer`



4.4 Tutorial “User Authentication OPC UA Server”

This chapter describes how to add user authentication to your OPC UA server. This tutorial is based on the base SampleServer project. It shows the basic steps required to add the different methods to add user authentication.

4.4.1 Goals of this tutorial

You will learn the following topics:

1. Minimum required changes to use user authentication.

4.4.2 Start adding user authentication

You need the following files to be able to develop your own OPC UA Server:

1. Copy the directory with the base OPC UA Server at \examples\Base\SampleServer, to your development directory, e.g. c:\solutions\examples\UserAuthentication\SampleServer

In the following chapters you will find some required and recommended changes necessary before you start changing the source code. For a first OPC UA Server we recommend changing only those values and keep everything else untouched. As you proceed further with this tutorial more and more changes and enhancements are done.

The solution provides the UserIdentity class which converts UA user identity tokens to and from the SecurityTokens. The solution currently supports

- Anonymous
- Username

user authentication.

4.4.2.1 ImpersonateUserEvent

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

```
#region Overridden Methods
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;
}
#endregion

#region User Validation Functions
/// <summary>
/// Called when a client tries to change its user identity.
/// </summary>
private void OnImpersonateUser(object sender, UaImpersonateUserEventArgs args)
{
    // check for a user name token.
    if (args.NewIdentity is UserNameIdentityToken userNameToken)
    {
        args.Identity = VerifyPassword(userNameToken);
    }
}
}
```



```
/// <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.
        var 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 following username/password combinations are supported:

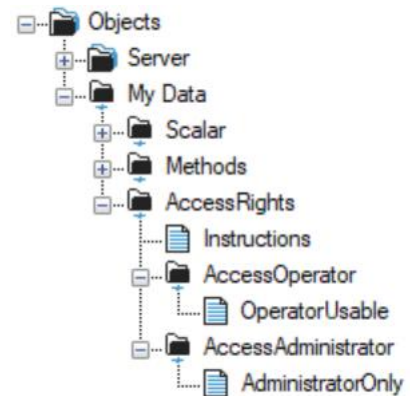
- operator / password1
- administrator / password2

4.4.3 Address Space creation

The sample 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 SampleCompany.SampleServerNodeManager.cs and involves the following methods:



4.4.3.1 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 and is done with:

```

#region Access Rights Handling
var folderAccessRights = CreateFolderState(root, "AccessRights", "AccessRights", null);
const string accessRights = "AccessRights_";
var accessRightsInstructions = CreateBaseDataVariableState(folderAccessRights, accessRights +
    "Instructions", "Instructions", null, DataTypeIds.String, ValueRanks.Scalar,
    AccessLevels.CurrentReadOrWrite, null);
accessRightsInstructions.Value = "This folder will be accessible to all authenticated users who enter, but
contents therein will be secured.";

#region Access Rights Operator Handling
// sub-folder for "AccessOperator"
var folderAccessRightsAccessOperator = CreateFolderState(folderAccessRights,
    "AccessRights_AccessOperator", "AccessOperator", null);
const string accessRightsAccessOperator = "AccessRights_AccessOperator_";

var arOperatorRW = CreateBaseDataVariableState(folderAccessRightsAccessOperator,
    accessRightsAccessOperator + "OperatorUsable", "OperatorUsable", null, BuiltInType.Int16, ValueRanks.Scalar,
    AccessLevels.CurrentReadOrWrite, null);
arOperatorRW.AccessLevel = AccessLevels.CurrentReadOrWrite;
arOperatorRW.UserAccessLevel = AccessLevels.CurrentReadOrWrite;
arOperatorRW.OnReadUserAccessLevel = OnReadOperatorUserAccessLevel;
arOperatorRW.OnSimpleWriteValue = OnWriteOperatorValue;
arOperatorRW.OnReadValue = OnReadOperatorValue;
dynamicNodes_.Add(arOperatorRW);
#endregion

#region Access Rights Administrator Handling
// sub-folder for "AccessAdministrator"
var folderAccessRightsAccessAdministrator = CreateFolderState(folderAccessRights,
    "AccessRights_AccessAdministrator", "AccessAdministrator", null);
const string accessRightsAccessAdministrator = "AccessRights_AccessAdministrator_";

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



4.4.4 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)
        {
            if (context.UserIdentity.GetIdentityToken() is UserNameIdentityToken user &&
                user.UserName == "administrator")
            {
                value = AccessLevels.CurrentReadOrWrite;
            }
            else
            {
                value = AccessLevels.None;
            }
        }
    }
    return ServiceResult.Good;
}
```

For reading a variable the following event handler was defined:

