Quick start guide

PLCcom.Opc.Ua.Sdk for .Net version 5

Indi.Systems GmbH

Flughafenallee 3
D-28199 Bremen
info@indi-systems.de

Tel + 49 421-989703-30
Fax + 49 421-989703-39

Table of contents

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

About this document

This document is intended to provide you with information about the provided functionalities. This is not a complete documentation, but a guide to help you getting started.

Further information can be found in

- Code examples within the supplied software package
- Code examples and FAQ on our website http://www.plccom.net/help_opcua_toolkit/help/index.html
- The online help (index.html) within the software package

All information is supplied without any liability. All rights reserved and subject to change. The contents of this document are protected under international copyright laws. Without prior written consent from the copyright holder, no part of this documentation may be reproduced by means of photocopying, microfilm or other processes, or transcribed or translated into another language or computer language in any form.

Note:
All product names or other names or brands referred to in this documentation are the trademarks or registered trademarks of their respective owners and are the property of those copyright owners.

There's no connection between any of the mentioned trademarks or trademark owner and the Fa. Indi.Systems GmbH. Any mention of brands serves purely as an indication to the intended purpose.

Important note

With the P PLCcom.Opc.Ua.Sdk you or the user will be able to monitor and control systems, machines or similar at your own discretion. For this purpose the user has to have the needed knowledge or activity.

Before the resulting work can be applied to the plant, machine or similar, the creator of a project must test all functions and check for function and interactions with the system, machine or similar. These tests are to be repeated after every software change and after every change to the system, machine or similar or the periphery (network, server, etc.).

If malfunctions occur or are detected, the PLCcom.Opc.Ua.Sdk must not be operated at the plant, machine or similar.

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

What is OPC.UA?

With the OPC UA specification, the OPC Foundation provides a newly developed communication model for the uniform transport of machine data. The goal was to adapt the OPC communication model to the requirements of future applications and to compensate for the existing disadvantages of the DCOM-based OPC interface. The OPC UA communication model is a complete new development, differs considerably from its predecessor and is also not based on the DCOM interface.

The first version of the OPC UA specification was made available in 2006, a revision took place in 2009.

With OPC UA, a future-proof standardized communication standard is provided that also covers the requirements of industry 4.0 applications.

What PLCcom.Opc.Ua.Sdk has to offer?

The PLCcom.Opc.Ua.Sdk is a highly optimized and modern component specially developed for .NET software developers to provide a convenient access to a client-side OPC UA interface, e.g. To read or write data.

The PLCcom.Opc.Ua.Sdk software is the successor / further development of the PLCcom.Opc.Ua.Toolkit.

The software is available for the .net framework and as .net standard component and is also available to developers for cross-platform .net Core or Xamarin applications.

Depending on the version, the libraries are 100% .Net files. The component can be directly linked as a reference, API calls are not necessary. It is easily to use the components in 32 or 64 bit environments as well as across platforms. The internal routines are optimized for high-performance access.

With the PLCcom.Opc.Ua.Sdk, you can create applications that support the most common OPC specifications.

These includes:

- DataAccess (most used)
- Alarm & Conditions
- Historical Data
- Historical Events

Included in the software package are extensive code examples and tutorials, which illustrate the easy connection of an OPC UA server via an OPC interface to your application and can also be used in your projects.

For development support, test server and client applications are included in the delivery package.

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Which advantages provide PLCcom.Opc.Ua.Sdk?

With the Sdk, .Net developers are able to add a standardized OPC.UA client access to their developed applications with a few lines of code, thereby accessing existing OPC UA server instances.

During the development of the toolkit, the focus has been placed on the possibility of rapid learning and using. For this reason, simple pre-configured commands have been provided for most functionalities.

The addressing of the OPC nodes can also be carried out as a string in the toolkit via the browse name. The complex finding of the NodeIDs leads the PLCcom.Opc.Ua.Sdk for you in the background, e.g. "Objects.Server.Data.Static.Scalar.Int64Value"

Here are some examples to illustrate the simple implementation of OPC UA functions.

Example Queries of existing endpoints of a server:

```
EndpointDescriptionCollection Endpoints =
          UaClient.GetEndpoints(new Uri("opc.tcp://localhost:4048"));
```

Example read a variable:

```
var value = client.ReadValue("Objects.Server.Data.Static.Scalar.Int64Value");
```

Example monitor a variable:

```
{
//create a new subscription
Subscription subscription = new Subscription();

//add new subscription to client
client.AddSubscription(subscription);

//Create a monitoring item and add to the subscription
string browsePath = "Objects.Server.Data.Dynamic.Scalar.Int64Value";
NodeId nodeId = client.GetNodeIdByPath(browsePath);
MonitoredItem monitoredItem = new MonitoredItem(subscription.DefaultItem)
          {
           StartNodeId = nodeId,
           DisplayName = nodeId.ToString()
          };

//register monitoring event
monitoredItem.Notification += Client_MonitorNotification;
//add Item to subscription
subscription.AddItem(monitoredItem);

//apply changes
subscription.ApplyChanges();
}
private void Client_MonitorNotification(MonitoredItem monitoredItem, MonitoredItemNotification
EventArgs e)
{
 MonitoredItemNotification notification = e.NotificationValue as MonitoredItemNotification;
 Console.WriteLine(monitoredItem.StartNodeId.Identifier + " Value: " + notification.Value +
 " Status: " + notification.Value.StatusCode.ToString());
}
```

