Integration Objects'

Toolkit for OPC UA Client Applications Development in .NET

OPC UA Client Toolkit

Version 2.0 Rev.1

USER GUIDE

OPC Compatibility

OPC Unified Architecture 1.04

OPC UA Client Toolkit User Guide Version 2.0 Rev.1
Published August 2024

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

PREFACE

ABOUT THIS USER GUIDE

This guide describes the functions provided by Integration Objects' OPC UA Client Toolkit and explains how to use this toolkit.

TARGET AUDIENCE

This user manual is intended for .NET developers of OPC UA client applications. It assumes that you have a working knowledge of OPC UA and programming with the .NET languages.

RELATED DOCUMENTATION

OPC Foundation (www.opcfoundation.org)

- OPC UA Specification

DOCUMENT CONVENTIONS

Convention	Description
Monospaced type	Indicates a file reference
	Information to be noted

CUSTOMER SUPPORT SERVICES

Phone	Email
Americas: +1 713 609 9208 **Europe-Africa-Middle East** +216 71 195 360	Support: customerservice@integrationobjects.com Sales: sales@integrationobjects.com Online: www.integrationobjects.com

INTRODUCTION

1. Overview

Integration Objects' OPC UA Client Toolkit is an API that handles all OPC UA details necessary to communicate with OPC UA servers. It is a tool for fast and easy programming of OPC UA client applications using the .NET framework.

Using this toolkit, developers will be able to build their own OPC UA client applications easily using C# and VB .NET and without having to be concerned with the details of the OPC UA standard. The generated .NET custom applications will be able to access real-time, historical and alarms and events data from any OPC UA server.

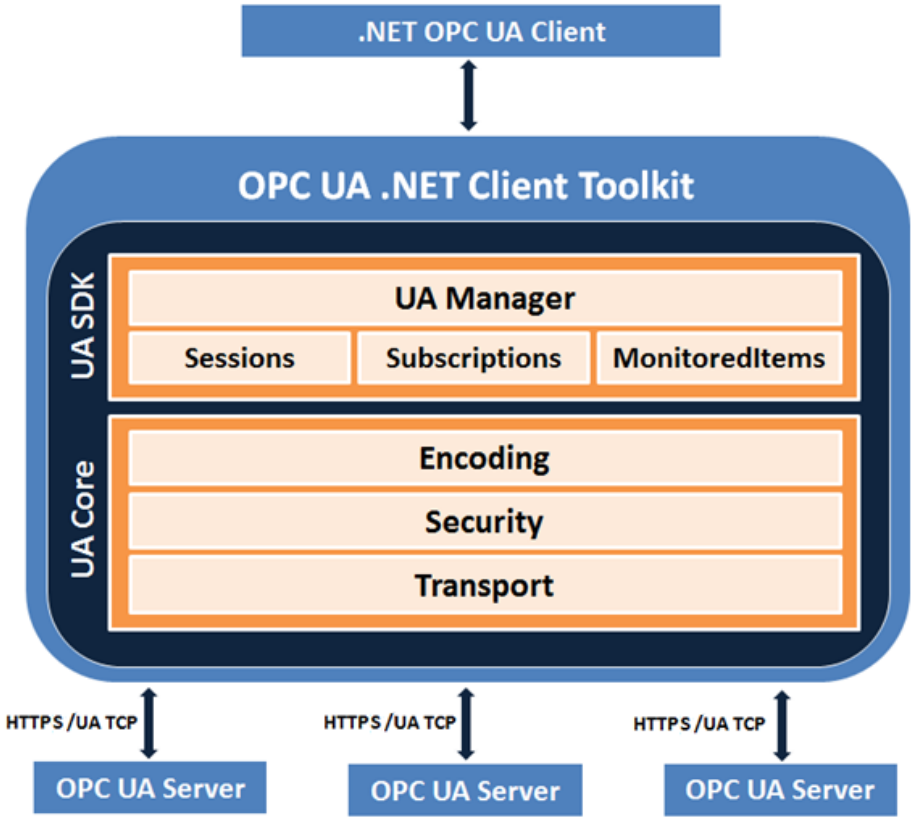

Figure 1: Overview of the OPC UA Client Toolkit

2. Features

The main features of OPC UA Client Toolkit are:

- Support of OPC UA specifications. This toolkit is fully compliant with OPC UA 1.04.
- Discovery of OPC UA servers available on the network.
- Managing local and remote connections to multiple OPC UA Servers.
- Offering the following OPC UA client capabilities:
  - Creating a secure session with OPC UA Server
  - Browsing OPC UA Server address space
  - Managing UA subscriptions
  - Monitoring real-time data and alarms & conditions,
  - Exploring history data.
  - Support of UA TCP and HTTPS transport protocols
  - Support of None, Sign and Sign & Encrypt security modes
  - Support of XML and Binary message encoding
  - Support of None, Basic128RSA15, Basic 256, Basic256Sha256, Aes128_Sha256_RsaOaep and Aes256_Sha256_RsaPss security policies
  - Support of Anonymous and User Name user authentication modes
  - Certificates Management
- Support of 32 and 64 bit applications
- Provides royalty free runtime distribution
- Support of .Net Core version 3.1 or higher
- Support of .Net Framework version 4.6.1 or higher
- Support of Visual Studio 2017 and higher

3. Operating Systems Compatibility

This Toolkit supports the following operating systems:

- Windows 11
- Windows 10
- Windows 8
- Windows 7

- Windows Server 2022
- Windows Server 2019
- Windows Server 2016
- Windows Server 2012
- Windows Server 2008

4. OPC Compatibility

- OPC Unified Architecture 1.04

GETTING STARTED

1. Pre-Installation Considerations

In order to properly run OPC UA Clients developed using the OPC UA Client Toolkit, you need to install the following software components on the target system:

- .NET Framework version 4.6.1 or higher
- .Net Core 3.1 Runtime or higher

2. Installation

To install the OPC UA Client toolkit, run the downloaded installation program using an administrator account and the installation wizard will take you through the different installation steps.

If you are evaluating the OPC UA Client Toolkit, make sure to select the demo version option in the select features dialog. Otherwise, select the full version. The evaluation license allows you to use the toolkit for 30 days and limits the runtime to 2 hours.

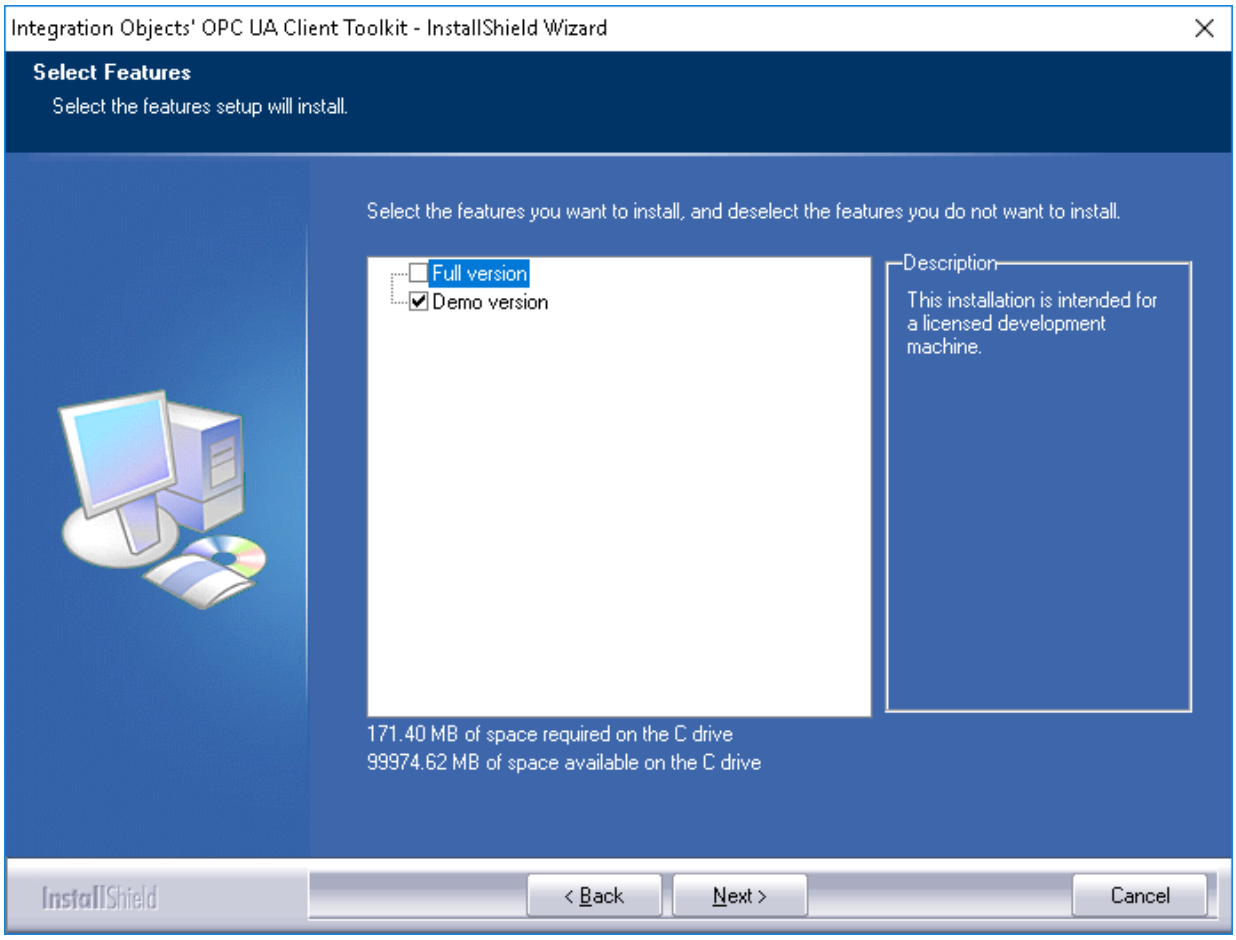

Figure 2: Select Features Dialog

Click the **Next** button and the dialog box for choosing to install the UA Local Discovery Server will be displayed as illustrated below.

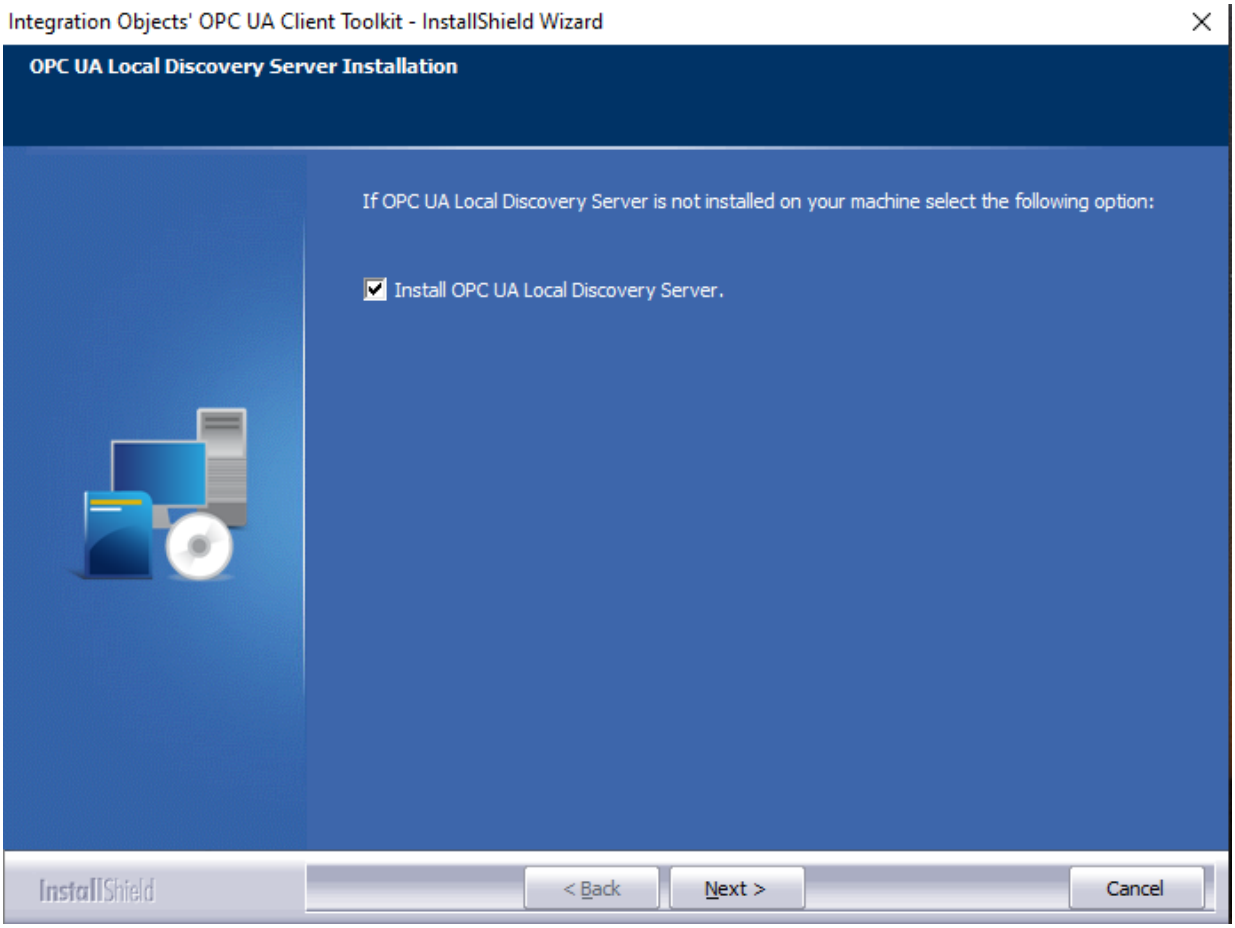

Figure 3: Install OPC UA Local Discovery Server

Once the installation is complete, you will have the following shortcuts in your start menu:

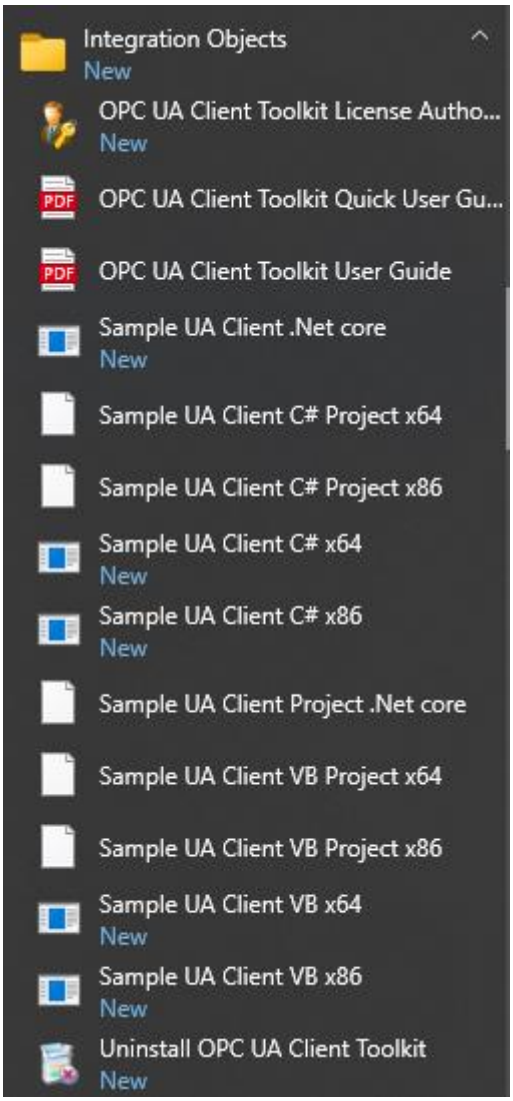

Figure 4: OPC UA Client Toolkit Start Menu

You will also get the following files in your system under the installation folder:

Files	Description
DLL Files	`IntegrationObjects.OpcUaNetClientToolkit.dll`: Release Any CPU DLL. It uses the following DLLs: • `License.dll` • `IntegrationObjects.Logger.SDK.dll` • `IntegrationObjects.Opc.Ua.Core.dll`
OPC Sample Demos	Contains the OPC UA Client samples for both x86 and x64 architectures: `OPCUAClient.exe` and `UAClientVBSample.exe` It contains also `UAClientNetCoreSample.exe` as a .Net core application example.
OPC Sample Projects	Contains the Visual Studio 2019 projects of the OPC UA Client samples.
Other files	• OPC UA Client Toolkit User Guide (this guide) • OPC UA Client Toolkit Quick User Guide • License authorization tool: Indicates the license status of each installed feature. • OPC UA Client Toolkit Uninstaller: used to uninstall the OPC UA Client Toolkit
Components	Contains the OPC UA Local Discovery Server installation program

Table 1: Installed Files Description

When changing your OPC UA Client Toolkit installation from a demo license to a full development license, make sure to reference the new dll files from the full version installation on your application project project or to copy them in your output folders.

3. Compiling and Linking Applications

This section details the steps to follow in order to compile and correctly link applications to develop a custom OPC UA client application using Integration Objects' OPC UA Client Toolkit and Microsoft Visual Studio 2019.