```
arAdministratorRW.OnReadValue = OnReadAdministratorValue;
```

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

T

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



4.4.5 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. The sample shown here only hides those variables from anonymous users. The operator user still can see the administrator related variables.

4.4.5.1 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 look like:

```
protected override bool IsReferenceAccessibleForUser(UaServerContext context,
                                                    UaContinuationPoint continuationPoint, IReference reference)
{
    if (context.UserIdentity == null || context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        if ((reference.TargetId.Identifier.ToString() == "AccessRights_AccessAdministrator") ||
            (reference.TargetId.Identifier.ToString() == "AccessRights_AccessOperator"))
        {
            return false;
        }
    }
    return true;
}
```

4.4.5.2 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;
}
```




4.4.6 Testing your OPC UA server

You should now be able to build and start the OPC UA server. Using the Unified Automation UaExpert you can use to connect to the OPC UA server and should see the following address space:

4.4.6.1 Anonymous user

- Root
 - Objects
 - Aliases
 - My Data
 - AccessRights
 - Instructions
 - Simulation
 - Static
 - Server
 - Types
 - Views

For the variables you should get the following:

#	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	NS2 Strin...	AdministratorOnly		Null	9:29:31.214 AM	9:29:31.215 AM	BadUserAccessDenied
2	NS2 Strin...	OperatorUsable	16308	Int16	9:30:45.224 AM	9:30:45.224 AM	Good

4.4.6.2 Operator or Administrator user

- Root
 - Objects
 - Aliases
 - My Data
 - AccessRights
 - AccessAdministrator
 - AdministratorOnly
 - AccessOperator
 - OperatorUsable
 - Instructions
 - Simulation
 - Static
 - Server
 - Types
 - Views

For the variables you should get the following:

#	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	NS2 Strin...	AdministratorOnly	26589	Int16	9:32:21.225 AM	9:32:21.225 AM	Good
2	NS2 Strin...	OperatorUsable	-15761	Int16	9:32:21.225 AM	9:32:21.225 AM	Good

4.4.7 SampleServer project

For your convenience the resulting sample server is also available in the distribution at

\examples\UserAuthentication\SampleServer



4.5 Tutorial “Reverse Connect enabled OPC UA Server”

This chapter describes how to add reverse connect handling to your OPC UA server. This tutorial is based on the base SampleServer project. It shows the basic steps required to add the different methods to add reverse connect functionality.

4.5.1 Goals of this tutorial

You will learn the following topics:

1. Minimum required changes to use support the reverse connect feature.

4.5.2 Start adding reverse connect functionality

You need the following files to be able to develop your own OPC UA Server:

1. Copy the directory with the base OPC UA Server at \examples\Base\SampleServer, to your development directory, e.g. c:\solutions\examples\ReverseConnect\SampleServer

In the following chapters you will find some required and recommended changes necessary before you start changing the source code. For a first OPC UA Server we recommend changing only those values and keep everything else untouched. As you proceed further with this tutorial more and more changes and enhancements are done.

To be able to use the Reverse Connect Feature the following code must be added to your server:

4.5.3 UaServerPlugin.cs

To enable the Reverse Connect Feature the UaServerPlugin must implement the [IUaReverseConnectServerPlugin](#) interface and the `OnUseReverseConnect()` method must return `true` as shown in the sample below:

```
public class UaServerPlugin : IUaServerPlugin, IUaOptionalServerPlugin,
                             IUaReverseConnectServerPlugin, IDisposable
{
    #region Reverse Connect Server Plugin methods
    public bool OnUseReverseConnect()
    {
        return true;
    }
    #endregion
}
```