Further advantages:

• Easy to use, many functions can be called by a single line of code
• At default, **automatic Connect, Reconnect, and Disconnect functionality**, the connection state does not need to be monitored by the developer
• The server state is tracked by **active keep-alive monitoring**
• **Extensive tutorials** for a quick introduction to the .Net languages C # and Visual Basic in the delivery package
• For the fast familiarization and tests, the example OPC server and OPC clients are installed

System requirements for creating applications with PLCcom.Opc.Ua.Sdk?

To create applications with the toolkit, advanced programming skills in a .Net programming language are advantageously required with C # or Visual Basic.

The following system components are also required for the operation of the PLCcom.Opc.Ua.Sdk:

- Microsoft .net Framework 4.7.2 or higher
- Microsoft .net core 3.1 or higher
- A development environment that is compatible with .net Standard 2.1 (e.g. Xamarin)

To execute the included examples you need to have

- Visual Studio 2017 or higher
- A code editor of a development environment that is compatible with .net Standard 2.1 (e.g. Xamarin)

How do I submit the licensing information?

The PLCcom.Opc.Ua.Sdk must be enabled by entering license information. This license information has been sent to you either after purchase or by sending a 30 day demo key.

The licensing information is passed during the creation of a client instance.

```
UaClient client = new UaClient(<Enter your UserName here>, <Enter your Serial here>);
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

How to use the Discovery functionality?

The communication between the UA client and the UA server is carried out via so-called endpoints, which are made available by the respective OPC UA server.

To create a client-side connection, either the endpoint information of the OPC UA server must be known, or this information can be determined using the discovery functions of the PLCcom.Opc.Ua.Sdk.
For this, the URL of the opc server must be known.

```
//C#
string url = "opc.tcp://localhost:50520/UA/DataAccessServer";

//get all servers
ApplicationDescriptionCollection servers = UaClient.FindServers(new Uri(url));
```

```
'Visual Basic'
Dim url As String = "opc.tcp://localhost:50520/UA/DataAccessServer"

'get all servers'
Dim servers As ApplicationDescriptionCollection = UaClient.FindServers(New Uri(url))
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

How to create a new client instance?

To create a new client instance, a session configuration has to created and parameterized first. For a simple session configuration the transfer of the server endpoint is sufficient.

```
//C#
//get the endpoints
EndpointDescriptionCollection ep = UaClient.GetEndpoints(new Uri("opc.tcp://localhost:4048"));

//create a a SessionConfiguration with the selected endpoint and application name
SessionConfiguration sessionConfiguration = SessionConfiguration.Build("Appname", ep[0]);
```

```
'Visual Basic'
'get the endpoints'
Dim ep As EndpointDescriptionCollection = UaClient.GetEndpoints(New Uri("opc.tcp://localhost:4
048"))

'create a a SessionConfiguration with the selected endpoint and application name'
Dim sessionConfiguration As SessionConfiguration = SessionConfiguration.Build("Appname",ep(0))
```

The second step is to create the client instance and pass the session configuration object:

```
//C#
UaClient client = new UaClient("<Enter your UserName here>",
                               "<Enter your Serial here>",
                               sessionConfiguration);
```

```
'Visual Basic'
Dim client As UaClient = New UaClient("<Enter your UserName here>",
                                      "<Enter your Serial here>",
                                      sessionConfiguration)
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Events can be registered to monitor the session:

```
//C#
client.ServerConnectionLost += Client_ServerConnectionLost;
client.ServerConnected += Client_ServerConnected;
client.KeepAlive += Client_KeepAlive;

private void Client_ServerConnected(object sender, EventArgs e)
{
   Console.WriteLine(DateTime.Now.ToLocalTime() + " Session connected");
}

private void Client_ServerConnectionLost(object sender, EventArgs e)
{
   Console.WriteLine(DateTime.Now.ToLocalTime() + " Session connection lost");
}

void Client_KeepAlive(Session session, KeepAliveEventArgs e)
{
   //catch the keepalive event
}
```

```
'Visual Basic
AddHandler client.ServerConnectionLost, AddressOf Client_ServerConnectionLost
AddHandler client.ServerConnected, AddressOf Client_ServerConnected
AddHandler client.KeepAlive, AddressOf Client_KeepAlive

Private Sub Client_ServerConnected(sender As Object, e As EventArgs)
    Console.WriteLine(DateTime.Now.ToLocalTime() & " Session connected")
End Sub

Private Sub Client_ServerConnectionLost(sender As Object, e As EventArgs)
    Console.WriteLine(DateTime.Now.ToLocalTime() & " Session connection lost")
End Sub

