# Beckhoff Training Series

## Module 4: Introduction to ADS-programming

Version: 19052012

This manual assumes that TwinCAT is already installed as well Visual Studio. This manual also assumes that the reader has a basic level of knowledge of .NET-programming.

## Contents

For more elaborate info, please take a look into the information System: **http://infosys.beckhoff.com/**

This page kept intentionally blank

# 1. ADS concept

## 1.1. ADS Architecture

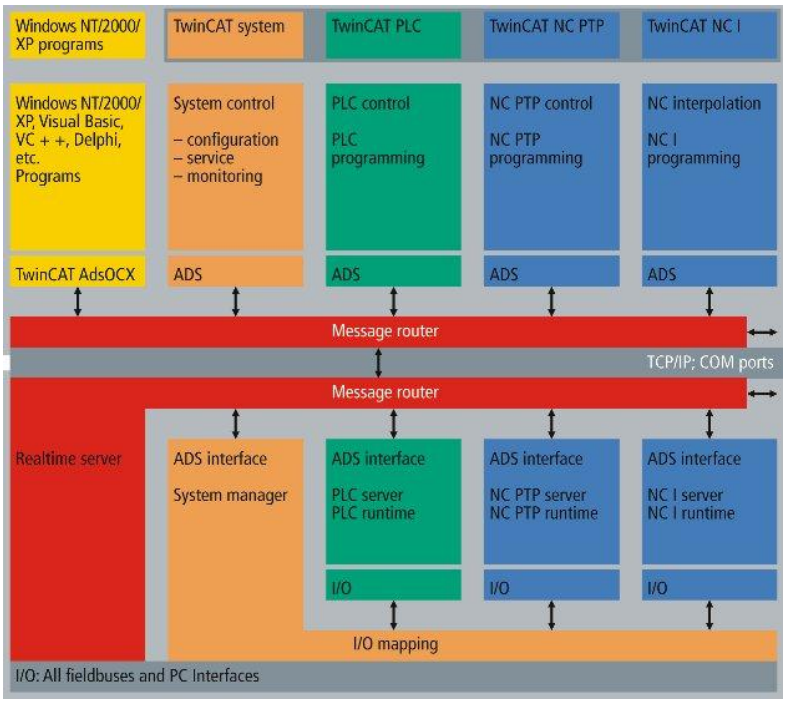
The TwinCAT system architecture allows the individual modules of the software (e.g. TwinCAT PLC, TwinCAT NC, ...) to be treated as independent devices: For every task there is a software module ("Server" or "Client"). The servers in the system are the executing working "devices" in the form of software, whose operating behaviour is exactly like that of a hardware device. For this reason we can speak of "virtual" devices implemented in the software. The "clients" are programs which request the services of the "servers", e.g. a visualisation, or even a "programming device" in the form of a program. It is thus possible for TwinCAT to grow, since there can always be new servers and clients for tasks such as camshaft controllers, oscilloscopes, PID controllers etc..

The messages between these objects are exchanged through a consistent ADS (**A**utomation **D**evice **S**pecification) interface by the "message router". This manages and distributes all the messages in the system and over the TCP/IP connections.

TwinCAT message routers exist on every TwinCAT PC and on every Beckhoff BCxxxx Bus Controller (e.g. BC3100, BC8100, ...). In this way PC systems can be linked (via TCP/IP) as CAN Bus Controllers (through serial interfaces and fieldbuses).

This allows all TwinCAT server and client programs to exchange commands and data, to send messages, transfer status information, etc..

The following diagram shows the TwinCAT device concept and seems complicated and will be discussed in later modules but just realize that the **TwinCAT system** is your computer. **Message router** talks to and through the TCP/IP port to the remote system (**ADS interface**):



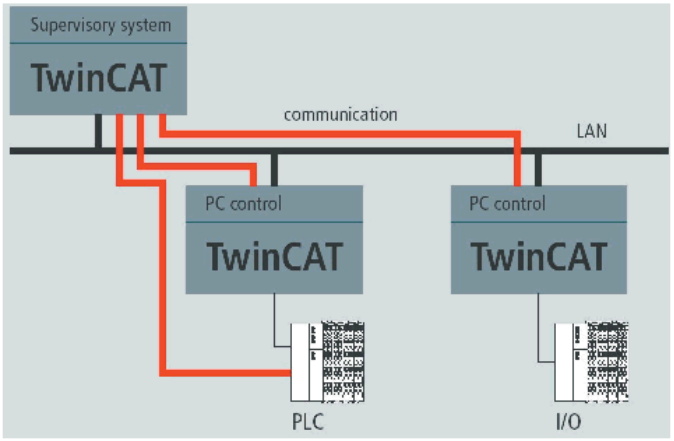
## 1.2. ADS-PORT

The ADS devices in a TwinCAT message router are uniquely identified by a number referred to as the ADS-PortNr. For ADS devices this has a fixed specification, whereas pure ADS client applications (e.g. a visualisation system) are allocated a variable ADS port number when they first access the message router.

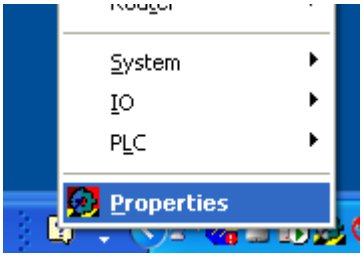
| ADS-PortNr | ADS device description | Constants port numbers |
|---|---|---|
| 100 | Logger (Only NT-Log) | AMSPORT_LOGGER |
| 110 | EventLogger | AMSPORT_EVENTLOG |
| 300 | IO | AMSPORT_R0_IO |
| 301 | Additional Task 1 | |
| 302 | Additional Task 2 | |
| | | |
| 500 | NC | AMSPORT_R0_NC |
| 800 | PLC Runtime System (Only at the buscontroller) | AMSPORT_R0_PLC |
| 801 | PLC Runtime System1 | AMSPORT_R0_PLC_RTS1 |
| 811 | PLC Runtime System2 | AMSPORT_R0_PLC_RTS2 |
| 821 | PLC Runtime System3 | AMSPORT_R0_PLC_RTS3 |
| 831 | PLC Runtime System4 | AMSPORT_R0_PLC_RTS4 |
| | | |
| 900 | CamShaft Controller | AMSPORT_R0_CAM |
| 950 | CamTool Server | AMSPORT_R0_CAMTOOL |
| 10000 | System Service | AMSPORT_R3_SYSSERV |
| 27110 | Scope server | AMSPORT_R3_SCOPESERVER |

## 1.3. ADS-AmsNetId

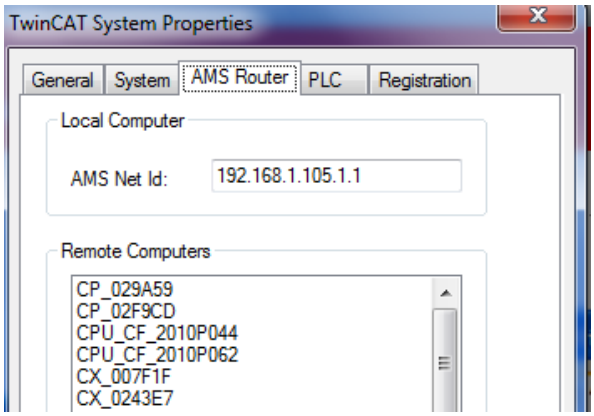
It is not only possible to exchange data between TwinCAT modules on one PC, it is even possible to do so by ADS methods between multiple TwinCAT PC's on the network.




**Configuration:**
The ADS-AmsNetId of a TwinCAT-PC can be set in the TwinCAT system service: A menu appears after the system service icon has been selected.




By selecting "Properties" you reach the "TwinCAT System Properties" dialogue.



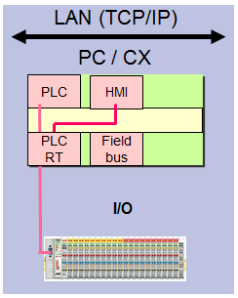

After the "AMS router" page has been selected, the desired identifier can be read or modified in the "AMS Net Id" field. By default, the TwinCAT installation constructs the AMSNetId from the TCP/IP address of the PC with an extension of ".1.1" (as if it were a "sub-net mask" for field busses, target bus controllers,..).

The AMSNetId can be chosen ‘freely’, one restriction on this matter is that all the AMSNetIds need uniquely be defined in the network.

When the services of an ADS device in the network are called on, its message router, or, more precisely, its AMSNetId, must be known. This is achieved through insertion of the target PC, as a result of which TwinCAT can establish the connection between the TCP/IP address of the target PC and the AMSNetId of the target message router address.

# 2. Setting up communication between devices

## 2.1. Setting up the route between the ADS-Devices

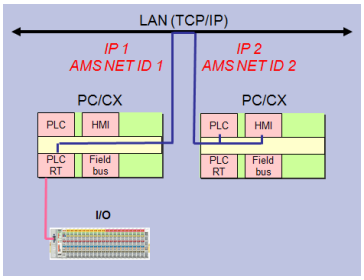



### Local connection:

In the case the HMI runs directly on the device, no specific settings need to be done.

### Remote connection:

The main element in remote connection is a system called a ***router***. As the name implies, the router, routes the messages to and from all remote devices to a single controller and manages them all.

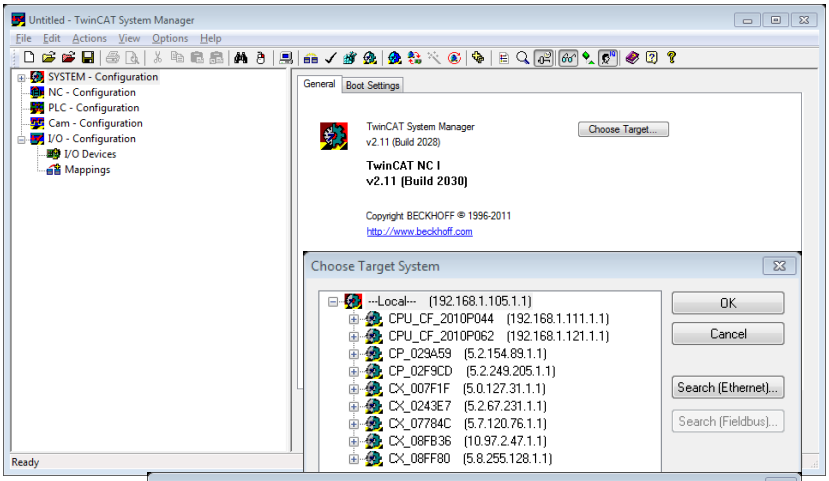



Remote connectivity involves ADS or Automation Device Specification. This is an interface that enables communication between ADS compliant devices and systems over TCP/IP.

**Establishing a Route**

The connection between a computer and a remote system is called a ***route***. To achieve a communication link between the systems, a route needs to be established.

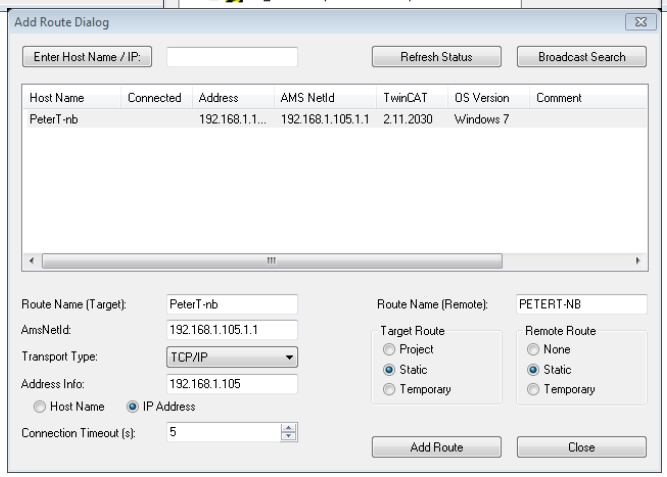
This can be done through the use of the system manager.




Choose "***System – configuration***" on the left side tree.

Click in the right side part of the screen on "***General***" –if not already selected- and click on the "***Choose Target***" Button.

The "***choose Target system***" screen will pop-up and shows all available systems.
The First time this is opened, only the local system will be displayed.




Click on the "***Search Ethernet***" button.

The "***Add route Dialog***" screen will pop-up. In this window click on the "***broadcast search***" button to begin discovering all connected systems.

The discovered systems will be displayed. On a LAN, the system will list every TwinCAT-enabled device which is in Config or Run mode.

Select the remote system you wish to add, and click the "***Add Route***" button.

In the "***login information***" screen that appears, enter the username and password for the remote system. And Click the "***OK***" button.

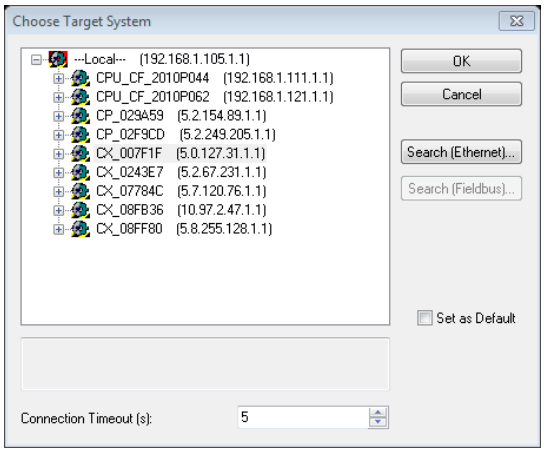
The remote system should now show an "***X***" in the connected column. This makes it easy to find connected systems when numerous systems are visible, such as when the systems are connected to the LAN.