- ## WINDOWS FORM APPLICATIONS USING .NET FRAMEWORK:

To build a .Net framework OPC UA Client application, follow the steps below:

Step 1: Create your Project

Start Visual Studio 2019 and choose **New Project**. The following window will be displayed.

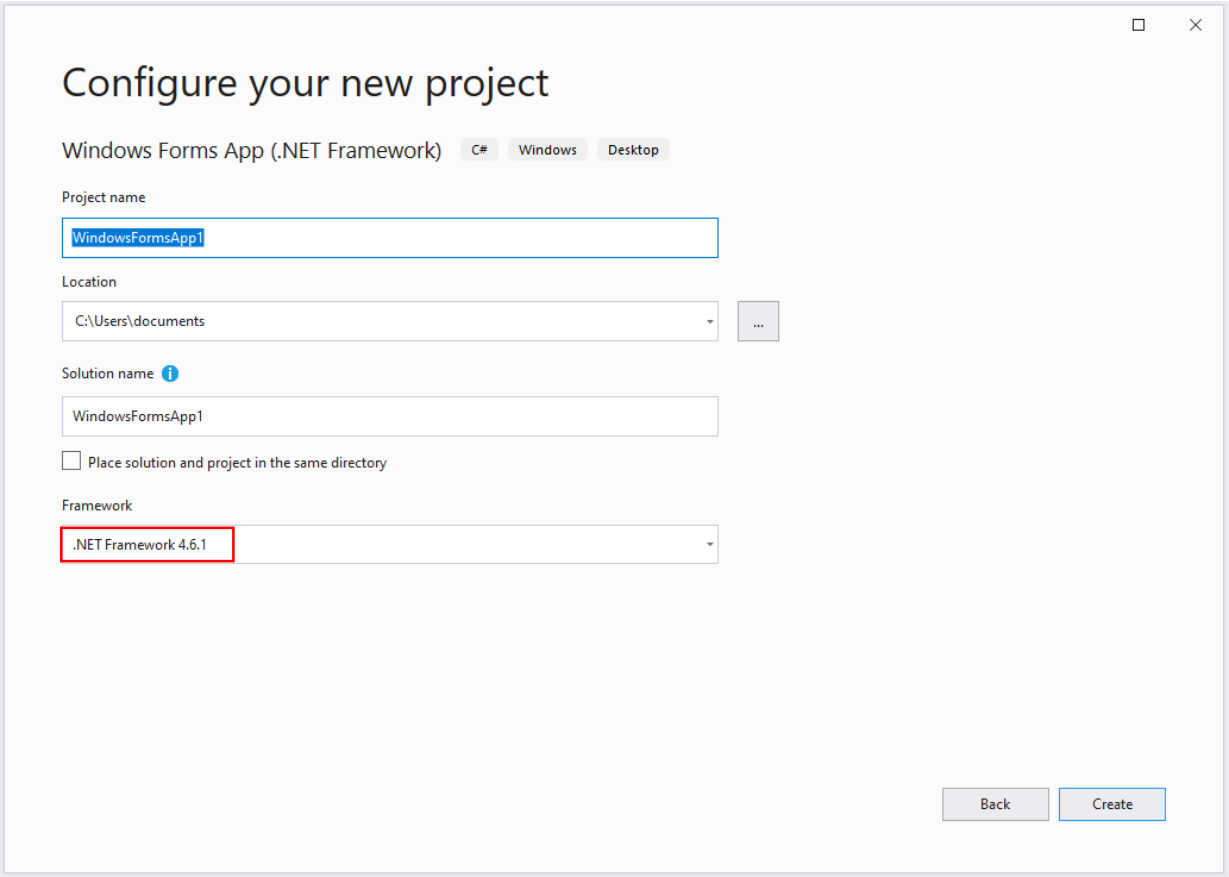

Figure 5: New Windows Form Project

Choose Visual C# **Windows Forms Application** Project and then click **OK**. A project named WindowFormsApplication with a form called Form1 will be automatically created.

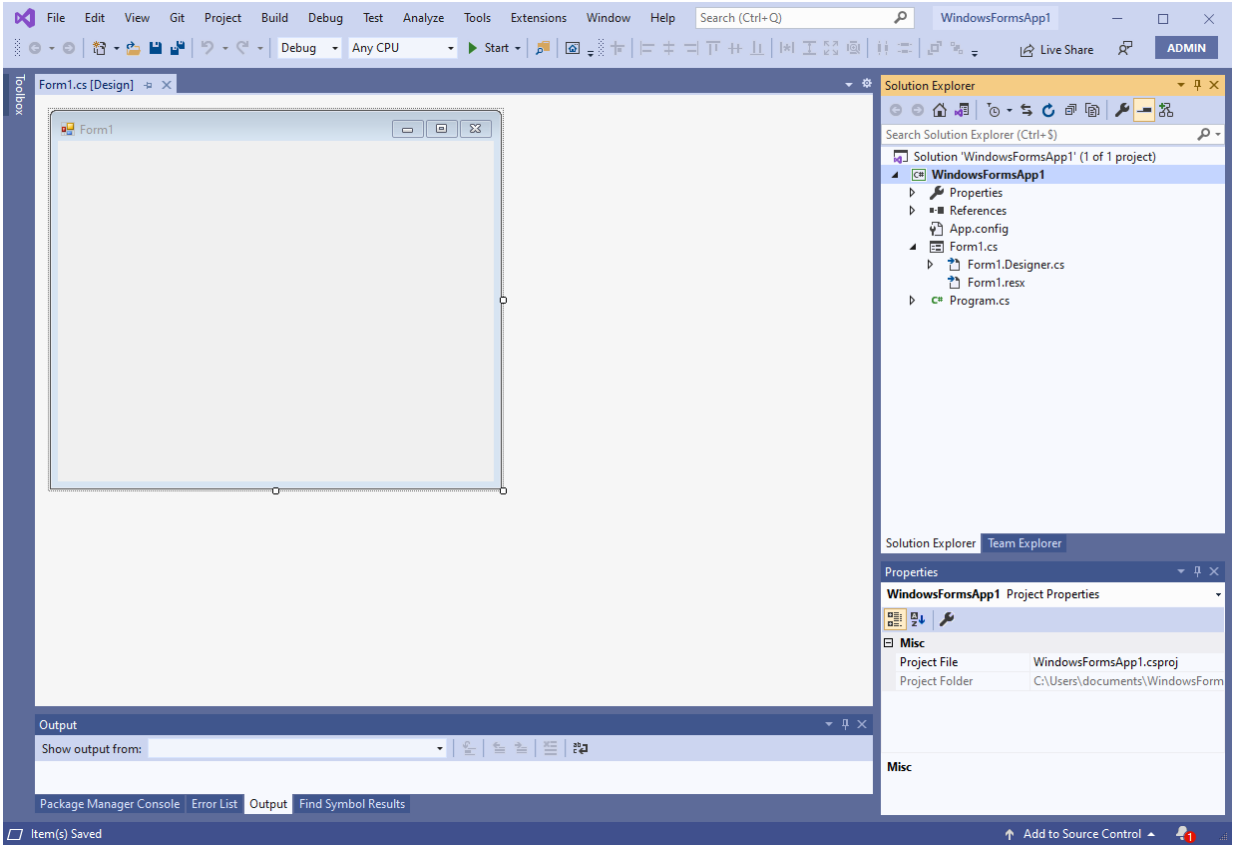

Figure 6: Windows Forms Project Template

Step 2: Add your References

1. Right click on **References** under the solution explorer then click **Add Reference…** from the displayed menu.

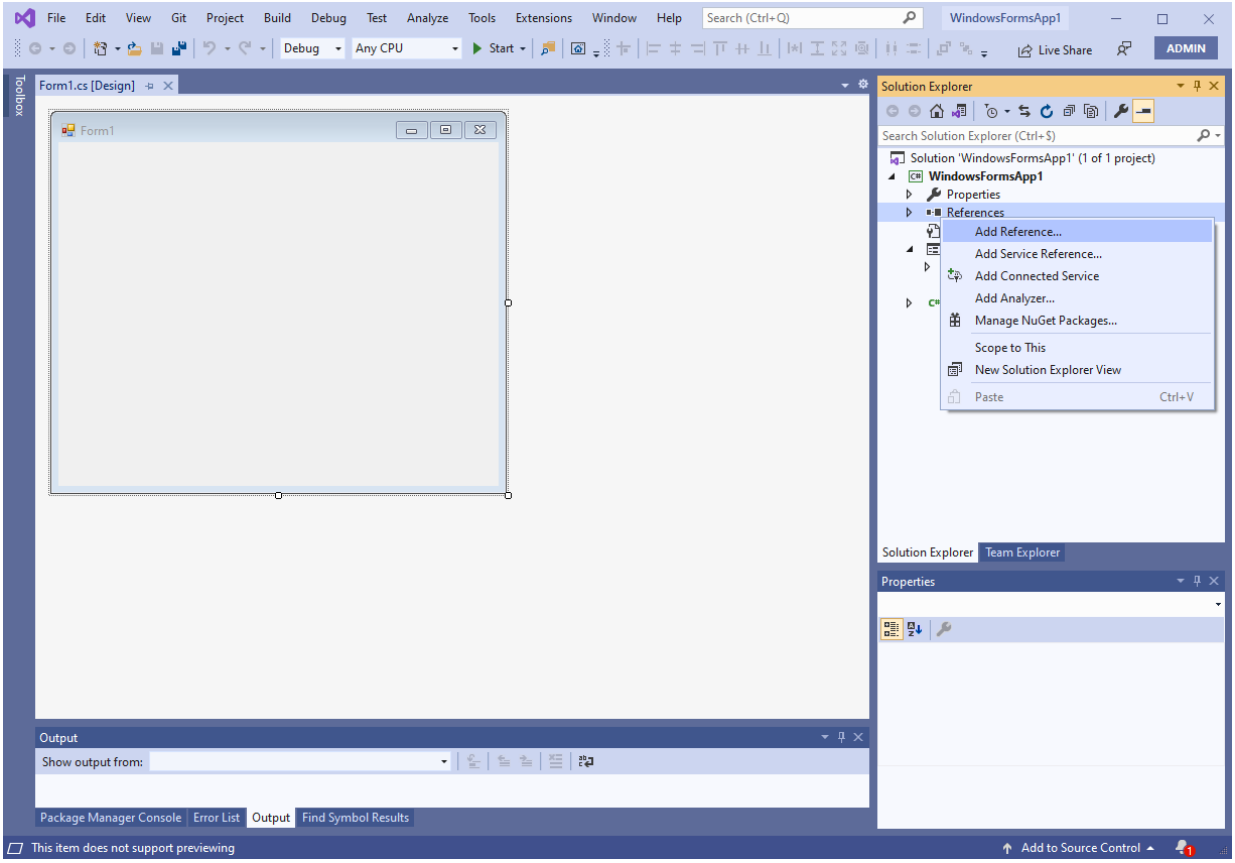

Figure 7: Solution Explorer

2. Select **Browse** tab from the displayed Add Reference window.

3. Select the following files located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin**":
    - IntegrationObjects.OpcUaNetClientToolkit.dll
    - IntegrationObjects.Opc.Ua.Core.dll
    - IntegrationObjects.Logger.SDK.dll
    - BouncyCastle.Crypto.dll
    - System.ServiceModel.Primitives.dll

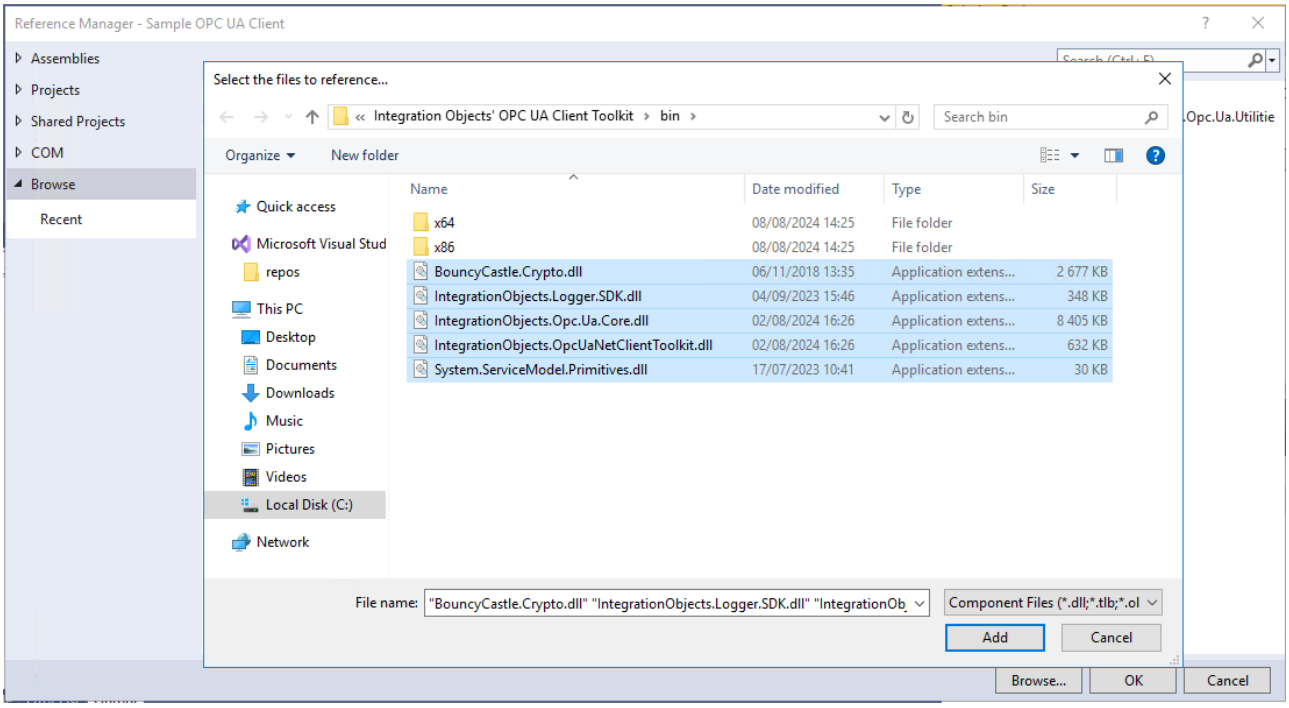

Figure 8: Choosing a reference

4. Copy the UA XML configuration file "*XXXX*.Config.xml" file located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin**" and paste it in the output project. *XXXX* is the name of your client application.

5. Copy the "license.dll" file located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin\x64**" and paste it in the output project. Make sure to choose the "license.dll" file located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin\x86**" if you are using the 32-bit version.

Step 4: Select your Platform

For users who have to build the application in a **32-bit** machine, the target platform has to be set to **x86** as illustrated in the screenshot below.

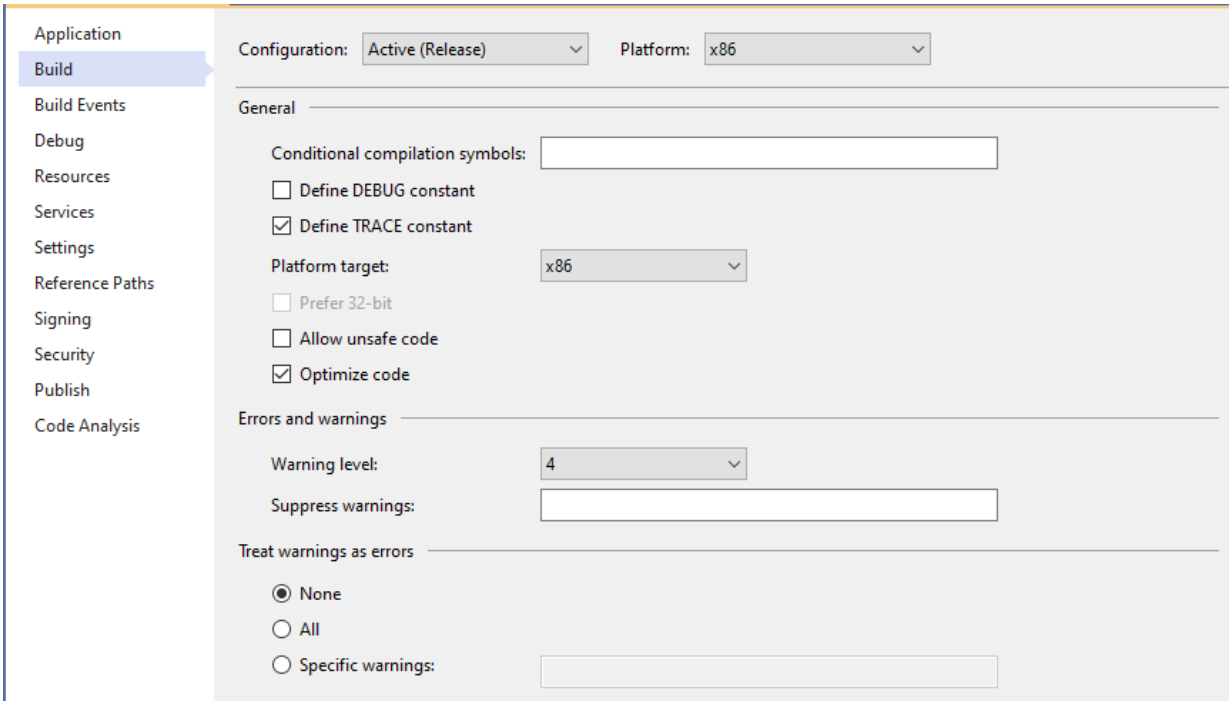

Figure 9: Platform for 32-bit Machine

For users who have to build the application in a **64-bit** machine, the target platform has to be set to **x64** as illustrated in the screenshot below.

