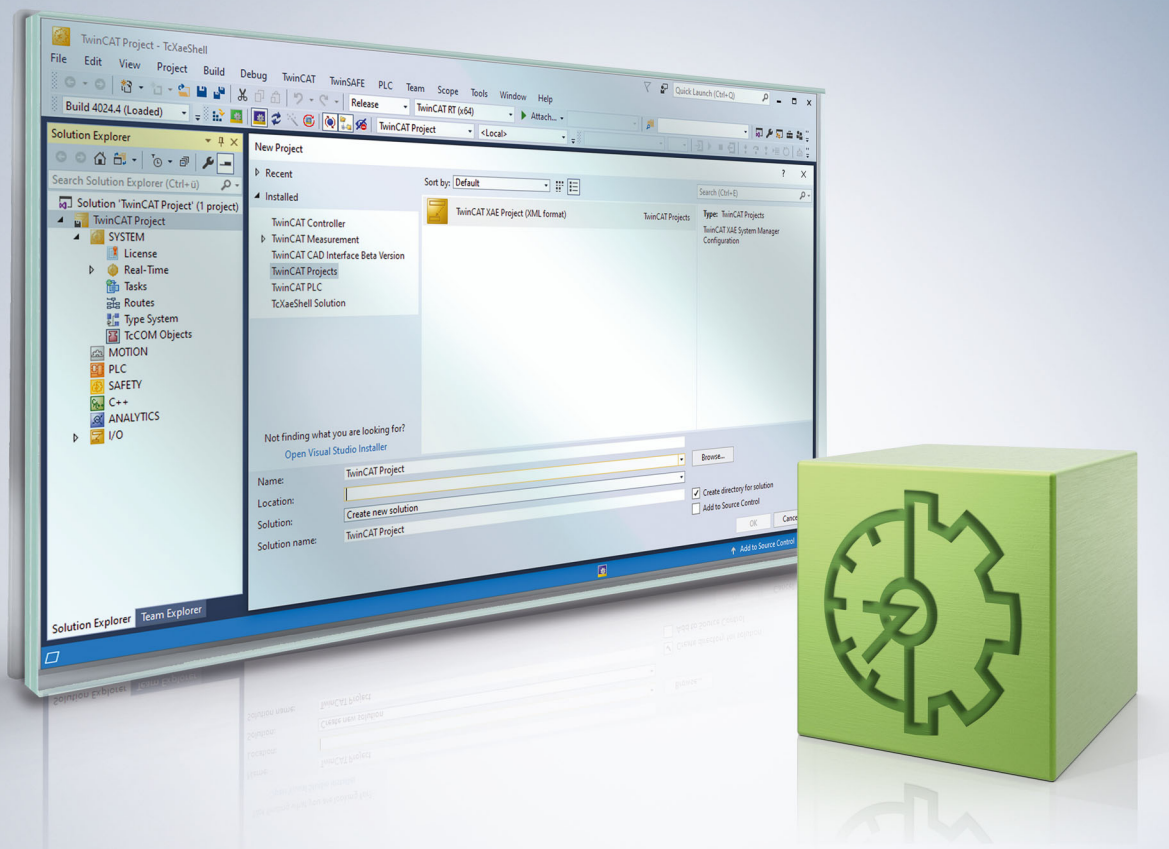
BECKHOFF New Automation Technology

Manual | EN

ADS-DLL .NET Samples

TwinCAT 3



2022-08-31 | Version: 1.2

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Integration

2.1 Linking into TwinCAT 3

Creating new project

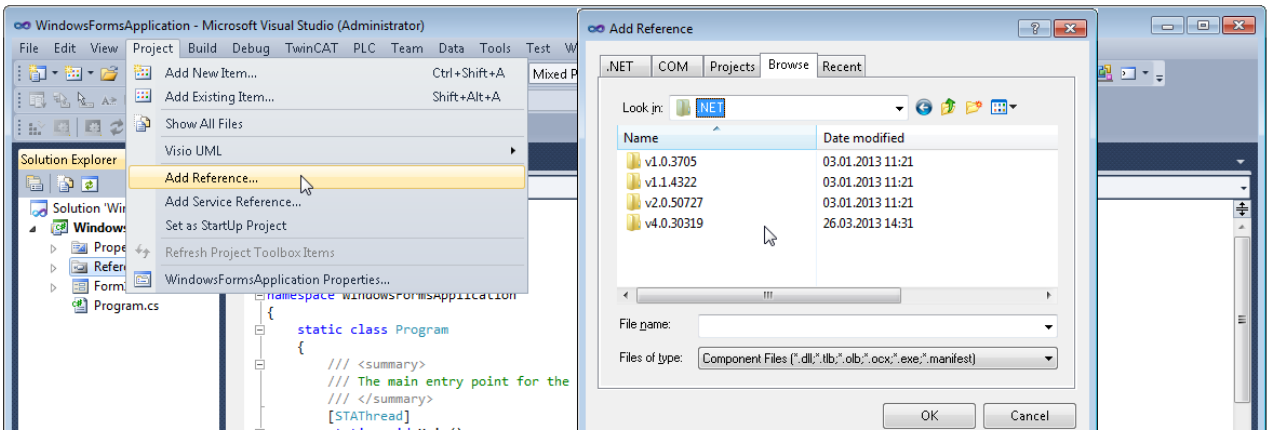
Start Microsoft Visual Studio and create new project (Windows Forms Application).

Adding reference

In order to select the TwinCAT.Ads class library you must choose the command **Add Reference...** under the **Project** menu . You will find the .Net Libraries per default in following TwinCAT folder:

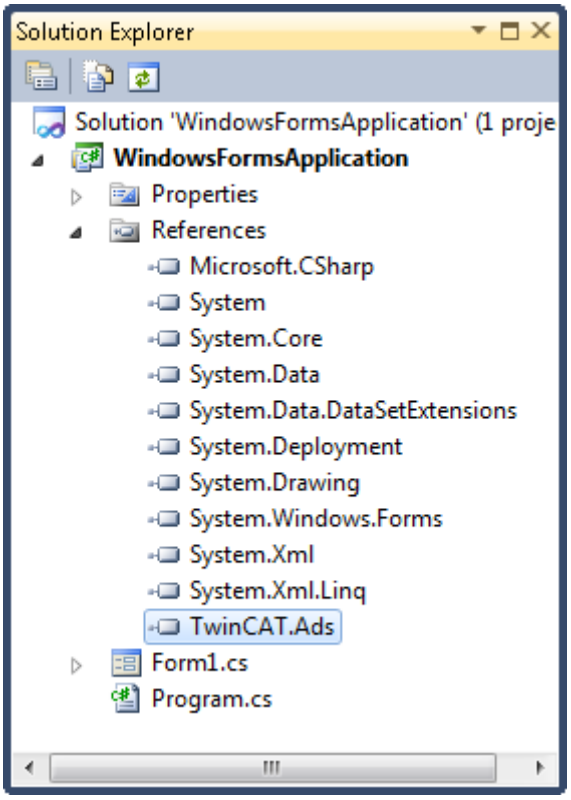
C:\TwinCAT\AdsApi\.NET

This opens the **Add Reference** dialog:



In this dialog you have to press the **Browse** button and select the file TwinCAT.Ads.dll for your used .NET Framework runtime.

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



All accessible types (classes,structures ...) belong to the namespace TwinCAT.Ads. Therefore one has to insert the following line at the beginning of the source :

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

This enables access to the types defined in TwinCAT.Ads without including the name of the namespace. The class TcAdsClient is the core of the TwinCAT.Ads class library and enables the user to communicate with an ads device. To begin with an instance of the class must be created. Then a connection to the ADS device is established by means of the Connect method.