Click the "***Close***" button.

The Remote system should now be displayed in the "***Choose target system***" screen. Select the remote system and click the "***OK***" button to return to the System manager.

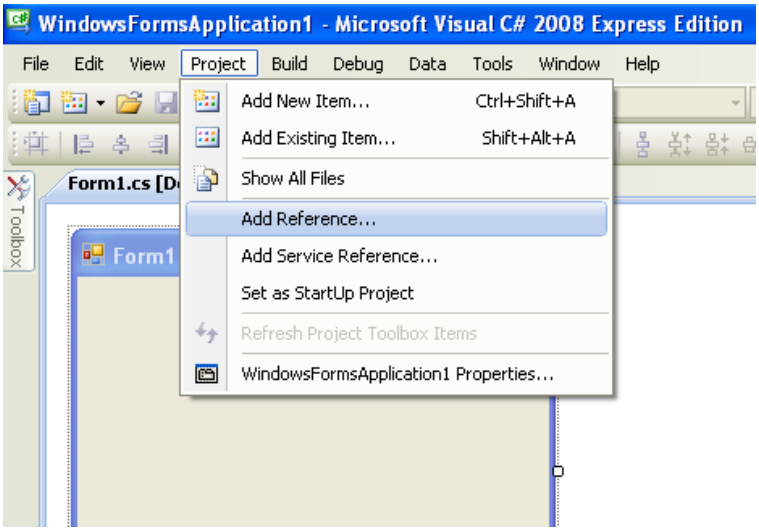
The system manager will now display extra tabs related to the remote system.

The remote system that is now connected is identified in the lower right corner in red.



## 2.2. Linking into Microsoft Visual Studio .NET

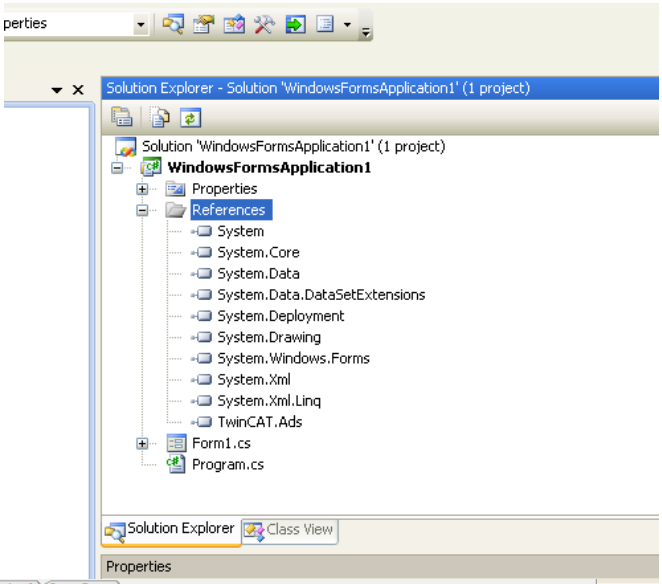
In order to select the TwinCAT.Ads class library you must choose the command *Add Reference...* under the *Project* menu. This opens the *Add Reference* dialog:




In this dialog you have to press the *Browse* button and select the file TwinCAT.Ads.dll.

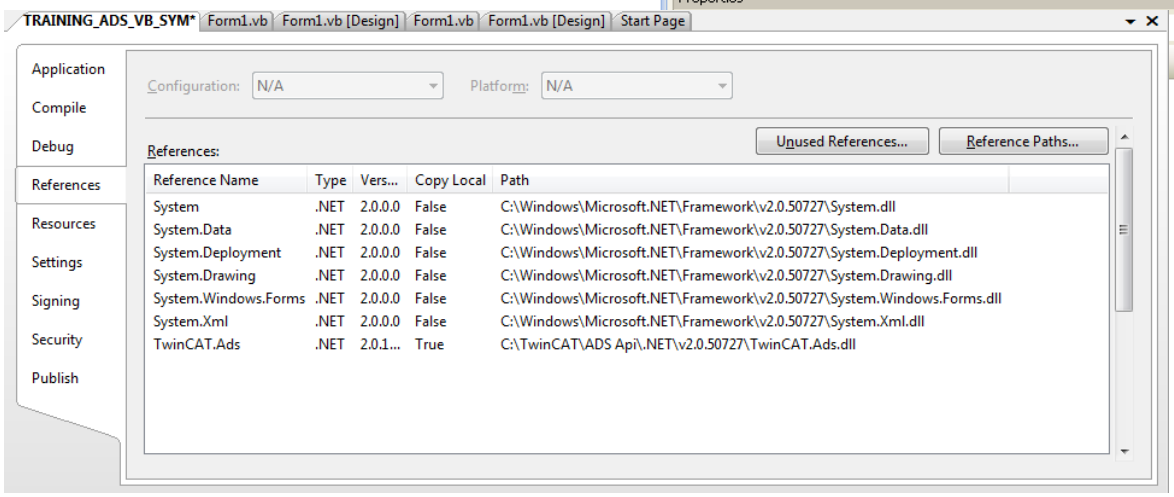
*C#:*

In the Solution Explorer, you can check if the component has been successfully added to the list of references.




*VB.NET:*

In the Solution Explorer, double click on ”MyProject”, on the next view select Tab “References” for the list of references



## 2.3. Setting up communication between “devices”

All accessible types (classes,structures ...) belong to the namespace TwinCAT.Ads.
Therefore one has to insert the “using/imports” at the beginning of the source, this enables access to the types defined in TwinCAT.Ads without including the fully qualified name of the namespace for every component. The class TcAdsClient is a wrapper for the TwinCAT.Ads class library and enables the user to communicate with an ads device.

The connection to the ADS device is established by means of the Connect method:

- public **void Connect**(**int** *srvPort*)
- public **void Connect**(**TwinCAT.Ads.AmsNetId** *netID*, **TwinCAT.Ads.AmsPort** *srvPort*)
- public **void Connect**(**TwinCAT.Ads.AmsNetId** *netID*, **int** *srvPort*)
- public **void Connect**(**byte[]** *netID*, **int** *srvPort*)
- public **void Connect**(**string** *netID*, **int** *srvPort*)

**Parameters:**
*netID*: NetId of the ADS server.
*srvPort*: Port number of the ADS server.

*C#:*

```
using TwinCAT.Ads;

namespace TRAINING_ADS_CNET
{
    public partial class Form1 : Form
    {
        TcAdsClient TcClient;

        /* unused code snipped out */

        private void Form1_Load(object sender, EventArgs e)
        {
            TcClient = new TcAdsClient();
            TcClient.Connect("127.255.255.1.1.1", 801);
        }
    }
}
```

*VB.NET:*

```
Imports TwinCAT.Ads

Public Class Form1
    Dim TcClient As TcAdsClient

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        TcClient = New TcAdsClient()
        TcClient.Connect(801)
    End Sub
End Class
```

# 3. Receiving and sending data

The Ads Client base class library is huge, and we will scarcely scratch the surface of its features in this chapter. That's a deliberate decision, because trying to cover more than a tiny fraction of the classes, methods and properties available would have effectively turned this chapter into a reference guide that simply listed classes and so on. We believe it's more important to understand the fundamental principles involved in ADS-communication; then you will be in a good position to explore the classes available yourself. (Full lists of all the classes and methods available are of course explained in depth in the Information System.)

## *3.1. IndexGroup and IndexOffset*

The READ and WRITE operations take place on the PLC interface (as defined by ADS) via two numbers: the index group (16 bit) and the index offset (32 bit). The ADS interface of the PLC will be described in more detail in the following pages with regard to the group and offset indices.

Indexgrp = The index group distinguishes different data within a port
IndexOffset = Indicates the offset, the byte from which reading or writing is to start.

**For example:**

| **Indexgroup** | **IndexGroup Description** | | |
|---|---|---|---|
| 16#4000 | PLC ADS services | | |
| | **Indexgroup** | **IndexOffset** | **Description** |
| | 16#4020 | 0x00000000-0x0000FFFF | READ_M - WRITE_M<br>PLC memory range (%M field). Offset is byte offset. |
| | 16#4021 | 0x00000000-0xFFFFFFFF | READ_MX - WRITE_MX<br>PLC memory range (%MX field). The index offset contains the bit address calculated from the byte number *8 + bit number |
| 16#F000 | General TwinCAT ADS system services | | |
| | **Indexgroup** | **IndexOffset** | **Description** |
| | 16#F020 | 0x00000000-0xFFFFFFFF | READ_I - WRITE_I<br>PLC process diagram of the physical inputs(%I field).<br>Offset is byte offset. |
| | 16#F021 | 0x00000000-0xFFFFFFFF | READ_IX - WRITE_IX<br>PLC process diagram of the physical inputs(%IX field).<br>The index offset contains the bit address which is calculated from byte number +8 + bit number |
| | 16#F030 | 0x00000000-0xFFFFFFFF | READ_Q - WRITE_Q<br>PLC process diagram of the physical outputs(%Q field).<br>Offset is byte offset. |
| | 16#F031 | 0x00000000-0xFFFFFFFF | READ_QX - WRITE_QX<br>PLC process diagram of the physical outputs(%QX field).<br>The index offset contains the bit address which is calculated from the byte number *8 + bit number. |

For more elaborate list on Index-groups/-offsets, please refer to the Information System

The “TwinCAT.Ads.dll” has and enumeration of the mostly used index-groups and offsets, for this have a look at "AdsReservedIndexGroups” and “AdsReservedIndexOffsets”.

## 3.2. AdsStream

The class AdsStream is a stream class used for Ads communication.

Derived from the System.IO.Memorystream, which is a stream whose backing store is memory.

A stream is an abstraction of a sequence of bytes, such as a file, an input/output device, an inter-process communication pipe, or a TCP/IP socket.

## 3.3. VariableHandle

A unique handle for an ADS variable.

## 3.4. Variable Types

The variable types from within the PLC-program would map into:

| System Manager | IEC61131-3 | Correspondent .NET type | C# Keyword | Visual Basic Keyword |
|---|---|---|---|---|
| BIT | BOOL | *System.Boolean* | *bool* | *Boolean* |
| BIT8 | BOOL | *System.Boolean* | *bool* | *Boolean* |
| BITARR8 | BYTE | *System.Byte* | *byte* | *Byte* |
| BITARR16 | WORD | *System.UInt16* | *ushort* | - |
| BITARR32 | DWORD | *System.UInt32* | *uint* | - |
| INT8 | SINT | *System.SByte* | *sbyte* | - |
| INT16 | INT | *System.Int16* | *short* | *Short* |
| INT32 | DINT | *System.Int32* | *int* | *Integer* |
| INT64 | LINT | *System.Int64* | *long* | *Long* |
| UINT8 | USINT | *System.Byte* | *byte* | *Byte* |
| UINT16 | UINT | *System.UInt16* | *ushort* | - |
| UINT32 | UDINT | *System.UInt32* | *uint* | - |
| UINT64 | ULINT | *System.UInt64* | *ulong* | - |
| FLOAT | REAL | *System.Single* | *float* | *Single* |
| DOUBLE | LREAL | *System.Double* | *double* | *Double* |

## 3.5. Read/Write Methods by Indexgroup and IndexOffset

*PLC-example Global Variables:*

```
BIT0 AT%MX10.0   :   BOOL;
BIT1 AT%MX10.1   :   BOOL;
BIT2 AT%MX10.2   :   BOOL;
BIT3 AT%MX10.3   :   BOOL;

iVar0 AT%MB12    :   INT;
diVar0 AT%MB14   :   DINT;
VisuPositions    :   ARRAY [0..19] OF LREAL;
```

### By Object

Read data synchronously from an ADS device and writes it to an object.

If the Type of the object to be read is a string type, the first element of the parameter args specifies the number of characters of the string. If the Type of the object to be read is an array type, the number of elements for each dimension has to be specified in the parameter args. Only 1 dimensional Arrays are supported.