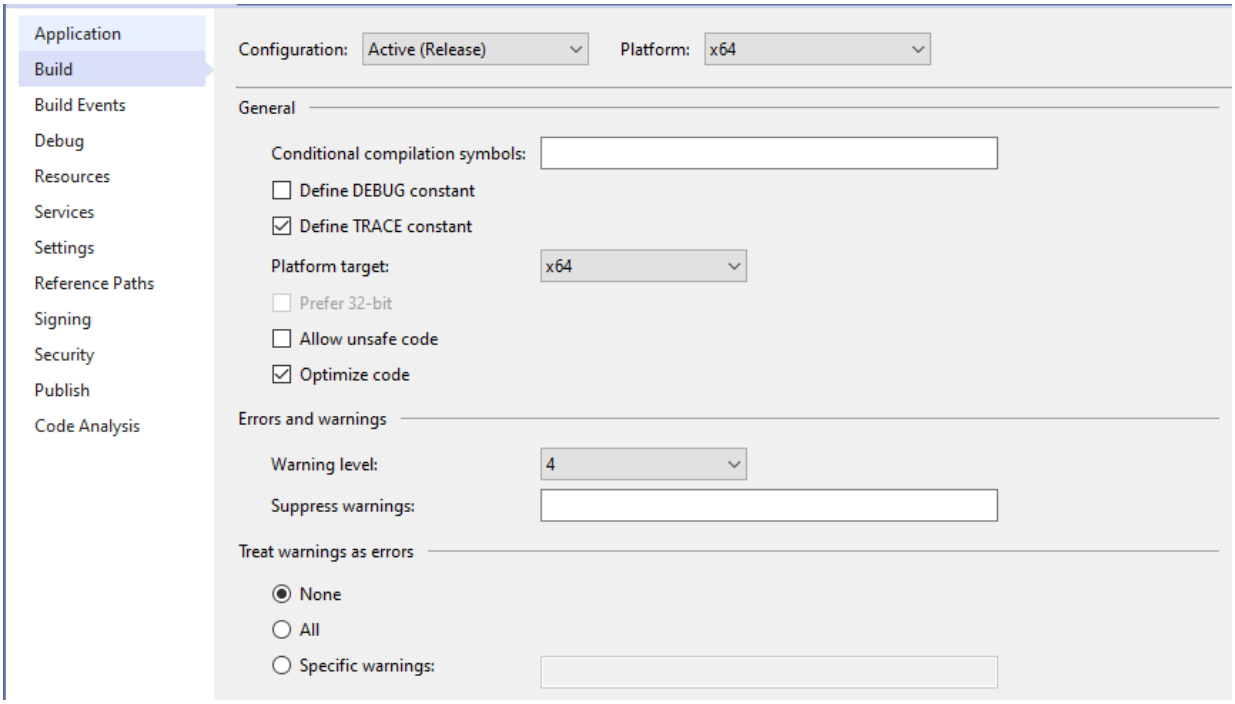

Figure 10: Platform for 64-bit Machine

- ## A CONSOLE APPLICATION USING .NET CORE:

To build a .Net Core OPC UA Client application, follow the steps below:

Step 1: Create your Project

Start Visual Studio 2019 and choose **New Project**. The following window will be displayed.

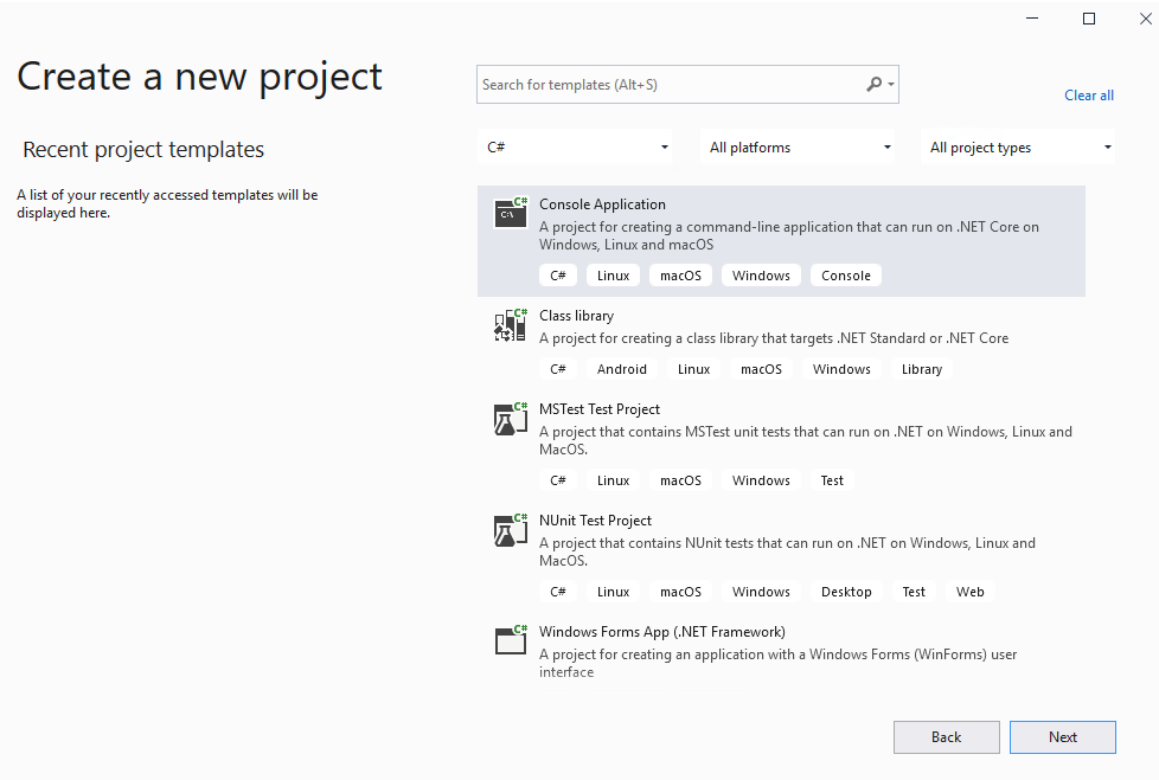

Figure 11: New Console Application Project

1. Choose Visual C# **Console Application** Project and then click **Next**.
2. From Additional information, select the .Net Core version you need to build with your application project and click Create.

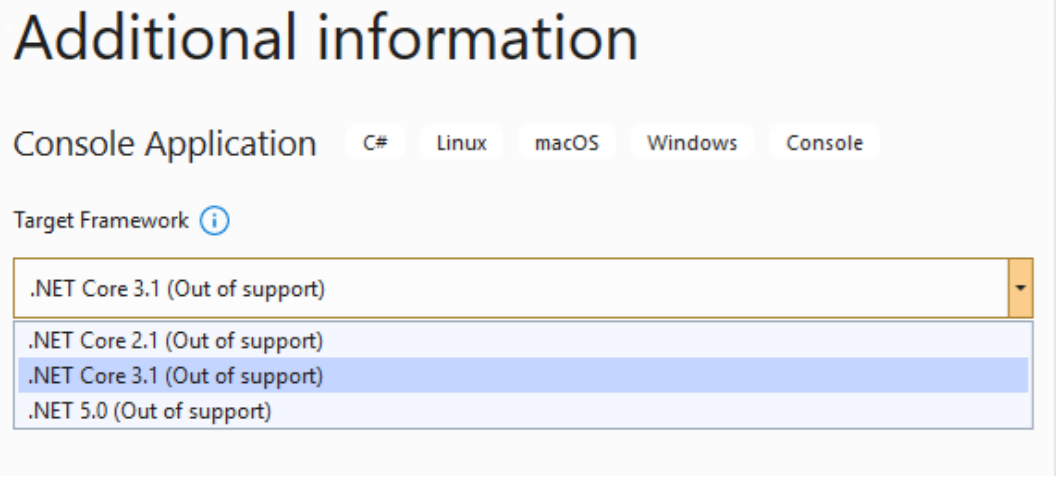

Figure 12: New Console Application Project

A project named ConsoleApp1 will be automatically created.

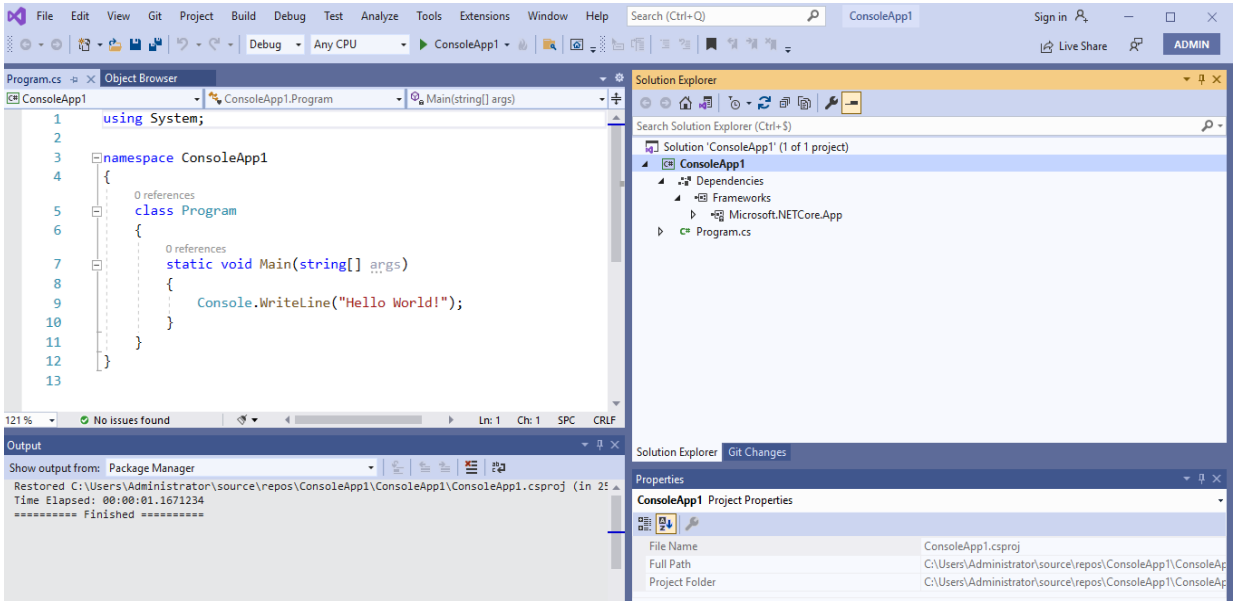

Figure 13: Console Application Project Template

Step 2: Add your References

1. Right click on **Dependencies** then click Add Project Reference… from the displayed menu.

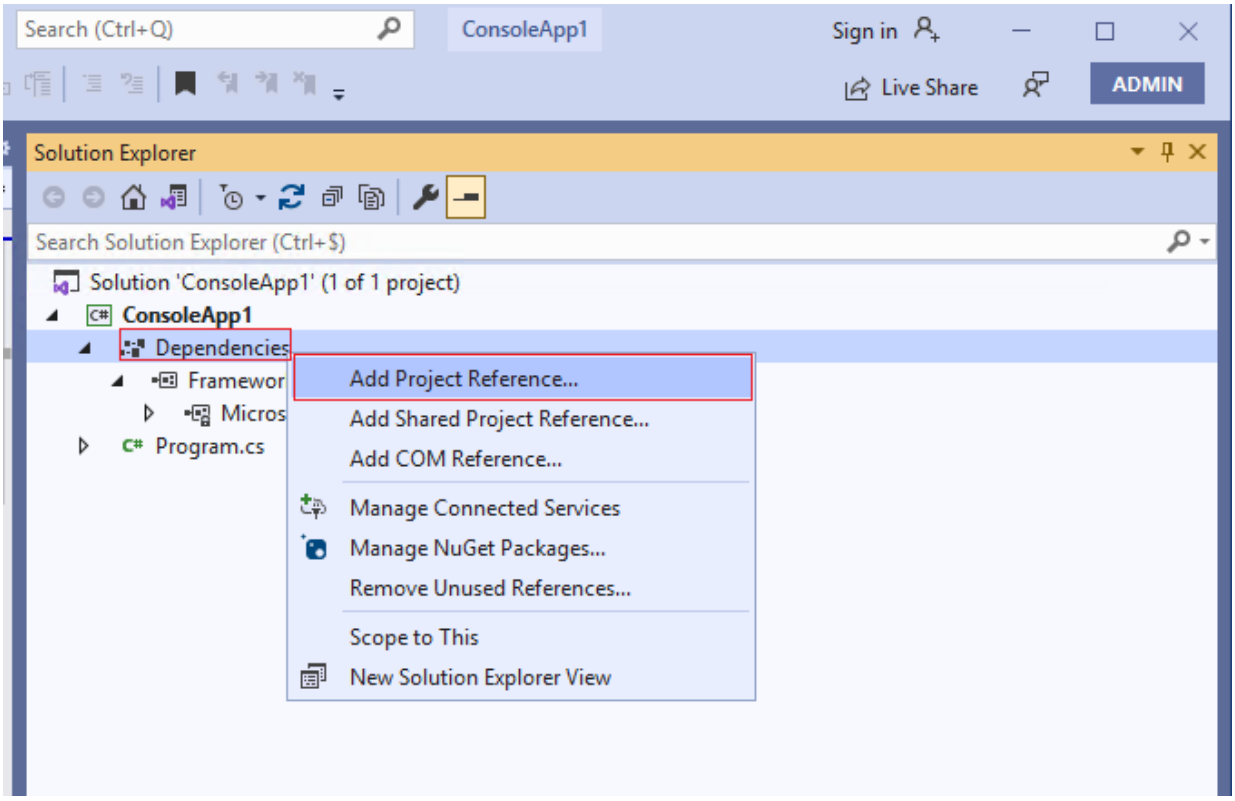

Figure 14: Solution Explorer

2. Select Browse tab from the displayed Add Reference window.
3. Select the following files located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin**":

- IntegrationObjects.OpcUaNetClientToolkit.dll
- IntegrationObjects.Opc.Ua.Core.dll
- IntegrationObjects.Logger.SDK.dll
- BouncyCastle.Crypto.dll
- System.ServiceModel.Primitives.dll

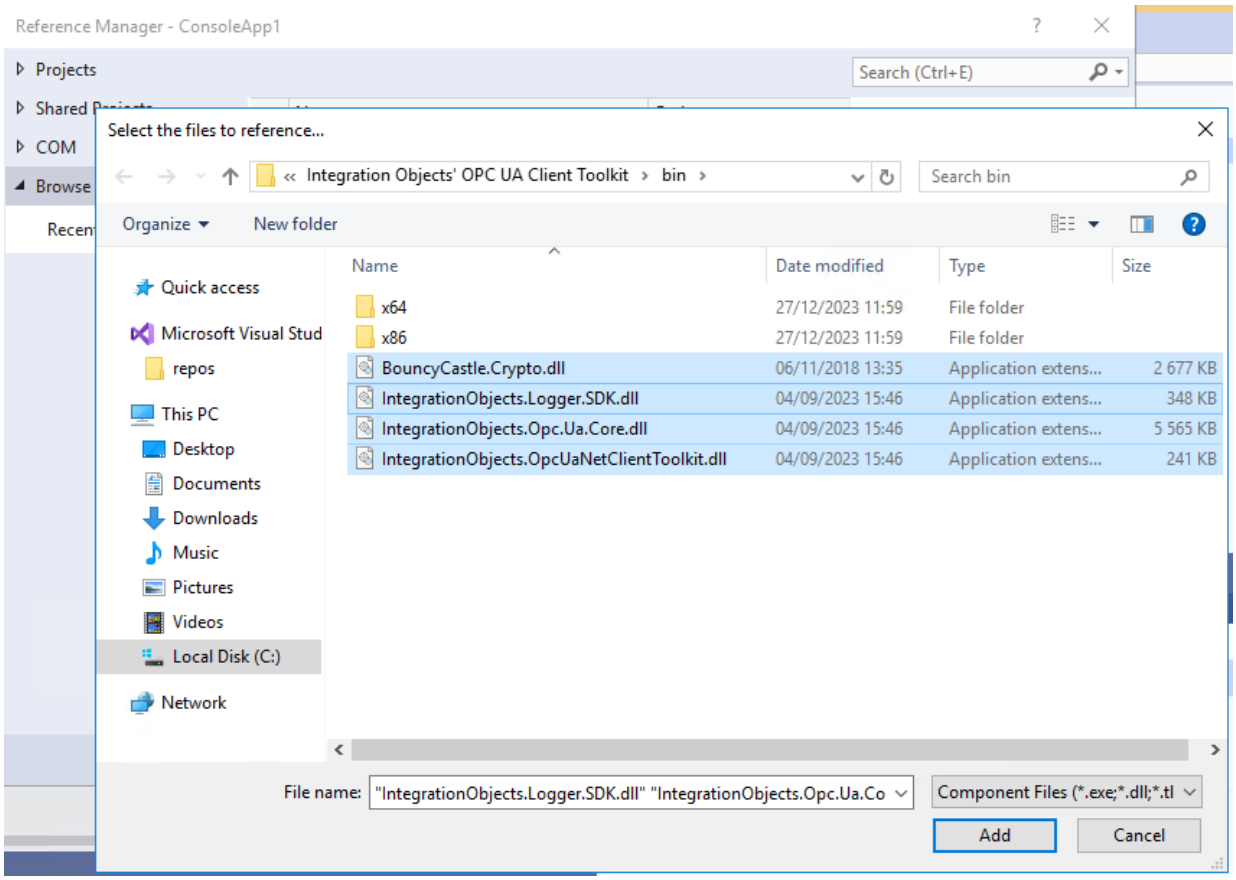

Figure 15: Choosing a Reference

3. Copy the UA XML configuration file "XXXX.Config.xml" file located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin**" and paste it in the output project.