3 Samples ADS .NET

Table 1: TwinCAT ADS .NET

Descripion
Linking into Microsoft Visual Studio [▶ 8]

Description	Minimun TwinCAT 3 Build	Visual C#
Sample 1: Accessing an array in the PLC [▶ 10]	TwinCAT 3.0 Build 3100	Sample01.zip
Sample 2: Transmitting a structure to the PLC [▶ 13]	TwinCAT 3.0 Build 3100	Sample02.zip
Sample 3: Event driven reading [▶ 16]	TwinCAT 3.0 Build 3100	Sample03.zip
Sample 4: Reading and writing of string variables [▶ 19]	TwinCAT 3.0 Build 3100	Sample04.zip
Sample 5: Reading and writing of DATE/TIME variables [▶ 22]	TwinCAT 3.0 Build 3100	Sample05.zip
Sample 6: Read PLC variable declaration [▶ 24]	TwinCAT 3.0 Build 3100	Sample06.zip
Sample 7: Reading and writing of PLC variables of any type [▶ 30]	TwinCAT 3.0 Build 3100	Sample07.zip
Sample 8: Detect state changes of TwinCAT router and PLC [▶ 37]	TwinCAT 3.0 Build 3100	Sample08.zip
Sample 9: ADS-Sum Command: Reading or writing several variables [▶ 39]	TwinCAT 3.0 Build 3100	Sample09.zip
Sample 10: Reserved		
Sample 11: Delete a handle of a PLC variable [▶ 43]	TwinCAT 3.0 Build 3100	Sample11.zip
Sample 12: Read flag synchronously from the PLC [▶ 43]	TwinCAT 3.0 Build 3100	Sample12.zip
Sample 13: Write flag synchronously into the PLC [▶ 44]	TwinCAT 3.0 Build 3100	Sample13.zip
Sample 14: Start/stop PLC [▶ 45]	TwinCAT 3.0 Build 3100	Sample14.zip
Sample 15: Access by variable name [▶ 46]	TwinCAT 3.0 Build 3100	Sample15.zip

3.1 Accessing an array in the PLC

Download

Requirements

Language / IDE	Unpack the example program
C# / Visual Studio	Sample01.zip

Task

The PLC contains an array that is to be read by the .Net application using a read command.

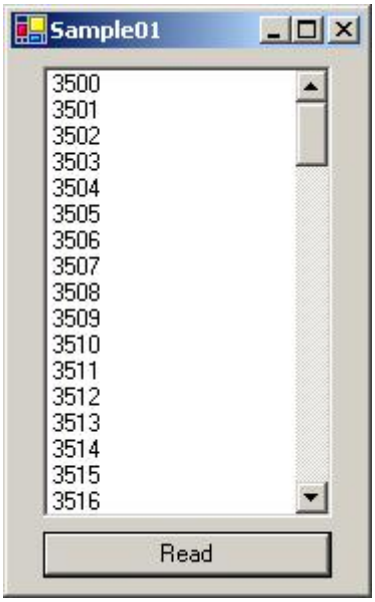
Description

The PLC contains an array of 100 elements of type integer (2 bytes). The array in the PLC is to be filled with the values from 3500 to 3599.

In the Form1_Load event method a new instance of the class TcAdsClient is created. Then the method TcAdsClient.Connect of the TcAdsClient object is called to establish a connection to the port 851. Finally the method TcAdsClient.CreateVariableHandle is used to fetch the handle of the PLC variable. When the program finishes, this handle is released in the Form1_Closing event method and the Dispose method of the TcAdsClient object is called.

When the user clicks the button on the form, the entire array is read from the PLC into the AdsStream dataStream by means of the TcAdsClient.Read method. The stream should be of the same size as the data in the SPS. Because we want to read 100 INTs (each 2 Bytes), the stream must be able to hold at least 2 * 100 Bytes.

The class System.IO.BinaryReader is necessary to read the individual fields of the array. First the position of the stream has to be set to 0. Then the individual fields of the array can be read by calling the method BinaryReader.ReadInt16 in a for loop. The position pointer of the stream is incremented automatically by the bytes (in this case 2) that have been read.



C# program

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using TwinCAT.Ads;
using System.IO;

namespace Sample01
{
public class Form1 : System.Windows.Forms.Form
{
private System.Windows.Forms.Button btnRead;
private System.ComponentModel.Container components = null;
private System.Windows.Forms.ListBox lbArray;

private int hVar;
private TcAdsClient tcClient;

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

```
 }

protected override void Dispose( bool disposing ) ...
private void InitializeComponent() ...

[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
// Create instance of class TcAdsClient
tcClient = new TcAdsClient();


// Connect to local PLC - Runtime 1 TwinCAT 3 Port=851
tcClient.Connect(851);

try
{
hVar = tcClient.CreateVariableHandle("MAIN.PLCVar");
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}

private void btnRead_Click(object sender, System.EventArgs e)
{
try
{
// AdsStream which gets the data
AdsStream dataStream = new AdsStream(100 * 2);
BinaryReader binRead = new BinaryReader(dataStream);

//read comlpete Array
tcClient.Read(hVar,dataStream);

lbArray.Items.Clear();
dataStream.Position = 0;
for(int i=0; i<100; i++)
{
lbArray.Items.Add(binRead.ReadInt16().ToString());
}
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
//enable resources
try
{
tcClient.DeleteVariableHandle(hVar);
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
tcClient.Dispose();
}
}
}
```

PLC program

```
PROGRAM MAIN
VAR
PLCVar : ARRAY [0..99] OF INT;
Index: BYTE;
```

```
END_VAR

FOR Index := 0 TO 99 DO
PLCVar[Index] := 3500 + INDEX;
END_FOR
```

3.2 Transmitting structures to the PLC

Download

Requirements

Language / IDE	Unpack the example program
C# / Visual Studio	Sample02.zip

Task

A structure is to be written into the PLC by the .NET application. The elements in the structure have different data types.

Description

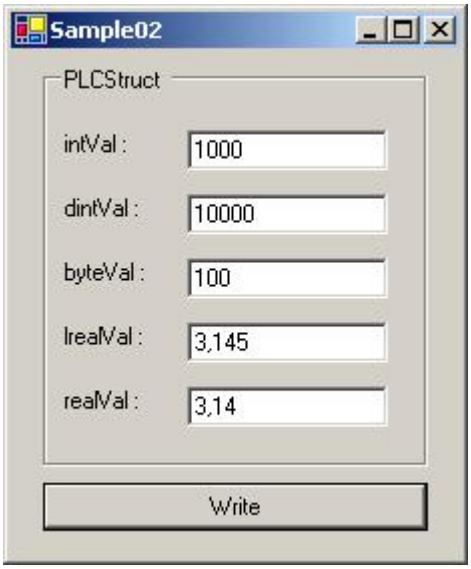
The structure we want to write into the PLC:

```
TYPE PLCStruct
STRUCT
intVal : INT; (*Offset 0*)
dintVal : DINT; (*Offset 2*)
byteVal : SINT; (*Offset 6*)
lrealVal : LREAL;(*Offset 7*)
realVal : REAL; (*Offset 15 --> Total size 19 Bytes*)
END_STRUCT
END_TYPE
```

In c# the structure would look like this:

```
public struct PLCStruct
{
public short intVal;
public int dintVal;
public byte byteVal;
public double lrealVal;
public float realVal;
}
```

This structure will not be created directly. Instead we will use a AdsStream as in Sample01. This object is initialized with a size of 19 bytes in the write event method. This is equivalent to the size of the data type PLCStruct in the SPS. A BinaryWriter is used to write the data from the textboxes to the stream. After setting the stream position to 0 the individual fields are written to the stream. Finally the total stream is written into the SPS with the help of TcAdsClient.Write.



C# Program

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using TwinCAT.Ads;

namespace Sample02
{
///
/// Summary description for Form1.
///
public class Form1 : System.Windows.Forms.Form
{
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.Button btnWrite;
private System.Windows.Forms.TextBox tbInt;
private System.Windows.Forms.TextBox tbDint;
private System.Windows.Forms.TextBox tbByte;
private System.Windows.Forms.TextBox tbLReal;
private System.Windows.Forms.TextBox tbReal;

private System.ComponentModel.Container components = null;

private int hVar;private TcAdsClient tcClient;

public Form1()
{
InitializeComponent();
}


protected override void Dispose( bool disposing )...

private void InitializeComponent()...