Private Sub Client_KeepAlive(session As Session, e As KeepAliveEventArgs)
    'catch the keepalive event
End Sub
```

Now, you can e.g. read data, write data, and monitor data or the browse the server via the client instance.

In the default setting, the client instance connects and disconnects itself.

Browse nodes

Browse with node path

The PLCcom.Opc.Ua.Sdk has also been used to simplify the browsing of nodes. For this purpose, the browse command is provided and executed by a configured client instance.

The commands

- GetReferenceDescriptionByNodeId,
- GetReferenceDescriptionByPath and
- GetNodeIdByPath

are used to convert a "browse path" to a NodeID or a ReferenceDescription, or to convert a NodeId to a ReferenceDescription.

This example browses the node "Objects.Server" forward. First create a BrowseDescription object:

```
//C#
//Set start NodeId by path
NodeId sourceNode = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar");

// find all of the components of the node.
BrowseDescription nodeToBrowse1 = new BrowseDescription();

nodeToBrowse1.NodeId = sourceNode;
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates;
nodeToBrowse1.IncludeSubtypes = true;
nodeToBrowse1.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse1.ResultMask = (uint)BrowseResultMask.All;

// find all nodes organized by the node.
BrowseDescription nodeToBrowse2 = new BrowseDescription();

nodeToBrowse2.NodeId = sourceNode;
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes;
nodeToBrowse2.IncludeSubtypes = true;
nodeToBrowse2.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse2.ResultMask = (uint)BrowseResultMask.All;

BrowseDescriptionCollection nodesToBrowse = new BrowseDescriptionCollection();
nodesToBrowse.Add(nodeToBrowse1);
nodesToBrowse.Add(nodeToBrowse2);
```

```
'Visual basic'
'Set start NodeId by path'
Dim sourceNode As NodeId = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar")

'Set start NodeId by path'
'find all of the components of the node. '
Dim nodeToBrowse1 As BrowseDescription = New BrowseDescription()
nodeToBrowse1.NodeId = sourceNode
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates
nodeToBrowse1.IncludeSubtypes = True
nodeToBrowse1.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse1.ResultMask = CUInt(BrowseResultMask.All)

' find all nodes organized by the node. '
Dim nodeToBrowse2 As BrowseDescription = New BrowseDescription()
nodeToBrowse2.NodeId = sourceNode
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

```
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes
nodeToBrowse2.IncludeSubtypes = True
nodeToBrowse2.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse2.ResultMask = CUInt(BrowseResultMask.All)
Dim nodesToBrowse As BrowseDescriptionCollection = New BrowseDescriptionCollection()
nodesToBrowse.Add(nodeToBrowse1)
nodesToBrowse.Add(nodeToBrowse2)
```

Now you can browse the "Objects.Server" node and get back a ReferenceDescriptionCollection object with the result of the operation:

```
//C#
//now, browse the node
ReferenceDescriptionCollection rdc = client.BrowseFull(nodesToBrowse);

if (rdc.Count > 0)
{
    foreach (ReferenceDescription rd in rdc)
    {
        Console.WriteLine("Child NodeID found => "+ rd.NodeId+
            " NodeClass => " + rd.NodeClass.ToString()  +
            " BrowseName => " + rd.BrowseName.ToString() +
            " DisplayName => " + rd.DisplayName.ToString());
    }
}
```

```
'Visual Basic'
'now, browse the node'
Dim rdc As ReferenceDescriptionCollection = client.BrowseFull(nodesToBrowse)

If rdc.Count > 0 Then

    For Each rd As ReferenceDescription In rdc
        Console.WriteLine("Child NodeID found => " & rd.NodeId.ToString() &
                  " NodeClass => " & rd.NodeClass.ToString() &
                  " BrowseName => " & rd.BrowseName.ToString() &
                  " DisplayName => " & rd.DisplayName.ToString())
    Next
End If
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Browse with NodeId

In this example, the object is browsed forward through the Objects node. To do this, we pass the NodeId `ObjectIds.ObjectsFolder` to the BrowseDescription:

```
//C#
//Set start
NodeId sourceNode = ObjectIds.ObjectsFolder

// find all of the components of the node.
BrowseDescription nodeToBrowse1 = new BrowseDescription();

nodeToBrowse1.NodeId = sourceNode;
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates;
nodeToBrowse1.IncludeSubtypes = true;
nodeToBrowse1.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse1.ResultMask = (uint)BrowseResultMask.All;

// find all nodes organized by the node.
BrowseDescription nodeToBrowse2 = new BrowseDescription();

nodeToBrowse2.NodeId = sourceNode;
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes;
nodeToBrowse2.IncludeSubtypes = true;
nodeToBrowse2.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse2.ResultMask = (uint)BrowseResultMask.All;

BrowseDescriptionCollection nodesToBrowse = new BrowseDescriptionCollection();
nodesToBrowse.Add(nodeToBrowse1);
nodesToBrowse.Add(nodeToBrowse2);
```

```
'Visual basic'
'Set start NodeId'
Dim sourceNode As NodeId = ObjectIds.ObjectsFolder

'Set start NodeId by path'
'find all of the components of the node. '
Dim nodeToBrowse1 As BrowseDescription = New BrowseDescription()
nodeToBrowse1.NodeId = sourceNode
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates
nodeToBrowse1.IncludeSubtypes = True
nodeToBrowse1.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse1.ResultMask = CUInt(BrowseResultMask.All)