4. Copy the "license.dll" file located under ".**:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin\x64**" and paste it in the output project.

Make sure to choose the "license.dll" file located under "**.:\Program Files (x86)\Integration Objects\Integration Objects' OPC UA Client Toolkit\bin\x86**" if you are using the 32-bit version.

Step 4: Select your Platform

The target platform has to be set to **x86** in case you are in a **32-bit** development machine or **x64** bit in case you are using in a **64-bit** machine as illustrated in the screenshot below.

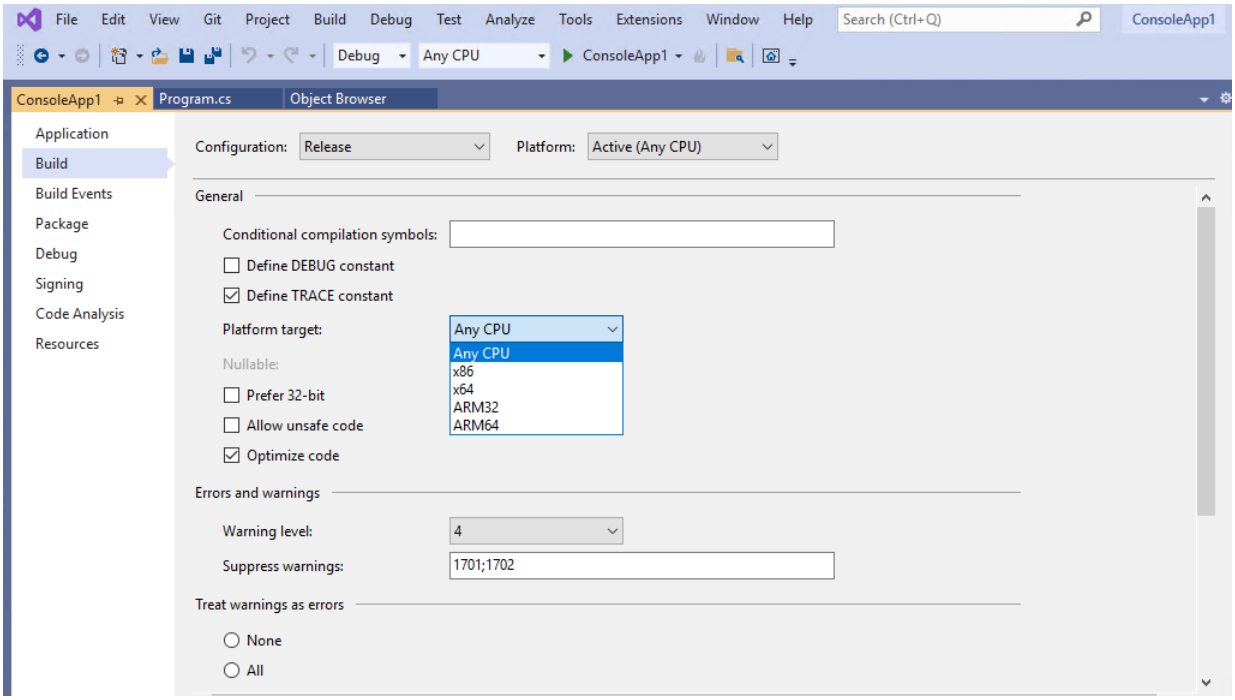

Figure 16: Target Platform

4. Runtime Deployment Steps

In order to deploy the developed client application from the development machine to the runtime machine, follow the steps below:

1. Create a new folder
2. Copy the following files:
   - Config.json
   - IntegrationObjects.Logger.SDK.dll
   - IntegrationObjects.OpcUaNetClientToolkit.dll
   - License.dll
   - IntegrationObjects.Opc.Ua.Core.dll
   - Your application executable and any other custom assembly dependencies

- The UA XML configuration file (*XXXX*.Config.xml, where *XXXX* is the name of your OPC UA client application)
- BouncyCastle.Crypto.dll
- System.ServiceModel.Primitives.dll
- ConnectionConfig.json (for the OPC UA .Net Core Client Toolkit)

3. Copy the folder to the runtime machine

Make sure that the OPC UA Client Toolkit is not installed in the runtime machine and that the path of the application folder does not include the key words "Debug" or "Release".

USING THE OPC UA CLIENT TOOLKIT

1. Initialization

The client application is responsible for properly initializing the OPC UA Client Toolkit using the **UAManager** class as follows:

```
UAManager objUAManager = new UAManager();
```

1.1. Set the UA XML configuration file

To set the UA XML configuration file path, the **UAManager** instance should be initialized instead as follows:

```
string strConfigFilePath = ".\OPCUANetClient.Config.xml";
UAManager objUAManager = new UAManager(strConfigFilePath);
```

Parameters

In/Out	Parameter	Description
In	`strConfigFilePath`	The UA XML configuration file path.

Table 2: Parameters of UAManager

2. OPC UA Servers Discovery

2.1. Discover Network Hosts

OPC UA Client Toolkit provides a method to discover all hosts on the network. The following table describes the parameters of the **BrowseLocalNetwork** function.

```
uint BrowseLocalNetwork(out List<string> lstHosts)
```

Parameters

In/Out	Parameter	Description
Out	**lstHosts**	Contains the list of the network hosts.

Table 3: Parameters of BrowseLocalNetwork

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 4: Returned Codes of BrowseLocalNetwork

2.2. Discover Endpoints

OPC UA Client Toolkit provides a way to discover OPC UA servers located in a machine. The following table describes the parameters of the **GetEndpoints** function.

```
uint GetEndpoints(string strHostName, out List<string> lstDiscoveredUrls)
```

Parameters

In/Out	Parameter	Description
In	**strHostName**	Name of the machine that hosts the endpoints.
Out	**lstDiscoveredUrls**	Contains the list of located UA endpoints.

Table 5: Parameters of GetEndpoints

Returned Codes

Return Code	Description
Good	The operation was successful.

Bad	The operation failed but no specific reason is known.

Table 6: Returned Codes of GetEndpoints

2.3. Get Endpoint Scheme

OPC UA Client Toolkit provides a method to get the list of endpoints descriptions from an endpoint URL. The following table describes the parameters of the **GetEndpointScheme** function.

```
uint GetEndpointScheme(string strdiscoveryUrl, out EndpointDescriptionCollection
lstEndpointsScheme)
```

Parameters

In/Out	Parameter	Description
In	**strdiscoveryUrl**	The endpoint URL.
Out	**lstEndpointsScheme**	Contains the list of the endpoints scheme.

Table 7: Parameters of GetEndpointScheme

EndpointDescription Attributes:

Setting	Description
EndpointUrl	The network endpoint to use when connecting to the server.
Server	The description of the server.
ServerCertificate	The server's application certificate.
SecurityMode	The security mode that must be used when connecting to the endpoint.
SecurityPolicyUri	The security policy to use when connecting to the endpoint.
UserIdentityTokens	The user identity tokens that can be used with this endpoint.

TransportProfileUri	The transport profile to use when connecting to the endpoint.
SecurityLevel	.A server assigned value that indicates how secure the endpoint is relative to other server endpoints.

Table 8: Endpoint Description Parameters

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 9: Returned Codes of GetEndpoints

3. Server Management

3.1. Connect to an OPC UA Server

This function establishes a session to a specified OPC UA server. The following table describes the parameters of the **CreateSession** function.

```
uint CreateSession(UAServer objUAServer, string strSessionName, out X509Certificate2
objServerCertificateToTrust, out Session objSession)
```

Parameters

In/Out	Parameter	Description
In	**objUAServer***	The server parameters.
In	**strSessionName**	The session name.
Out	**objServerCertificateToTrust**	Contains the server certificate if the server certificate is not trusted and null if it is already trusted.

Out	**objSession**	If the call succeeds, this parameter will contain the created session. If the call fails, the parameter will contain a null object.

Table 10: Parameters of CreateSession

UAServer attributes:

Setting	Description
ServerName	The server's name.
Protocol	The server's binary protocol, which can be **OPC UA TCP** or **HTTPS**.
SecurityMode	The security mode which can be **None**, **Sign** or **SignAndEncrypt**.
SecurityPolicy	Specifies which security mechanisms are to be used, it includes the following information: • algorithms for signing and encryption • algorithm for key derivation
UserIdentity	The UserIdentity mappings can be based on user names, user certificates or user groups.
UserIdentityString	The string that represents a UserIdentity, it can be **"Anonymous"**, **"UserName"** or **"Certificate"**.
CertificationPath	Defines the location of the directory store where the certificate will be placed.
CertificationPassword	Defines the password to be associated with the new generated certificate.
CertificationStore	Defines a place where Certificates and Private Keys can be stored on a file system.
UserName	The server's user name.
UserPassword	The server's user password.

IsSecurityStoreEnabled	Indicates whether the security store path is enabled.

Table 11: UAServer Parameters

Session attributes:

Setting	Description
SessionName	The session name.
SessionTimeout	The period for which the server will maintain the session if there is no communication from the client.
KeepAliveInterval	Specifies how frequently the server is pinged to see if communication is still working.
ConfiguredEndpoint	The endpoint used to connect to the server.
NamespaceUris	The table of namespace uris known to the server.
Subscriptions	The session subscription list.

Table 12: Session Parameters

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_SecureChannelIdInvalid	The specified secure channel is no longer valid.
Bad_SecurityChecksFailed	An error occurred while verifying security.
Bad_CertificateTimeInvalid	The Certificate has expired or is not yet valid.

Bad_CertificateIssuerTimeInvalid	An Issuer Certificate has expired or is not yet valid.
Bad_CertificateHostNameInvalid	The HostName used to connect to a Server does not match a HostName in the Certificate.
Bad_CertificateUriInvalid	The URI specified in the ApplicationDescription does not match the URI in the Certificate.
Bad_CertificateUseNotAllowed	The Certificate may not be used for the requested operation.
Bad_CertificateIssuerUseNotAllowed	The Issuer Certificate may not be used for the requested operation.
Bad_CertificateUntrusted	The Certificate is not trusted.
Bad_CertificateRevocationUnknown	It was not possible to determine if the Certificate has been revoked.
Bad_CertificateIssuerRevocationUnknown	It was not possible to determine if the Issuer Certificate has been revoked.
Bad_CertificateRevoked	The Certificate has been revoked.
Bad_CertificateIssuerRevoked	The Issuer Certificate has been revoked.
Bad_TooManySessions	The server has reached its maximum number of sessions.
Bad_ServerUriInvalid	The Server URI is not valid.
Bad_IdentityTokenInvalid	The user identity token is not valid.
Bad_IdentityTokenRejected	The user identity token is valid but the server has rejected it.
Bad_UserAccessDenied	User does not have permission to perform the requested operation.
Bad_ApplicationSignatureInvalid	The signature provided by the client application is missing or invalid.
Bad_UserSignatureInvalid	The user token signature is missing or invalid.

Bad_NoValidCertificates	The Client did not provide at least one Software Certificate that is valid and meets the profile requirements for the Server.
Bad_IdentityChangeNotSupported	The Server does not support changing the user identity assigned to the session.

Table 13: Returned Codes of CreateSession

3.2. Disconnect from OPC UA Server

To disconnect from the server, the client may use the method called **CloseSession** and provide the name of the session**: strSessionName**. The following table describes the parameters of this function.

```
uint CloseSession(string strSessionName)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The name of the session to close.

Table 14: Parameters of Disconnect

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_SessionIdInvalid	The session id is not valid.

Table 15: Returned Codes of Disconnect

3.3. Browse OPC UA Server Address Space

3.3.1. Set Root Browser

This function initializes a browser for the session and sets its root node. The following table describes the parameters of this function.

```
uint SetRoot(Session objSession, BrowseViewType objBrowseViewType, NodeId objViewId, out
ReferenceDescription objReferenceDescription)
```

Parameters

In/Out	Parameter	Description
In	**objSession**	The session to initialize the browser for.
In	**objBrowseViewType**	The type views that can be used when browsing the address space.
In	**objViewId**	Contains a NodeId if the BrowseViewType is a ServerDefinedView, otherwise, it contains a null object.
Out	**objReferenceDescription**	Contains the root node of the Browser.

Table 16: Parameters of SetRoot

BrowseViewType:

Type	Description
All	All nodes and references in the address space.
Objects	The object instance hierarchy.
Types	The type hierarchies.
ObjectTypes	The object type hierarchies.
EventTypes	The event type hierarchies.

DataTypes	The data type hierarchies.
ReferenceTypes	The reference type hierarchies.
ServerDefinedView	A server defined view.

Table 17: Type of BrowseViewType

NodeId:

Attributes	Description
IdType	The type of node identifier used.
NamespaceIndex	The index of the namespace URI in the server's namespace array.
Identifier	The node identifier.
IsNullNodeId	Specifies Whether the object represents a Null NodeId.

Table 18: NodeId Attributes

ReferenceDescription attributes:

Setting	Description
BinaryEncodingId	The UA type identifier for binary encoding.
TypeId	The UA type identifier.
TypeDefinition	The type definition of the target node.
NodeClass	The node class of the target node.
DisplayName	The display name of the target node.
BrowseName	The browse name of the target node.
NodeId	The id of the target node.
IsForward	TRUE if the reference is a forward reference.
ReferenceTypeId	The type of references.

XmlEncodingId	The UA type identifier for the XML encoding id.
Unfiltered	True if the reference filter has not been applied.

Table 19: ReferenceDescription Parametres

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 20: Returned Codes of SetRoot

3.3.2. Browse Children

This function browses the children of a specified node in the address space. The following table describes the parameters of this function.

```
uint BrowseChildren(ReferenceDescription objReferenceDescription, string strSessionName,
out ReferenceDescriptionCollection objReferenceDescriptionCollection)
```

Parameters

In/Out	Parameter	Description
In	**objReferenceDescription**	The parent node.
In	**strSessionName**	The session name.
Out	**objReferenceDescriptionCollection**	Collection of the children nodes.

Table 21: Parameters of BrowseChildren

*ReferenceDescriptionCollection: the list of ReferenceDescription.

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_ViewIdUnknown	The view id does not refer to a valid view Node.
Bad_ViewTimestampInvalid	The view timestamp is not available or not supported.
Bad_ViewParameterMismatchInvalid	The view parameters are not consistent with each other.
Bad_ViewVersionInvalid	The view version is not available or not supported.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.

Table 22: Returned Codes of BrowseChildren

4. Subscription Management

4.1. Create Subscription

This method is used to create a subscription. Subscriptions monitor a set of items for notifications and return them to the client in response to Publish requests.

```
uint CreateSubscription(string strSessionName, ref Subscription objSubscription)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	Name of session.

In/Out	**objSubscription**	Contains the parameter of the subscription.

Table 23: Parameters of CreateSubcription

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_TooManySubscriptions	The Server has reached its maximum number of subscriptions.

Table 24: Returned Codes of CreateSubcription

Subscription attributes:

Setting	Description
Publishing Interval	This interval defines the cyclic rate that the subscription is being requested to return notifications to the client. This interval is expressed in milliseconds
Keep Alive Count	This setting defines the number of consecutive publishing cycles in which there have been no notifications to report to the client. When the maximum keep-alive count is reached, a Publish request is de-queued and used to return a keep alive message. This keep-alive message informs the client that the subscription is still active.
Lifetime Count	When the publishing timer has expired this number of times without a publish request being available to send a notification

	message, then the subscription shall be deleted by the server.
Max Notifications per Publish	The maximum number of notifications that the client wishes to receive in a single Publish response. A value of zero indicates that there is no limit.
Priority	This setting indicates the relative priority of the subscription. When more than one Subscription needs to send notifications, the server should dequeue a publish request to the subscription with the highest priority number. For subscriptions with equal priority the server should de-queue Publish requests in a round-robin fashion.
Publishing Enabled	A Boolean parameter with the following values: -TRUE: publishing is enabled for the subscription. -FALSE: publishing is disabled for the subscription.