[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
```

```
 // Create instance of class TcAdsClient
tcClient = new TcAdsClient();

// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
tcClient.Connect(851);

try
{
hVar = tcClient.CreateVariableHandle("MAIN.PLCVar");
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}

private void btnWrite_Click(object sender, System.EventArgs e)
{
AdsStream dataStream = new AdsStream(19);
BinaryWriter binWrite = new BinaryWriter(dataStream);

dataStream.Position = 0;
try
{
//Fill stream according to the order with data from the text boxes
binWrite.Write(short.Parse(tbInt.Text));
binWrite.Write(int.Parse(tbDint.Text));
binWrite.Write(byte.Parse(tbByte.Text));
binWrite.Write(double.Parse(tbLReal.Text));
binWrite.Write(float.Parse(tbReal.Text));

//Write complete stream in the PLC
tcClient.Write(hVar,dataStream);
}
catch( Exception err)
{
MessageBox.Show(err.Message);
}
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
//Enable resources
try
{
tcClient.DeleteVariableHandle(hVar);
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
tcClient.Dispose();
}
}
}
```

PLC program

```
TYPE PLCStruct
STRUCT
intVal : INT;
dintVal : DINT;
byteVal : SINT;
lrealVal : LREAL;
realVal : REAL;
END_STRUCT
END_TYPE

PROGRAM MAIN
VAR
PLCVar : PLCStruct;
END_VAR
```

3.3 Event driven reading

Download

Requirements

Language / IDE	Unpack the example program
C# / Visual Studio	Sample03.zip

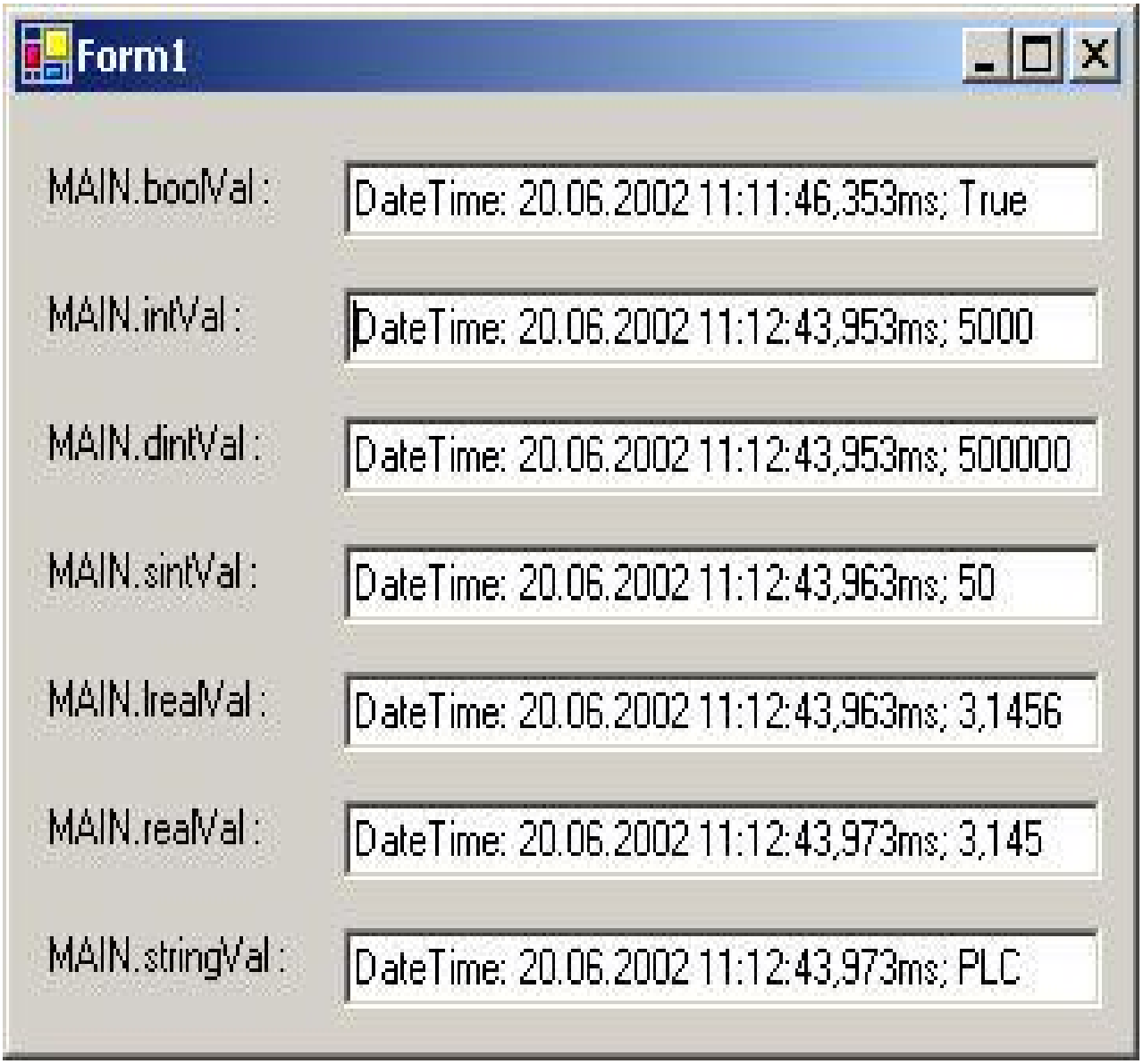
Task

There are 7 global variables in the PLC. Each of these PLC variables is of a different data type. The values of the variables should be read in the most effective manner, and the value with its timestamp is to be displayed on a form.

Description

In the form's load event, a connection to each of the PLC variables is created with the TcAdsClient.AddDeviceNotification() method. The handle for this connection is stored in a array.
The parameter TransMode specifies the type of data exchange. AdsTransMode.OnChange has been selected here. This means that the value of the PLC variable is only transmitted if its value within the PLC has changed (see the AdsTransMode data type). The parameter cycleTime indicates that the PLC is to check whether the corresponding variable has changed every 100 ms. MaxDelay allows to collect notification for a specified interval. If the maxDelay elapse, all notifications will be send at once.
When the PLC variable changes, the TcAdsClient.AdsNotification() event is called. The parameter e of the event handling method is of the type AdsNotificationEventArgs and contains the time stamp, the handle, the value and the control in which the value is to be displayed. The connections are released again in the closing event by means of the TcAdsClient.DeleteDeviceNotification() method. It is essential that you do this, since every connection established by TcAdsClient.AddDeviceNotification() uses resources.
You should also choose appropriate values for the cycle time, since too many write / read operations load the system so heavily that the user interface becomes much slower.

Advice: Don't use time intensive executions in callbacks (OnNotification()).



C# program

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using TwinCAT.Ads;

namespace Sample03
{
public class Form1 : System.Windows.Forms.Form
{
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label label9;
private System.Windows.Forms.Label label10;
private System.Windows.Forms.TextBox tbInt;
private System.Windows.Forms.TextBox tbDint;
```

```
private System.Windows.Forms.TextBox tbSint;
private System.Windows.Forms.TextBox tbLreal;
private System.Windows.Forms.TextBox tbReal;
private System.Windows.Forms.TextBox tbString;
private System.Windows.Forms.Label label11;
private System.Windows.Forms.TextBox tbBool;
private System.ComponentModel.Container components = null;

private TcAdsClient tcClient;
private int[] hConnect;
private AdsStream dataStream;
private BinaryReader binRead;

public Form1()
{
InitializeComponent();
}

protected override void Dispose( bool disposing ) ...

private void InitializeComponent() ...

[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
dataStream = new AdsStream(31);
//Encoding is set to ASCII, to read strings
binRead = new BinaryReader(dataStream, System.Text.Encoding.ASCII);
// Creaste instance of class TcAdsClient
tcClient = new TcAdsClient();

// PLC1 Port - TwinCAT 3=851
tcClient.Connect(851);

hConnect = new int[7];

try
{
hConnect[0] = tcClient.AddDeviceNotification("MAIN.boolVal",dataStream,0,1,
AdsTransMode.OnChange,100,0,tbBool);
hConnect[1] = tcClient.AddDeviceNotification("MAIN.intVal",dataStream,1,2,
AdsTransMode.OnChange,100,0,tbInt);
hConnect[2] = tcClient.AddDeviceNotification("MAIN.dintVal",dataStream,3,4,
AdsTransMode.OnChange,100,0,tbDint);
hConnect[3] = tcClient.AddDeviceNotification("MAIN.sintVal",dataStream,7,1,
AdsTransMode.OnChange,100,0,tbSint);
hConnect[4] = tcClient.AddDeviceNotification("MAIN.lrealVal",dataStream,8,8,
AdsTransMode.OnChange,100,0,tbLreal);
hConnect[5] = tcClient.AddDeviceNotification("MAIN.realVal",dataStream,16,4,
AdsTransMode.OnChange,100,0,tbReal);
hConnect[6] = tcClient.AddDeviceNotification("MAIN.stringVal",dataStream,20,11,
AdsTransMode.OnChange,100,0,tbString);

tcClient.AdsNotification += new AdsNotificationEventHandler(OnNotification);
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}

private void OnNotification(object sender, AdsNotificationEventArgs e)
{
DateTime time = DateTime.FromFileTime(e.TimeStamp);
e.DataStream.Position = e.Offset;
string strValue = "";

if( e.NotificationHandle == hConnect[0])
strValue = binRead.ReadBoolean().ToString();
else if( e.NotificationHandle == hConnect[1] )
strValue = binRead.ReadInt16().ToString();
else if( e.NotificationHandle == hConnect[2] )
strValue = binRead.ReadInt32().ToString();
else if( e.NotificationHandle == hConnect[3] )
strValue = binRead.ReadSByte().ToString();
```

```
else if( e.NotificationHandle == hConnect[4] )
strValue = binRead.ReadDouble().ToString();
else if( e.NotificationHandle == hConnect[5] )
strValue = binRead.ReadSingle().ToString();
else if( e.NotificationHandle == hConnect[6] )
{
strValue = new String(binRead.ReadChars(11));
}

((TextBox)e.UserData).Text = String.Format("DateTime: {0},{1}ms;
{2}",time,time.Millisecond,strValue);
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
try
{
for(int i=0; i<7; i++)
{
tcClient.DeleteDeviceNotification(hConnect[i]);
}
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
tcClient.Dispose();
}

}
}
```