4.5.4 Program.cs

The configuration of the client Url's to be used for reverse connect can be done after starting the server. For this example, like this:

```
public static int Main(string[] args)
{
    Console.WriteLine("Technosoftware {0} OPC UA Simple Server", Utils.IsRunningOnMono() ?
        "Mono" : ".NET Core");

    // command line options
    var showHelp = false;
    var stopTimeout = 0;
    var autoAccept = false;
    string reverseConnectUrlString = null;
    Uri reverseConnectUrl = null;

    var options = new Mono.Options.OptionSet {
        { "h|help", "show this message and exit", h => showHelp = h != null },
        { "a|autoaccept", "auto accept certificates (for testing only)",
            a => autoAccept = a != null },
        { "t|timeout=", "the number of seconds until the server stops.", (int t) => stopTimeout = t },
        { "rc|reverseconnect=", "Connect using the reverse connection.",
            url => reverseConnectUrlString = url },
    };

    try
    {
        IList<string> extraArgs = options.Parse(args);
        foreach (var extraArg in extraArgs)
        {
            Console.WriteLine("Error: Unknown option: {0}", extraArg);
            showHelp = true;
        }
        if (reverseConnectUrlString != null)
        {
            reverseConnectUrl = new Uri(reverseConnectUrlString);
        }
    }
    catch (OptionException e)
    {
        Console.WriteLine(e.Message);
        showHelp = true;
    }

    if (showHelp)
    {
        Console.WriteLine(Utils.IsRunningOnMono() ? "Usage: mono Technosoftware.SimpleServer.exe"
            "[OPTIONS]" : "Usage: dotnet Technosoftware.SimpleServer.dll [OPTIONS]");
        Console.WriteLine();

        Console.WriteLine("Options:");
        options.WriteOptionDescriptions(Console.Out);
        return (int)ExitCode.ErrorInvalidCommandLine;
    }

    var server = new MySampleServer(autoAccept, stopTimeout, reverseConnectUrl);
    server.Run();

    return (int)MySampleServer.ExitCode;
}
```



```
public class MySampleServer
{
    #region Properties
    public Task Status { get; private set; }
    public DateTime LastEventTime { get; private set; }
    public int ServerRunTime { get; private set; }
    public static bool AutoAccept { get; private set; }
    public static ExitCode ExitCode { get; private set; }
    public Uri ReverseConnectUrl { get; private set; }

    private static UaServer.UaServer uaServer_ = new UaServer.UaServer();
    private static UaServerPlugin uaServerPlugin_ = new UaServerPlugin();
    #endregion

    #region Constructors, Destructor, Initialization
    public MySampleServer(bool autoAccept, int stopTimeout)
        : this(autoAccept, stopTimeout, null)
    {
    }

    public MySampleServer(bool autoAccept, int stopTimeout, Uri reverseConnectUrl)
    {
        AutoAccept = autoAccept;
        ServerRunTime = stopTimeout == 0 ? Timeout.Infinite : stopTimeout * 1000;
        ReverseConnectUrl = reverseConnectUrl;
    }

    private async Task SampleServer()
    {
        // start the server.
        await uaServer_.StartAsync(uaServerPlugin_, "SampleCompany.SampleServer", null);

        if (ReverseConnectUrl != null)
        {
            uaServer_.BaseServer.AddReverseConnection(ReverseConnectUrl);
        }

        var reverseConnections = uaServer_.BaseServer.GetReverseConnections();
        if (reverseConnections?.Count > 0)
        {
            // print reverse connect info
            Console.WriteLine("Reverse Connect Clients:");
            foreach (var connection in reverseConnections)
            {
                Console.WriteLine(connection.Key);
            }
        }

        // print endpoint info
        Console.WriteLine("Server Endpoints:");
        var endpoints = uaServer_.BaseServer.GetEndpoints().Select(e => e.EndpointUrl).Distinct();
        foreach (var endpoint in endpoints)
        {
            Console.WriteLine(endpoint);
        }

        // start the status thread
        Status = Task.Run(StatusThread);

        // print notification on session events
        uaServer_.BaseServer.CurrentInstance.SessionManager.SessionActivatedEvent += EventStatus;
        uaServer_.BaseServer.CurrentInstance.SessionManager.SessionClosingEvent += EventStatus;
        uaServer_.BaseServer.CurrentInstance.SessionManager.SessionCreatedEvent += EventStatus;
    }
}
```

4.5.5 Testing your OPC UA server

You should now be able to build and start the OPC UA server. To enable reverse connection functionality, you have to add the following as application argument:

```
-rc=opc.tcp://localhost:55557
```

Of course, you can choose a different port.

Using the Unified Automation UaExpert you can enable the reverse connect feature and add the chosen port, e.g. 55557 to the port property. That will look then similar to:

Server Settings - SampleCompany OPC UA Sample Server

Configuration

Configuration Name: SampleCompany OPC UA Sample Server

Server Information

Endpoint Url: opc.tcp://technosoftware:55555/SampleServer

Reverse Connect: ☒

Client Endpoint URL: opc.tcp://technosoftware:55557

Host: technosoftware ☐ Override

Port: 55557

Server Certificate: Length=1281, Content=308204fd308203e5a003020102020a ... Load file...