Table 25: Subscription Parameters

4.2. Remove Subscription

This function removes a subscription from the OPC UA session. The following table describes the parameters of this function.

```
uint RemoveSubscription(Subscription objSubscription)
```

Parameters

In/Out	Parameter	Description

In	**objSubscription**	The subscription to be removed.

Table 26: Parameters of RemoveSubscription

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Bad_SubscriptionIdInvalid	The subscription id is not valid.

Table 27: Returned Codes of RemoveSubscription

4.3. Set Publishing Mode

This function updates the publishing mode of a subscription. The following table describes the parameters of this function.

```
uint SetPublishingMode(Subscription objSubscription, bool bEnabled)
```

Parameters

In/Out	Parameter	Description
In	**objSubscription**	The subscription to be updated
In	**bEnabled**	Set to true if the publishing mode will be enabled and to false if the publishing mode will be disabled

Table 28 : Parameters of SetPublishingMode

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.

Table 29: Returned Codes of SetPublishingMode

4.4. Create Monitored Item

This function creates a monitored item and assigns it to a subscription. The following table describes the parameters of this function.

```
uint CreateMonitoredItem(ReferenceDescription objReferenceDescription, ref Subscription
objSubscription, bool bUseDataChangeFilter = false, DataChangeFilter _filter = null);
```

Parameters

In/Out	Parameter	Description
In	**objReferenceDescription**	The item to be subscribed.
In/Out	**objSubscription**	Contains the parameter of the subscription.
In	bUseDataChangeFilter	Set to true if the DataChangeFilter will be enabled and to false if the DataChangeFilterwill be disabled
In	filter	Contains the parameter of the data change filer

Table 30: Parameters of CreateMonitoredItem

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Bad_TimestampsToReturnInvalid	The timestamps to return parameter is invalid.
Bad_SubscriptionIdInvalid	The subscription id is not valid.
Bad_MonitoringModeInvalid	The monitoring mode is invalid.
Bad_NodeIdInvalid	The syntax of the node id is not valid.
Bad_NodeIdUnknown	The node id refers to a node that does not exist in the server address space.
Bad_AttributeIdInvalid	The attribute is not supported for the specified node.
Bad_IndexRangeInvalid	The syntax of the index range parameter is invalid.
Bad_IndexRangeNoData	No data exists within the range of indexes specified.
Bad_DataEncodingInvalid	The data encoding is invalid. This result is used if no dataEncoding can be applied because an Attribute other than Value was requested or the DataType of the Value Attribute is not a subtype of the Structure DataType.
Bad_DataEncodingUnsupported	The server does not support the requested data encoding for the node.

	This result is used if a dataEncoding can be applied but the passed data encoding is not known to the Server.
Bad_MonitoredItemFilterInvalid	The monitored item filter parameter is not valid.
Bad_MonitoredItemFilterUnsupported	The server does not support the requested monitored item filter.
Bad_FilterNotAllowed	A monitoring filter cannot be used in combination with the attribute specified.
Bad_TooManyMonitoredItems	The Server has reached its maximum number of monitored items.

Table 31: Returned Codes of CreateMonitoredItem

4.5. Create Monitored Items

This function creates a list of monitored items and assigns them to a subscription. The following table describes the parameters of this function.

```
List<uint> CreateMonitoredItems(List<ReferenceDescription> lstReferenceDescription,
ref Subscription objSubscription, List<bool> bUseDataChangeFilter = null,
List<DataChangeFilter> _filters = null)
```

Parameters

In/Out	Parameter	Description
In	**lstReferenceDescription**	The list of reference descriptions.
In/Out	**objSubscription**	Contains the parameter of the subscription.
In	**bUseDataChangeFilter**	Set to true if the DataChangeFilter will be enabled and to false if the DataChangeFilterwill be disabled
In	**filter**	Contains the parameter of the data change filer

Table 32: Parameters of CreateMonitoredItems

Returned Codes

This methods outputs the same codes returned by the CreateMonitoredItem method. Refer to Table 31 for more details.

4.6. Delete Monitored Items

This function removes the added items from the subscription. The following table describes the parameters of this function.

uint DeleteMonitoredItems(List<ReferenceDescription> lstReferenceDescriptionref, ref Subscription subscription)

Parameters

In/Out	Parameter	Description
In	**lstReferenceDescription**	The list of reference descriptions.
In/Out	**objSubscription**	Contains the parameter of the subscription.

Table 33: Parameters of DeleteMonitoredItems

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Bad_SubscriptionIdInvalid	The subscription id is not valid.
Bad_MonitoredItemIdInvalid	The monitoring item id does not refer to a valid monitored item.

Table 34: Returned Codes of DeleteMonitoredItems

4.7. Acknowledge

This function acknowledges the state of a condition or an alarm. The following table describes the parameters of this function.

```
uint Acknowledge(string strSessionName, NodeId objConditionId, byte[] yArrEventId,
string strComment)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session name
In	**objConditionId**	Contains the parameter of the subscription.
In	**yArrEventId**	The event identifier
In	**strComment**	The acknowledge comment

Table 35: Parameters of Acknowledge

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 36: Returned Codes of Acknowldge

4.8. Confirm

This function confirms the state of a condition or an alarm. The following table describes the parameters of this function.

```
uint Confirm(string strSessionName, NodeId objConditionId, byte[] yArrEventId, string
strComment)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session name
In	**objConditionId**	Contains the parameter of the subscription.
In	**yArrEventId**	The event identifier
In	**strComment**	The acknowledge comment

Table 37: Parameters of Confirm

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 38:Returned Codes of Confirm

4.9. Refresh

This function requests the server to refresh all conditions being monitored by the subscription.

```
void  ConditionRefresh()
```

5. Read

5.1. Read Value

This function reads the value of a node. The following table describes the parameters of this function.

```
uint ReadValue(string strSessionName, string strNodeId, out DataValue objDataValue)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session name.
In	**strNodeId**	The identifier of the node to be read.
Out	**objDataValue**	The data value of the node.

Table 39: Parameters of ReadValue

DataValue attributes:

Attribute	Description
ServerTimestamp	The server timestamp associated with the value.
SourceTimestamp	The source timestamp associated with the value.
WrappedValue	The value of the data value.
StatusCode	The status code associated with the value.

Table 40: Parameters of DataValue

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Bad_MaxAgeInvalid	The max age parameter is invalid.

Bad_TimestampsToReturnInvalid	The timestamps to return parameter is invalid.
Bad_NodeIdInvalid	The syntax of the node id is not valid.
Bad_NodeIdUnknown	The node id refers to a node that does not exist in the server address space.
Bad_AttributeIdInvalid	The attribute is not supported for the specified node.
Bad_IndexRangeInvalid	The syntax of the index range parameter is invalid.
Bad_IndexRangeNoData	No data exists within the range of indexes specified.
Bad_DataEncodingInvalid	The data encoding is invalid. This result is used if no dataEncoding can be applied because an Attribute other than Value was requested or the DataType of the Value Attribute is not a subtype of the Structure DataType.
Bad_DataEncodingUnsupported	The server does not support the requested data encoding for the node. This result is used if a dataEncoding can be applied but the passed data encoding is not known to the Server.
Bad_NotReadable	The access level does not allow reading or subscribing to the Node.
Bad_UserAccessDenied	User does not have permission to perform the requested operation.
Bad_SecurityModeInsufficient	The security level is not high enough to complete the operation. A user may have the right to receive the data but the data can only be transferred through an encrypted channel or may require other settings with higher security level.

Table 41: Returned Codes of ReadValue

5.2. Read Values

This function reads the values of a list of nodes. The following table describes the parameters of this function.

```
List<uint> ReadValues(string strSessionName, List<string> lstNodeId, out List<DataValue>
lstDataValue)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session name.
In	**lstNodeId**	The list of nodes identifiers.
Out	**lstDataValue**	The list of data values.

Table 42: Parameters of ReadValues

Returned Codes

This methods outputs the same codes returned by the ReadValue method. Refer to Table 41 for more details.

The returned value shall not be used when having a bad/uncertain or failed status code.

6. Write

6.1. Write Value

This function writes a value to a node. The following table describes the parameters of this function.

```
uint WriteValue(string strSessionName, string strNodeId, string strValueToWrite)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session name.
In	**strNodeId**	The identifier of the node to be written.
In	**strValueToWrite**	The value to be written.

Table 43: Parameters of WriteValue

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Good_CompletesAsynchronously	The value was successfully written to an intermediate system but the Server does not know if the data source was updated properly.
Bad_NodeIdInvalid	The syntax of the node id is not valid.
Bad_NodeIdUnknown	The node id refers to a node that does not exist in the server address space.
Bad_AttributeIdInvalid	The attribute is not supported for the specified node.
Bad_IndexRangeInvalid	The syntax of the index range parameter is invalid.
Bad_IndexRangeNoData	No data exists within the range of indexes specified.
Bad_WriteNotSupported	If a Client attempts to write any value, quality, timestamp combination and the

	Server does not support the requested combination (which could be a single quantity such as just timestamp), than the Server shall not perform any write on this Node and shall return this StatusCode for this Node. It is also used if writing an IndexRange is not supported for a Node.
Bad_NotWritable	The access level does not allow writing to the Node.
Bad_UserAccessDenied	The current user does not have permission to write the attribute.
Bad_OutOfRange	If a Client attempts to write a value outside the valid range like a value not contained in the enumeration data type of the Node, the Server shall return this StatusCode for this Node.
Bad_TypeMismatch	The value supplied for the attribute is not of the same type as the attribute's value.
Bad_DataEncodingUnsupported	The data encoding is invalid.
Bad_NoCommunication	Communication with the data source is defined, but not established, and there is no last known value available. This status/sub-status is used for cached values before the first value is received or for Write and Call if the communication is not established.
Bad_LocaleNotSupported	The locale in the requested write operation is not supported.

Table 44: Returned Codes of WriteValue

6.2. Write Values

This function writes a list of values to a list of nodes. The following table describes the parameters of this function.

```
List<uint> WriteValues(string strSessionName, List<string> lstNodeId, List<string>
lstValueToWrite)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session name.
In	**lstNodeId**	The list of nodes identifiers.
In	**lstValueToWrite**	The list of values to write.

Table 45: Parameters of WriteValues

Returned Codes

This methods outputs the same codes returned by the WriteValue method. Refer to Table 44 for more details.

7. History Read

7.1. Read Raw

This function reads the historical values for the specified time domain for a list of items. The following table describes the parameters of this function.

```
List<uint> ReadRaw(bool bIsReadModified, DateTime dateStartDateTime, DateTime
dateEndDateTime, int iMaxReturnVal, List<string> lstNodeId, string strSessionName, out
HistoryReadResultCollection objHistoryReadResult)
```

Parameters

In/Out	Parameter	Description
In	**bIsReadModified**	True if it is Read Modified and false if it is Read Raw.
In	**dateStartDateTime**	The start of the history time period to be read.

In	**dateEndDateTime**	The end of the history time period to be read.
In	**iMaxReturnVal**	The maximum values to be returned.
In	**lstNodeId**	The list of nodes identifiers.
In	**strSessionName**	The session name.
Out	**objHistoryReadResult**	The result of the history read operation.

Table 46: Parameters of ReadRaw

* HistoryReadResultCollection: the list of HistoryReadResult.

HistoryReadResult Attributes:

Setting	Description
StatusCode	The status code associated with the result.
ContinuationPoint	Marks a continuation point to read if the values could not be returned in one response.
HistoryData	The history data.
TypeId	The UA type identifier.
BinaryEncodingId	The UA type identifier for binary encoding.
XmlEncodingId	The UA type identifier for the XML encoding id.

Table 47: HistoryReadResult Parameters

7.2. Read at Time

This function reads the values from the history database for a specified timestamp for a list of items. The following table describes the parameters of this function.

```
List<uint> ReadAtTime(DateTime dateStartDateTime, List<string> lstNodeId, string
strSessionName, out HistoryReadResultCollection objHistoryReadResult
```

Parameters

In/Out	Parameter	Description
In	**dateStartDateTime**	The datetime for the requested data.
In	**lstNodeId**	The list of nodes identifiers.
In	**strSessionName**	The session name.
Out	**objHistoryReadAtTimeResult**	The result of the history read at time operation.

Table 48: Parameters of ReadAtTime

7.3. Read Processed

This function returns aggregate values from data in the history database for the specified time domain for a list of items. The following table describes the parameters of this function.

```
List<uint> ReadProcessed(DateTime dateStartDateTime, DateTime dateEndDateTime, int
iProcessInterval, string strAggregate, List<string> lstNodeId, string strSessionName,
out HistoryReadResultCollection objHistoryReadResult)
```

Parameters

In/Out	Parameter	Description
In	**dateStartDateTime**	The start of the history time period to be read.
In	**dateEndDateTime**	The end of the history time period to be read.
In	**iProcessInterval**	Interval between returned values.
In	**strAggregate**	The calculation to be performed on the raw data to create the values to be returned.
In	**lstNodeId**	The list of nodes identifiers.
In	**strSessionName**	The session name.
Out	**objReadProcessedResult**	The result of the read processed operation.

Table 49: Parameters of ReadProcessed

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Bad_TimestampsToReturnInvalid	The timestamps to return parameter is invalid.
Bad_HistoryOperationInvalid	The history details parameter is not valid.
Bad_HistoryOperationUnsupported	The requested history operation is not supported by the server.
Bad_NodeIdInvalid	The syntax of the node id is not valid.
Bad_NodeIdUnknown	The node id refers to a node that does not exist in the server address space.
Bad_DataEncodingInvalid	The data encoding is invalid.
Bad_DataEncodingUnsupported	The server does not support the requested data encoding for the node. This result is used if a dataEncoding can be applied but the passed data encoding is not known to the Server.
Bad_UserAccessDenied	User does not have permission to perform the requested operation.
Bad_ContinuationPointInvalid	The continuation point provided is no longer valid. This status is returned if the continuation point was deleted or the address space was changed between the browse calls.

Bad_InvalidTimestampArgument	The defined timestamp to return was invalid.
Bad_HistoryOperationUnsupported	The requested history operation is not supported for the requested node.
Bad_NoContinuationPoints	The operation could not be processed because all continuation points have been allocated.

Table 50: Returned Codes of HistoryRead

8. Acknowledge Event

The Acknowledge method is used to acknowledge an event notification for a condition instance state. The following table describes the parameters of this function.

```
uint Acknowledge(string strSessionName, NodeId objConditionId, byte[] yArrEventId,
string strComment)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session Name.
In	**objConditionId**	The condition NodeId.
In	**yArrEventId**	The event identifier.
In	**strComment**	The acknowledge comment.

Table 51: Parameters of Acknowledge

Returned Codes

Return Code	Description
Good	The operation was successful.

Bad	The operation failed but no specific reason is known.
Bad_ConditionBranchAlreadyAcked	The EventId does not refer to a state that needs acknowledgement.
Bad_MethodInvalid	The method id does not refer to a method for the specified object or ConditionId.
Bad_EventIdUnknown	The specified EventId is not known to the Server.
Bad_NodeIdInvalid	The specified ObjectId is not valid or the Method was called on the ConditionType Node.

Table 52: Returned Codes of Acknowledge

9. Call Method

9.1 Fetch method arguments

The function FetchArgumentForMethod returns the input arguments or the output argument of a specific method. The following table describes the parameters of this function.

```
 void FetchArgumentForMethod(string sessionName,string methodId,bool bInput ,out
object[] dataypes, out object[] names, out object[] desc)
```

Parameters

In/Out	Parameter	Description
In	**sessionName**	The session Name.
In	**methodId**	The method Name.
In	**bInput**	• True: for input arguments • False: for output arguments
Out	**dataypes**	Array of arguments dataype
Out	**names**	Array of arguments name
Out	**desc**	Array of arguments description

Table 53:Parameters of FetchArgumentForMethod

9.2 Call Method

The CallMethod function calls a specific method to be executed by the server. The following table describes the parameters of this function.

```
uint CallMethod(string strSessionName, ReferenceDescription methodReference, ReferenceDescription methodParentReference, object[] objValues, out VariantCollection outputArguments)
```

Parameters

In/Out	Parameter	Description
In	**strSessionName**	The session Name.
In	**methodReference**	The method to be called
In	**methodParentReference**	The parent node of the method to be called
In	**objValues**	The array of input arguments values
Out	**outputArguments**	The output arguments of the method

Table 54: Parameters of CallMethod

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.
BadReferenceNotAllowed	The reference could not be created because it violates constraints imposed by the data model.
BadMethodInvalid	The method id does not refer to a method for the specified object.

Table 55: Returned Codes of CallMethod

10. Certificate Management

10.1. Trust Certificate

This function trusts the certificate by adding it to the trusted certificate store of the OPC UA Client defined in its XML configuration file. The following table describes the parameters of this function.