PLC program

```
PROGRAM MAIN
VAR
boolVal : BOOL;
intVal : INT;
dintVal : DINT;
sintVal : SINT;
lrealVal : LREAL;
realVal : REAL;
stringVal : STRING(10);
END_VAR

PROGRAM MAIN
VAR
;
END_VAR
```

3.4 Reading and writing of string variables

Download

Requirements

Language / IDE	Unpack the sample program
C# / Visual Studio	Sample04.zip

Task

A .Net application should read a string from the PLC and write a string to the PLC.

Description

The PLC contains the string MAIN.text.

In the Form1_Load event method a new instance of the class TcAdsClient is created. Then the method TcAdsClient.Connect of the TcAdsClient object is called to establish a connection to the port 851. Finally the method TcAdsClient.CreateVariableHandle is used to fetch the handle of the PLC variable . When the program finishes, the Dispose method of the TcAdsClient object is called.

When the user clicks the "Read" button on the form, the string is read by means of the TcAdsClient.Read method and is displayed in the text box. When the user clicks the "Write" button ont the form, the string is wirtten to PLC and is displayed in the text box.

TwinCAT.Ads.NET version >= 1.0.0.10:

The classes AdsBinaryReader and AdsBinaryWriter can be used to read and write strings (see commented section in sample program). These classes derive from the BinaryReader/Writer classes. To read a string from the stream one has to call the method AdsBinaryReader.ReadPlcString. To write a string to the stream one has to call the method AdsBinaryWriter.WritePLCString. The length of the AdsStream must be equal to size of the string in the PLC (without terminating zero character). The length is passed to these method and have to be equal to the length of the string in the PLC (without the terminating zero character).

C# program

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using TwinCAT.Ads;
using System.IO;

namespace Sample04
{
public class Form1 : System.Windows.Forms.Form
{
private System.ComponentModel.Container components = null;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.Button btnRead;
private System.Windows.Forms.Button btnWrite;
private System.Windows.Forms.Label label1;
private TcAdsClient adsClient;
private int varHandle;

public Form1()
{
InitializeComponent();
}

protected override void Dispose( bool disposing )
{
...
}

private void InitializeComponent()
{
...
}
```

```
[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
try
{
adsClient = new TcAdsClient();

// PLC1 Port - TwinCAT 3=851
tcClient.Connect(851);

varHandle = adsClient.CreateVariableHandle("MAIN.text");
}
catch( Exception err)
{
MessageBox.Show(err.Message);
}
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
adsClient.Dispose();
}

private void btnRead_Click(object sender, System.EventArgs e)
{
try
{
//length of the stream = length of string in sps + 1
AdsStream adsStream = new AdsStream(31);
BinaryReader reader = new BinaryReader(adsStream,System.Text.Encoding.ASCII);

int length = adsClient.Read(varHandle, adsStream);
string text = new string(reader.ReadChars(length));
//necessary if you want to compare the string to other strings
//text = text.Substring(0,text.IndexOf('\0'));
textBox1.Text = text;
}
catch(Exception err)
{
MessageBox.Show(err.Message)
}
}

private void btnWrite_Click(object sender, System.EventArgs e)
{
try
{
//length of the stream = length of string + 1
AdsStream adsStream = new AdsStream(textBox1.Text.Length+1);
BinaryWriter writer = new BinaryWriter(adsStream, System.Text.Encoding.ASCII);
writer.Write(textBox1.Text.ToCharArray());
//add terminating zero
writer.Write('\0');
adsClient.Write(varHandle,adsStream);
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}


/*From version 1.0.0.10 and higher the classes AdsBinaryReader and AdsBinaryWriter
can be used to read and write strings
private void btnRead_Click(object sender, System.EventArgs e)
{
try
{
AdsStream adsStream = new AdsStream(30);
AdsBinaryReader reader = new AdsBinaryReader(adsStream);
adsClient.Read(varHandle, adsStream);
textBox1.Text = reader.ReadPlcString(30);
}
catch(Exception err)
```

```
{
MessageBox.Show(err.Message);
}
}

private void btnWrite_Click(object sender, System.EventArgs e)
{
try
{
AdsStream adsStream = new AdsStream(30);
AdsBinaryWriter writer = new AdsBinaryWriter(adsStream);
writer.WritePlcString(textBox1.Text, 30);
adsClient.Write(varHandle, adsStream);
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}
*/