Security Settings

Security Policy: None

Message Security Mode: None

Authentication Settings

☒ Anonymous

☐ Username: ☐ Store

☐ Password:

☐ Certificate: ...

☐ Private Key: ...

Session Settings

Session Name: urn:technosoftware:UnifiedAutomation:UaExpert

OK Cancel



The server output then should look similar to:

```
C:\opcua-solution-net-src-build\distr\dotnetcore\examples\ReverseConnect\SampleServer\bin\Debug\netcoreapp3.1\SampleCompany....
SampleCompany .NET Core OPC UA Sample Server
Reverse Connect Clients:
opc.tcp://localhost:55557/
Server Endpoints:
opc.tcp://technosoftware:55555/SampleServer
https://technosoftware:55556/SampleServer/
Server started. Press Ctrl-C to exit...
Created:urn:technosoftware:UnifiedAutomation:UaExpert:: Anonymous:ns=3;i=477761461
Activated:urn:technosoftware:UnifiedAutomation:UaExpert:: Anonymous:ns=3;i=477761461
-Status-:urn:technosoftware:UnifiedAutomation:UaExpert:Last Event:11:51:30
-Status-:urn:technosoftware:UnifiedAutomation:UaExpert:Last Event:11:51:35
-Status-:urn:technosoftware:UnifiedAutomation:UaExpert:Last Event:11:51:40
```

4.5.6 SampleServer project

For your convenience the resulting sample server is also available in the distribution at
`\examples\ReverseConnect\SampleServer`



5 Use of the different Methods

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

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

```
// start the server.
await uaServer_.StartAsync(uaServerPlugin_, "SampleCompany.SampleServer", null);
```

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

5.1.2 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)
{
    Utils.Trace(Utils.TraceMasks.Information, "OnStartup(): Server is starting up.");
    return StatusCodes.Good;
}
```

5.1.3 OnGetLicenseInformation

This method is called from the generic server to get the license information. The base implementation looks like:

```
public void OnGetLicenseInformation(out string serialNumber)
{
    Utils.Trace(Utils.TraceMasks.Information,
        "OnGetLicenseInformation(): Request the license information.");
    serialNumber = "";
}
```

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



5.1.4 OnGetServer

To be able to use advanced features of the solution the base implementation returns here an instance of the `EmptyServer` which then can override some methods of the `UaBaseServer` class and looks like:

```
public UaBaseServer OnGetServer()
{
    Utils.Trace(Utils.TraceMasks.Information,
        "OnGetServer(): Request the instance of the server.");
    return new EmptyServer();
}
```

All example servers are now using this feature and therefore override this method.

5.1.5 OnGetServerProperties

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

```
public ServerProperties OnGetServerProperties()
{
    Utils.Trace(Utils.TraceMasks.Information,
        "OnGetServerProperties(): Request some standard information of the server.");
    var properties = new ServerProperties
    {
        ManufacturerName = "EmptyCompany",
        ProductName = "EmptyCompany OPC UA Empty Server",
        ProductUri = "http://emptycompany.com/EmptyServer/v1.0",
        SoftwareVersion = GetAssemblySoftwareVersion(),
        BuildNumber = GetAssemblyBuildNumber(),
        BuildDate = GetAssemblyTimestamp()
    };

    return properties;
}
```

5.1.6 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 `EmptyServer`. It looks like:

```
public string[] OnGetNamespaceUris()
{
    Utils.Trace(Utils.TraceMasks.Information,
        "OnGetNamespaceUris(): Request the supported namespace Uris.");
    // set one namespace for the type model.
    var namespaceUris = new string[1];
    namespaceUris[0] = Namespaces.EmptyServer;
    return namespaceUris;
}
```




5.1.7 OnGetNodeManager

To be able to use advanced features of the solution the EmptyServer returns here an instant of the [EmptyServerNodeManager](#) 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)
{
    Utils.Trace(Utils.TraceMasks.Information,
        "OnGetNodeManager(): Request the instance of the node manager.");
    return new EmptyServerNodeManager(opcServer, this, uaServer, configuration,
        namespaceUris);
}
```

The [EmptyServerNodeManager](#) 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.