' find all nodes organized by the node. '
Dim nodeToBrowse2 As BrowseDescription = New BrowseDescription()
nodeToBrowse2.NodeId = sourceNode
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes
nodeToBrowse2.IncludeSubtypes = True
nodeToBrowse2.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse2.ResultMask = CUInt(BrowseResultMask.All)
Dim nodesToBrowse As BrowseDescriptionCollection = New BrowseDescriptionCollection()
nodesToBrowse.Add(nodeToBrowse1)
nodesToBrowse.Add(nodeToBrowse2)
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Nun können Sie den Knoten "Objects" browsen und erhalten ein ReferenceDescriptionCollection-Objekt mit dem Ergebnis der Operation zurück:

```
//C#
//now, browse the node
ReferenceDescriptionCollection rdc = client.BrowseFull(nodesToBrowse);

if (rdc.Count > 0)
{
    foreach (ReferenceDescription rd in rdc)
    {
        Console.WriteLine("Child NodeID found => "+ rd.NodeId+
            " NodeClass => " + rd.NodeClass.ToString()  +
            " BrowseName => " + rd.BrowseName.ToString() +
            " DisplayName => " + rd.DisplayName.ToString());
    }
}
```

```
'Visual Basic'
'now, browse the node'
Dim rdc As ReferenceDescriptionCollection = client.BrowseFull(nodesToBrowse)

If rdc.Count > 0 Then

    For Each rd As ReferenceDescription In rdc
        Console.WriteLine("Child NodeID found => " & rd.NodeId.ToString() &
                  " NodeClass => " & rd.NodeClass.ToString() &
                  " BrowseName => " & rd.BrowseName.ToString() &
                  " DisplayName => " & rd.DisplayName.ToString())
    Next
End If
```

Read and write data

The usual reading and writing of values can be performed with a single line of code. A prerequisite is a configured client instance (see above).

By default, the PLCcom.Opc.Ua.Sdk automatically connects to the server and also monitors its connection.

As you already know, you can also simply address the node with the complete browse name during monitoring, and the conversion to the NodeID is performed automatically in the background in the toolkit.

The following examples illustrates how to read and write data from/to an Opc Ua server.

How to synchronous reading of data from node

The following example illustrates the synchronous reading of nodes of an OPC UA server. You can use the AttributeId parameter to specify which attribute you want to read:

```
//C#
ReadValueId nodeToRead = new ReadValueId();
nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value");
nodeToRead.AttributeId = Attributes.Value;

//reading the node synchronous
DataValue readresults = client.Read(nodeToRead);
```

```
'Visual Basic'
ReadValueId nodeToRead = New ReadValueId()
nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value")
nodeToRead.AttributeId = Attributes.Value

'reading the nodes synchronous'
Dim readresults As DataValueCollection = client.Read(nodeToRead)
```

How to synchronous writing of data in nodes

The following example demonstrates the synchronous writing to nodes of an OPC UA server. With the AttributeId parameter you can specify which attribute you want to write:

```
//C#
//write value 123 to OPC UA Node and receive the status code

WriteValue nodeToWrite = new WriteValue();
nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value");
nodeToWrite.AttributeId = Attributes.Value;
nodeToWrite.Value = new DataValue((Int64)(123));

//writing the node synchronous
StatusCode writeResult = client.Write(nodeToWrite);
```

```
'Visual Basic'
'write value 123 to OPC UA Node and receive the status code'
Dim nodeToWrite As New WriteValue()
nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value")
nodeToWrite.AttributeId = Attributes.Value
nodeToWrite.Value = New DataValue(CLng(123))

'writing the nodes synchronous'
Dim writeResult As StatusCode = client.Write(nodeToWrite)
```

How to asynchronous reading of data from node

The following example demonstrates the asynchronous reading of nodes of an OPC UA server.

With the BeginRead command you initiate the asynchronous reading process and also transfer the corresponding method for data evaluation. You can use the AttributeId parameter to specify which attribute you want to read:

```
//C#
private void ReadData()
{
   ReadValueId nodeToRead = new ReadValueId();
   nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value");
   nodeToRead.AttributeId = Attributes.Value;

   //reading the nodes asynchronous
   client.BeginRead(nodeToRead, GetReadAsyncResult, nodesToRead);

}

private void GetReadAsyncResult(IAsyncResult res)
{
    DataValueCollection values = null;
    DiagnosticInfoCollection diagnostic = null;
    ResponseHeader readresults = client.EndRead(res, out values, out diagnostic);

    ReadValueIdCollection req = res.AsyncState as ReadValueIdCollection;

    if (req != null)
    {
        for (int i = 0; i < values.Count; i++)
        {
            Console.WriteLine("asynchronous read result " +
                req[i].NodeId.ToString() + " Value => " +
                values[i].ToString() + " StatusCode => " +
                values[i].StatusCode.ToString());
        }
    }
}
```

```
'Visual Basic'
Private Sub ReadData ()

   ReadValueId nodeToRead = New ReadValueId()
   nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value")
   nodeToRead.AttributeId = Attributes.Value

         'reading the nodes asynchronous'
   client.BeginRead(nodesToRead, New AsyncCallback(AddressOf GetReadAsyncResult), nodesToRead)

End Sub

Private Sub GetReadAsyncResult(ByVal res As IAsyncResult)
    Dim values As DataValueCollection = Nothing
    Dim diagnostic As DiagnosticInfoCollection = Nothing
    Dim readresults As ResponseHeader = client.EndRead(res, values, diagnostic)
    Dim req As ReadValueIdCollection = TryCast(res.AsyncState, ReadValueIdCollection)

    If req IsNot Nothing Then

        For i As Integer = 0 To values.Count - 1
            Console.WriteLine("asynchronous read result " & req(i).NodeId.ToString() &
                              " Value => " & values(i).ToString() &
                              "  StatusCode => " & values(i).StatusCode.ToString())
        Next
    End If
End Sub
```