}
}
```

PLC program

```
PROGRAM MAIN
VAR
text : STRING[30] := 'hello';
END_VAR
```

3.5 Reading and writing of TIME/DATE variables

Download

Requirements

Language / IDE	Unpack the example program
C# / Visual Studio	Sample05.zip

Task

A .Net application should read and write a date and a time.

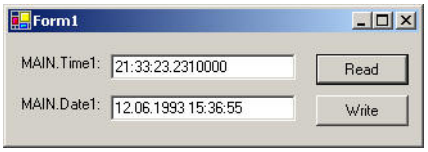
Description

The PLC contains the TIME variable MAIN.Time1 and the DT variable MAIN.Date1.

In the Form1_Load event method a new instance of the class TcAdsClient is created. Then the method TcAdsClient.Connect of the TcAdsClient object is called to establish a connection to the port 851. Finally the method TcAdsClient.CreateVariableHandle is used to fetch the handle of the PLC variables . When the program finishes, the Dispose method of the TcAdsClient object is called.

When the user clicks the "Read" button on the form, the variables MAIN.Time1 and MAIN.Date1 are read by means of the TcAdsClient.Read method. The class AdsBinaryReader, that derives from BinaryReader, is used to read the time and date from the PLC. The method AdsBinaryReader.ReadPlcTIME reads the time from the AdsStream and converts it to the .NET type TimeSpan. The method AdsBinaryReader.ReadPlcDATE reads the date from the AdsStream and converts it to the .NET type DateTime.

When the user clicks the "Write" button ont the form, the variables MAIN.Time1 and MAIN.Date1 are wirtten to the PLC. The class AdsBinaryWriter, that derives from BinaryWriter, is used to write the time and date to the PLC. The method AdsBinaryWriter.WritePlcType writes the .NET types TimeSpan and DateTime in the PLC format for time and date types to the AdsStream.



C# program

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using TwinCAT.Ads;
using System.IO;

namespace Sample04
{
public class Form1 : System.Windows.Forms.Form
{
private System.ComponentModel.Container components = null;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.Button btnRead;
private System.Windows.Forms.Button btnWrite;
private System.Windows.Forms.Label label1;
private TcAdsClient adsClient;
private int[] varHandles;

public Form1()
{
InitializeComponent();
}

protected override void Dispose( bool disposing )
{
...
}

private void InitializeComponent()
{
...
}

[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
try
{
adsClient = new TcAdsClient();

// PLC1 Port - TwinCAT 3 = 851
adsClient.Connect(851);
varHandles = new int[2];
varHandles[0] = adsClient.CreateVariableHandle("MAIN.Time1");
varHandles[1] = adsClient.CreateVariableHandle("MAIN.Date1");
}
catch( Exception err)
{
MessageBox.Show(err.Message);
}
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
```

```
adsClient.Dispose();
}

private void btnRead_Click(object sender, System.EventArgs e)
{
try
{
AdsStream adsStream = new AdsStream(4);
AdsBinaryReader reader = new AdsBinaryReader(adsStream);
adsClient.Read(varHandles[0], adsStream);
textBox1.Text = reader.ReadPlcTIME().ToString();
adsStream.Position = 0;
adsClient.Read(varHandles[1], adsStream);
textBox2.Text = reader.ReadPlcDATE().ToString();
}
catch(Exception err)
{
MessageBox.Show(err.Message)
}
}

private void btnWrite_Click(object sender, System.EventArgs e)
{
try
{
AdsStream adsStream = new AdsStream(4);
AdsBinaryWriter writer = new AdsBinaryWriter(adsStream);
writer.WritePlcType(TimeSpan.Parse(textBox1.Text));
adsClient.Write(varHandles[0], adsStream);

adsStream.Position = 0;
writer.WritePlcType(DateTime.Parse(textBox2.Text));
adsClient.Write(varHandles[1], adsStream)
}
catch(Exception err)
{
MessageBox.Show(err.Message);
}
}
}
}
```

PLC program

```
PROGRAM MAIN
VAR
Time1:TIME := T#21h33m23s231ms;
Date1:DT:=DT#1993-06-12-15:36:55.40;
END_VAR
```

3.6 Read PLC variable declaration

Download

Requirements

Language / IDE	Unpack the example program
C# / Visual Studio	Sample06.zip

Task

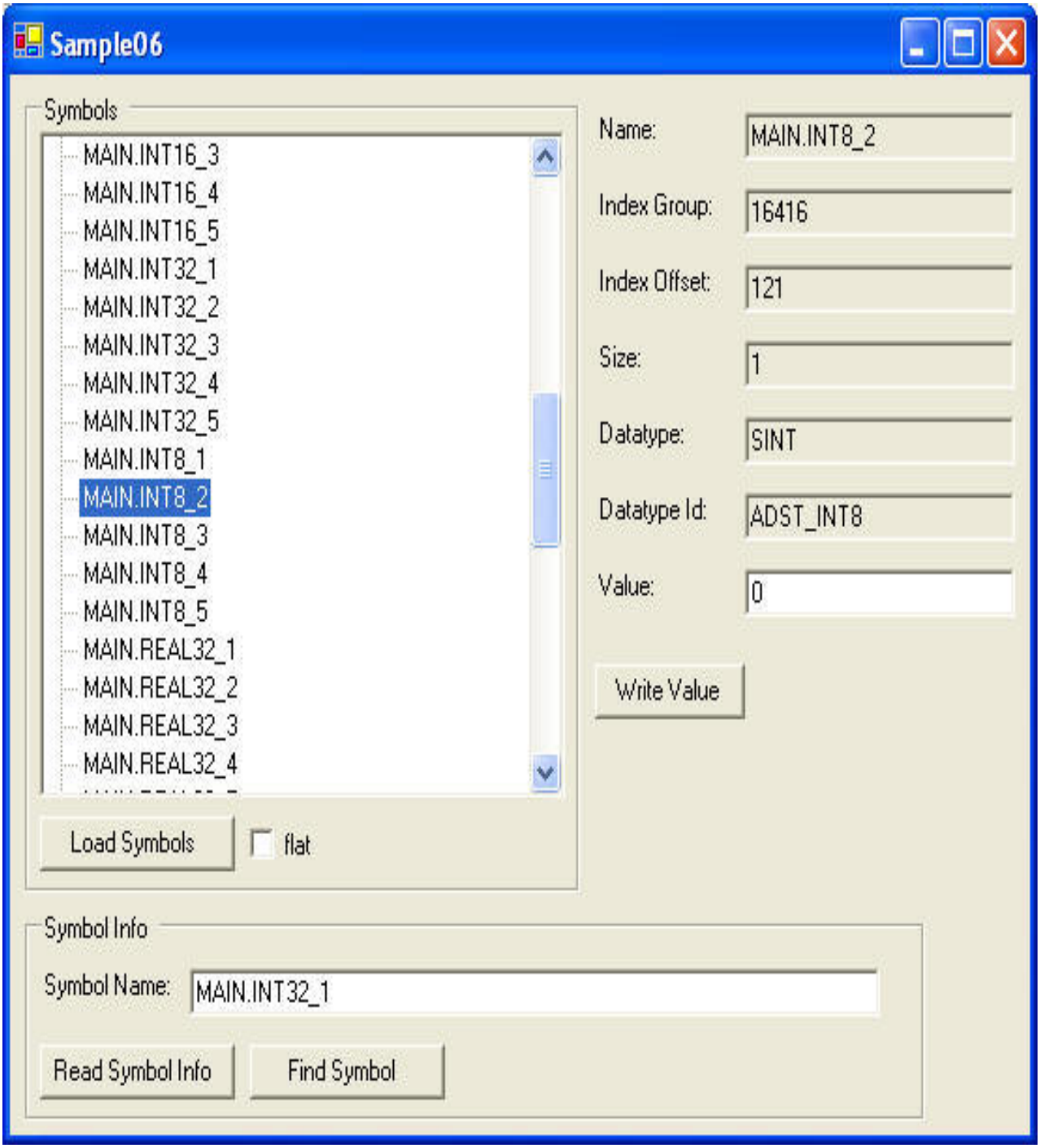
All variables that are declared in the PLC should be displayed in a tree view.

Description

In the Form1_Load method an instance of the TcAdsSymbolInfoLoader is created with a call to TcAdsClient.CreateSymbolInfoLoader. This class is responsible for loading the symbol information from the PLC. By clicking the *Load Symbols* button the symbols are loaded by means of the TcAdsSymbolInfoLoader.GetFirstSymbol method. This method returns the first loaded symbol as a TcAdsSymbolInfo object. The methods TcAdsSymbolInfo.NextSymbol und TcAdsSymbolInfo.FirstSubSymbol are used to iterate over the symbols and to display the symbols hierarchically in the tree view. If the check box *flat* is checked, the symbols are displayed in a flat list. In this case foreach is used to enumerate over the symbols loaded by the TcAdsSymbolInfoLoader object. This includes the sub symbols.

By selecting a tree view item the edit boxes are filled with the symbol information. Additionally the value of the variable is read with the help of TcAdClient.ReadSymbol and is displayed in the value edit box. To write a value to a variable one has to click the *Write Value* button, which leads to a call of TcAdsCient.WriteSymbol.