- public **object ReadAny**(**long** *indexGroup*, **long** *indexOffset*, **Type** *type*);
- public **object ReadAny**(**long** *indexGroup*, **long** *indexOffset*, **Type** *type*, **int[]** *args*);

The write counterparts:

- public **void WriteAny**(**long** *indexGroup*, **long** *indexOffset*, **object** *value*);
- public **void WriteAny**(**long** *indexGroup*, **long** *indexOffset*, **object** *value*, **int[]** *args*);

For MX10.0 would give us index Offset 10 * 8 + 0 = 80

*C#:*

```
Using TwinCAT.Ads;
Using System;

    public partial class Form1 : Form
    {
        /*Unused code snipped out*/

        private void btnReadAny_Click(object sender, EventArgs e)
        {
                lblReadAny.Text = TcClient.ReadAny(0x4021, 80,
typeof(bool)).ToString();
        }
    }
```

*VB.NET:*

```
Imports TwinCAT.Ads
Imports System.Type

Public Class Form1

    /*Unused code snipped out*/

    Private Sub BtnRead_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BtnRead.Click
            LblReadAny.Text = TcClient.ReadAny(&H4021, 80,
GetType(Boolean))
    End Sub
End Class
```

So for MB12, type **INT** in our PLC program would give us type **INT16** in **.NET**

*C#:*

```
Using TwinCAT.Ads;
Using System;

    public partial class Form1 : Form
    {
        private void btnReadAny_Click(object sender, EventArgs e)
        {
                lblReadAny.Text = TcClient.ReadAny(0x4020, 12,
typeof(Int16)).ToString();
        }
    }
```

*VB.NET:*

```
Imports TwinCAT.Ads
Imports System.Type

Public Class Form1

    Private Sub BtnRead_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BtnRead.Click
            LblReadAny.Text = TcClient.ReadAny(&H4021, 80,
GetType(Int16))
    End Sub
End Class
```

For WRITING MB14 (diVar0), this would give us:

*C#:*

```
Using TwinCAT.Ads;
Using System;

    public partial class Form1 : Form
    {
        private void btnWriteAny_Click(object sender, EventArgs e)
        {
                TcClient.WriteAny(0x4020, 14,
Int16.Parse(txtBoxWriteAny.Text));
        }
  }
```

*VB.NET:*

```
Imports TwinCAT.Ads
Imports System.Type

Public Class Form1

    Private Sub btnWriteAny_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnWriteAny.Click
            TcClient.WriteAny(&H4020, 14,
Int16.Parse(txtBoxWriteAny.Text))
    End Sub
End Class
```

### By AdsStream

Reads data synchronously from an ADS device and writes it to the given stream.

Each read method returns the number of successfully data bytes being read.

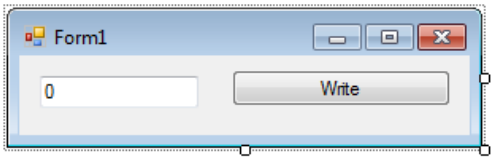
- public **int Read**(**int** *indexGroup*, **int** *indexOffset*, **AdsStream** *dataStream*);
- public **int Read**(**long** *indexGroup*, **long** *indexOffset*, **AdsStream** *dataStream*);
- public **int Read**(**long** *indexGroup*, **long** *indexOffset*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*);

Write counterparts:

- public **void Write**(**int** *indexGroup*, **int** *indexOffset*, **AdsStream** *dataStream*);
- public **void Write**(**int** *indexGroup*, **int** *indexOffset*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*);
- public **void Write**(**long** *indexGroup*, **long** *indexOffset*, **AdsStream** *dataStream*);
- public **void Write**(**long** *indexGroup*, **long** *indexOffset*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*);

As for our last example WRITING diVar0, this would give us:

*C#:*

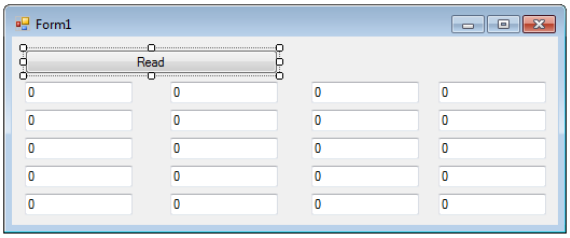



```
using System.IO;
using TwinCAT.Ads;

    public partial class Form1 : Form
    {
        TcAdsClient client = new TcAdsClient();
        int hVar;

        public Form1()
        {
            InitializeComponent();
            client.Connect(801);
        }

        private void BtnWrite_Click(object sender, EventArgs e)
        {
            AdsStream dataStream = new AdsStream();
            BinaryWriter binWrite = new BinaryWriter(dataStream);

            dataStream.Position = 0;
            binWrite.Write(Int32.Parse(TxtbxWrite.Text));
            client.Write(0x4020, 14, dataStream);
        }
    }
```

*VB.NET:*

```
Imports System.IO
Imports TwinCAT.Ads

Public Class Form1
    Dim client As New TcAdsClient

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        client.Connect(801)
    End Sub

    Private Sub BtnWrite_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BtnWrite.Click
        Dim dataStream As New AdsStream
        Dim binWrite As New BinaryWriter(dataStream)

        dataStream.Position = 0
        binWrite.Write(Integer.Parse(TxtBoxWrite.Text))
        client.Write(&H4020, 14, dataStream)
    End Sub
End Class
```

Reading out the array VisuPositions of type LREAL, this would give us:




*C#:*

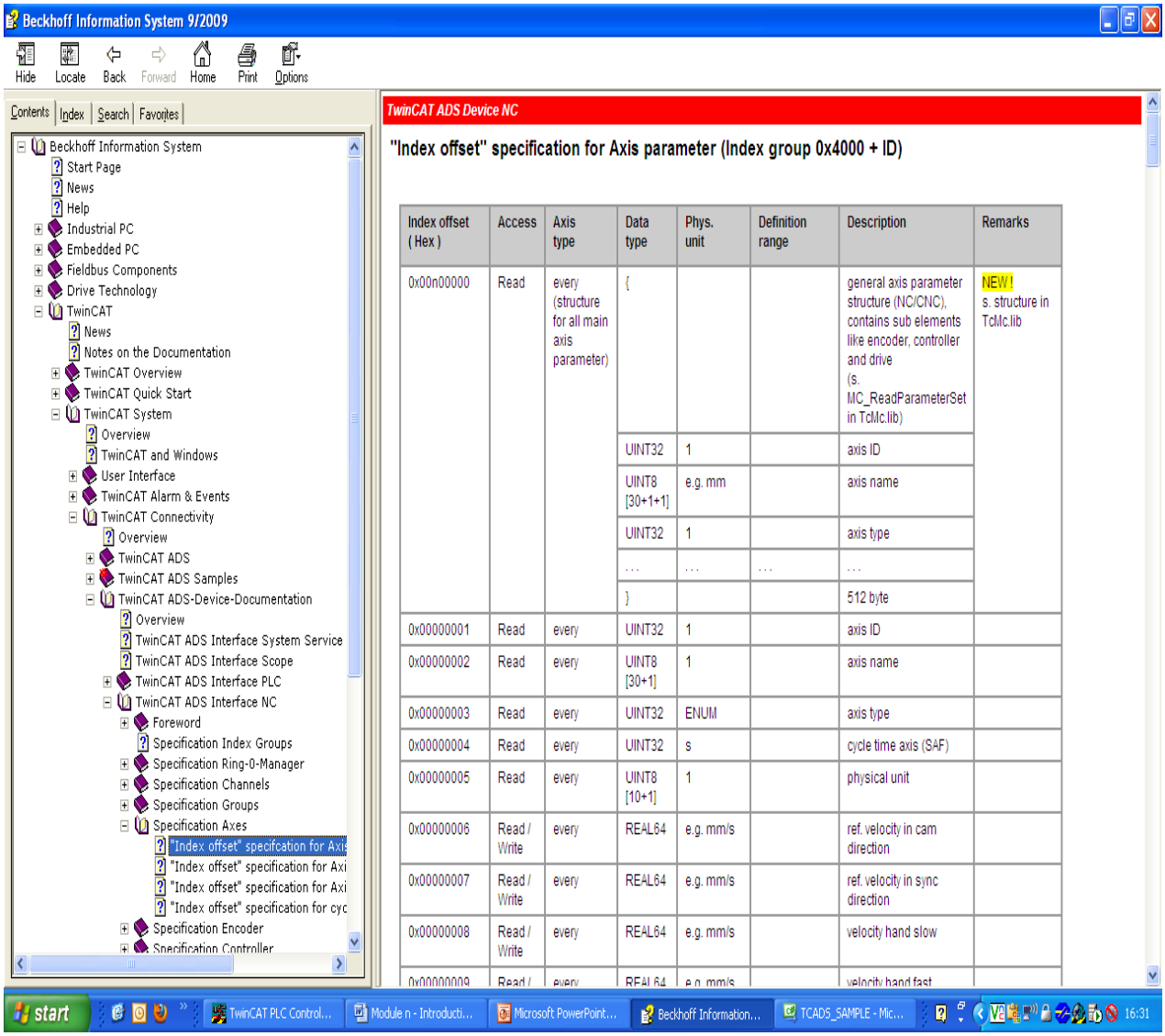
```
using System.IO;
using TwinCAT.Ads;

    public partial class Form1 : Form
    {
        TcAdsClient client = new TcAdsClient();
        int hVar;

        public Form1()
        {
            InitializeComponent();
            client.Connect(801);
        }

        private void btnRead_Click(object sender, EventArgs e)
        {
            AdsStream dataStream = new AdsStream(20 * 8);
            BinaryReader binRead = new BinaryReader(dataStream);

            client.Read(0x4040, 38, dataStream);
            TxtBoxRead1.Text = binRead.ReadDouble().ToString();
            TxtBoxRead2.Text = binRead.ReadDouble().ToString();
            //. redundant code snipped out
            TxtBoxRead19.Text = binRead.ReadDouble().ToString();
            TxtBoxRead20.Text = binRead.ReadDouble().ToString();
        }
    }
```

*VB.NET:*

```
Imports System.IO
Imports TwinCAT.Ads

Public Class Form1
    Dim client As New TcAdsClient

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        client.Connect(801)
    End Sub

    Private Sub btnRead_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnRead.Click
        Dim dataStream As New AdsStream(20 * 8)
        Dim binRead As New BinaryReader(dataStream)

        client.Read(&H4040, 38, dataStream)

        TxtBoxRead1.Text = binRead.ReadDouble()
        TxtBoxRead2.Text = binRead.ReadDouble()
        '. Redundant code snipped out
        TxtBoxRead19.Text = binRead.ReadDouble()
        TxtBoxRead20.Text = binRead.ReadDouble()
    End SubEnd Class
```

### PLC-example NC-struct

In the information System we find that we can read on Indexgrp 0x4000 + AxisID offset 0 the axis parameter from a configured axis.




Instantiate an ADS Client port 500.

```
tcClient = new TcAdsClient();
tcClient.Connect(801);
NcClient = new TcAdsClient();
NcClient.Connect(500);
```

Create a stream to receive the data

```
AdsStrNcParaset = new AdsStream(512);
```

And read the data through its indexgrp and offset

```
NcClient.Read(0x4001, 0x0, AdsStrNcParaset);

BinaryReader BinRead = new BinaryReader(AdsStrNcParaset, Encoding.ASCII);
UInt32 Id = BinRead.ReadUInt32();
string tekst = new string(BinRead.ReadChars(32));
```

OR you could also use the **AdsBinaryReader**

```
NcClient.Read(0x4001, 0x0, AdsStrNcParaset);

AdsBinaryReader AdsBinRead = new AdsBinaryReader(AdsStrNcParaset);

UInt32 Id = AdsBinRead.ReadUInt32();
string tekst = new string(AdsBinRead.ReadChars(32));
```

AdsBinaryReader derives from BinaryReader and reads primitive as well as PLC data types as binary values.

| | |
|---|---|
| ReadPlcDATE | Reads a PLC Date type from the current stream. |
| ReadPlcString | Reads a PLC string from the current stream. |
| ReadPlcTIME | Reads a PLC Time type from the current stream. |

For our example this would give us:

```
NcClient.Read(0x4001, 0x0, AdsStrNcParaset);

AdsBinaryReader AdsBinRead = new AdsBinaryReader(AdsStrNcParaset);

UInt32 Id = AdsBinRead.ReadUInt32();
string tekst = AdsBinRead.ReadPlcString(31);
```

## 3.6. Read/Write Methods by Variable Handle

- public **int CreateVariableHandle**(**string** *variableName*);

In the end or at a error, we have to release the handle:

- public **void DeleteVariableHandle**(**int** *variableHandle*);

*PLC-example Global Variables:*

```
BIT0 AT%MX10.0   :   BOOL;
BIT1 AT%MX10.1   :   BOOL;
BIT2 AT%MX10.2   :   BOOL;
BIT3 AT%MX10.3   :   BOOL;

iVar0 AT%MB12    :   INT;
diVar0 AT%MB14   :   DINT;
VisuPositions    :   ARRAY [0..19] OF LREAL;
```

*C#:*

```
Using TwinCAT.Ads;
Using System;

    public partial class Form1 : Form
    {
        /* unused code snipped out */
        int varHandle = TcClient.CreateVariableHandle(".BIT0");
        /* unused code snipped out */

    }
```

*VB.NET:*

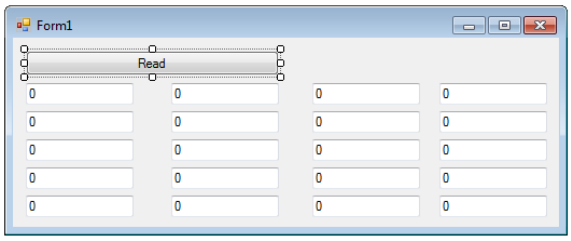
```
Imports TwinCAT.Ads
Imports System.Type

Public Class Form1
    /* unused code snipped out */
    Dim varHandle As Integer = TcClient.CreateVariableHandle(".BIT0")
    /* unused code snipped out */
End Class
```

## By Object

- public **object ReadAny**(**int** *variableHandle*, **Type** *type*);
- public **object ReadAny**(**int** *variableHandle*, **Type** *type*, **int[]** *args*);
- public **void WriteAny**(**int** *variableHandle*, **object** *value*);
- public **void WriteAny**(**int** *variableHandle*, **object** *value*, **int[]** *args*);

Example of reading out the array VisuPositions of type LREAL:

*C#:*




```
Using TwinCAT.Ads;
Using System;

    public partial class Form1 : Form
    {
        TcAdsClient client = new TcAdsClient();
        int hVar;

        public Form1()
        {
            InitializeComponent();
            client.Connect(801);
            hVar = client.CreateVariableHandle(".VisuPositions");
        }

        private void btnRead_Click(object sender, EventArgs e)
        {
             double[] tempvar = (Double[])client.ReadAny(hVar,typeof(Double[]),
new int[] { 20 });

            TxtBoxRead1.Text = tempvar[0].ToString();
            TxtBoxRead2.Text = tempvar[1].ToString();
            //.
            //. redundant code snipped out
            //.
            TxtBoxRead20.Text = tempvar[19].ToString();
        }
    }
```

## By AdsStream

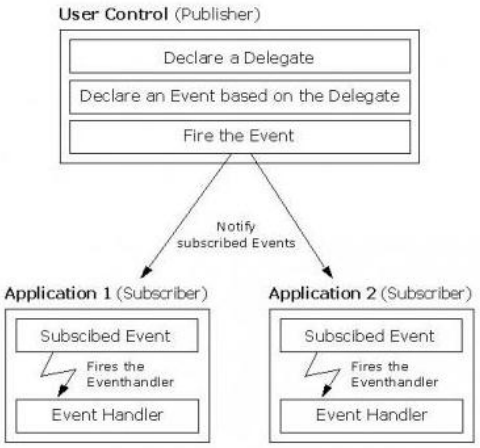
- public **int Read**(**int** *variableHandle*, **AdsStream** *dataStream*);
- public **int Read**(**int** *variableHandle*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*);
- public **void Write**(**int** *variableHandle*, **AdsStream** *dataStream*);
- public **void Write**(**int** *variableHandle*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*);

## 3.7. Update methods by notification

Connects a variable to the ADS client. The ADS client will be notified by the AdsNotification event.