How to asynchronous writing of data in node

The following example demonstrates asynchronous writing to nodes of an OPC UA server. With the AttributeId parameter you can specify which attribute you want to write:

With the BeginWrite command you initiate the asynchronous write process and also transfer the corresponding method for data evaluation. With the AttributeId parameter you can specify which attribute you want to write:

```
//C#
private void WriteData()
{
   //write value 123 to OPC UA Node and receive the status code
   WriteValue nodeToWrite = new WriteValue();
   nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value");
   nodeToWrite.AttributeId = Attributes.Value;
   nodeToWrite.Value = new DataValue((Int64)(123));

   //writing the nodes asynchronous
   client.BeginWrite(nodeToWrite, GetWriteAsyncResult, nodesToWrite);
}

private void GetWriteAsyncResult(IAsyncResult res)
{
    StatusCodeCollection statuscodes = null;
    DiagnosticInfoCollection diagnostic = null;
    ResponseHeader readresults = client.EndWrite(res, out statuscodes, out diagnostic);

    WriteValueCollection req = res.AsyncState as WriteValueCollection;
    if (req != null)
    {
        for (int i = 0; i < statuscodes.Count; i++)
        {
            Console.WriteLine("asynchronous write result " + req[i].NodeId.ToString() +
                " Value => " + req[i].Value.ToString() +
                " StatusCode => " + statuscodes[i].ToString());
        }
    }
}
```

```
'Visual Basic'
Private Sub WriteData()

 'write value 123 to OPC UA Node and receive the status code'
 Dim nodeToWrite As New WriteValue()
 nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value")
 nodeToWrite.AttributeId = Attributes.Value
 nodeToWrite.Value = New DataValue(CLng(123))

  'writing the nodes asynchronous'
 client.BeginWrite(nodeToWrite, New AsyncCallback(AddressOf GetWriteAsyncResult), nodeToWrite)
End Sub

Private Sub GetWriteAsyncResult(ByVal res As IAsyncResult)
    Dim statuscodes As StatusCodeCollection = Nothing
    Dim diagnostic As DiagnosticInfoCollection = Nothing
    Dim readresults As ResponseHeader = client.EndWrite(res, statuscodes, diagnostic)
    Dim req As WriteValueCollection = TryCast(res.AsyncState, WriteValueCollection)

    If req IsNot Nothing Then
        For i As Integer = 0 To statuscodes.Count - 1
            Console.WriteLine("asynchronous write result " & req(i).NodeId.ToString() &
                              " Value => " & req(i).Value.ToString() &
                              " StatusCode => " & statuscodes(i).ToString())
        Next
    End If
End Sub
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Available Attributes read or write

In PLCcom.Opc.Ua.Sdk the possible attributes for reading or writing are preconfigured in the class PLCcom.Opc.Ua.Attributes:

The class contains the following objects:

```
namespace PLCcom.Opc.Ua
{
    public static partial class Attributes
    {
        // The canonical identifier for the node.
        public const uint NodeId = 1;

        // The class of the node.
        public const uint NodeClass = 2;

        // A non-localized, human readable name for the node.
        public const uint BrowseName = 3;

        // A localized, human readable name for the node.
        public const uint DisplayName = 4;

        // A localized description for the node.
        public const uint Description = 5;

        // Indicates which attributes are writeable.
        public const uint WriteMask = 6;

        // Indicates which attributes are writeable by the current user.
        public const uint UserWriteMask = 7;

        // Indicates that a type node may not be instantiated.
        public const uint IsAbstract = 8;

        // Indicates that forward and inverse references have the same meaning.
        public const uint Symmetric = 9;

        // The browse name for an inverse reference.
        public const uint InverseName = 10;

        // Indicates that following forward references within a view will not cause a loop.
        public const uint ContainsNoLoops = 11;

        // Indicates that the node can be used to subscribe to events.
        public const uint EventNotifier = 12;

        // The value of a variable.
        public const uint Value = 13;

        // The node id of the data type for the variable value.
        public const uint DataType = 14;

        // The number of dimensions in the value.
        public const uint ValueRank = 15;

        // The length for each dimension of an array value.
        public const uint ArrayDimensions = 16;