To read the symbol information for a specific variable, one can either click the *Read Symbol Info* or the *Find Symbol* button. In the first case the method TcAdsClient.ReadSymbolInfo is called. This leads to an ADS call, to load the information for this symbol from the PLC. In the second case the method TcAdsSymbolInfoLoader.FindSymbol is called, to search for the symbol in the list of the previous loaded symbols. If no symbols have been loaded before, all symbols are loaded from the PLC.



C# Programm

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using TwinCAT.Ads;


namespace Sample06
{
/// <summary>
/// Summary description for Form1.
/// </summary>
public class Form1 : System.Windows.Forms.Form
```

```
{
private System.Windows.Forms.TreeView treeViewSymbols;
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Button btnLoad;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.TextBox tbDatatype;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox tbIndexGroup;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox tbIndexOffset;
private System.Windows.Forms.Button btnFindSymbol;
private System.Windows.Forms.Button btnReadSymbolInfo;
private System.Windows.Forms.GroupBox groupBox2;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.TextBox tbDatatypeId;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.TextBox tbValue;
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components = null;
private TcAdsClient adsClient;
private System.Windows.Forms.TextBox tbSymbolname;
private System.Windows.Forms.Label label9;
private System.Windows.Forms.TextBox tbSize;
private System.Windows.Forms.Button btnWrite;
private TcAdsSymbolInfoLoader symbolLoader;
private System.Windows.Forms.TextBox tbName;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.CheckBox cbFlat;
private ITcAdsSymbol currentSymbol = null;

public Form1()
{
InitializeComponent();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
if( disposing )
{
if (components != null)
{
components.Dispose();
}
}
base.Dispose( disposing );
}

#region Windows Form Designer generated code
...
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
try
{
adsClient = new TcAdsClient();

// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
adsClient.Connect(851);

symbolLoader = adsClient.CreateSymbolInfoLoader();
}
catch(Exception err)
{
MessageBox.Show(err.Message);
```

```
 }
}

private void btnLoad_Click(object sender, System.EventArgs e)
{
treeViewSymbols.Nodes.Clear();

if( !cbFlat.Checked )
{
TcAdsSymbolInfo symbol = symbolLoader.GetFirstSymbol(true);
while( symbol != null )
{
treeViewSymbols.Nodes.Add(CreateNewNode(symbol));
symbol = symbol.NextSymbol;
}
}
else
{
foreach( TcAdsSymbolInfo symbol in symbolLoader )
{
TreeNode node = new TreeNode(symbol.Name);
node.Tag = symbol;
treeViewSymbols.Nodes.Add(node);
}
}
}

private void btnReadSymbolInfo_Click(object sender, System.EventArgs e)
{
try
{
ITcAdsSymbol symbol = adsClient.ReadSymbolInfo(tbSymbolname.Text);
if( symbol == null)
{
MessageBox.Show("Symbol " + tbSymbolname.Text + " not found");
return;
}
SetSymbolInfo(symbol);
}
catch( Exception err )
{
MessageBox.Show(err.Message);
}
}

private void btnFindSymbol_Click(object sender, System.EventArgs e)
{
try
{
ITcAdsSymbol symbol = symbolLoader.FindSymbol(tbSymbolname.Text);
if( symbol == null)
{
MessageBox.Show("Symbol " + tbSymbolname.Text + " not found");
return;
}
SetSymbolInfo(symbol);
}
catch( Exception err )
{
MessageBox.Show(err.Message);
}
}

private void btnWrite_Click(object sender, System.EventArgs e)
{
try
{
if( currentSymbol != null )
adsClient.WriteSymbol(currentSymbol, tbValue.Text);
}
catch(Exception err)
{
MessageBox.Show("Unable to write Value. " + err.Message);
}
}

private void treeViewSymbols_AfterSelect(object sender, System.Windows.Forms.TreeViewEventArgs e)
{
if( e.Node.Text.Length > 0 )
{
```

```
if( e.Node.Tag is TcAdsSymbolInfo )
{
SetSymbolInfo((ITcAdsSymbol)e.Node.Tag);
}
}
}

private TreeNode CreateNewNode(TcAdsSymbolInfo symbol)
{
TreeNode node = new TreeNode(symbol.Name);

node.Tag = symbol;
TcAdsSymbolInfo subSymbol = symbol.FirstSubSymbol;
while( subSymbol != null )
{
node.Nodes.Add(CreateNewNode(subSymbol));
subSymbol = subSymbol.NextSymbol;
}
return node;
}

private void SetSymbolInfo(ITcAdsSymbol symbol)
{
currentSymbol = symbol;
tbName.Text = symbol.Name.ToString();
tbIndexGroup.Text = symbol.IndexGroup.ToString();
tbIndexOffset.Text = symbol.IndexOffset.ToString();
tbSize.Text = symbol.Size.ToString();
tbDatatype.Text = symbol.Type;
tbDatatypeId.Text = symbol.Datatype.ToString();
try
{
tbValue.Text = adsClient.ReadSymbol(symbol).ToString();
}
catch( AdsDatatypeNotSupportedException err )
{
tbValue.Text = err.Message;
}
catch(Exception err)
{
MessageBox.Show("Unable to read Symbol Info. " + err.Message);
}
}
}
}
```

PLC program

```
PROGRAM MAIN
VAR
REAL32_1 AT %MB0 : REAL; (* 1 *)
REAL32_2 AT %MB4 : REAL; (* 2 *)
REAL32_3 AT %MB8 : REAL; (* 3 *)
REAL32_4 AT %MB12: REAL; (* 4 *)
REAL32_5 AT %MB16: REAL; (* 5 *)

REAL64_1 AT %MB20 : LREAL; (* 6 *)
REAL64_2 AT %MB28 : LREAL; (* 7 *)
REAL64_3 AT %MB36 : LREAL; (* 8 *)
REAL64_4 AT %MB44 : LREAL; (* 9 *)
REAL64_5 AT %MB52 : LREAL; (* 10 *)

INT32_1 AT %MB60 : DINT; (* 11 *)
INT32_2 AT %MB64 : DINT; (* 12 *)
INT32_3 AT %MB68 : DINT; (* 13 *)
INT32_4 AT %MB72 : DINT; (* 14 *)
INT32_5 AT %MB76 : DINT; (* 15 *)

UINT32_1 AT %MB80 : UDINT; (* 16 *)
UINT32_2 AT %MB84 : UDINT; (* 17 *)
UINT32_3 AT %MB88 : UDINT; (* 18 *)
UINT32_4 AT %MB92 : UDINT; (* 19 *)
UINT32_5 AT %MB96 : UDINT; (* 20 *)

INT16_1 AT %MB100 : INT; (* 21 *)
INT16_2 AT %MB102 : INT; (* 22 *)
```

```
INT16_3 AT %MB104 : INT; (* 23 *)
INT16_4 AT %MB106 : INT; (* 24 *)
INT16_5 AT %MB108 : INT; (* 25 *)

UINT16_1 AT %MB110 : UINT; (* 26 *)
UINT16_2 AT %MB112 : UINT; (* 27 *)
UINT16_3 AT %MB114 : UINT; (* 28 *)
UINT16_4 AT %MB116 : UINT; (* 29 *)
UINT16_5 AT %MB118 : UINT; (* 30 *)

INT8_1 AT %MB120 : SINT; (* 31 *)
INT8_2 AT %MB121 : SINT; (* 32 *)
INT8_3 AT %MB122 : SINT; (* 33 *)
INT8_4 AT %MB123 : SINT; (* 34 *)
INT8_5 AT %MB124 : SINT; (* 35 *)

UINT8_1 AT %MB125 : USINT; (* 36 *)
UINT8_2 AT %MB126 : USINT; (* 37 *)
UINT8_3 AT %MB128 : USINT; (* 38 *)
UINT8_4 AT %MB129 : USINT; (* 39 *)
UINT8_5 AT %MB130 : USINT; (* 40 *)