For every demanded notification you will receive an event, explanation taken from MSDN-website:

*"An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic. The object that raises the event is called the event sender. The object that captures the event and responds to it is called the event receiver."*




The AdsNotificationEx-EventHandler occurs when the ADS device sends a notification to the client.

Explanation taken from MSDN-website:

*"The standard signature of an event handler delegate defines a method that does not return a value, whose first parameter is of type **Object** and refers to the instance that raises the event, and whose second parameter is derived from type **EventArgs** and holds the event data. If the event does not generate event data, the second parameter is simply an instance of **EventArgs**. Otherwise, the second parameter is a custom type derived from **EventArgs** and supplies any fields or properties needed to hold the event data."*

### Impact of CycleTime & MaxDelayTime

| Subscript | CycleTime | MaxDelayTime | Response |
|---|---|---|---|
| Cyclic | 500 | 10000 | You reveice every 10 seconds a new value. |
| Cyclic | 1000 | 10000 | You receive every 10 seconds a new value. |
| Cyclic | 1000 | 0 | You receive every second a new value. |
| OnChange | 500 | 10000 | Same behaviour as with cyclical notification. Differerence when value freezes for t > MaxDelayTime, no new values will be transmit. When the value starts changing again, it could take up to t = MaxDelayTime before value is received at the client level. |

*PLC-example Global Variables:*

```
BIT0 AT%MX10.0   :          BOOL;
BIT1 AT%MX10.1   :          BOOL;
BIT2 AT%MX10.2   :          BOOL;
BIT3 AT%MX10.3   :          BOOL;

iVar0 AT%MB12    :          INT;
diVar0 AT%MB14   :          DINT;
VisuPositions    :          ARRAY [0..19] OF LREAL;
```

## By Object

- public **int AddDeviceNotificationEx**(long indexGroup, long indexOffset, AdsTransMode transMode, int cycleTime, int maxDelay, object userData, Type type);

- public **int AddDeviceNotificationEx**(long indexGroup, long indexOffset, AdsTransMode transMode, int cycleTime, int maxDelay, object userData, Type type, int[] args);
- public **int AddDeviceNotificationEx**(string *variableName*, AdsTransMode *transMode*, int *cycleTime*, int *maxDelay*, object *userData*, Type *type*);
- public **int AddDeviceNotificationEx**(string *variableName*, AdsTransMode *transMode*, int *cycleTime*, int *maxDelay*, object *userData*, Type *type*, int[] *args*);

**TransMode**
Specifies if the event should be fired cyclically or only if the variable has changed.
**CycleTime**
The ADS server checks whether the variable has changed after this time interval. The unit is in ms.
**maxDelay**
The AdsNotification event is fired at the latest when this time has elapsed. The unit is ms.
**userData**
This object can be used to store user specific data.
**type**
Type of the object stored in the event argument.
**args**
Additional arguments.

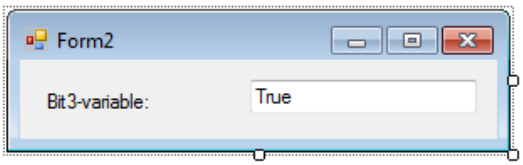
The event handler receives a **AdsNotificationExEventArgs** containing data related to the **AdsNotificationEx** event.

The following **AdsNotificationExEventArgs** properties provide information specific to this event:

| Property | Description |
|---|---|
| NotificationHandle | Gets the handle of the connection. |
| TimeStamp | Gets the timestamp. |
| UserData | Gets the user object. |
| Value | Value of the ads variable. |

For the “BIT3”-variable in our PLC-example as global variable this would give us

*C#:*




```
using TwinCAT.Ads;

namespace Notif
{

    public partial class Form1 : Form
    {
        TcAdsClient client = new TcAdsClient();
        int notifHandle;

        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            client.Connect(801);
            client.AdsNotificationEx +=
                new AdsNotificationExEventHandler(client_AdsNotificationEx);
            notifHandle = client.AddDeviceNotificationEx(
                ".Bit3",
                AdsTransMode.OnChange,
                100,
                100,
                TxtBoxValue,
                typeof(bool));
        }

        void client_AdsNotificationEx(
            object sender, AdsNotificationExEventArgs e)
        {
            if (e.NotificationHandle == notifHandle)
            {
                //TextBox txtbx = (TextBox)e.UserData;
                TextBox txtbx = e.UserData as TextBox;

                if (txtbx != null)
                {
                    txtbx.Text = e.Value.ToString();
                }
            }
        }
    }
}
```

With “TxtBoxValue” being our TextBox added to the form.
While creating the notificationHandle, we pass the TextBox instance as an object to the handler.

*VB.NET:*

```
Imports TwinCAT.Ads

Partial Public Class Form1
    Inherits Form
    Private client As New TcAdsClient()
    Private notifHandle As Integer

    Private Sub Form1_Load(sender As Object, e As EventArgs)
        client.Connect(801)
        AddHandler client.AdsNotificationEx, AddressOf client_AdsNotificationEx
        notifHandle = client.AddDeviceNotificationEx(
            ".Bit3", AdsTransMode.OnChange, 100, 100,
            TxtBoxValue, GetType(Boolean))
    End Sub

    Private Sub client_AdsNotificationEx(
        sender As Object, e As AdsNotificationExEventArgs)
        If e.NotificationHandle = notifHandle Then
            Dim txtbx As TextBox = TryCast(e.UserData, TextBox)

            If txtbx IsNot Nothing Then
                txtbx.Text = e.Value.ToString()
            End If
        End If
    End Sub
End Class
```

*Taken from the MSDN website – only valid for VB.Net:*

If an attempted conversion fails, **CType** and **DirectCast** both throw an InvalidCastException error. This can adversely affect the performance of your application. **TryCast** returns Nothing (Visual Basic), so that instead of having to handle a possible exception, you need only test the returned result against **Nothing**.

For the visupositions-variable in our PLC-example as global variable this would give us

*C#:*

```
using TwinCAT.Ads;

namespace Notif
{

    public partial class Form1 : Form
    {
        TcAdsClient client = new TcAdsClient();
        int notifHandleArray;

        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            client.Connect(801);
            client.AdsNotificationEx +=
                new AdsNotificationExEventHandler(client_AdsNotificationEx);
            notifHandleArray = client.AddDeviceNotificationEx(
                ".VisuPositions",
                AdsTransMode.OnChange,
                100,
                100,
                null,
                typeof(double[]),
                new int[] { 20 });
        }

        void client_AdsNotificationEx(
            object sender, AdsNotificationExEventArgs e)
        {

            if (e.NotificationHandle == notifHandleArray)
            {
                double[] array = (double[])e.Value;
            }
        }
    }
}
```

*VB.NET:*

```
Imports TwinCAT.Ads

Public Class Form1
    Dim TcClient As TcAdsClient
    Dim notifHandleArray As Int32
    Dim array() As Double

    Private Sub Form1_Load(
                          ByVal sender As System.Object,
                          ByVal e As System.EventArgs) Handles MyBase.Load
        TcClient = New TcAdsClient()
        TcClient.Connect(801)

        AddHandler TcClient.AdsNotificationEx, AddressOf client_adsNotification
        notifHandleArray = TcClient.AddDeviceNotificationEx(
              ".VisuPositions",
              AdsTransMode.OnChange,
              100,
              100,
              Nothing,
              GetType(Double()),
              New Int32() {20})

    End Sub

    Protected Sub client_adsNotification(
                                        ByVal sender As Object,
                                        ByVal e As AdsNotificationExEventArgs)
        If notifHandleArray.Equals(e.NotificationHandle) Then
            array = e.Value
        End If
    End Sub
End Class
```

### By AdsStream

- public **int AddDeviceNotification**(**int** *indexGroup*, **int** *indexOffset*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*, **AdsTransMode** *transMode*, **int** *cycleTime*, **int** *maxDelay*, **object** *userData*);
- public **int AddDeviceNotification**(**int** *indexGroup*, **int** *indexOffset*, **AdsStream** *dataStream*, **AdsTransMode** *transMode*, **int** *cycleTime*, **int** *maxDelay*, **object** *userData*);
- public **int AddDeviceNotification**(**long** *indexGroup*, **long** *indexOffset*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*, **AdsTransMode** *transMode*, **int** *cycleTime*, **int** *maxDelay*, **object** *userData*);
- public **int AddDeviceNotification**(**long** *indexGroup*, **long** *indexOffset*, **AdsStream** *dataStream*, **AdsTransMode** *transMode*, **int** *cycleTime*, **int** *maxDelay*, **object** *userData*);
- public **int AddDeviceNotification**(**string** *variableName*, **AdsStream** *dataStream*, **int** *offset*, **int** *length*, **AdsTransMode** *transMode*, **int** *cycleTime*, **int** *maxDelay*, **object** *userData*);
- public **int AddDeviceNotification**(**string** *variableName*, **AdsStream** *dataStream*, **AdsTransMode** *transMode*, **int** *cycleTime*, **int** *maxDelay*, **object** *userData*);

**TransMode**
: Specifies if the event should be fired cyclically or only if the variable has changed.

**CycleTime**
: The ADS server checks whether the variable has changed after this time interval. The unit is in ms.

**maxDelay**
: The AdsNotification event is fired at the latest when this time has elapsed. The unit is ms.

**userData**
: This object can be used to store user specific data.

The event handler receives a **AdsNotificationEventArgs** containing data related to the **AdsNotification** event.