        // How a variable may be accessed.
        public const uint AccessLevel = 17;

        // How a variable may be accessed after taking the user's access rights into account.
        public const uint UserAccessLevel = 18;

        // Specifies (in milliseconds) how fast the server can reasonably sample the value for
 changes.
        public const uint MinimumSamplingInterval = 19;

       // Specifies whether the server is actively collecting historical data for the variable
```

```
        public const uint Historizing = 20;

        // Whether the method can be called.
        public const uint Executable = 21;

        // Whether the method can be called by the current user.
        public const uint UserExecutable = 22;

        // Provides the metadata and encoding information for custom DataTypes.
        public const uint DataTypeDefinition = 23;

        // The permissions for the node granted to roles.
        public const uint RolePermissions = 24;

        /// The subset of permissions available for the roles available to the current session
        public const uint UserRolePermissions = 25;

        // The access restrictions assigned to the node.
        public const uint AccessRestrictions = 26;

        // How a variable may be accessed.
        public const uint AccessLevelEx = 27;
    }
}
```

Monitoring knots

The monitoring functions of the PLCcom.Opc.Ua.Sdk have also been made extremely convenient and easy for the developer.

A prerequisite is a configured client instance (see above).

By default, the PLCcom.Opc.Ua.Sdk automatically connects to the server and also monitors its connection. You will be notified of an upcoming connection via an event.

As you already know, you can also address the node with the complete browse name during monitoring, and the conversion to the NodeID is performed automatically in the background in the PLCcom.Opc.Ua.Sdk.

Create subscription instances

In order to be able to generate a new subscription, the corresponding constructor must be called. You can then set the desired properties and register existing events to monitor the subscription. Finally, the subscription must be transferred to the UaClient instance.

```
//C#
//create a new subscription
Subscription subscription = new Subscription();

subscription.PublishingInterval = 1000;
subscription.DisplayName = "mySubsription";

//register subscription events
subscription.StateChanged += Subscription_StateChanged;
subscription.PublishStatusChanged += Subscription_PublishStatusChanged;

//add new subscription to client
client.AddSubscription(subscription);
```

```
'Visual Basic'
'create a new subscription'
Dim subscription As Subscription = New Subscription()
subscription.PublishingInterval = 1000
subscription.PublishingEnabled = False
subscription.DisplayName = "mySubsription"

'register subscription events'
AddHandler subscription.StateChanged, AddressOf Subscription_StateChanged
AddHandler subscription.PublishStatusChanged, AddressOf Subscription_PublishStatusChanged

'add new subscription to client'
client.AddSubscription(subscription)
```

Edit subscription parameters

With the methods of Toolkit you can edit your subscriptions. The following code edit the parameter ‘publishing interval’ to 500ms and setting ‘publishing’ mode to true.

```
//C#
//enable publishing mode of subscription and set PublishingInterval
subscription.PublishingInterval = 500;
subscription.SetPublishingMode(true);
subscription.Modify();
```

```
'Visual Basic'
'enable publishing mode of subscription And set PublishingInterval'
subscription.PublishingInterval = 500
subscription.SetPublishingMode(True)
subscription.Modify()
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Management of MonitoredItems

A MonitoredItems object corresponds to a node of an Opc Ua server to be monitored. One or many MonitoredItems are logically bundled in a subscription. The Subscription object can be used to create, modify or delete MonitoredItems objects.

Create a subscription object. Then create one or more MonitoredItems objects and add these objects to your subscription.

You link the "Notification Event" with a "Notification Method" (see example). Via the "Notification-Event" the value changes of the MonitoredItem object are now made available to you.