BOOL_1 AT %MX131.0 : BOOL; (* 41 *)
BOOL_2 AT %MX131.1 : BOOL; (* 42 *)
BOOL_3 AT %MX131.2 : BOOL; (* 43 *)
BOOL_4 AT %MX131.3 : BOOL; (* 44 *)
BOOL_5 AT %MX131.4 : BOOL; (* 45 *)

ARRAY_1 : ARRAY[1 .. 10] OF SINT; (* 46 *)
ARRAY_2 : ARRAY[1 .. 10] OF INT; (* 47 *)
ARRAY_3 : ARRAY[1 .. 10] OF DINT; (* 48 *)
ARRAY_4 : ARRAY[1 .. 10] OF LREAL;(* 49 *)
ARRAY_5 : ARRAY[1 .. 10] OF BOOL; (* 50 *)

STRING_1 : STRING(20);
END_VAR
```

3.7 Reading and writing of PLC variables of any type (ReadAny, WriteAny)

Download

Requirements

Language / IDE	Unpack the example program
C# / Visual Studio	Sample07.zip

Task

Read and Write variables of any type with the help of the ReadAny and WriteAny methods.

Description

ReadAny

In the event method btnRead_Click the method TcAdsClient.ReadAny is used to read a variable by handle:

```
public object ReadAny(int variableHandle, Type type)
public object ReadAny(int variableHandle, Type type, int[] args)
```

ReadAnyReadAny

The type of the variable is passed to the method in the parameter **type**. In case the method was successfull, the read data will be returned as a object. The type of the object is equal to the type passed in the parameter **type**. Because some data types(arrays and strings) need additional information, an overload of the method ReadAny exists, that takes an additional parameter **args**. E.g. with strings one has to pass an integer array of the length 1. Full list of supported types can be found in the documentation of the overloaded method.

Example:

A PLC variable of the type ARRAY[0..3] OF DINT should be read:

```
int hArr;
int[] arr;

hArr = adsClient.CreateVariableHandle(".arr")
arr = (int[]) adsClient.ReadAny(hArr, typeof(int[]), new int[] {4})

...
adsClient.DeleteVariableHandle(hArr)
```

WriteAny

In the event method btnWrite_Click the method TcAdsClient.WriteAny is used to write to a variable by handle:

```
public void WriteAny(int variableHandle, object value)
public void WriteAny(int variableHandle, object value, int[] args)
```

WriteAnyWriteAny

The parameter **value** is a reference to the object, that should be written to the PLC variable. Full list of supported types of the object **value** can be found in the documentation of the overloaded method.

Example:

A PLC variable of the type ARRAY[0..3] OF DINT should be written:

```
int hArr;
int[] arr = new int[]{1,2,3,4}

hArr = adsClient.CreateVariableHandle(".arr")
adsClient.WriteAny(hArr, arr)

...
adsClient.DeleteVariableHandle(hArr)
```

Reading and writing of structures:

(not possible with the Compact Framework(CE))

To be able to read or write PLC structures the memory layout of the structure or class in .NET must be the same as in the PLC. The layout of a structure or class can be specified with the attribute StructLayoutAttribute. The LayoutKind must be set to LayoutKind.Sequential and the pack must be set to 1. Therefore the class SimpleStruct is defined as followed:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public class SimpleStruct
{
public double lrealVal;
public int dintVal1;
}
```

If arrays, strings or boolean values are define the class, one has to specify how these fields should be marshalled. This is accomplished with help of the MarshalAs attribute. Because arrays and strings do not have a fixed length in .NET, the property SizeConst is necessary for arrays and strings. It is not possible to marshal multidimensional arrays or arrays of structures with the :NET Framework 1.1. Multidimensional arrays in the PLC must be mapped to one dimensional arrays in .NET.

In the example the MarshalAsAttribute is used in the class ComplexStruct:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public class ComplexStruct
{
public short intVal;
//specifies how .NET should marshal the array
//SizeConst specifies the number of elements the array has.
[MarshalAs(UnmanagedType.ByValArray, SizeConst=4)]
public int[] dintArr = new int[4];
[MarshalAs(UnmanagedType.I1)]
public bool boolVal;
public byte byteVal;
//specifies how .NET should marshal the string
//SizeConst specifies the number of characters the string has.
//'(inclusive the terminating null ).
[MarshalAs(UnmanagedType.ByValTStr, SizeConst=6)]
public string stringVal = "";
public SimpleStruct simpleStruct1 =new SimpleStruct();
}
```

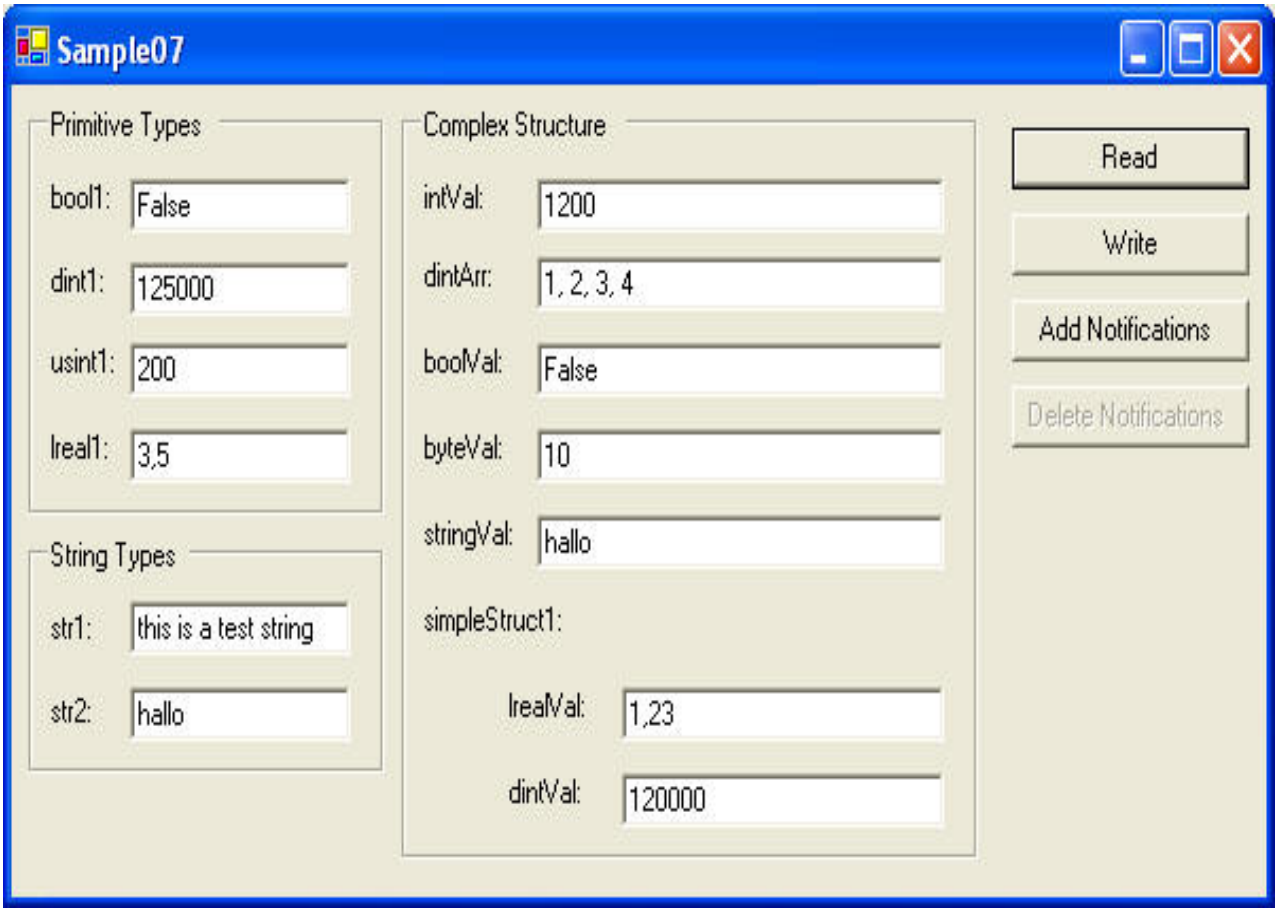
Register ADS notifcations

In the event method btnAddNotifications_Click the method AddDeviceNotificationEx is used to register notifications for a PLC variable. If the value of a variable changes the event AdsNotificationEx is fired. The difference to the event AdsNotification, is that the value of the variable is stored in an object instead of in an AdsStream. Therefore one has to pass the type of the object to the method AddDeviceNotificationEx:

```
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.dint1", AdsTransMode.OnChange, 100,
0, tbDint1, typeof(int)));
```

As user object the textbox that should display the value is passed. If the event is fired, the event method **adsClient_AdsNotificationEx** is called. For this the event must be registered in the Form_Load method.