The following **AdsNotificationEventArgs** properties provide information specific to this event:

| Property | Description |
|---|---|
| DataStream | Streams that holds the notification data. |
| Length | Gets the Length of the data in the stream. |
| NotificationHandle | Gets the handle of the connection. |
| Offset | Gets the Offset of the data in the stream. |
| TimeStamp | Gets the timestamp. |
| UserData | Gets the user object. This object is passed by to AddDeviceNotification and can be used to store data. |

*PLC-example Global Variables:*

```
Axisstate1      :       strAxisState; (*Axe struct V0.1*)
```

With strAxisState defined as

```
TYPE strAxisstate :
STRUCT
        AxisName    :       STRING(19);
        Ready       :       BOOL;
        bError      :       BOOL;
        actVelocity :       LREAL;
        actPosition :       LREAL;
        setVelocity :       LREAL;
        setPosition :       LREAL;
        Lagdistance :       LREAL;
END_STRUCT
END_TYPE
```

This will give us:

*C#:*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using TwinCAT.Ads;

namespace TRAINING_ADS_CNET
{
    public partial class Form1 : Form
    {
        public struct AxisStruct
        {
            public string AxisName;
            public bool Ready;
            public bool Error;
            public double ActVelocity;
            public double ActPosition;
            public double SetVelocity;
            public double SetPosition;
            public double LagDistance;
        }

        TcAdsClient TcClient;
        AdsStream AdsStrAxisStruct;
        int notifHandle;

        AxisStruct axisStruct1 = new AxisStruct();

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            TcClient = new TcAdsClient();
            TcClient.Connect("192.168.1.105.1.1", 801);

            TcClient.AdsNotification += new
AdsNotificationEventHandler(TcClient_AdsNotification);
            AdsStrAxisStruct = new AdsStream(62);
            notifHandle =
TcClient.AddDeviceNotification(".Axisstate1",AdsStrAxisStruct ,
AdsTransMode.OnChange,100,0,null);
        }
    }
}
```

*VB.NET:*

```
Imports TwinCAT.Ads
Imports System.Type

Public Structure AxisStruct
    Dim AxisName As String
    Dim Ready As Boolean
    Dim bError As Boolean
    Dim actVelocity As Double
    Dim actPosition As Double
    Dim setVelocity As Double
    Dim setPosition As Double
    Dim Lagdistance As Double
End Structure

Public Class Form1
    Dim TcClient As TcAdsClient
    Dim AdsStrAxisStruct As AdsStream
    Dim notifHandle As Int32

    Dim axisStruct1 As New AxisStruct()

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        TcClient = New TcAdsClient()
        TcClient.Connect(801)

        Dim AdsStrAxisStruct As New AdsStream(62)

        AddHandler TcClient.AdsNotification, AddressOf
client_adsNotification
        notifHandle = TcClient.AddDeviceNotification(".Axisstate1",
AdsStrAxisStruct, AdsTransMode.OnChange, 100, 0, Nothing)
    End Sub
```

An example event-handling function could be:

*C#:*

```
        void TcClient_AdsNotification(object sender,
AdsNotificationEventArgs e)
        {
            if (e.NotificationHandle == notifHandle)
            {
                e.DataStream.Position = 0;
                AdsBinaryReader adsbinreader = new
AdsBinaryReader(e.DataStream);
                axisStruct1.AxisName = adsbinreader.ReadPlcString(20);
                axisStruct1.Ready = adsbinreader.ReadBoolean();
                axisStruct1.bError = adsbinreader.ReadBoolean();
                axisStruct1.ActVelocity = adsbinreader.ReadDouble();
                axisStruct1.ActPosition = adsbinreader.ReadDouble();
                axisStruct1.SetVelocity = adsbinreader.ReadDouble();
                axisStruct1.SetPosition= adsbinreader.ReadDouble();
                axisStruct1.LagDistance = adsbinreader.ReadDouble();
            }
        }
    }
}
```

*VB.NET:*

```
          Protected Sub client_adsNotification(ByVal sender As
Object, ByVal e As AdsNotificationEventArgs)
        If e.NotificationHandle = notifHandle Then
            e.DataStream.Position = 0
            Dim adsbinreader As AdsBinaryReader = New
AdsBinaryReader(e.DataStream)
            axisStruct1.AxisName = adsbinreader.ReadPlcString(20)
            axisStruct1.Ready = adsbinreader.ReadBoolean()
            axisStruct1.bError = adsbinreader.ReadBoolean()
            axisStruct1.actVelocity = adsbinreader.ReadDouble()
            axisStruct1.actPosition = adsbinreader.ReadDouble()
            axisStruct1.setVelocity = adsbinreader.ReadDouble()
            axisStruct1.setPosition = adsbinreader.ReadDouble()
            axisStruct1.Lagdistance = adsbinreader.ReadDouble()
        End If
    End Sub
```

## 3.8. ADS-Sum Command: Reading or writing several variables

Using the ADS Sum Command it is possible to read or write several variables in one command. Designed as TcAdsClient.ReadWrite it is used as a container, which transports all sub-commands in one ADS stream.

### Background:

ADS offer powerful and fast communication to exchange any kind of information. It's possible to read single variables or complete arrays and structures with each one single ADS-API call.
This new ADS command offers to read with one single ADS call multiple variables which are not structured within a linear memory.

As a result the ADS caller application (like scada Systems etc.) can extremely speed up cyclic polling.

Sample:

- Until now : Polling 4000 single variables which are not in a linear area (like array / structure / fixed PLC address ) would cause 4000 single Ads-ReadReq with each 1-2 ms protocol time.
  **As a result the scanning of these variables takes 4000ms-8000ms.**
- New Ads-Command allows to read multiple variables with one single ADS-ReadReq : 4000 single variables are handled with e.g. 8 single Ads-ReadReq (each call requesting 500 variables) with each 1-2 ms protocol time.
  **As a result the scanning of these variables takes just few 10ms.**

### Version of Target device:

ADS itself is just the transport layer, but the requested ADS device has to support the ADS-Command.

TwinCAT 2.11 >= Build 1550 required

### Bytes length of requested data:

Requesting a large list of values from variables is fine, but the requested data of the Ads-response (the data-byte-length) have to pass the AMS-router (size by default a 2048kb)
So the caller has to limit the requested variables based on calculation of requested data-byte-length.

## Number of Sub-ADS calls:

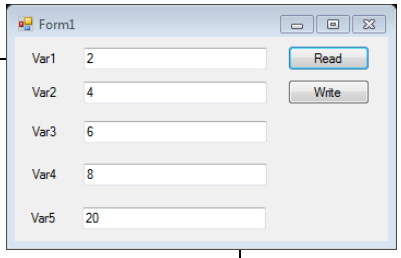
Highly recommended to a maximum of 500 sub-calls.

If the PLC is processing one ADS request, it will completely work on this single ADS request **BEFORE** starting neyt PLC cycle.
As a result one single ADS request with 200.000 sub-Ads-requests would cause that PLC would collect and copy 200.000 variables into one single ADS response, before starting next PLC.
So this large number of ads-sub-commands will jitter the PLC execution!

*PLC-example Global Variables:*

```
VAR1 AT%MB10    :    INT;
VAR2 AT%MB20    :    INT;
VAR3 AT%MB30    :    INT;
VAR4 AT%MB34    :    REAL;
VAR5 AT%MB45    :    INT;
```

*C#:*




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using TwinCAT.Ads;
using System.IO;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        // Structure declaration for handles
        internal struct VariableInfo
        {
            public int indexGroup;
            public int indexOffset;
            public int length;
        }

        TcAdsClient tcClient = new TcAdsClient();
        private string[] variableNames;
        private int[] variableLengths;
        VariableInfo[] variables;
```

```
    public Form1()
        {
            InitializeComponent();

            tcClient.Connect(801);

            //Fill structures with name and size of PLC variables
            variableNames = new string[] { ".VAR1", ".VAR2", ".VAR3",
".VAR4", ".VAR5" };
            variableLengths = new int[] { 2, 2, 2, 4, 2 };

            // Write handle parameter into structure
            variables = new VariableInfo[variableNames.Length];
            for (int i = 0; i < variables.Length; i++)
            {
                variables[i].indexGroup =
(int)AdsReservedIndexGroups.SymbolValueByHandle;
                variables[i].indexOffset =
tcClient.CreateVariableHandle(variableNames[i]);
                variables[i].length = variableLengths[i];
            }
        }

       private AdsStream BlockRead(VariableInfo[] variables)
        {
            //Allocate memory
            int wrLength = variables.Length * 12;
            //3* INT length for every variable (group-offset-length)

            int rdLength = variables.Length * 4;
            //int-length for every variable (error)
            // Write data for handles into the ADS Stream
            BinaryWriter writer = new BinaryWriter(new
AdsStream(wrLength));
            for (int i = 0; i < variables.Length; i++)
            {
                writer.Write(variables[i].indexGroup);
                writer.Write(variables[i].indexOffset);
                writer.Write(variables[i].length);
                rdLength += variables[i].length;
            }
            // Sum command to read variables from the PLC
            AdsStream rdStream = new AdsStream(rdLength);
            tcClient.ReadWrite(0xF080, variables.Length, rdStream,
(AdsStream)writer.BaseStream);

            // Return the ADS error codes
            return rdStream;
        }
```

```
        private AdsStream BlockWrite(VariableInfo[] variables)
        {
            //Allocate memory
            int wrLength = variables.Length * 12;
            //3* INT length for every variable (group-offset-length)
            // +Valuesize for every variable
            for (int i = 0; i < variables.Length; i++)
            {
                wrLength += variables[i].length;
            }

            int rdLength = variables.Length * 4;
            //int-length for every variable (error)
            // Write data for handles into the ADS Stream
            BinaryWriter writer = new BinaryWriter(new
AdsStream(wrLength));
            for (int i = 0; i < variables.Length; i++)
            {
                writer.Write(variables[i].indexGroup);
                writer.Write(variables[i].indexOffset);
                writer.Write(variables[i].length);
            }
            writer.Write(Int16.Parse(textBox1.Text));
            writer.Write(Int16.Parse(textBox2.Text));
            writer.Write(Int16.Parse(textBox3.Text));
            writer.Write(Single.Parse(textBox4.Text));
            writer.Write(Int16.Parse(textBox5.Text));

            // Sum command to read variables from the PLC
            AdsStream rdStream = new AdsStream(rdLength);
            tcClient.ReadWrite(0xF081, variables.Length, rdStream,
(AdsStream)writer.BaseStream);
            // Return the ADS error codes
            return rdStream;
        }

        private void btnRead_Click(object sender, EventArgs e)
        {
            if (tcClient == null)
                return;

            //Get the ADS return codes and examine the errors
            BinaryReader binReader = new
BinaryReader(BlockRead(variables));
            for (int i = 0; i < variables.Length; i++)
            {
                int error = binReader.ReadInt32();
                if (error != (int)AdsErrorCode.NoError)
                    System.Diagnostics.Debug.WriteLine(
                        String.Format("Unable to read variable {0}
(Error = {1})", i, error));
            }
            textBox1.Text = binReader.ReadInt16().ToString();
            textBox2.Text = binReader.ReadInt16().ToString();
            textBox3.Text = binReader.ReadInt16().ToString();
            textBox4.Text = binReader.ReadSingle().ToString();
            textBox5.Text = binReader.ReadInt16().ToString();
        }
```

```
        private void btnWrite_Click(object sender, EventArgs e)
        {
            if (tcClient == null)
                return;

            // Get the ADS return codes and examine for errors
            BinaryReader reader = new
BinaryReader(BlockWrite(variables));
            for (int i = 0; i < variables.Length; i++)
            {
                int error = reader.ReadInt32();
                if (error != (int)AdsErrorCode.NoError)
                    System.Diagnostics.Debug.WriteLine(
                        String.Format("Unable to read variable {0}
(Error = {1})", i, error));
            }
        }
```

Alternative InitClient, to show you don't **have** to use handles for sumup

*C#:*

```
        private void InitClient()
        {
            tcClient.Connect(801);

            //Fill structures with name and size of PLC variables
            variableNames = new string[] { ".VAR1", ".VAR2", ".VAR3",
".VAR4", ".VAR5" };
            variableLengths = new int[] { 2, 2, 2, 4, 2 };

            // Write handle parameter into structure
            variables = new VariableInfo[variableNames.Length];
            int offset = 10;
            for (int i = 0; i < variables.Length; i++)
            {
                variables[i].indexGroup =
(int)AdsReservedIndexGroups.PlcRWMB;
                variables[i].indexOffset = offset;
                variables[i].length = variableLengths[i];
                offset = offset + 10;
            }
        }
```

*VB.NET:*

```
    Private Sub Btn_ReadData_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Btn_ReadData.Click
        AantalVar = 5
        SizeOfVar = 12 '(4xInt + 1xReal)
        dsReadData = New AdsStream(AantalVar * 4 + SizeOfVar)
        dsWriteData = New AdsStream(AantalVar * 12)
        binWriteData = New BinaryWriter(dsWriteData,
System.Text.Encoding.ASCII)
        binReadData = New BinaryReader(dsReadData, System.Text.Encoding.ASCII)

        ' WriteData invullen
        dsWriteData.Position = 0

        ' IntVar1 at %MB10
        binWriteData.Write(CInt(&H4020))    'IndexGroup
        binWriteData.Write(CInt(10))        'IndexOffset
        binWriteData.Write(CInt(2))         'Length

        ' IntVar2 at %MB20
        binWriteData.Write(CInt(&H4020))    'IndexGroup
        binWriteData.Write(CInt(20))        'IndexOffset
        binWriteData.Write(CInt(2))         'Length

        ' IntVar3 at %Mb30
        binWriteData.Write(CInt(&H4020))    'IndexGroup
        binWriteData.Write(CInt(30))        'IndexOffset
        binWriteData.Write(CInt(2))         'Length

        ' RealVar4 at %Mb40
        binWriteData.Write(CInt(&H4020))    'IndexGroup
        binWriteData.Write(CInt(40))        'IndexOffset
        binWriteData.Write(CInt(4))         'Length
```

```
        ' IntVar5 at %MB50
        binWriteData.Write(CInt(&H4020))    'IndexGroup
        binWriteData.Write(CInt(50))        'IndexOffset
        binWriteData.Write(CInt(2))         'Length

        ' Read Data
        tcClient.ReadWrite(&HF080, AantalVar, dsReadData, 0, AantalVar * 4 +
SizeOfVar, dsWriteData, 0, AantalVar * 12)

        ' Display Result
        dsReadData.Position = AantalVar * 4
        ' IntVar1 at %MB10
        TextBoxIntVar1.Text = binReadData.ReadInt16.ToString()
        ' IntVar2 at %MB20
        TextBoxIntVar2.Text = binReadData.ReadInt16.ToString()
        ' IntVar3 at %Mb30
        TextBoxIntVar3.Text = binReadData.ReadInt16.ToString()
        ' RealVar4 at %Mb40
        TextBoxIntVar4.Text = binReadData.ReadSingle.ToString()
        ' IntVar5 at % mb50
        TextBoxIntVar5.Text = binReadData.ReadInt16.ToString()
    End Sub
End Class
```