Finally, call the method subscription.ApplyChanges ().

```
//C#
//create a new subscription
Subscription subscription = new Subscription();

subscription.PublishingInterval = 1000;
subscription.DisplayName = "mySubsription";

//register subscription events
subscription.StateChanged += Subscription_StateChanged;
subscription.PublishStatusChanged += Subscription_PublishStatusChanged;

//add new subscription to client
client.AddSubscription(subscription);

//Create a monitoring item and add to the subscription
NodeId nodeId = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar.Int64Value");
MonitoredItem monitoredItem = new MonitoredItem(subscription.DefaultItem)
                              {
                                  StartNodeId = nodeId,
                                  SamplingInterval = 500,
                                  QueueSize = UInt32.MaxValue,
                                  DisplayName = nodeId.ToString()
                              };

//register monitoring event
monitoredItem.Notification += Client_MonitorNotification;
//add Item to subscription
subscription.AddItem(monitoredItem);

//apply changes
subscription.ApplyChanges();

...

private void Client_MonitorNotification(MonitoredItem monitoredItem, MonitoredItemNotification
EventArgs e)
{
    MonitoredItemNotification notification = e.NotificationValue as MonitoredItemNotification;
    Console.WriteLine(monitoredItem.StartNodeId.Identifier +
        " Value: " + notification.Value +
        " Status: " + notification.Value.StatusCode.ToString());
}
```

```
'Visual Basic'
'create a new subscription'
Dim subscription As Subscription = New Subscription()
subscription.PublishingInterval = 1000
subscription.PublishingEnabled = False
subscription.DisplayName = "mySubsription"

'register subscription events'
AddHandler subscription.StateChanged, AddressOf Subscription_StateChanged
AddHandler subscription.PublishStatusChanged, AddressOf Subscription_PublishStatusChanged

'add new subscription to client'
client.AddSubscription(subscription)

'Create a monitoring item and add to the subscription'
Dim nodeId As NodeId = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar.Int64Value")
Dim monitoredItem As MonitoredItem = New MonitoredItem(subscription.DefaultItem) With {
                                    .StartNodeId = nodeId,
                                    .SamplingInterval = 500,
                                    .QueueSize = UInteger.MaxValue,
                                    .DisplayName = nodeId.ToString()
                                }

'register monitoring event'
AddHandler monitoredItem.Notification, AddressOf Client_MonitorNotification

'add Item to subscription'
subscription.AddItem(monitoredItem)

'apply changes'
subscription.ApplyChanges()

...

Private Sub Client_MonitorNotification(ByVal monitoredItem As MonitoredItem, ByVal e As Monito
redItemNotificationEventArgs)
    Dim notification As MonitoredItemNotification = TryCast(e.NotificationValue, MonitoredItem
Notification)
    Console.WriteLine(monitoredItem.StartNodeId.Identifier &
                      " Value: " & notification.Value &
                      " Status: " & notification.Value.StatusCode.ToString())
End Sub
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Call methods

Use a call-method to call server methods from the client.

Attention:
Each Opc Ua server only supports the call calls specifically defined for it.
See the documentation of your Opc-Server-manufacturer.

In the delivery package you will find several code examples within the tutorials for the execution of call calls with the transfer of simple flat data and even complex structures.

Following, the definition of data structure to passing with method call: //C#

```
 /*
let´s starting a method call, step by step
In this simple case, we pass a simple structure named as 'DataStructure_One" constructed as
follows:

structure DataStructure_One =
{
  int myIntValue1,
  string myStringValue2,
  int myIntValue3,
  int myIntValue4,
  string myStringValue5
}

Object to which the method should be applied is named as "myObjectNode"
Method is named as "myMethodNode"
*/

int myIntValue1 = 1;
string myStringValue2 = "testvalue";
int myIntValue3 = 3333;
int myIntValue4 = 4444;
string myStringValue5 = "a_string_value";
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Next step, preparation and execution of the call method:

```
//C#
//Call Methods

//create a Encoder instance
BinaryEncoder encoder = new BinaryEncoder(client.getMessageContext());

//put objects to encoder with given order
encoder.WriteInt32("", myIntValue1);
encoder.WriteString("", myStringValue2);
encoder.WriteInt32("", myIntValue3);
encoder.WriteInt32("", myIntValue4);

//read byte array from encoder
byte[] argumentByteArray = new byte[encoder.BaseStream.Length];
encoder.BaseStream.Position = 0;
encoder.BaseStream.Read(argumentByteArray, 0, argumentByteArray.Length);

//create an extension object and pass arguments to ExtensionObject.Body
ExtensionObject extensionObjectWithInputArguments = new ExtensionObject();
extensionObjectWithInputArguments.Body = argumentByteArray;

//set type of structure, create a new ExpandedNodeId by name and namespace
extensionObjectWithInputArguments.TypeId = new ExpandedNodeId("DataStructure_One", 3);

//create your InputArguments with extensionObject
VariantCollection inputArguments = new VariantCollection();
inputArguments.Add(new Variant(extensionObjectWithInputArguments));

//create a new NodeId for the Object to which the method should be applied by name and
//namespace
NodeId objectNode = new NodeId("myObjectNode", 3);

//create a new NodeId for the Method by name and namespace
NodeId methodNode = new NodeId("myMethodNode", 3);

//create a CallMethodRequest instance and pass your arguments
CallMethodRequest request = new CallMethodRequest();
request.ObjectId = objectNode;
request.MethodId = methodNode;
request.InputArguments = inputArguments;

//call your method
CallMethodResult result = client.Call(request);

//finaly evaluate your results,
if (StatusCode.IsGood(result.StatusCode))
{
    foreach (Variant outputArgument in result.OutputArguments)
    {
            if (outputArgument != Variant.Null)
                Console.WriteLine("output argument: " + outputArgument.ToString());
    }
}
```

Following, the same example in visual basic programmer language:

```
'Visual Basic'
 'Structure DataStructure_One = '
'{'
'    int myIntValue1, '
'    string myStringValue2, '
'    int myIntValue3, '
'    int myIntValue4, '
'    string myStringValue5'
'}'

'Object to which the method should be applied Is named as "myObjectNode"'
'Method Is named as "myMethodNode"'


Dim myIntValue1 As Integer = 1
Dim myStringValue2 As String = "testvalue"
Dim myIntValue3 As Integer = 3333
Dim myIntValue4 As Integer = 4444
Dim myStringValue5 As String = "a_string_value"
```

