OPC UA Server SDK .NET Standard

Develop OPC UA Servers with C# / VB.NET

Tutorial Workshop Server

Document Control

Version	Date	Comment
1.0.8	18-MAY-2019	Initial version based on SDK 1.0.8
1.0.9	31-MAY-2019	Updated chapter 2

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

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

Summary

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

Referenced OPC Documents

Documents
This document partly uses extracts taken from the OPC UA specifications to be able to give at least a short introduction into the specifications. The specifications itself are available from: http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC#Specifications
OPC Unified Architecture Textbook, written by Wolfgang Mahnke, Stefan-Helmut Leitner and Matthias Damm: http://www.amazon.com/OPC-Unified-Architecture-Wolfgang-Mahnke/dp/3540688986/ref=sr_1_1?ie=UTF8&s=books&qid=1209506074&sr=8-1
[UA Part 1] OPC UA Specification: Part 1 – Concepts http://www.opcfoundation.org/UA/Part1/
[UA Part 2] OPC UA Specification: Part 2 – Security http://www.opcfoundation.org/UA/Part2/
[UA Part 3] OPC UA Specification: Part 3 – Address Space Model http://www.opcfoundation.org/UA/Part3/
[UA Part 4] OPC UA Specification: Part 4 – Services http://www.opcfoundation.org/UA/Part4/
[UA Part 5] OPC UA Specification: Part 5 – Information Model http://www.opcfoundation.org/UA/Part5/
[UA Part 6] OPC UA Specification: Part 6 – Mappings http://www.opcfoundation.org/UA/Part6/
[UA Part 7] OPC UA Specification: Part 7 – Profiles http://www.opcfoundation.org/UA/Part7/
[UA Part 8] OPC UA Specification: Part 8 – Data Access http://www.opcfoundation.org/UA/Part8/
[UA Part 9] OPC UA Specification: Part 9 – Alarm & Conditions http://www.opcfoundation.org/UA/Part9/
[UA Part 10] OPC UA Specification: Part 10 – Programs http://www.opcfoundation.org/UA/Part10/
[UA Part 11] OPC UA Specification: Part 11 – Historical Access http://www.opcfoundation.org/UA/Part11/
[UA Part 12] OPC UA Specification: Part 12 – Discovery and Global Services http://www.opcfoundation.org/UA/Part12/
[UA Part 13] OPC UA Specification: Part 13 – Aggregates http://www.opcfoundation.org/UA/Part13/
[UA Part 14] OPC UA Specification: Part 14 – PubSub https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub/

Other Referenced Documents

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

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

SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts

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

XML Encryption: XML Encryption Syntax and Processing

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

XML Signature:: XML-Signature Syntax and Processing

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

WS Security: SOAP Message Security 1.1

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

WS Addressing: Web Services Addressing (WS-Addressing)

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

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

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

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

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

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

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

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

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

:X509: X.509 Public Key Certificate Infrastructure

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

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

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

HTTPS: RFC 2818: HTTP Over TLS

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

IS Glossary: Internet Security Glossary

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

NIST 800-12: Introduction to Computer Security

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

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

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

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

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

IEC 62351: Data and Communications Security

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

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

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

SHA-1: Secure Hash Algorithm RFC

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

PKI: Public Key Infrastructure article in Wikipedia

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

X509 PKI: Internet X.509 Public Key Infrastructure

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

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

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

Table of Contents

Disclaimer

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

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

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

Trademark Notice

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

1 Installation

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

1. **OPC Sample Binaries .NET Standard**
   A suite of OPC Clients and Servers that demonstrate OPC UA, OPC DA, OPC AE and OPC HDA technology and its most popular functionality. All examples are ready to run without any configuration.
   You can donload it from https://technosoftware.com/download/opc-sample-binaries-net-standard/
2. **OPC UA Bundle SDK .NET Standard**
   The OPC UA Bundle SDK .NET Standard offers a fast and easy access to the OPC UA Client & Server technology. Develop OPC compliant UA Clients and Servers with C#/VB.NET targeting the .NET Standard.
   The download includes examples for .NET 4.6.1, .NET 4.7.2 and for .NET Standard 2.0.
   You can download it from https://technosoftware.com/download/opc-ua-bundle-sdk-net-standard-evaluation/

Important:

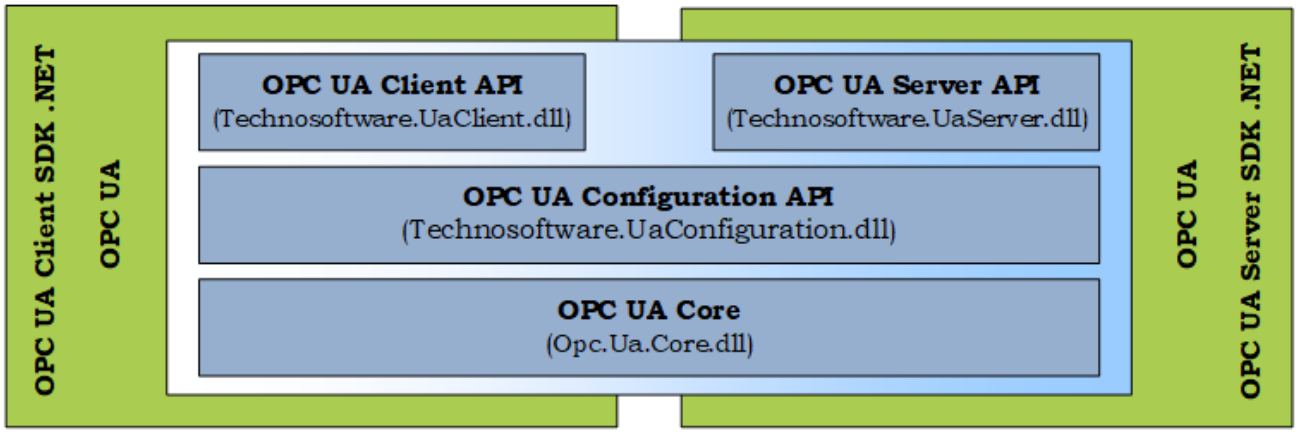
An installation guide is available with the SDK or from https://technosoftware.com/download/opc-ua-net-installation/. Please read that one first and then follow this guide.

2 Sample Applications

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

2.1 Required SDK DLLs

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



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

Name	Description
Opc.Ua.Core.dll	The OPC UA Stack. Based on the OPC Unified Architecture .NET Standard. We deliver three versions, one for .NET 4.6.1, one for .NET 4.7.2 and one for .NET Standard 2.0.
Technosoftware.UaConfiguration.dll	Contains configuration related classes like, e.g. `ApplicationInstance`. We deliver three versions, one for .NET 4.6.1, one for .NET 4.7.2 and one for .NET Standard 2.0.
Technosoftware.UaServer.dll	The DLL containing the classes and methods usable for OPC UA Server development. We deliver three versions, one for .NET 4.6.1, one for .NET 4.7.2 and one for .NET Standard 2.0.

2.2 Directory Structure

The basic directory layout is as follows:

- **bin/**
  - **net461/**
    Standard SDK Executables and DLL's for the .NET 4.6.1 Framework
  - **net472/**
    Standard SDK Executables and DLL's for the .NET 4.7.2 Framework
  - **netstandard2.0/**
    Standard SDK Executables and DLL's for the .NET Standard 2.0 and .NET Core 2.0 Framework
  - **redist/**
    - **OPC UA Local Discovery Server 1.03/**
      The installer and Merge-Module for the OPC UA Local Discovery Server
- **doc/**
  Additional documentation
- **examples/**
  Sample applications
  - **net/**
    - **C#/**
      - **AlarmConditions/**
        Server with Windows Forms UI showing the Alarm&Condition features
      - **CommonControls/**
        Windows Forms Controls used by the WorkshopClientSample. Contains the TitleBarControl which allows adapting logo and text of the title
      - **DataAccess/**
        Server with Windows Forms UI showing the Data Access features
      - **HistoricalAccess/**
        Server with Windows Forms UI showing the Historical Access features
      - **Reference/**
        Server used for testing with the OPC Foundation Complianc Test Tool
      - **ServerControls/**
        Windows Forms Controls used by the WorkshopServerForms
      - **Views/**
        Server with Windows Forms UI showing the Views features.
  - **Workshop/**
    - **ServerConsole/**
      Server as Console application used for this introduction. Features .NET 4.6.1, .NET 4.7.2 and .NET Core 2.0 compilation within one solution.
    - **ServerForms/**
      Server with Windows Forms used for this introduction
- **keys/**
  The dummy Key for signing the executables and DLL's
- **scripts/**
  Scripts and executables used for building the applications