```
uint TrustCertificate(X509Certificate2 objCertificateToTrust)
```

Parameters

In/Out	Parameter	Description
In	**objCertificateToTrust**	The certificate to be trusted by the UA Client.

Table 56: Parameters of TrustCertificate

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 57: Returned Codes of TrustCertificate

10.2. Reject Certificate

This function rejects the certificate by adding it to the rejected certificate store of the OPC UA Client defined in its XML configuration file. The following table describes the parameters of this function.

```
uint RejectCertificate(X509Certificate2 objCertificateToReject)
```

Parameters

In/Out	Parameter	Description
In	**objCertificateToReject**	The certificate to be rejected by the UA Client.

Table 58: Parameters of RejectCertificate

Returned Codes

Return Code	Description
Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 59: Returned Codes of RejectCertificate

10.3. Assign Certificate

This function assigns a certificate to the OPC UA Client by providing the certificate path and the certificate password. The following table describes the parameters of this function.

```
uint AssignCertificate(string strCertificatePath, string strCertificatePassword)
```

Parameters

In/Out	Parameter	Description
In	**strCertificatePath**	The certificate path to be assigned to the UA Client. The certificate should be a .pfx file.
In	**strCertificatePassword**	The password of the certificate.

Table 60: Parameters of AssignCertificate

Returned Codes

Return Code	Description

Good	The operation was successful.
Bad	The operation failed but no specific reason is known.

Table 61: Returned Codes of AssignCertificate

Note that the certificates paths and other parameters are configurable from an XML configuration file that should be located in the same folder as the OPC UA client application and should be named *XXXX*.Config.xml. *XXXX* is the name of your OPC UA client or should be located in the specified path in case the file path was configured by the user.

11. Publish Errors Handling

The following delegate is used to handle publishing errors:

```
void PublishErrorEventHandler(Session session, PublishErrorEventArgs e)
```

To properly set up the error handling, you need to perform the following steps:

- Step 1: Define the delegate

```
private  PublishErrorEventHandler m_sessionNotificationError;
```

- Step 2: Initialize the delegate

The publish errors handler constuctor expects as input a `void` method with `Session` and `PublishErrorEventArgs` as input paramters.

Below is an implementation example :

```
m_sessionNotificationError = new PublishErrorEventHandler(Session_publishError);

private void Session_publishError(Session session, PublishErrorEventArgs e)
        {
           //your error handling logic here
        }
```

- Step 3: Assign the session's PublishError field to the delegate

The final step is to add to the delegate the used session's PublishError field :

```
m_session.PublishError += m_sessionNotificationError;
```

OPC UA CLIENT SAMPLE

This chapter describes the required steps on how to use the OPC UA sample available within the installation of OPC UA Client Toolkit.

1. Step 1: Open OPC UA Sample Client

The OPC UA Sample Client allows you to manage multiple sessions, to monitor data, events and alarms, and to explore historical data.

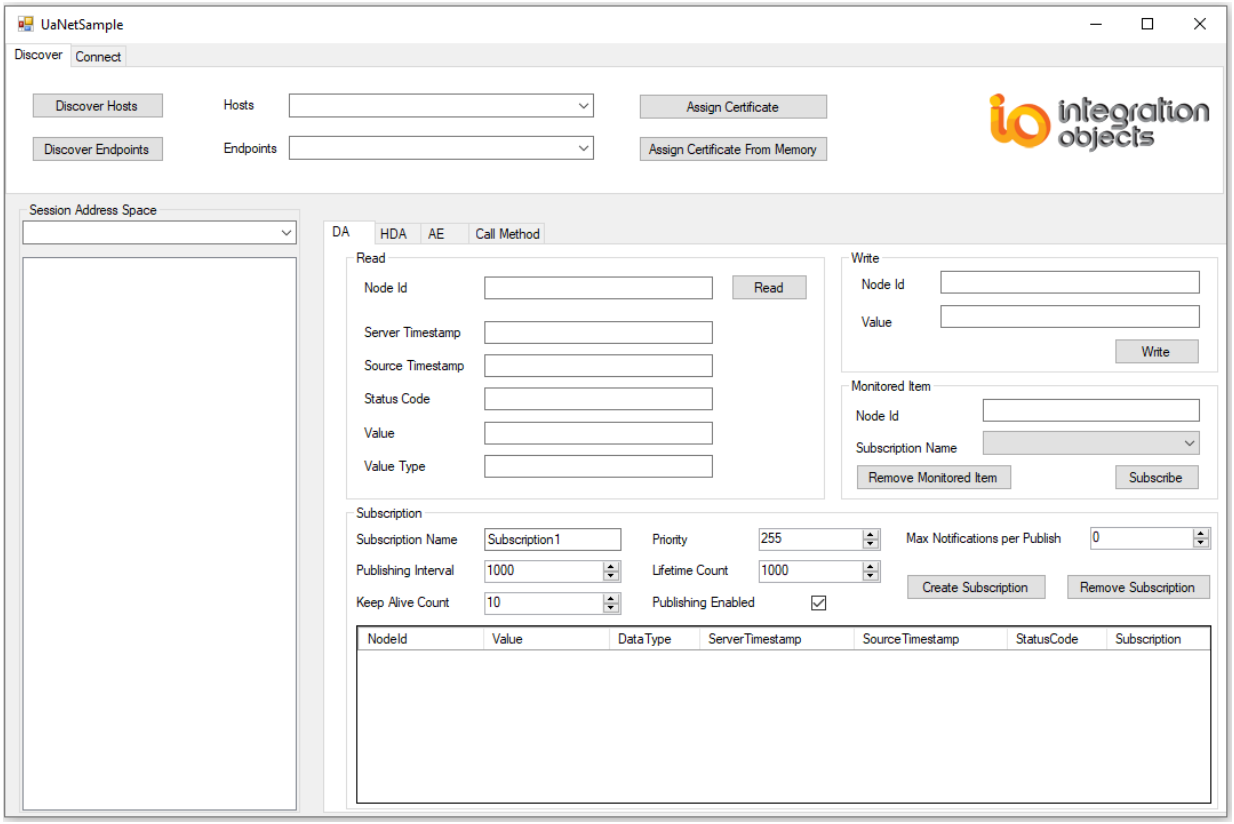

Figure 17: OPC UA Sample Client User Interface

2. Step 2: Discover OPC UA Servers

To list all the available OPC UA Servers endpoints, select the **Discover Hosts** button to get the list of hosts on the network then select a host name from the **Hosts** comboBox and click the **Discover Endpoints** button as shown below:

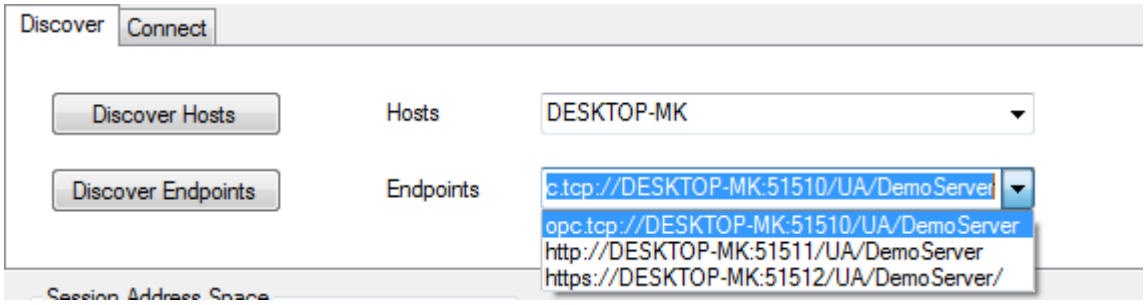

Figure 18: Discover OPC UA Servers Endpoints

3. Step 3: Connect

To connect to an UA endpoint, set the server endpoint URL, the session name, the transport protocol, the security parameters, the user identity mode and click the **Connect** button or the **Connect2** button **(**if you have assigned a certificate from memory and you want to connect using the added certificate) as illustrated in the figure below:

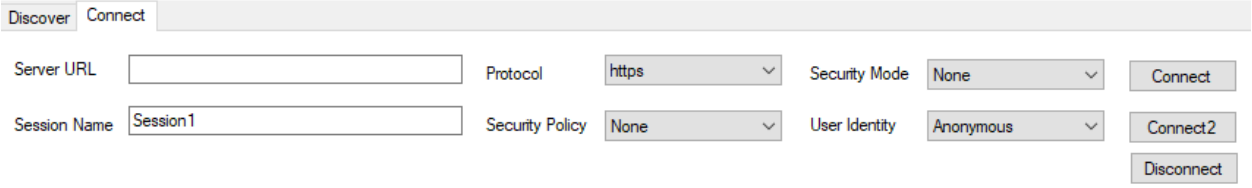

Figure 19: Connect to an UA Server

If the certificate of the server is not trusted, it will be returned by the **CreateSession** method and then trusted by calling the **TrustCertificate** method.

4. Step 4: Browse Address Space

To browse the address space, select a session name from the **session** combobox and expand the treeview nodes as shown below:

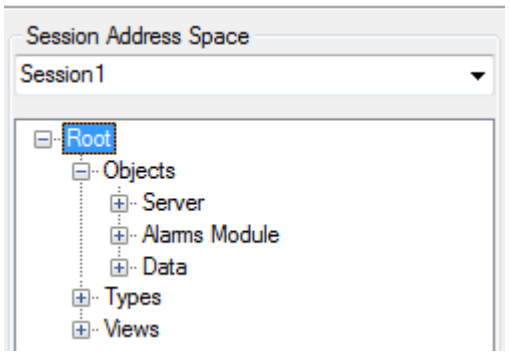

Figure 20: Browse UA Server Address Space

5. Step 5: Subscribe

To create a subscription, fill the subscription parameters then click **Create Subscription** button as shown in the figure below:

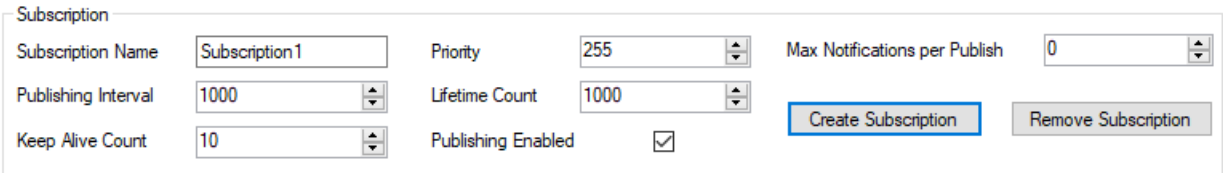

Figure 21: Create a Subscription

To subscribe to a DA monitored item, select a node from the address space, select a subscription name from the **Subscription Name** combobox and click the **Subscribe** button.

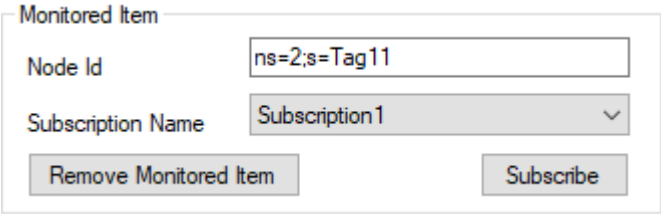

Figure 22: Subscribe to a DA Monitored Item

The data change notifications will be displayed on the datagridview as follows:

NodeId	Value	DataType	ServerTimestamp	SourceTimestamp	StatusCode	Subscription
ns=2;s=Tag11	11	Int16	2023-08-15 14:47:07.976	2023-08-15 14:47:07.739	Good	Subscription1
ns=2;s=Tag12	3	Int32	2023-08-15 14:47:22.046	2023-08-15 14:47:22.046	Good	Subscription1

Figure 23: Display Data Change Notifications

To subscribe to an Event Notifier, select the node from the address space, type the subscription name if it is already created or fill the subscription parameters to create a new one then click **Create Subscription** button. The alarms and events will be displayed on the datagridview as shown below:

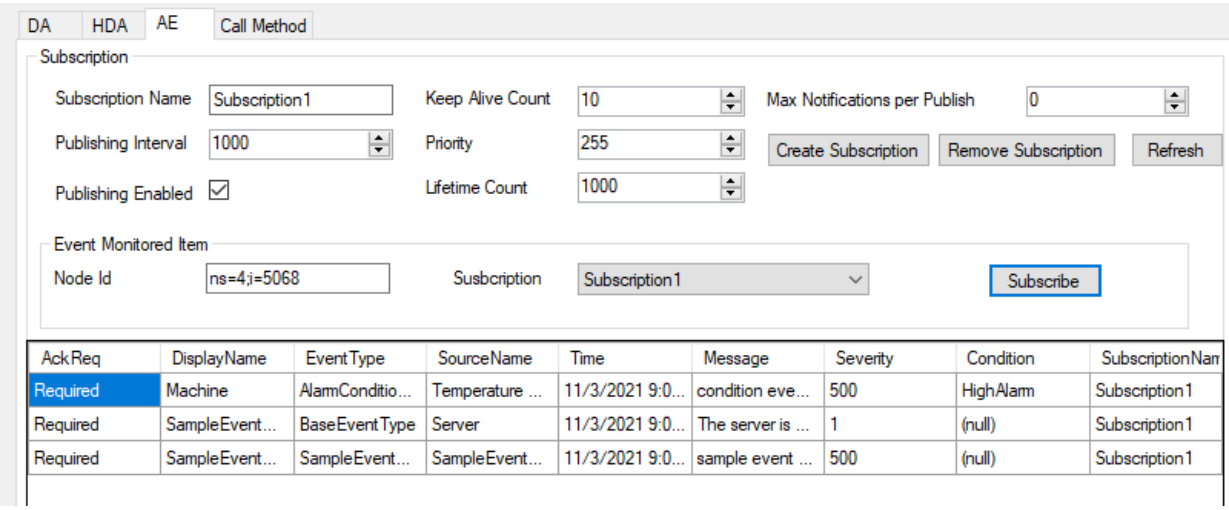

Figure 24: Display Alarms and Events

6. Step 6: Read

To read the value of a node, select a node from the server address space and click **Read** button as follows:

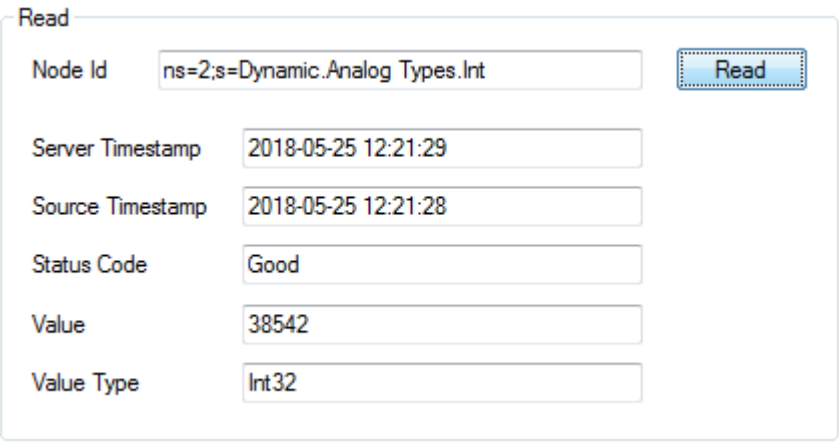

Figure 25: Read

7. Step 7: Write

To write a value to a node, select a node from the server address space, type the value to be written and click **Write** button as follows:

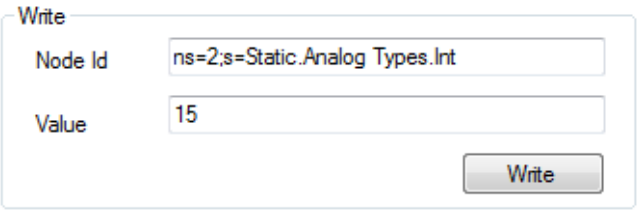

Figure 26: Write

8. Step 8: History Read

To read the historical values of an item, select a node from the server address space, fill the history read parameters, click **OK** button and the history result will be displayed in the datagridview as illustrated in the figure below:

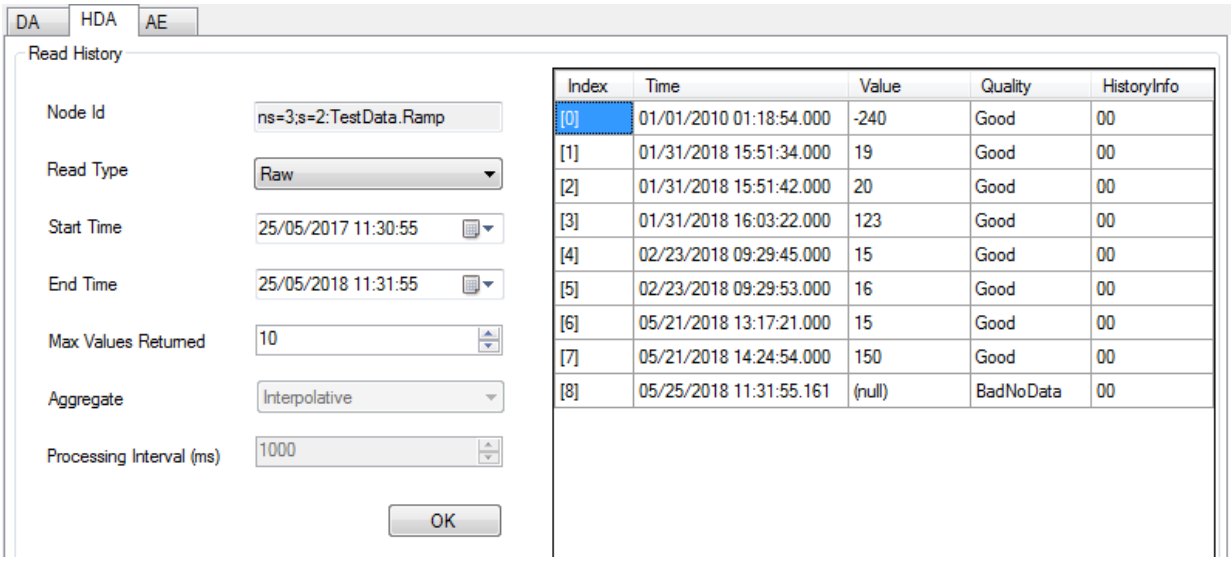

Figure 27: History Read

9. Step 9: Refresh Condition

The Condition Refresh allows a client to request a refresh of all condition instances that currently are in an interesting state. A Client would typically invoke this Method when it initially connects to a server and following any situations.

To refresh the condition, create an AE subscription then click **Refresh** button.

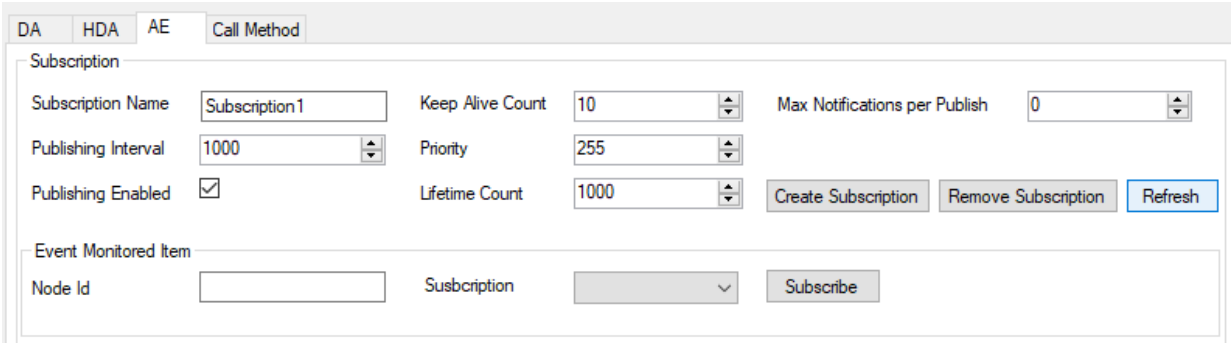

Figure 28: Refresh Condition

10. Step 10: Call Method

To call a method, select a node (method name) from the server address space click on the "Call Method" tab.

Click on **Fetch Arguments** button the Input and output arguments will be displayed in the grid views.

Enter the input values in the “Input Value” column and click **Call** button and the result will be returned in the Output Value column.

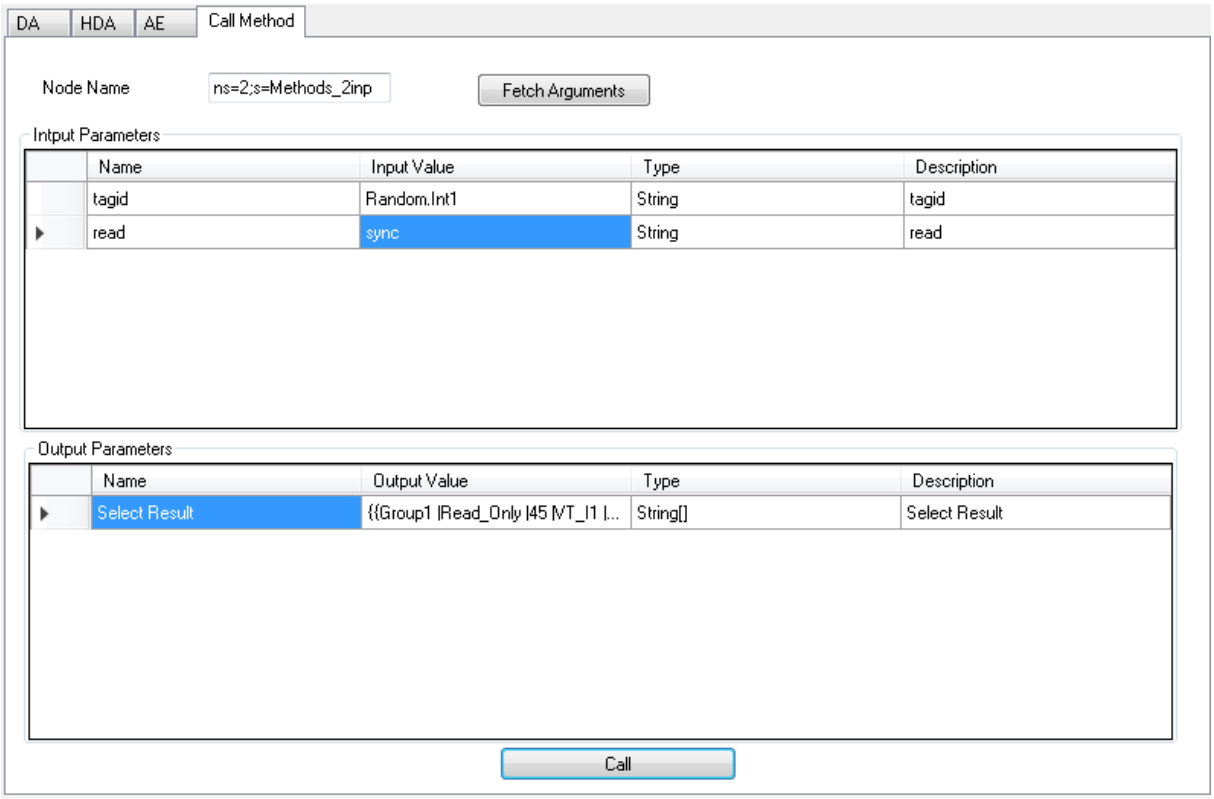

Figure 29: Call Method

11. Step 11: Assign Certificate

11.1. Assign Certificate

To assign a certificate to the OPC UA Client application, click **Assign Certificate** button, select your .pfx certificate and type your certificate password as follows:

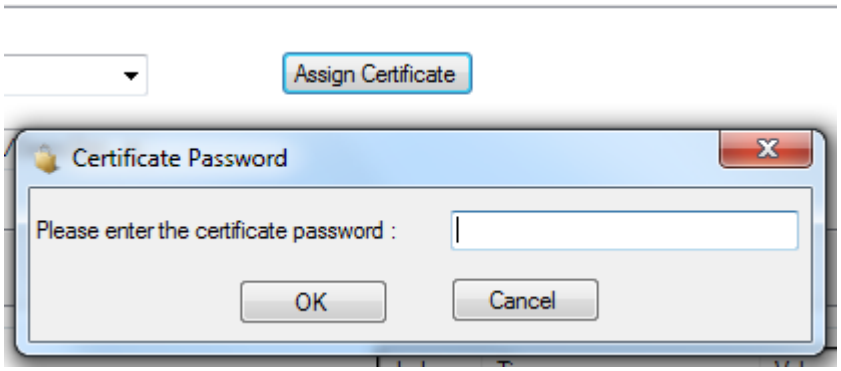

Figure 30: Assign Certificate

11.2. Assign Certificate from Memory

To assign a certificate to the OPC UA Client application from memory, click **Assign Certificate from Memory** button then select a certificate from the displayed list as follows:

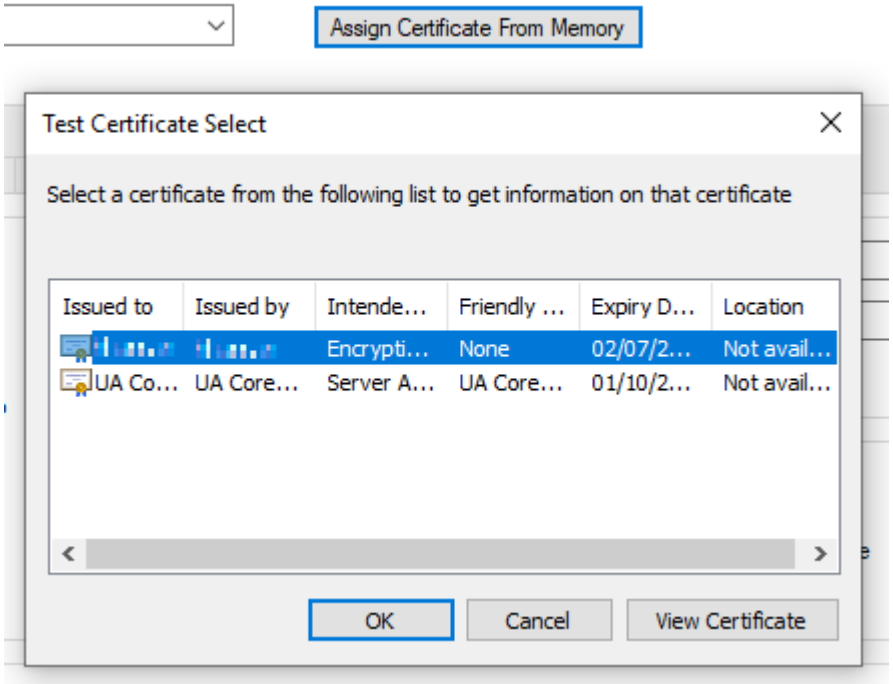

Figure 31: Assign Certificate from Memory

OPC UA CLIENT .NET CORE CONSOLE SAMPLE

This chapter describes the required steps on how to use the OPC UA .Net Core Console sample available within the installation of OPC UA Client Toolkit.

1. Step 1: Configuration

The OPC UA .Net Core Console Sample runs based on the settings specified in the JSON Configuration file named **ConnectionConfig.json**, which allows you to configure the following parameters:

- Connection parameters
- DA node to read.
- DA node and values to be written
- Historical data nodeIds
- DA node to monitor
- AE node to monitor

Following is an example:

```
{
    "m_server": {
        "Protocol": "opc.tcp",
        "SecurityPolicy": "None",
        "UserIdentity": "Anonymous",
        "MessageEncoding": "binary",
        "SecurityMode": "None",
        "ServerName": "opc.tcp://localhost:62640/IntegrationObjects/ServerSimulator",
        "UserName": "",
        "UserPassword": ""
    },
    "NodesToRead": [
        "ns=2;s=Tag1"
    ],
    "WriteValues": [
        {
            "NodeId": "ns=2;s=Tag1",
            "Value": "285"
        }
    ],
    "DANodesToMonitor": [
        "ns=2;s=Tag1"
        "ns=2;s=Tag12"
        "ns=2;s=Tag17"
        "ns=2;s=Tag17"
    ],
    "AENodesToMonitor": [
        "ns=0;i=2253"
    ],
    "HistoryVariableIds": [
        "ns=2;s=1:Tag17"
    ]
}
```

Figure 32: Configuration Settings

2. Step 2: Open OPC UA .Net Core Console Sample

The OPC UA .Net Core Sample Client allows the creation of a unique session,subscription and creation of data and events monitored items, and the exploration of historical data.

```
==================================================================================================
==                                Integration Objects' OPC UA Client Toolkit                    ==
==                                            Version : 2.0.1                                   ==
==                                 Copyright © 2020-2024 Integration Objects                    ==
==================================================================================================



-------------------------------------------------------
 - Press x to close client                            -
-------------------------------------------------------
 - Press 1 to connect                                 -
```

Figure 33: Startup Menu

3. Step 3: Connect

To connect to OPC UA server according the current configuration, press “1”. If the connection succeeded the following menu will be dispalyed.

```
-----------------------------------------------------
 - Press x to close client                          -
-----------------------------------------------------
 - Press 1 to connect                               -
1The session [UANetCoreSession] was created successfully.

Successfully connected to opc.tcp://localhost:62640/IntegrationObjects/ServerSimulator


-----------------------------------------------------
 - Press 0 to disconnect                            -
-----------------------------------------------------
 - Press 1 to read nodes                            -
 - Press 2 to write values                          -
 - Press 3 to browse server                         -
 - Press 4 to create a subscription                 -
 - Press 5 to delete a subscription                 -
 - Press 6 to add data monitored items              -
 - Press 7 to add event monitored items             -
 - Press 8 to delete monitored items                -
 - Press 9 to history read data                     -
 - Press A to acknowldge alarms                     -
 - Press B to Confirm alarms                         -
```

Figure 34: Connected Menu

4. Step 4: Read

To read the values of nodes configured in the JSON file, press “1”.

```
1
7941

Read succeeded


-----------------------------------------------------
 - Press 0 to disconnect                            -
-----------------------------------------------------
 - Press 1 to read nodes                            -
 - Press 2 to write values                          -
 - Press 3 to browse server                         -
 - Press 4 to create a subscription                 -
 - Press 5 to delete a subscription                 -
 - Press 6 to add data monitored items              -
 - Press 7 to add event monitored items             -
 - Press 8 to delete monitored items                -
 - Press 9 to history read data                     -
```

Figure 35: Read Output

5. Step 5: Write

To write values in the nodes configured in the JSON file, press “2”. If the write succeeded, the result will be displayed as well as the following menu.

```
2
Write succeeded

Write result 0


-------------------------------------------------------
 - Press 0 to disconnect                              -
-------------------------------------------------------
 - Press 1 to read nodes                              -
 - Press 2 to write values                            -
 - Press 3 to browse server                           -
 - Press 4 to create a subscription                   -
 - Press 5 to delete a subscription                   -
 - Press 6 to add data monitored items                -
 - Press 7 to add event monitored items               -
 - Press 8 to delete monitored items                  -
 - Press 9 to history read data                       -
```

Figure 36: Write Output

6. Step 6: Browse the OPC UA Server

To browse the OPC UA server address space, press “3”. If the browse succeded, the list of all nodes of the server address space will be listed.

```
ns=2;s=1:Tag1
ns=2;s=1:Tag10
ns=2;s=1:Tag11
ns=2;s=1:Tag12
ns=2;s=1:Tag13
ns=2;s=1:Tag14
ns=2;s=1:Tag15
ns=2;s=1:Tag16
ns=2;s=1:Tag17
ns=2;s=1:Tag18
ns=2;s=1:Tag19
ns=2;s=1:Tag2
ns=2;s=1:Tag20
ns=2;s=1:Tag3
ns=2;s=1:Tag4
ns=2;s=1:Tag5
ns=2;s=1:Tag6
ns=2;s=1:Tag7
ns=2;s=1:Tag8
ns=2;s=1:Tag9
ns=2;s=1:Tag1?Annotations
ns=2;s=1:Tag1?HA Configuration
ns=2;s=1:Tag1?HA Configuration/AggregateConfiguration
ns=2;s=1:Tag1?HA Configuration/AggregateFunctions
ns=2;s=1:Tag1?HA Configuration/Stepped
ns=2;s=1:Tag1?HA Configuration/Definition
ns=2;s=1:Tag1?HA Configuration/MaxTimeInterval
ns=2;s=1:Tag1?HA Configuration/MinTimeInterval
ns=2;s=1:Tag1?HA Configuration/ExceptionDeviation
ns=2;s=1:Tag1?HA Configuration/ExceptionDeviationFormat
ns=2;s=1:Tag1?HA Configuration/StartOfArchive
ns=2;s=1:Tag1?HA Configuration/StartOfOnlineArchive
ns=2;s=1:Tag1?HA Configuration/AggregateConfiguration/TreatUncertainAsBad
ns=2;s=1:Tag1?HA Configuration/AggregateConfiguration/PercentDataBad
ns=2;s=1:Tag1?HA Configuration/AggregateConfiguration/PercentDataGood
ns=2;s=1:Tag1?HA Configuration/AggregateConfiguration/UseSlopedExtrapolation
ns=2;s=1:Tag10?Annotations
ns=2;s=1:Tag10?HA Configuration
ns=2;s=1:Tag10?HA Configuration/AggregateConfiguration
ns=2;s=1:Tag10?HA Configuration/AggregateFunctions
ns=2;s=1:Tag10?HA Configuration/Stepped
ns=2;s=1:Tag10?HA Configuration/Definition
ns=2;s=1:Tag10?HA Configuration/MaxTimeInterval
ns=2;s=1:Tag10?HA Configuration/MinTimeInterval
ns=2;s=1:Tag10?HA Configuration/ExceptionDeviation
ns=2;s=1:Tag10?HA Configuration/ExceptionDeviationFormat
ns=2;s=1:Tag10?HA Configuration/StartOfArchive
ns=2;s=1:Tag10?HA Configuration/StartOfOnlineArchive
ns=2;s=1:Tag10?HA Configuration/AggregateConfiguration/TreatUncertainAsBad
ns=2;s=1:Tag10?HA Configuration/AggregateConfiguration/PercentDataBad
ns=2;s=1:Tag10?HA Configuration/AggregateConfiguration/PercentDataGood
ns=2;s=1:Tag10?HA Configuration/AggregateConfiguration/UseSlopedExtrapolation
ns=2;s=1:Tag11?Annotations
ns=2;s=1:Tag11?HA Configuration
ns=2;s=1:Tag11?HA Configuration/AggregateConfiguration
ns=2;s=1:Tag11?HA Configuration/AggregateFunctions
ns=2;s=1:Tag11?HA Configuration/Stepped
ns=2;s=1:Tag11?HA Configuration/Definition
ns=2;s=1:Tag11?HA Configuration/MaxTimeInterval
ns=2;s=1:Tag11?HA Configuration/MinTimeInterval
ns=2;s=1:Tag11?HA Configuration/ExceptionDeviation
ns=2;s=1:Tag11?HA Configuration/ExceptionDeviationFormat
```

Figure 37: Browse Output

7. Step 7: Create a Subscription

To create a subscription, press “4”. If the subscription is created succefully, a confirmation message will be displayed.