5.1.8 OnInitialized

This method is called after the node manager is initialized. So if something fails in the [EmptyServerNodeManager](#) this method may not be reached. It looks like:

```
public void OnInitialized(IUaServer opcServer, ApplicationConfiguration configuration)
{
    Utils.Trace(Utils.TraceMasks.Information, "OnInitialized(): Server is initialized.");
    opcServer_ = opcServer;
}
```

5.1.9 OnRunning

This method is called from the generic server when the server was successfully started and is running. This could be the place where the UI is initialized and started up:

```
public StatusCode OnRunning()
{
    Utils.Trace(Utils.TraceMasks.Information, "OnRunning(): Server is running.");
    return StatusCodes.Good;
}
```

5.1.10 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)
{
    Utils.Trace(Utils.TraceMasks.Information,
        "OnShutdown(): Server is shutting down because of {0}.", reason);
    return StatusCodes.Good;
}
```



5.2 Address Space Creation

The creation of the address space is done in the node manager 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.

5.2.1 Creating variables in the address space

Variables are created always in the following sequence:

- 1) Create the root folder

```
var root = CreateFolderState(null, "My Data", new LocalizedText("en", "My Data"),  
                             new LocalizedText("en", "Root folder of the Sample Server"));  
References.Add(new NodeStateReference(ReferenceTypes.Organizes, false, root.NodeId));  
root.EventNotifier = EventNotifiers.SubscribeToEvents;  
opcServer_.AddRootNotifier(root);
```

- 2) Create the sub folder

```
var staticFolder = CreateFolderState(root, "Static", "Static",  
                                     "A folder with a sample static variable.");
```

- 3) Create the variable

```
const string scalarStatic = "Static_";  
CreateBaseDataVariableState(staticFolder, scalarStatic + "String", "String", null,  
                             DataTypeIds.String, ValueRanks.Scalar, AccessLevels.CurrentReadOrWrite, null);
```

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 and are located under the `root` the server must add them to the list of predefined nodes. This is done at the end of the method `CreateAddressSpace`:

```
AddPredefinedNode(SystemContext, root);
```

After that call the address space of the OPC-UA server is ready.



6 Configuration

6.1 Application Configuration

The solution 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: <code>http://machine1/OPC/UASampleServer/4853DB1C-776D-4ADA-9188-00CAA737B780</code>
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 <code>Utils.Trace(...)</code> functions.
Extensions	XmlElementCollection	Allows developers to add additional information to the file.

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 is normally in the same directory as the executable. It can be loaded as shown in the following example:

```
// start the server.  
await uaServer_.StartAsync(uaServerPlugin_, "Technosoftware.EmptyServer", null);
```

The Application Configuration file of the EmptyServer can be found in the file `Technosoftware.EmptyServer.Config.xml`.



6.1.1 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://emptycompany.com/EmptyServer">
      <ConfigurationFile>.\EmptyServerConfiguration.xlsx</ConfigurationFile>
    </WorkshopServerConfiguration>
  </ua:XmlElement>
</Extensions>
```

Important:

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

6.1.2 Tracing Output

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

EmptyServer:

%LocalApplicationData%/Logs/Technosoftware.EmptyServer.log

where

%CommonApplicationData% typically points to C:\ProgramData



7 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 Administrator (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 solution 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.





8 UA Model Compiler

The Model Compiler (Technosoft\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 Technosoft\UaModelCompiler.exe we only guarantee that the version delivered fits the solution.

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.

8.1 Command Line Usage

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

```
Technosoft\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.



8.2 XSD Specification

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

8.2.1 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:Namespace>
  <opc:Namespace Name="Engineering" Prefix="Technosoftware.WorkshopServerForms.Engineering"
InternalPrefix="Technosoftware.WorkshopServerForms.Engineering">http://emptycompany.com/WorkshopServerForms/Engineering</opc:Namespace>
  <opc:Namespace Name="Operations" Prefix="Technosoftware.WorkshopServerForms.Operations"
InternalPrefix="Technosoftware.WorkshopServerForms.Operations">http://emptycompany.com/WorkshopServerForms/Operations</opc:Namespace>
  <opc:Namespace Name="WorkshopServerForms" Prefix="Technosoftware.WorkshopServerForms.Model"
InternalPrefix="Technosoftware.WorkshopServerForms.Model">http://emptycompany.com/WorkshopServerForms/Model</opc:Namespace>
</opc:Namespaces>
```

8.2.2 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>
```




8.2.3 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>
```



8.2.4 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>
```

8.2.5 Variable

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

8.2.6 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>
```

8.2.7 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>
```

8.2.8 Property

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

T

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 are, 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