2.3 OPC UA Server Solution

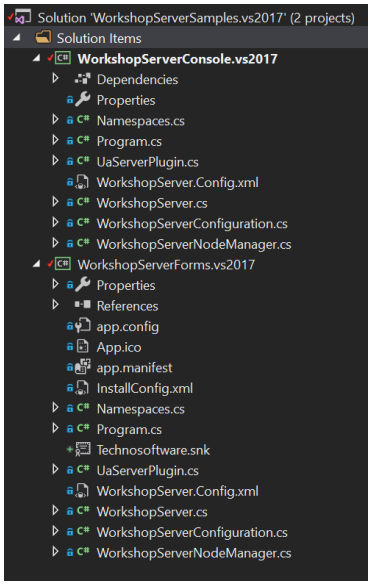
The main OPC UA Server Solutions can be found at \examples\Workshop\ and are named

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

and uses in addition the output of the following solutions:

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

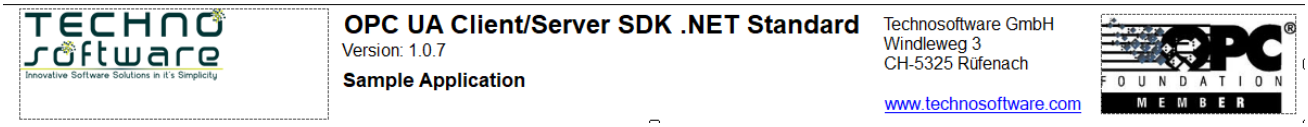
The solution contains two sample servers, one a console-based server and one a Windows Forms based server. The OPC UA functionality of both is the same.



2.3.1 CommonControls

2.3.1.1 Customizing the TitleBarControl

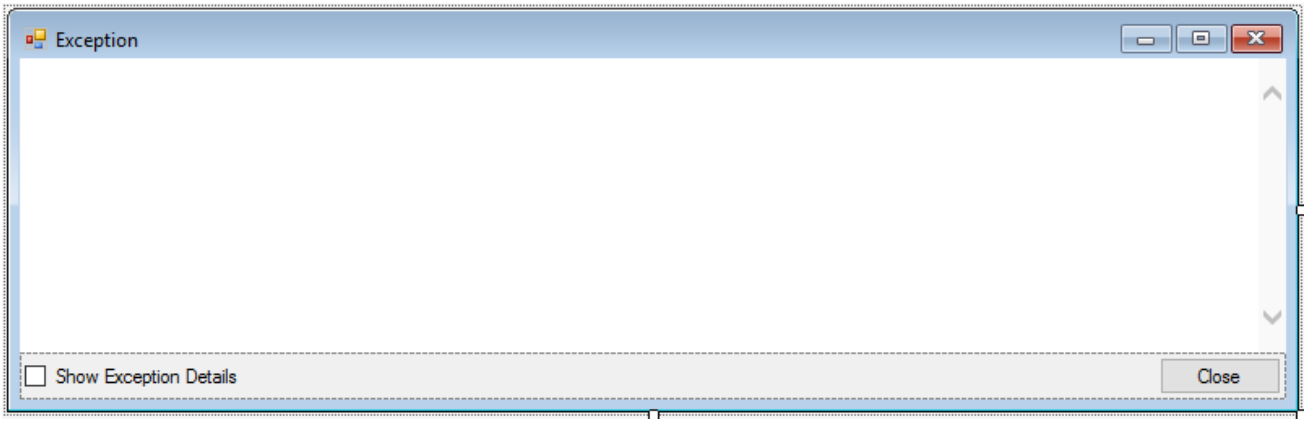
The TitleBarControl contains the header of all Windows Forms based sample server and sample client solutions provided with the SDK, e.g. WorkshopServerForms. The following picture shows how it looks like as default:



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

2.3.1.2 Customizing the ExceptionDlg

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

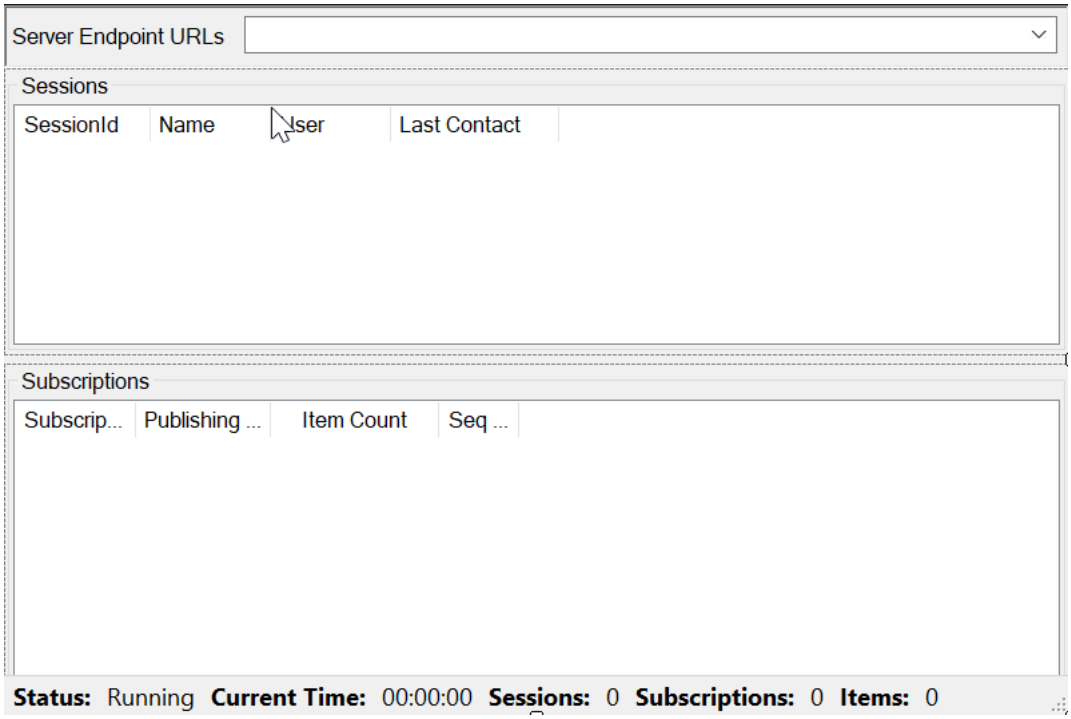


By changing this dialog, you can adapt the layout of the exception dialog for the WorkshopServerForms to your needs.

2.3.2 ServerControls

2.3.2.1 Customizing the ServerDiagnosticControl

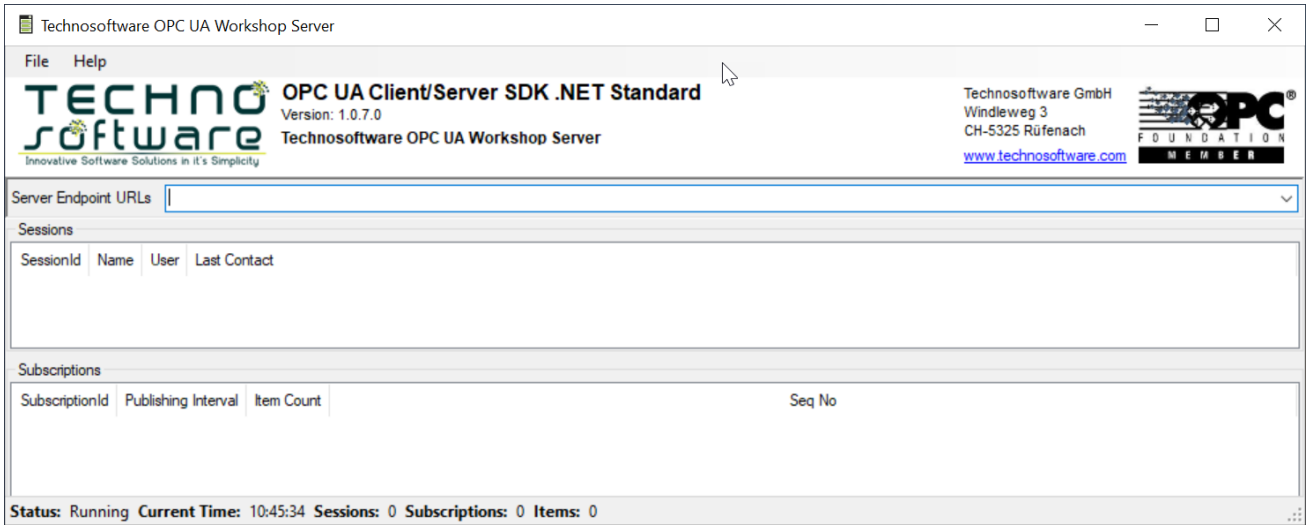
The ServerDiagnosticControl display information about the active sessions and subscriptions made by one or more OPC UA Clients. The following picture shows how it looks like as default:



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

2.3.2.2 Customizing the ServerForm

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



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

3 Configuration

3.1 Application Configuration

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

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

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

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

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

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

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

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

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

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

3.1.1 Extensions

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

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

The Extension looks like:

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

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

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

string configurationFile = configuration_.ConfigurationFile;
```

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

3.1.2 Tracing Output

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

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

where

%CommonApplicationData% typically points to C:\ProgramData

3.2 Installed Application

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

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

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

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

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

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

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

</s0:InstalledApplication>
```

4 Certificate Management and Validation

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

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

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

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

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

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

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

5 UserIdentity and UserIdentityTokens

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

- Anonymous
- Username
- X.509 certificate

user authentication.