## 3.9. SymbolInfo:

### A single variable

- public **ITcAdsSymbol** **ReadSymbolInfo**(**string** *name*);

Call this method to obtain information about the individual symbols (variables) in ADS devices. The returned **ITcAdsSymbolinfo** contains:

| | |
|---|---|
| Comment | Gets the comment behind the variable declaration. |
| DataType | Data type of the symbol. |
| IndexGroup | Gets the index group of the symbol. |
| IndexOffset | Gets the index offset of the symbol. |
| Name | Gets the name of the symbol. |
| Size | Gets the size of the symbol. |
| Type | Gets the name of the symbol data type. |

For our previously defined “Axisstate1” structure (see 3.7 Update methods by notification):

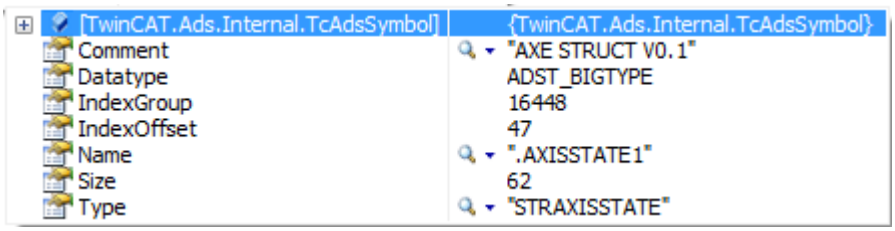
*C#:*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using TwinCAT.Ads;

namespace Training_ADS_CNET_SYM
{
    public partial class Form1 : Form
    {
        TcAdsClient TcClient = new TcAdsClient();
        public Form1()
        {
            InitializeComponent();

            TcClient.Connect(801);
            ReadSymbolInfo();
        }

        private void ReadSymbolInfo()
        {
            ITcAdsSymbol symbol =
TcClient.ReadSymbolInfo(".Axisstate1");
        }
    }
}
```

*VB.NET:*

```
      Imports TwinCAT.Ads

Public Class Form1
    Dim TcClient As TcAdsClient

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        TcClient = New TcAdsClient()
        TcClient.Connect(801)
        ReadSymbolInfo()
    End Sub

    Private Sub ReadSymbolInfo()
        Dim symbol As ITcAdsSymbol =
TcClient.ReadSymbolInfo(".Axisstate1")
    End Sub
End Class
```

We would receive in both cases:



- public **object** **ReadSymbol**(**string** *name*, **Type** *type*, **bool** *reloadSymbolInfo*);
- public **object** **ReadSymbol**(**ITcAdsSymbol** *symbol*);

**Name**
Name of the ADS symbol.
**Type**
Managed type of the ADS symbol.
**Symbol**
The symbol that should be read.

Both methods will read out the value of a symbol and returns it as an object. Strings and all primitive datatypes(UInt32, Int32, Bool etc.) are supported. The difference between both is that Arrays and Structures cannot be read with the ITcAdsSymbol-method.

If we would extend our previous example:

*C#:*

```
namespace Training_ADS_CNET_SYM
{
    public partial class Form1 : Form
    {
        public struct AxisStruct
        {
            public string AxisName;
            public bool Ready;
            public bool Error;
            public double ActVelocity;
            public double ActPosition;
            public double SetVelocity;
            public double SetPosition;
            public double LagDistance;
        }

        TcAdsClient TcClient = new TcAdsClient();
        public Form1()
        {
            InitializeComponent();

            TcClient.Connect(801);
            ReadSymbolInfo();
        }

        private void ReadSymbolInfo()
        {
            ITcAdsSymbol symbol =
TcClient.ReadSymbolInfo(".Axisstate1");

            //This will read out our struct without an error
            AxisStruct Axestruct =
(AxisStruct)TcClient.ReadSymbol(".Axisstate1", typeof(AxisStruct),
true);
            //Next would result in an errormessage
            //AxisStruct Axestruct =
(AxisStruct)TcClient.ReadSymbol(symbol);
        }
    }
}
```

## All variables

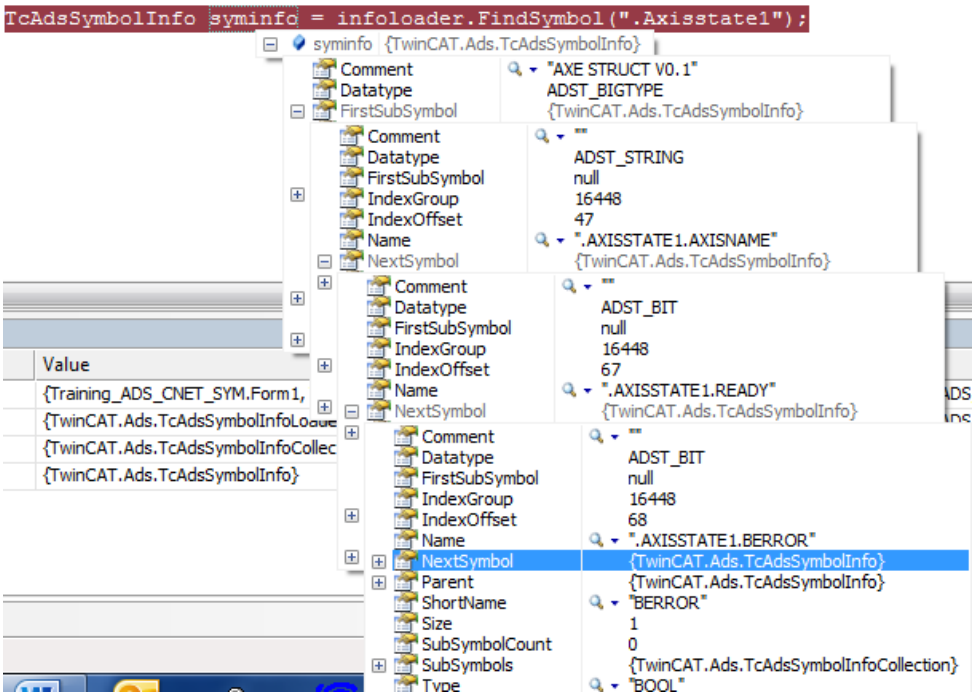
The class TcAdsSymbolInfoLoader is responsible for downloading the list of declared variables from an ADS Server. This loader is created through the use of our TcAdsClient.

- public **TcAdsSymbolInfoLoader** **CreateSymbolInfoLoader**();

The TcAdsSymbolInfoLoader exposes public methods like:

- public **TcAdsSymbolInfo** **FindSymbol**(**string** *name*);

  The Findsymbol method does the same as the ReadSymbolInfo, except the fact that this method returns a TcAdsSymbolInfo which exposes more info like how a managed type is built up (possible structs etc.), so one could parse through them.




As the screenshot shows, our “Axisstate1”-variabele is made up (through the use of “NextSymbol”), of *name*, *Ready, bError,…*

- public **TcAdsSymbolInfoCollection** **GetSymbols**(**bool** *forceReload*);

  Loads the declared symbols from the ADS device and returns them as a collection of TcAdsSymbolInfo objects.

*C#:*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using TwinCAT.Ads;

namespace Training_ADS_CNET_SYM
{
    public partial class Form1 : Form
    {
        TcAdsClient TcClient = new TcAdsClient();
        public Form1()
        {
            InitializeComponent();

            TcClient.Connect(801);
            TcAdsSymbolInfoLoader infoloader =
TcClient.CreateSymbolInfoLoader();
            TcAdsSymbolInfoCollection adscollection =
infoloader.GetSymbols(true);

              /* unused code snipped out */
        }
    }
}
```

# 4. Appendix

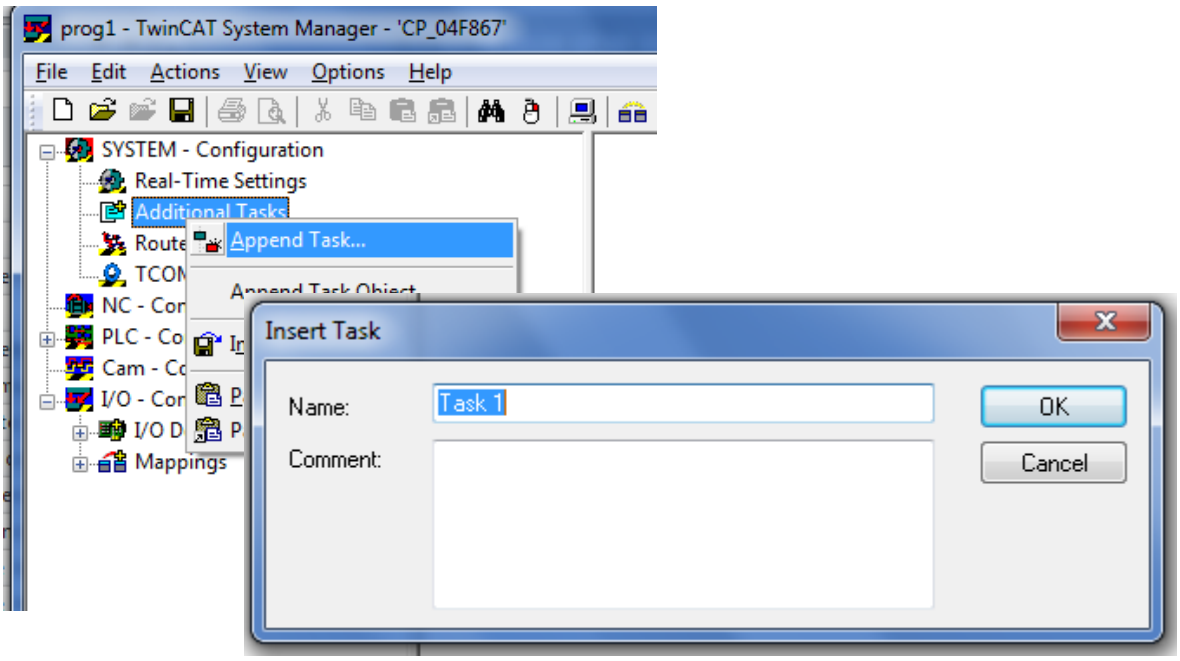
## 4.1. System manager Task

In addition to four PLC tasks per run-time system (and max. four PLC run-time systems), the TwinCAT system also supports further (non-PLC) software tasks which may possess I/O variables. These tasks are managed in the TwinCAT System Manager under the option ”Additional Tasks” (below ”**SYSTEM - Configuration**” in TwinCAT v2.9). These tasks can be used if a PLC is not available. Access to variables of these tasks can be gained directly from applications like TwinCAT OPC Server, Visual Basic, Delphi, VB.NET, VC++, C#.NET, etc.. e.g. via ADS-OCX, ADS-DLL or TcADS-DLL.

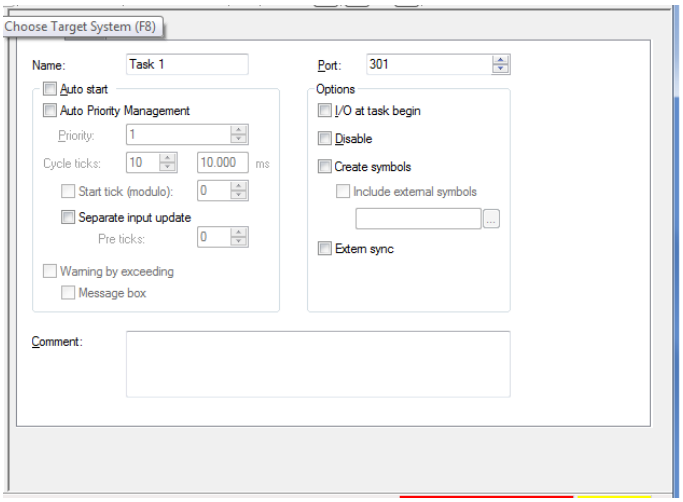
### Creating a user task

An additional task (user task) allows to refresh the I/O linked on the peripherals of the PC/PLC. The managing program is written in a higher language which can have access through the use of the DLL.

Under **SYSTEM-Configuration** , right mouse click on “Additional Tasks” and choose “Append Task” from the context menu.



On the right side, you have access to all parameters of this task.




- **Name**

Edits the internal name of the task

- **Port**

Defines the AMS port number of the task. This value must be specified! For some tasks the value is already set (e.g. for PLC tasks).

- **Auto-Start**

Causes the TwinCAT System Manager to create the start command for the task so that when restarting TwinCAT the task is automatically started with configured settings.

- **Priority**

Defines the priority of the Task within TwinCAT (you should ensure that priorities are not duplicated). The priority is only relevant when Auto-Start is selected.

- **Cycle Ticks**

Sets the cycle time in ticks (depends upon the pre-set TwinCAT Base Time of the task). The cycle time is only relevant when Auto-Start is selected.

- **Warning by exceeding**

Causes the TwinCAT to issue a warning when the pre-set task cycle time is exceeded.

- **Message box**

Outputs the warning (above) also as a message box.

- **I/O at task begin**

An I/O cycle is carried out at the beginning of the task.

- **Disable**

Allows occasional task-disablement, i.e. the task is ignored when generating the I/O information (e.g. during commissioning). The link information is, however, retained.

- **Create Symbols**

Allows access to variables of the corresponding task via ADS (e.g. from TwinCAT Scope View).