Next step, preparation and execution of the call method:

```
'Visual Basic'
'Call Methods'
'create a Encoder instance'
Dim encoder As BinaryEncoder = New BinaryEncoder(client.getMessageContext)

'put objects to encoder with given order'
encoder.WriteInt32("", myIntValue1)
encoder.WriteString("", myStringValue2)
encoder.WriteInt32("", myIntValue3)
encoder.WriteInt32("", myIntValue4)
encoder.WriteString("", myStringValue5)

'read byte array from encoder'
Dim argumentByteArray() As Byte = New Byte((encoder.BaseStream.Length) - 1) {}
encoder.BaseStream.Position = 0
encoder.BaseStream.Read(argumentByteArray, 0, argumentByteArray.Length)

'create an extension object and pass arguments to ExtensionObject.Body'
Dim extensionObjectWithInputArguments As ExtensionObject = New ExtensionObject()
extensionObjectWithInputArguments.Body = argumentByteArray

'set type of structure, create a new ExpandedNodeId by name and namespace'
extensionObjectWithInputArguments.TypeId = New ExpandedNodeId( _
New NodeId("DataStructure_One", 3))

'create your InputArguments with extensionObject'
Dim inputArguments As VariantCollection = New VariantCollection
inputArguments.Add(New PLCcom.Opc.Ua.Variant(extensionObjectWithInputArguments))

'create a new NodeId for the Object to which the method should be applied by name and '
'namespace'
Dim objectNode As NodeId = New NodeId("myObjectNode", 3)

'create a new NodeId for the Method by name and namespace'
Dim methodNode As NodeId = New NodeId("myMethodNode", 3)

'create a CallMethodRequest instance and pass your arguments'
Dim request As CallMethodRequest = New CallMethodRequest
request.ObjectId = objectNode
request.MethodId = methodNode
request.InputArguments = inputArguments

'call your method'
Dim result As CallMethodResult = client.Call(request)

'finaly evaluate your results'
If StatusCode.IsGood(result.StatusCode) Then

  For Each outputArgument As PLCcom.Opc.Ua.Variant In result.OutputArguments
      If (outputArgument <> PLCcom.Opc.Ua.Variant.Null) Then
          Console.WriteLine(("output argument: " + outputArgument.ToString))
      End If

  Next

Else
  Console.WriteLine(("Method call failed " + result.StatusCode.ToString))
End If
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Certificate management

For secure communication between the OPC UA server and the OPC UA client, it is possible to exchange certificates. For this purpose, the PLCcom.OPC.UA.Toolkit provides a default certificate port.

The path to the certificate store is in the session configuration and can be viewed or modified using the "CertificateStorePath" property. By default, the certificate store is placed under the path% ProgramData% \ <application name> \ CertificateStores.

The validation of a certificate can also be intercepted and processed via the "CertificateValidation" event of the UaClient instance. The Sdk accepts invalid server certificates in default settings. If this is not desired, the "AutoAcceptUntrustedCertificates" property can be set to "false" within the session configuration.

Special attention:

The attempt to connect a UaClient instance automatically checks the certificate store for a suitable certificate if no certificate exists, it creates automatically a self signed certificate. This automatically created certificate does not contain information about a trusted certification authority. If the use of this certificate is nevertheless desired, it must be installed under trustworthy certification authorities.

If a secure connection with certificate exchange is desired, the server certificate must be installed in the client certificate store and the client certificate in the server certificate store. This certificate exchange must be carried out manually via the file system.

```
//C#
//catching certificate validate event
client.CertificateValidation += clie_CertificateValidation;

...

void clie_CertificateValidation(CertificateValidator sender,CertificateValidationEventArgs e)
{
    //external certificate validation
    if (ServiceResult.IsGood(e.Error))
        e.Accept = true;
    else if ((e.Error.StatusCode.Code == StatusCodes.BadCertificateUntrusted ||
        e.Error.StatusCode.Code == StatusCodes.BadCertificateChainIncomplete) &&
        autoAcceptUntrustedCertificates)
        e.Accept = true;
    else
    {
      throw new Exception(string.Format("Failed to validate certificate error code {0}: {1}",
                          e.Error.Code,
                          e.Error.AdditionalInfo));
    }
}
```

```
'Visual Basic'
 ' catching certificate validate event'  AddHandler client.CertificateValidation, AddressOf
client_CertificateValidation
…
Private Sub client_CertificateValidation(ByVal sender As CertificateValidator, ByVal e As Cert
ificateValidationEventArgs)
    'external certificate validation'
    If ServiceResult.IsGood(e.Error) Then
        e.Accept = True
    ElseIf (e.Error.StatusCode.Code = StatusCodes.BadCertificateUntrusted OrElse
        e.Error.StatusCode.Code = StatusCodes.BadCertificateChainIncomplete) AndAlso
        autoAcceptUntrustedCertificates Then
        e.Accept = True
    Else
        Throw New Exception("Failed to validate certificate with error code " &
                            e.Error.Code & " " & e.Error.AdditionalInfo)
    End If
End Sub
```

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Any questions?

Call our free hotline +49 421 98970330 or write an email to support@indi-systems.de.

We will process your request promptly or respond to you directly.

Indi.Systems GmbH
Flughafenallee 3
D-28199 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39