```
4
Create subscrition succeeded


-------------------------------------------------------
 - Press 0 to disconnect                              -
-------------------------------------------------------
 - Press 1 to read nodes                              -
 - Press 2 to write values                            -
 - Press 3 to browse server                           -
 - Press 4 to create a subscription                   -
 - Press 5 to delete a subscription                   -
 - Press 6 to add data monitored items                -
 - Press 7 to add event monitored items               -
 - Press 8 to delete monitored items                  -
 - Press 9 to history read data                       -
```

Figure 38: Create Subscription Output

8. Step 8: Delete the Subscription

To remove the created subscription, press “5”. If the operation succeeded, a confirmation message will be displayed.

```
5
Delete Subscription succeeded


-------------------------------------------------------
 - Press 0 to disconnect                              -
-------------------------------------------------------
 - Press 1 to read nodes                              -
 - Press 2 to write values                            -
 - Press 3 to browse server                           -
 - Press 4 to create a subscription                   -
 - Press 5 to delete a subscription                   -
 - Press 6 to add data monitored items                -
 - Press 7 to add event monitored items               -
 - Press 8 to delete monitored items                  -
 - Press 9 to history read data                       -
```

Figure 39: Delete Subscription Output

9. Step 9: Add Data Monitored Items

If the subscription is created successfully, add data monitored items to this subscription by pressing “6”. If the operation succeeded, the client will start receiving notifications from the server for the data changes.

```
6
Create MonitoredItems succeeded
[ns=2;s=Tag2] Create MonitoredItem succeeded
ns=2;s=Tag2, 1044627583, Int32, 8/16/2021 3:22:04 PM, 8/16/2021 2:58:49 PM, Good, MySampleSubscription
ns=2;s=Tag2, 1044627583, Int32, 8/16/2021 3:22:04 PM, 8/16/2021 2:58:49 PM, Good, MySampleSubscription


-----------------------------------------------------
 - Press 0 to disconnect                            -
-----------------------------------------------------
 - Press 1 to read nodes                            -
 - Press 2 to write values                          -
 - Press 3 to browse server                         -
 - Press 4 to create a subscription                 -
 - Press 5 to delete a subscription                 -
 - Press 6 to add data monitored items              -
 - Press 7 to add event monitored items             -
 - Press 8 to delete monitored items                -
 - Press 9 to history read data                     -
```

Figure 40: Add Data Monitored items Output

10. Step 10: Add Event Monitored Items

If the subscription is created successfully, you can subscribe to an Event Notifier using this subscription by pressing “7”. If the operation succeeded, the client will start receiving notifications from the server when an event occurs.

11. Step 11: Delete Monitored Items

To remove all the monitored items in the active subscription, press “8”. If the operation succeeded, the following message will be displayed.

```
8
DeleteMonitoredItems succeeded.


-------------------------------------------------------
 - Press 0 to disconnect                              -
-------------------------------------------------------
 - Press 1 to read nodes                              -
 - Press 2 to write values                            -
 - Press 3 to browse server                           -
 - Press 4 to create a subscription                   -
 - Press 5 to delete a subscription                   -
 - Press 6 to add data monitored items                -
 - Press 7 to add event monitored items               -
 - Press 8 to delete monitored items                  -
 - Press 9 to history read data                       -
```

Figure 41: Delete Monitored Items Output

12. Step 12: Read History Data

To read historical data values of configured node, press “9”. If the operation succeeded, the list of values with their server timestamps will be displayed.

```
9
Node ns=2;s=1:Tag1
8/16/2021 2:02:53 PM: 285
8/16/2021 2:05:06 PM: 285
8/16/2021 2:47:01 PM: 25823


-------------------------------------------------------
 - Press 0 to disconnect                               -
-------------------------------------------------------
 - Press 1 to read nodes                               -
 - Press 2 to write values                             -
 - Press 3 to browse server                            -
 - Press 4 to create a subscription                    -
 - Press 5 to delete a subscription                    -
 - Press 6 to add data monitored items                 -
 - Press 7 to add event monitored items                -
 - Press 8 to delete monitored items                   -
 - Press 9 to history read data                        -
```

Figure 42: Read History Data Output

13. Step 13: Acknowledge Alarms

To acknowldge alarms, press “A”. If the operation succeeded, the following menu will be displayed:

```
-------------------------------------------------------
 - Press ESC to return to connect menu                 -
-------------------------------------------------------
 - Press a to acknowledge all alarms                   -
 - Press number to acknowledge single alarm            -
 - Number: Acknowledged - Time - Severity - Source - Message
alarms:C0-20-8C-D2-A8-34-F5-49-91-9E-79-FC-A4-A1-26-50
 - Alarm 0:  C0-20-8C-D2-A8-34-F5-49-91-9E-79-FC-A4-A1-26-50 - Unacknowledged - 11/2/2021 3:17:44 PM - 500 - 11/2/2021
:17:44 PM - condition event message
alarms:7B-65-3F-08-45-61-FC-44-BE-72-19-1A-A1-76-2D-E3
 - Alarm 1:  7B-65-3F-08-45-61-FC-44-BE-72-19-1A-A1-76-2D-E3 -  - 11/2/2021 3:26:26 PM - 500 - 11/2/2021 3:26:26 PM - s
mple event message
alarms:1D-95-61-E3-61-E8-BE-4E-93-AC-33-7D-B8-1B-7A-AA
 - Alarm 2:  1D-95-61-E3-61-E8-BE-4E-93-AC-33-7D-B8-1B-7A-AA - Unacknowledged - 11/2/2021 3:26:35 PM - 500 - 11/2/2021
:26:35 PM - condition event message
```

Figure 43: Acknowldge Alarms Menu

14. Step 14: Confirm Alarms

After acknowledging the alarms, you can confirm by pressing “B” in your keyboard. If the operation succeeded, the following menu will be displayed:

```
 - Press ESC to return to connect menu                -
-------------------------------------------------------
 - Press a to Confirm all alarms                      -
 - Press number to confirm single alarm               -
 - Number: Acknowledged - Time - Severity - Source - Message
alarms:C0-20-8C-D2-A8-34-F5-49-91-9E-79-FC-A4-A1-26-50
 - Alarm 0:  C0-20-8C-D2-A8-34-F5-49-91-9E-79-FC-A4-A1-26-50 - Acknowledged - 11/2/2021 3:17:44 PM - 500 - 11/2/2021 3:1
7:44 PM - condition event message
alarms:7B-65-3F-08-45-61-FC-44-BE-72-19-1A-A1-76-2D-E3
 - Alarm 1:  7B-65-3F-08-45-61-FC-44-BE-72-19-1A-A1-76-2D-E3 -  - 11/2/2021 3:26:26 PM - 500 - 11/2/2021 3:26:26 PM - sa
mple event message
alarms:1D-95-61-E3-61-E8-BE-4E-93-AC-33-7D-B8-1B-7A-AA
 - Alarm 2:  1D-95-61-E3-61-E8-BE-4E-93-AC-33-7D-B8-1B-7A-AA - Acknowledged - 11/2/2021 3:26:35 PM - 500 - 11/2/2021 3:2
6:35 PM - condition event message
```

Figure 44: Confirm Alarms Menu

TOOLKIT TRACING CAPABLITIES

The toolkit has tracing capabilities to allow developers to record the toolkit errors and debugging information in a log file named `OPCUANetClientToolkitLog.LOG`. If difficulties occur with the toolkit, the log file can be extremely valuable for troubleshooting.

This log file is generated at start-up where the client executable file is located. The toolkit incorporates a configuration file `Config.json` that includes several logging parameters. All these parameters have default settings and can be changed by editing the configuration file.

To change this file:

1. Open `Config.json` in a text editor.
2. Edit any of the parameters listed in the following tables:

Log Setting	Description	Default Value
Auto Append	Set to true to continue writing log messages in the existed log file or to false to create a new file.	True
Buffer Size	The maximum number of messages to be stored in the runtime memory before launching a write action in the hard disk. The specified value must be greater than 100.	100
Log File Max Size	This is the maximum log file size, in Mega-Bit. Once it is reached the OPC UA Client Toolkit will automatically create a new log file and archive the last one.	10MB
Level	There are five log levels: 1. Control: Logs only control messages. This log level is the lowest level. 2. Error: Logs error and control messages.	Error

	3. Warning: Logs warning, error and control messages 4. Inform: Logs information, warning, error and control messages. 5. Debug: Logs all messages. This is the highest level. The higher the log level, the more information are recorded.	
Maximum Files	Set to 0 means that log files will be created in an unlimited way.	0
Accept Bad Quality On Write	Set to false to write values into "good" quality tags only or to true to write values into "bad" quality tags.	False

Table 62: Log Settings

3. Save the file and restart your client application for the changes to take effect.

Sample Configuration File:

```
{
"FileLogConfiguration": {
"AutoAppend": true,
"BufferSize": 100,
"MaximumFiles": 0,
"Level": "Error",
"FileMaxSize": 10,
“AcceptBadQualityOnWrite”:false
}
}
```

TROUBLESHOOTING

Problem 1: Unable to Discover the OPC UA Servers

If you are not able to discover your OPC UA Server from the OPC UA client but you can directly connect to its endpoint using its URL, make sure that the Local Discovery Service is installed and running on the OPC UA Server machine.

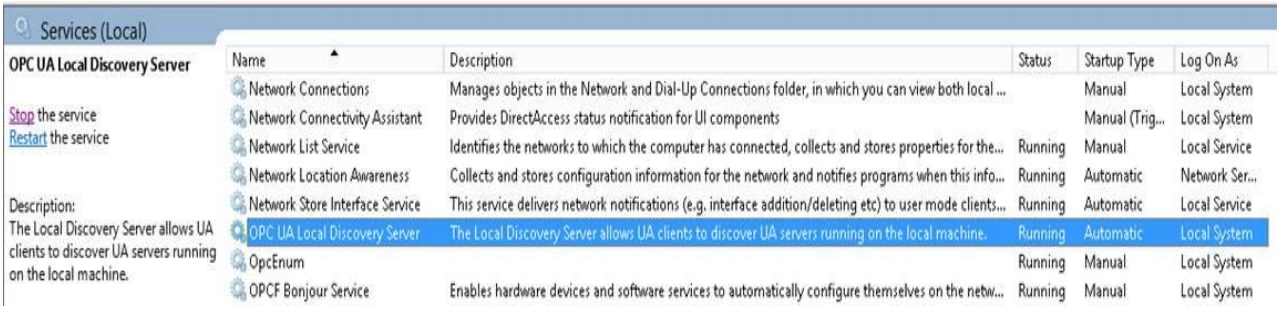

Figure 45:OPC UA Local Discovery Server

Problem 2: “This is not a development machine” Error Message

You have a full version of the toolkit and when you run the application on the runtime machine, the following error message is prompted: “This is not a development machine”. To run your application properly using a full runtime version of the toolkit, make sure that the OPC UA Client Toolkit is not installed as a demo version in the deployment machine. If it is the case, you will need to uninstall it. Also, verify that the following criteria are met:

1. The path of the application folder does not include the words “Debug” or “Release”.
2. The application deployment folder contains the following files:
    - Config.json
    - IntegrationObjects.Logger.SDK.dll
    - IntegrationObjects.OpcUaNetClientToolkit.dll
    - License.dll

- IntegrationObjects.Opc.Ua.Core.dll
- BouncyCastle.Crypto.dll
- System.ServiceModel.Primitives.dll
- The application executable and any other custom assembly dependencies
- The UA XML configuration file (*XXXX*.Config.xml, where *XXXX* is the name of your OPC UA client application or the name you have set if you had configured the application configuration file)
- ConnectionConfig.json (when using the OPC UA .Net Core Client Toolkit)

Problem 3: Unable to Assign a New Certificate

When you run the application, a new instance certificate is not generated or when assigning a new certificate, it is not be added.

For resolution, open the file named *XXXX*.Config.xml where *XXXX* is the name of your OPC UA client or the name you set by configuring the application configuration file, then apply the following changes:

1. Change the <SubjectName> to the one that is originally set
2. Delete the <Thumbprint> and <RawData> tags


```
<SubjectName>CN=test, DC=User-pc</SubjectName>
<Thumbprint>2184704B75BA92246CBDCA9A758A847C031B30E9</Thumbprint>
<RawData>
MIIC4DCCAkmgAwIBAgIRANMrxXNyfyBBs3DFfTbfzl4wDQYJKoZIhvcNAQEFBQAwLjEcMBoGCgmSJomT8ixkARkWDHNzb3VmYXJnaS1
wYzEOMAwGA1UEAxMFdGVzdDIwIBcNMTkwODI3MTI0MjM1WhgPMjA2ODEyMDcxMjQyMzVaMC4xHDAaBgoJkiaJk/IsZAEZFgxzc291Zm
FyZ2ktcGMxDjAMBgNVBAMTBXRlc3QyMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQClxWbnrGBGwCZ+o85roFF/OaNsu5E+eE/+r
nrhhWYW5kpakXywf8RKsVMqAyEhgNJiinrswBk1GD2/M5CA4SLFkVCreBVshZbApCMbkG+6qtPVVRjBEBCx8gtdgoc+3pqCKUdGxuZn
RnGanr0sSRiPExe+3XL0FjvGddsgxKNvkwIDAQABo4H7MIH4MB0GA1UdDgQWBBSPYVXEvY6iviUhq7Jdgy1w3gMnLzBmBgNVHSMEXzB
dgBSPYVXEvY6iviUhq7Jdgy1w3gMnL6EypDAwLjEcMBoGCgmSJomT8ixkARkWDHNzb3VmYXJnaS1wYzEOMAwGA1UEAxMFdGVzdDKCEQ
DTK8Vzcn8gQbNwxX02385eMAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BAQDAgL0MCAGA1UdJQEB/wQWMBQGCCsGAQUFBwMBBggrBgEFB
QcDAjAvBgNVHREEKDAmhhZ1cm46bG9jYWxob3N0OklPOnRlc3Qyggxzc291ZmFyZ2ktcGMwDQYJKoZIhvcNAQEFBQADgYEAV+Fzfqs/
Am19umIzjXaxy3jcWeS6FUiK1NcgXUrDMc+IrFg19tUZE8LwhdMqsZo5O6WS/jExeTf9hXCjEpVIZP3xEgJQmeXpqrTYYyWsltm48oz
h5wzgIfS9MgW+jBS9bMxF7VHr1HJ6Gm+fG58vSoi6jnG3UoUiqnhFdyEQ4Uc=</RawData>
```


Figure 46:XML Configuration File

Problem 4: “This is not a valid license” Error Message

When you run the application, the following error message is prompted: “This is not a valid license”.

Open the license authorization tool and check the license status. In case the license is valid, check that the License.dll exists in your application output folder.

Problem 5: I Sent the User ID to Integration Objects. Can I Close the Setup Program Now?

You can close the setup program. The user ID will not change the next time you run the setup. Once you receive the activation code, run the setup program using an administrator account and enter the provided code.

Problem 6: Do I Have to Buy a Third Party Library to Be Able to Use This Toolkit?

No. The only license to be purchased is the OPC UA Client Toolkit development license.

Problem 7: By Purchasing the Rights to the OPC UA Client Toolkit, Are We Entitled to Install the Library Only on 1 Machine?

The OPC UA Client Toolkit is licensed per development machine. Meaning, one license can be installed on a single development machine. With respect to runtime, you can deliver as many as you want for free.

Problem 8: Is it Possible to Integrate the Library with Windows Service?

Yes, you can use the OPC UA Client Toolkit to develop your application as Windows service.

Problem 9: Does the Toolkit Support 64-bit?

The toolkit supports 64 bit and 32 bit applications.

For additional information on this guide, questions or problems to report, please contact:

Offices

- Americas: +1 713 609 9208
- Europe-Africa-Middle East: +216 71 195 360

Email

- Support Services:customerservice@integrationobjects.com
- Sales:sales@integrationobjects.com

To find out how you can benefit from other Integration Objects' products and custom-designed solutions, please visit our website www.integrationobjects.com