- **Extern Sync**

Is this option activated, this Task will be synchronized with a configured device with Master Sync Interrupt (e.g. a SERCOS card).

Tick “Auto Start” and “Create Symbols” to get things running.

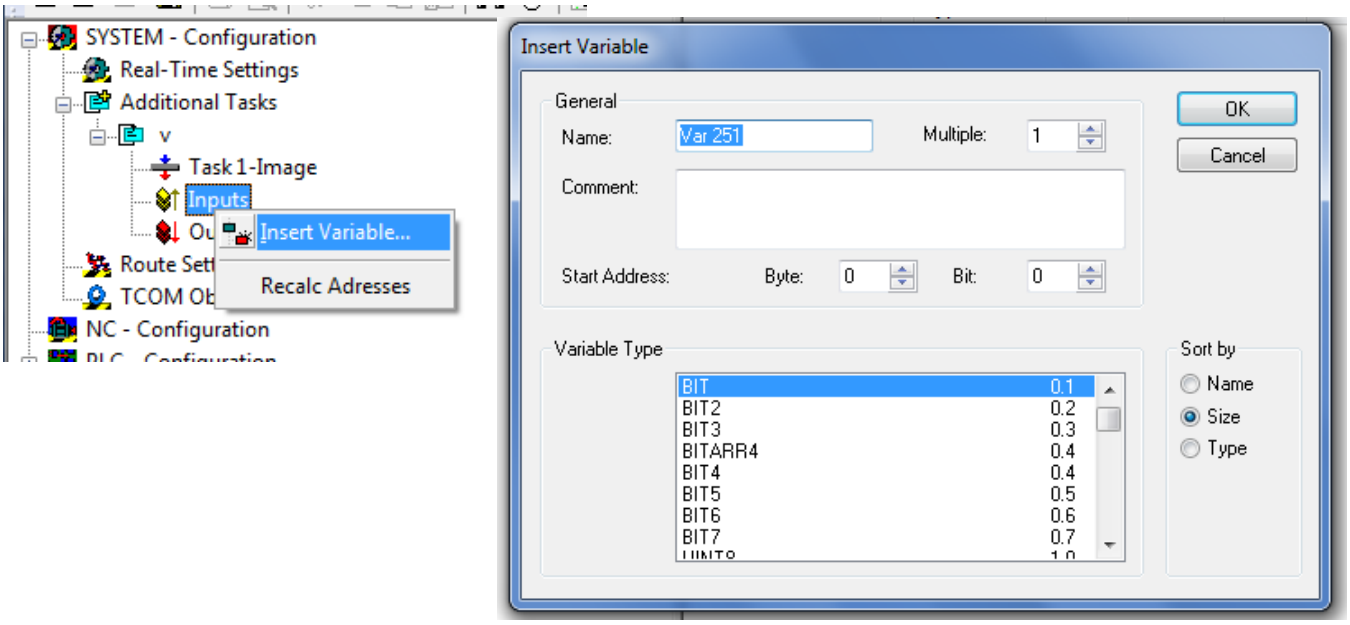
## Creating the variables

In and output variables can be added to process images. These can then be linked with the various I/O devices or variables of other tasks.

Variables assigned to the task process image can be added under the menu Inputs and Outputs sub-options.

***Note:*** Variables for PLC tasks cannot be added inside the System Manager. These variables have to be declared within the PLC project using TwinCAT PLC Control (via defined PLC process image address or *%Q** resp. *%I**)

In the newly created Task, right mouse click on “INPUTS|OUTPUTS” and choose “Insert Variables” from the context menu.




This will open a dialogue.

- **Name**

Defines the name of the variable.

- **Comment**

Defines an optional comment on the new variable.

- **Start Address**

Specifies the address of the variable in the task’s process image.

- **Multiple**

Multiple variables of the same type can be created and added with sequential addresses.

- **Variable Type**

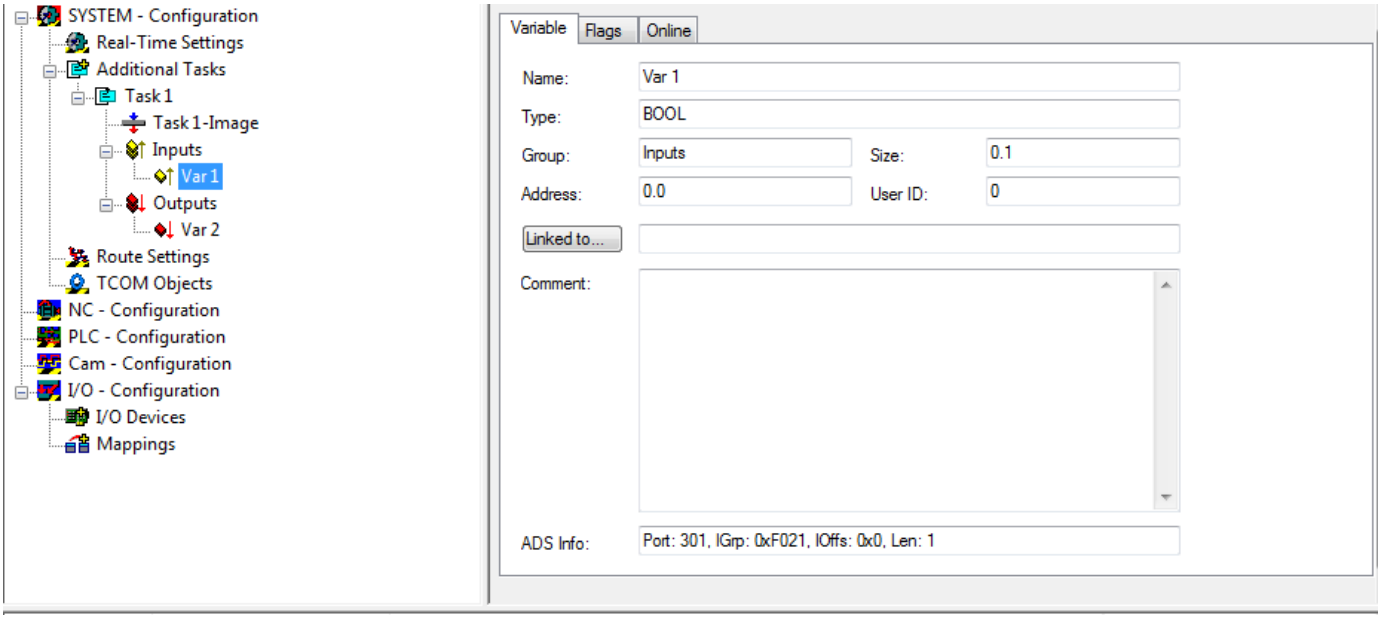
Lists all currently recognised data types in TwinCAT System Manager from which the type of the new variable(s) can be selected.

- **Sort by**

Allows the list of variable types to be sorted accordingly.

## Receiving and sending data

From this moment on you can read/write variables as if you were connected through a PLC program.




To connect through “VariableName” you have to use the fully qualified name for it, for instance:

- Var 1’s qualified name will be “TASK 1.INPUTS.Var 1”.
- Var 2’s qualified name would be “TASK 1.OUTPUS.Var 2”.

To connect through the use of “Indexgroup and index offset” we see,
on the right side panel “ADS Info”.
This provides information required to access selected variable via ADS
(e.g. TwinCAT Scope View, …). The information comprises Port Number, Index Group, Index Offset and Variable Length in bytes.

## 4.2. Debugging CE project form VS2005 and VS2008

At some point, it becomes essential to connect to the device from within Visual Studio, and debug the application online, running on the device hardware and not just an emulator.

After pressing F5 to deploy the application, it will prompt you to choose a device.
The Windows CE Device is the correct choice, but unless connection options are set correctly, you will only get the dreaded error "Unable to connect to device".
Before reading the rest of this post, refer to the MSDN article below.

Below are the needed steps to connect to your Beckhoff device.

### Copy the Windows CE debug files to the device

There are five files you need to copy to the Windows CE controller. The most important is ***ConmanClient2.exe***. Executing this file on the device enables the device to listen for an incoming connection from Visual Studio.

- Clientshutdown.exe
- ConmanClient2.exe
- CMaccept.exe
- eDbgTL.dll
- TcpConnectionA.dll

These files are located at the paths below in a standard Visual Studio 2005 SP1 or Visual Studio 2008 installation. You can copy the files to the device using FTP or by removing the compact flash card and inserting it into a USB reader.

***X86 Processor (CX1000 Series):***
C:\Program Files\Common Files\microsoft shared\CoreCon\1.0\Target \wce400\x86

***ARM Processor (CX9000 Series):***
C:\Program Files\Common Files\microsoft shared\CoreCon\1.0\Target\wce400 \armv4i

Note, these will only work for Visual Studio 2005 SP1 or higher.

## CoreConOverrideSecurity Registry Key

This step and also the next step will require that you have access to the device. You will need to plug a monitor and mouse into the controller, or you can use the CE
Remote Host terminal tool (*CERHOST*) and connect remotely into the device. The mouse and monitor route is usually the easiest option.

After copying and executing ConmanClient2.exe, Visual Studio will still not be able to connect to the device. The problem seems to be because of a missing registry key.
On the device, you can use the mouse and keyboard and go to **Start > Run > Regedit** to add this key that MSDN documents below.
*You can eliminate the CMaccept step by disabling security on the device. To do so, use the Remote Registry Editor to set* ***HLKM\System\CoreConOverrideSecurity = 1 DWORD*** *value. Disabling security exposes your device to malicious attack and is not recommended unless you have provided appropriate safeguards.*

*If this key does not exist, use the regedit menu to add a new DWORD key and set the value to 1.*

## Run ConmanClient2.exe on the Device

On the device, double click ConmanClient2.exe to start listening for a connection from Visual Studio. MSDN says you should use a command prompt, and this is works as well.
After executing ConmanClient2.exe at the command prompt, it will appear to just hang, with a flashing cursor.

This is not a problem; it is just running waiting for a connection from Visual Studio.
If you need to restart the connection, run ClientsShutdown.exe then ConmanClient2.exe.
CMaccept.exe should not make any difference since the registry key is set to 1.

## Configure Visual Studio Options

Visual Studio must be configured to connect to the device's IP Address.
Go to the **“Tools > Options > Device Tools > Devices”** menu to configure this setting.

## Deploy and debug the application

You can create and debug a .NET Windows CE Smart Device project much like any other type of application in Visual Studio.

Be sure to select Windows CE as the target and not Windows Mobile. You can use the 2.0 or 3.5 version of the Compact Framework. The framework will be deployed along with the application when you press F5.

Press F5 to deploy and debug the application. A dialog to select the Windows CE smart device will appear. If all goes well, the connection will be made to the Beckhoff device across the IP connection.

If Visual Studio cannot connect, it will give the error "Connect to Device Failed" error message. Review the steps outlined above, unfortunately there is not a lot of information about how to debug this problem. Make sure the IP address is set correctly and that you can ping the device.

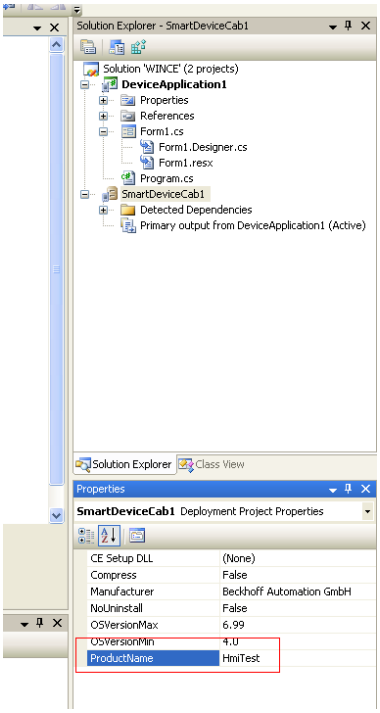
If all goes well, the application starts on the Windows CE device in debug mode.

## 4.3. Creating setup files for Windows CE using Visual Studio 2005

Visual Studio 2005 has great support for building Windows Mobile applications such as Pocket PC and Smartphone applications. But what happens after an application is completed and ready to be deployed onto the user's device? In Visual Studio 2005, deploying an application to a device is simply a matter of connecting that device to the development machine using ActiveSync and then pressing the F5 key. However, this method of deployment is not acceptable to end users. End users are accustomed to GUI setup applications that hide the details of the installation process.

### Packaging the CAB files

The first step toward deploying the project is to first package the application into a Cabinet (CAB) file. The CAB file can then be deployed to devices, where it is then expanded and installed. A CAB file is an executable archive file that contains your application, dependencies such as DLLs, resources, Help files, and any other files you want to include in it.




- Add a new project to your current solution by going to **File(Add Project…**.
  In the Project Types column, expand the **Other Project Types** node and select **Setup and Deployment**.
  Select the **Smart Device CAB Project** template.
  Name the project **C:\SmartDeviceCab1** and click OK.

- In Solution Explorer, click the **SmartDeviceCab1** project and set the **ProductName** property to **MySampleApp**.
  This will be the name of the folder that is created on the CE Device when you install the application.

- Right-click on **SmartDeviceCab1** in Solution Explorer and select **Properties**.
  Set the output file name to **Debug\ SmartDeviceCab1.cab**
  This will be the name and location of the CAB file.