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

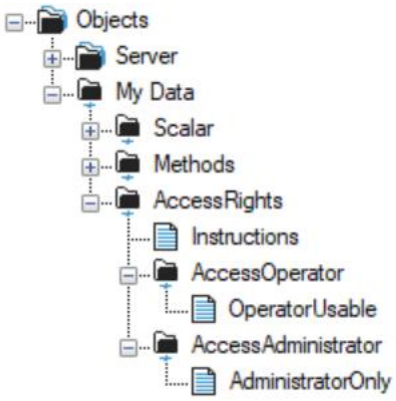
```
protected override void OnServerStarted(IUaServerData server)
{
    base.OnServerStarted(server);

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

}


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

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

The following username/password combinations are supported:

- operator / password1
- administrator / password2

The Workshop Server creates two variables under the AccessRights folder

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



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

CreateAddressSpace():

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

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

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

BaseDataVariableState arAdministratorRW = CreateVariable(folderAccessRightsAccessAdministrator,
accessRightsAccessAdministrator + "AdministratorOnly", "AdministratorOnly", BuiltInType.Int16,
ValueRanks.Scalar);
                    arAdministratorRW.AccessLevel = AccessLevels.CurrentReadOrWrite;
                    arAdministratorRW.UserAccessLevel = AccessLevels.CurrentReadOrWrite;
                    variables.Add(arAdministratorRW);

arAdministratorRW.OnReadUserAccessLevel = OnReadAdministratorUserAccessLevel;
arAdministratorRW.OnSimpleWriteValue = OnWriteAdministratorValue;
#endregion
```

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 (context.UserIdentity == null || context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        value = AccessLevels.CurrentRead;
    }
    else
    {
        if (context.UserIdentity.TokenType == UserTokenType.UserName)
        {
            UserNameIdentityToken user = context.UserIdentity.GetIdentityToken() as UserNameIdentityToken;
            if (user.UserName == "administrator")
            {
                value = AccessLevels.CurrentReadOrWrite;
            }
            else
            {
                value = AccessLevels.CurrentRead;
            }
        }
    }
    return ServiceResult.Good;
}
```

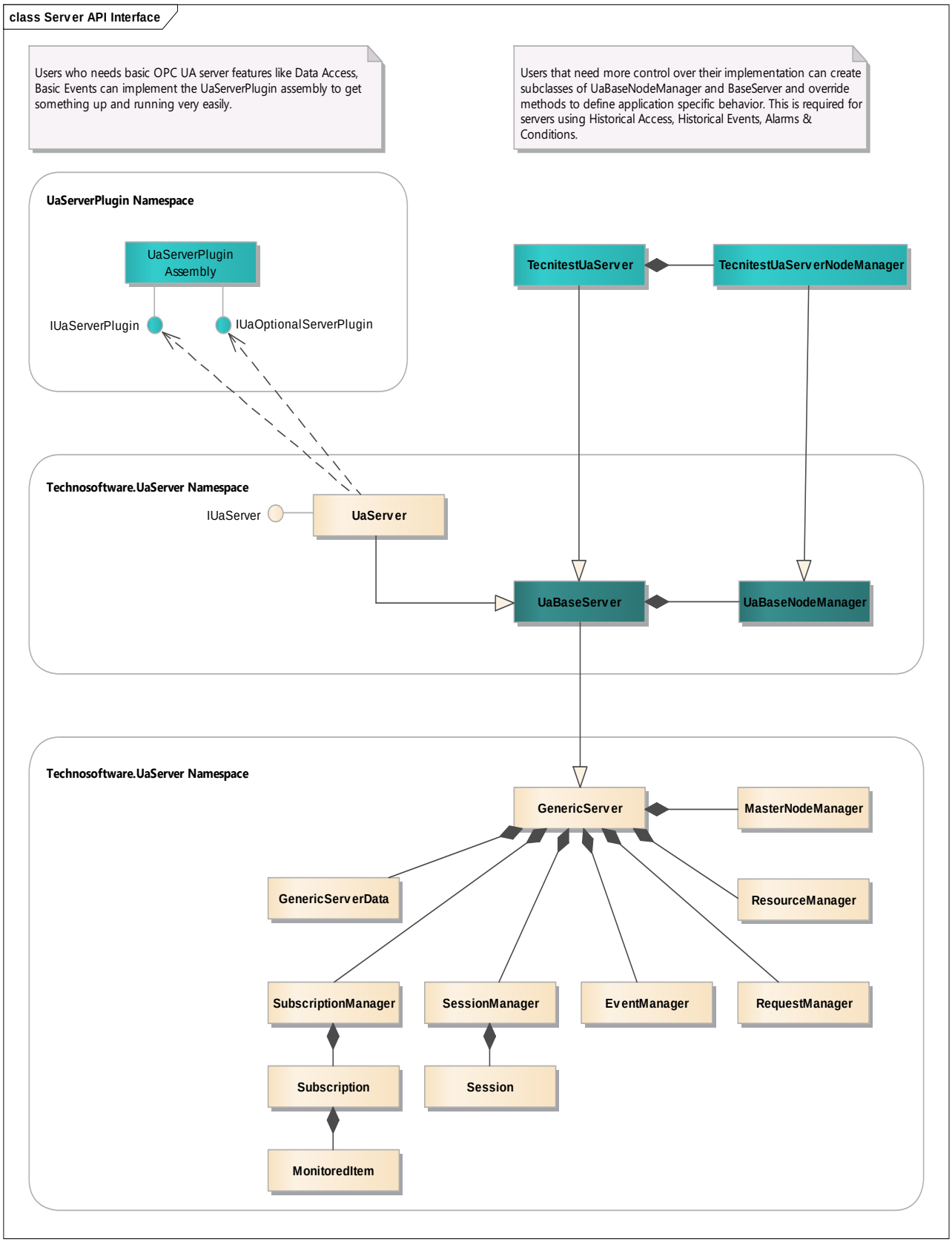
Also, during creation of the address space, the following event handler was defined for writing the variable:

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

and is implemented like this:

```
public ServiceResult OnWriteAdministratorValue(ISystemContext context, NodeState node, ref object value)
{
    if (context.UserIdentity == null || context.UserIdentity.TokenType == UserTokenType.Anonymous)
    {
        TranslationInfo info = new TranslationInfo(
                    "BadUserAccessDenied",
                    "en-US",
                    "User cannot change value.");

        return new ServiceResult(StatusCodes.BadUserAccessDenied, new LocalizedText(info));
    }

    if (context.UserIdentity.TokenType == UserTokenType.UserName)
    {
        UserNameIdentityToken user = context.UserIdentity.GetIdentityToken() as UserNameIdentityToken;
        if (user.UserName != "administrator")
        {
            TranslationInfo info = new TranslationInfo(
                        "BadUserAccessDenied",
                        "en-US",
                        "User cannot change value.");

            return new ServiceResult(StatusCodes.BadUserAccessDenied, new LocalizedText(info));
        }
    }

    return ServiceResult.Good;
}
```

6 UA Server Design



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

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

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

The WorkshopServerSample only use the Plugin Layer for Startup and uses the TechnitestUaServer and TechnitestUaServerNodeManager classes to be able to use more advanced functionality not available through the UaServerPlugin class.

6.1 Server Startup / Shutdown

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

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

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

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

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

6.1.1 OnStartup

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

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

6.1.2 OnGetLicenseInformation

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

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

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

6.1.3 OnGetServer

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

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

One sample for overriding is the implementation os user authentication.

6.1.4 OnGetServerProperties

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

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

    return properties;
}
```

6.1.5 OnGetNamespaceUris

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

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

6.1.6 OnGetNodeManager

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

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

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

6.1.7 OnInitialized

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

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

6.1.8 OnRunning

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

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

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

    return StatusCodes.Good;
}
```

6.1.9 OnShutdown

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

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

7 Address Space Creation

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

7.1 Creating variables in the address space

Variables are created always in the following sequence:

1) Create the root folder

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

2) Create the sub folder

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

3) Create the variable

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

or

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

4) Add the variable to a list of variables

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

or

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

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

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

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

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

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

Why Technosoftware GmbH?...

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

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

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

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

Technosoftware GmbH
Windleweg 3, CH-5235 Rüfenach

sales@technosoftware.com
www.technosoftware.com