- Under **Build (Configuration Manager…**in the Build column**)** tick the option so that the **CAB file project** is build automatically on each build of your project.

## 4.4. Marshaling

This doesn't work under CE framework!!!

*PLC-example Global Variables:*

```
VAR_GLOBAL
        varTest                 :       ST_CodeFunction;
END_VAR
```

The variable type ST_CodeFunction could be almost everything you can imagine, for instance:

*Struct-example ST_CodeFunction:*

```
TYPE ST_CodeFunction :
STRUCT                                          (*Handled in .NET as...*)
        bEnable         :       BOOL;           (*Byte sized boolean*)
        ActionStep      :       INT:=1;         (*INT16*)
        Name            :       STRING(20);     (*STRING*)
        Code            :       WORD:=0;        (*UINT16*)
        TimeOut         :       TIME:=t#1ms;    (*UINT32...TIME takes Dword size*)
        Position_X      :       REAL:=0;        (*Single*)
        Position_Y      :       REAL:=0;        (*Single*)
        Position_Z      :       REAL:=0;        (*Single*)
        Volume          :       DWORD:=0;       (*UINT32*)
END_STRUCT
END_TYPE
```

This would translate into:

*C#:*

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public class St_CodeFunction
{
    [MarshalAs(UnmanagedType.I1)]
    public Boolean bEnable;
    public Int16 ActionStep;
    //our string in PLC is 20 long + 1 ending-Char
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 21)]
    public String Name;
    public UInt16 Code;
    public UInt32 TimeOut;
    public Single Position_X;
    public Single Position_Y;
    public Single Position_Z;
    public UInt32 Volume;
}
```

## By Object

To read/write this variable in one Read/write ADS-call this would resolve to:

*C#:*

```
using TwinCAT.Ads;
using System.Runtime.InteropServices;

namespace MarshalingC
{
    public partial class Form1 : Form
    {
        TcAdsClient tcClient = new TcAdsClient();
        St_CodeFunction dataToRead;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            tcClient.Connect(801);
            try
            {
                dataToRead = (St_CodeFunction)tcClient.ReadAny(
                    tcClient.CreateVariableHandle(".varTest"),
                    typeof(St_CodeFunction));
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

For the marshaling you could use a “class” or a “struct” in .NET for our codeFunction example. But this only works for the outermost type, innermost sub-types must always be structs.

*VB.Net:*

```
Imports TwinCAT.Ads
Imports System.Runtime.InteropServices

Partial Public Class Form1
    Inherits Form

    Private tcClient As New TcAdsClient()
    Private dataToRead As St_CodeFunction

    Private Sub Form1_Load(sender As Object, e As EventArgs)
        tcClient.Connect(801)
        Try
            dataToRead = CType(
                tcClient.ReadAny(
                    tcClient.CreateVariableHandle(".varTest"),
                    GetType(St_CodeFunction)),
                St_CodeFunction)
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub
End Class
```

## By AdsStream

Instead of reading/writing each item from/to the stream, you could also opt to use the strength of marshaling.

*C#:*

```
using TwinCAT.Ads;
using System.Runtime.InteropServices;

namespace MarshalingC
{
    public partial class Form2 : Form
    {
        [StructLayout(LayoutKind.Sequential, Pack = 1)]
        public class St_CodeFunction
        {
            [MarshalAs(UnmanagedType.I1)]
            public Boolean bEnable;
            public Int16 ActionStep;
            //our string in PLC is 20 long + 1 ending-Char
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 21)]
            public String Name;
            public UInt16 Code;
            public UInt32 TimeOut;
            public Single Position_X;
            public Single Position_Y;
            public Single Position_Z;
            public UInt32 Volume;

            public Byte[] ToByteArray()
            {
                //Create the buffer
                byte[] buff = new byte[Marshal.SizeOf(typeof(St_CodeFunction))];
                //Tell the garbage collector to keep hands off
                GCHandle handle = GCHandle.Alloc(buff, GCHandleType.Pinned);
                //Marshall the structure
                Marshal.StructureToPtr(
                    this,
                    handle.AddrOfPinnedObject(),
                    false);
                handle.Free();
                return buff;
            }
        }

        TcAdsClient tcClient = new TcAdsClient();
        St_CodeFunction dataToRead;

        public Form2()
        {
            InitializeComponent();
        }
```

```
        private void Form2_Load(object sender, EventArgs e)
        {
            tcClient.Connect(801);
            try
            {
                //Create Buffer
                AdsStream adsStream =
                    new AdsStream(
                        new byte[Marshal.SizeOf(typeof(St_CodeFunction))]);
                //Read in the data
                int bytesRead =
                    tcClient.Read(
                    tcClient.CreateVariableHandle(".varTest"),
                    adsStream);
                //Make sure that the Garbage Collector doesn't move our buffer
                GCHandle handle =
                    GCHandle.Alloc(
                    adsStream.GetBuffer(),
                    GCHandleType.Pinned);
                //Marshal the bytes
                dataToRead =
                    (St_CodeFunction)Marshal.PtrToStructure(
                    handle.AddrOfPinnedObject(),
                    typeof(St_CodeFunction));
                //Give control of the buffer back to the GC
                handle.Free();

                dataToRead.Name = "Response from app";

                tcClient.Write(
                    tcClient.CreateVariableHandle(".varTest"),
                    new AdsStream(dataToRead.ToByteArray()));
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

*VB.Net:*

```
Imports TwinCAT.Ads
Imports System.Runtime.InteropServices

Partial Public Class Form2
    Inherits Form
    <StructLayout(LayoutKind.Sequential, Pack:=1)> _
    Public Class St_CodeFunction
        <MarshalAs(UnmanagedType.I1)> _
        Public bEnable As Boolean
        Public ActionStep As Int16
        'our string in PLC is 20 long + 1 ending-Char
        <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=21)> _
        Public Name As String
        Public Code As UInt16
        Public TimeOut As UInt32
        Public Position_X As Single
        Public Position_Y As Single
        Public Position_Z As Single
        Public Volume As UInt32

        Public Function ToByteArray() As Byte()
            'Create the buffer
            Dim buff As Byte() =
                New Byte(Marshal.SizeOf(
                    GetType(St_CodeFunction)) - 1) {}
            'Tell the garbage collector to keep hands off
            Dim handle As GCHandle =
                GCHandle.Alloc(buff, GCHandleType.Pinned)
            'Marshall the structure
            Marshal.StructureToPtr(
                Me,
                handle.AddrOfPinnedObject(),
                False)
            handle.Free()
            Return buff
        End Function
    End Class

    Private tcClient As New TcAdsClient()
    Private dataToRead As St_CodeFunction
```

```
    Private Sub Form2_Load(sender As Object, e As EventArgs)
        tcClient.Connect(801)
        Try
            'Create Buffer
            Dim adsStream As New AdsStream(
                New Byte(
                    Marshal.SizeOf(GetType(St_CodeFunction)) - 1) {})
            'Read in the data
            Dim bytesRead As Integer =
                tcClient.Read(
                    tcClient.CreateVariableHandle(".varTest"),
                    adsStream)
            'Make sure that the Garbage Collector doesn't move our buffer
            Dim handle As GCHandle =
                GCHandle.Alloc(adsStream.GetBuffer(), GCHandleType.Pinned)
            'Marshal the bytes
            dataToRead =
                DirectCast(
                    Marshal.PtrToStructure(
                        handle.AddrOfPinnedObject(),
                        GetType(St_CodeFunction)),
                    St_CodeFunction)
            'Give control of the buffer back to the GC
            handle.Free()

            dataToRead.Name = "Response from app"

            tcClient.Write(
                tcClient.CreateVariableHandle(".varTest"),
                New AdsStream(dataToRead.ToByteArray()))
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub
End Class
```

For a more detailed explanation, have a look at the “Extending the struct” sub-chapter.

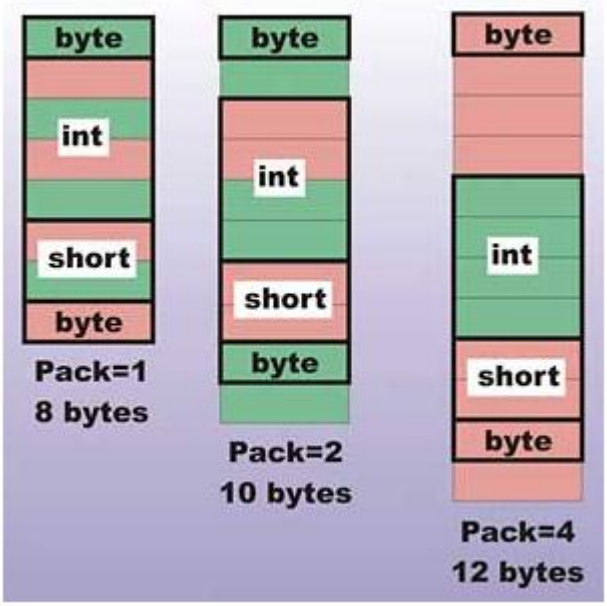
## Details

```
using System.Runtime.InteropServices;
```

This namespace provides a wide variety of members that support interoperating with unmanaged code.
One important attribute you could use of this namespace, would be the **`MarshalAs`** Attribute, which you use to specify how data is marshaled between managed and unmanaged memory.

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
```

This attribute lets you control the physical layout of the data fields of a class or structure.
**LayoutKind:** lets you control how the members of the object are laid out.
Sequential: The members are in the order in which they appear.
The members are laid out according to the packing specified in
The **StructLayoutAttribute.Pack**, and can be noncontiguous.
**Pack:** controls the alignment of data fields of a class or structure in memory.




A good explanation about marshaling and packsizes can be found at:
http://www.developerfusion.com/article/84519/mastering-structs-in-c/

Example:

**TC2:**
X86 -> 1Byte alignment `[StructLayout(LayoutKind.Sequential, Pack=1)]`
ARM -> 4Byte alignment `[StructLayout(LayoutKind.Sequential, Pack=4)]`

**TC3:** 8Byte alignment `[StructLayout(LayoutKind.Sequential, Pack=8 /*or 0*/)]`

```
[MarshalAs(UnmanagedType.I1)]
```

A Boolean in the IEC61131 would actually take up a full byte size:

I1:
A 1-byte signed integer. You can use this member to transform a Boolean value into a 1-byte, C-style **bool** (**true** = 1, **false** = 0).

```
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 21)]
```

Used for in-line, fixed-length character arrays that appears within a structure. The character type used with **ByValTStr** is determined by the **System.Runtime.InteropServices.CharSet** argument of the **System.Runtime.InteropServices.StructLayoutAttribute** applied to the containing structure. Always use the **MarshalAsAttribute.SizeConst** field to indicate the size of the array.

As a string in the PLC would take up his size + 1 byte (null terminated string), in our example 20 bytes string would take up 21 characters.

## Extending the struct

*PLC-example Global Variables:*

```
VAR_GLOBAL
        varTest2                :       ST_Receipt;
END_VAR
```

The variable type ST_CodeFunction could be almost everything you can imagine, for instance:

*Struct-example ST_Receipt:*

```
TYPE ST_Receipt :
STRUCT
        arrayData       :       ARRAY[1..20] OF ST_CodeFunction;
        strArray        :       ARRAY[1..20] OF STRING;
END_STRUCT
END_TYPE
```

*C#:*

```
        [StructLayout(LayoutKind.Sequential, Pack = 1)]
        public Class St_Receipt
        {
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = 20)]
            public St_CodeFunction[] ArrayData;
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = 20)]
            public String80[] StrArray;
        }

        [StructLayout(LayoutKind.Sequential, Pack = 1)]
        public struct String80
        {
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 81)]
            public string tekst;
        }
```

*VB.Net:*

```
       <StructLayout(LayoutKind.Sequential, Pack:=1)> _
        Public Class St_Receipt
            <MarshalAs(UnmanagedType.ByValArray, SizeConst:=20)> _
            Public ArrayData As St_CodeFunction()
            <MarshalAs(UnmanagedType.ByValArray, SizeConst:=20)> _
            Public StrArray As String80()
        End Class

        <StructLayout(LayoutKind.Sequential, Pack:=1)> _
        Public Structure String80
            <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=81)> _
            Public tekst As String
        End Structure
```

As you can’t marshal AND the size of the string-variable AND the size of the array, you have to split things up a bit, as shown in the above example.

*C#:*

```
using TwinCAT.Ads;
using System.Runtime.InteropServices;

namespace MarshalingC
{
    public partial class Form1 : Form
    {
         TcAdsClient tcClient = new TcAdsClient();
        St_Receipt dataToRead;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            tcClient.Connect(801);
            dataToRead = (St_Receipt)tcClient.ReadAny(
                tcClient.CreateVariableHandle(".varTest2"),
                typeof(St_Receipt));
        }
    }
}
```

*VB.Net:*

```
Imports TwinCAT.Ads
Imports System.Runtime.InteropServices

Partial Public Class Form1
    Inherits Form

    Private tcClient As New TcAdsClient()
    Private dataToRead As St_CodeFunction

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        tcClient.Connect(801)
        dataToRead = DirectCast(
            tcClient.ReadAny(
                tcClient.CreateVariableHandle(".varTest"),
                GetType(St_CodeFunction)),
            St_CodeFunction)
    End Sub
End Class
```