```
adsClient.AdsNotificationEx+=new AdsNotificationExEventHandler(adsClient_AdsNotificationEx);
```



C# program

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using TwinCAT.Ads;


namespace Sample07
{
/// <summary>
/// Summary description for Form1.
/// </summary>
public class Form1 : System.Windows.Forms.Form
{
internal System.Windows.Forms.Button btnDeleteNotifications;
internal System.Windows.Forms.Button btnAddNotifications;
internal System.Windows.Forms.Button btnWrite;
internal System.Windows.Forms.Button btnRead;
internal System.Windows.Forms.GroupBox GroupBox3;
internal System.Windows.Forms.TextBox tbComplexStruct_dintArr;
internal System.Windows.Forms.Label Label14;
internal System.Windows.Forms.TextBox tbComplexStruct_ByteVal;
internal System.Windows.Forms.Label Label13;
internal System.Windows.Forms.TextBox tbComplexStruct_SimpleStruct1_lrealVal;
internal System.Windows.Forms.Label Label12;
internal System.Windows.Forms.TextBox tbComplexStruct_SimpleStruct_dintVal;
internal System.Windows.Forms.Label Label11;
internal System.Windows.Forms.Label Label5;
internal System.Windows.Forms.TextBox tbComplexStruct_stringVal;
internal System.Windows.Forms.Label Label3;
internal System.Windows.Forms.TextBox tbComplexStruct_boolVal;
internal System.Windows.Forms.Label Label9;
internal System.Windows.Forms.TextBox tbComplexStruct_IntVal;
internal System.Windows.Forms.Label Label10;
internal System.Windows.Forms.GroupBox GroupBox2;
internal System.Windows.Forms.TextBox tbStr2;
internal System.Windows.Forms.Label Label7;
internal System.Windows.Forms.TextBox tbStr1;
internal System.Windows.Forms.Label Label8;
internal System.Windows.Forms.GroupBox GroupBox1;
internal System.Windows.Forms.TextBox tblreal1;
internal System.Windows.Forms.Label Label6;
internal System.Windows.Forms.TextBox tbUsint1;
internal System.Windows.Forms.Label Label4;
internal System.Windows.Forms.TextBox tbDint1;
internal System.Windows.Forms.Label Label2;
internal System.Windows.Forms.TextBox tbBool1;
internal System.Windows.Forms.Label Label1;

/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components = null;

//PLC variable handles
private int hdint1;
private int hbool1;
private int husint1;
private int hlreal1;
private int hstr1;
private int hstr2;
private int hcomplexStruct;
private ArrayList notificationHandles;

private TcAdsClient adsClient;

public Form1()
{
//
// Required for Windows Form Designer support
//
InitializeComponent();
}
```

```
 /// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
if( disposing )
{
if (components != null)
{
components.Dispose();
}
}
base.Dispose( disposing );
}

#region Windows Form Designer generated code
..
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
adsClient = new TcAdsClient();
notificationHandles = new ArrayList();
try
{
// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
adsClient.Connect(851);
adsClient.AdsNotificationEx+=new AdsNotificationExEventHandler(adsClient_AdsNotificationEx);
btnDeleteNotifications.Enabled = false;
//create handles for the PLC variables;
hbool1 = adsClient.CreateVariableHandle("MAIN.bool1");
hdint1 = adsClient.CreateVariableHandle("MAIN.dint1");
husint1 = adsClient.CreateVariableHandle("MAIN.usint1");
hlreal1 = adsClient.CreateVariableHandle("MAIN.lreal1");
hstr1 = adsClient.CreateVariableHandle("MAIN.str1");
hstr2 = adsClient.CreateVariableHandle("MAIN.str2");
hcomplexStruct = adsClient.CreateVariableHandle("MAIN.ComplexStruct1");
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}

private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
adsClient.Dispose();
}

private void btnRead_Click(object sender, System.EventArgs e)
{
try
{
//read by handle
//the second parameter specifies the type of the variable
tbDint1.Text = adsClient.ReadAny(hdint1, typeof(int)).ToString();
tbUsint1.Text = adsClient.ReadAny(husint1, typeof(Byte)).ToString();
tbBool1.Text = adsClient.ReadAny(hbool1, typeof(Boolean)).ToString();
tblreal1.Text = adsClient.ReadAny(hlreal1, typeof(Double)).ToString();
//with strings one has to additionally pass the number of characters
//specified in the PLC project(default 80).
//This value is passed is an int array.
tbStr1.Text = adsClient.ReadAny(hstr1, typeof(String), new int[] {80}).ToString();
tbStr2.Text = adsClient.ReadAny(hstr2, typeof(String), new int[] {5}).ToString();
FillStructControls((ComplexStruct)adsClient.ReadAny(hcomplexStruct, typeof(ComplexStruct)));
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}
```

```
private void btnWrite_Click(object sender, System.EventArgs e)
{
try
{
//write by handle
//the second parameter is the object to be written to the PLC variable
adsClient.WriteAny(hdint1, int.Parse(tbDint1.Text));
adsClient.WriteAny(husint1, Byte.Parse(tbUsint1.Text));
adsClient.WriteAny(hbool1, Boolean.Parse(tbBool1.Text));
adsClient.WriteAny(hlreal1, Double.Parse(tblreal1.Text));
//with strings one has to additionally pass the number of characters
//the variable has in the PLC(default 80).
adsClient.WriteAny(hstr1, tbStr1.Text, new int[] {80});
adsClient.WriteAny(hstr2, tbStr2.Text, new int[] {5});
adsClient.WriteAny(hcomplexStruct, GetStructFromControls()) ;
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}

private void btnAddNotifications_Click(object sender, System.EventArgs e)
{
notificationHandles.Clear();
try
{
//register notification
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.dint1", AdsTransMode.OnChange, 100,
0, tbDint1, typeof(int)));
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.usint1", AdsTransMode.OnChange, 100,
0, tbUsint1, typeof(Byte)));
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.bool1", AdsTransMode.OnChange, 100,
0, tbBool1, typeof(Boolean)));
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.lreal1", AdsTransMode.OnChange, 100,
0, tblreal1, typeof(Double)));
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.str1", AdsTransMode.OnChange, 100,
0, tbStr1, typeof(String), new int[] {80}));
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.str2", AdsTransMode.OnChange, 100,
0, tbStr2, typeof(String), new int[] {5}));
notificationHandles.Add(adsClient.AddDeviceNotificationEx("MAIN.complexStruct1",
AdsTransMode.OnChange, 100, 0, tbDint1, typeof(ComplexStruct)));
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
btnDeleteNotifications.Enabled = true;
btnAddNotifications.Enabled = false;
}

private void btnDeleteNotifications_Click(object sender, System.EventArgs e)
{
//delete registered notifications.
try
{
foreach(int handle in notificationHandles)
adsClient.DeleteDeviceNotification(handle);
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
notificationHandles.Clear();
btnAddNotifications.Enabled = true;
btnDeleteNotifications.Enabled = false;
}

private void adsClient_AdsNotificationEx(object sender, AdsNotificationExEventArgs e)
{
TextBox textBox = (TextBox)e.UserData;
Type type = e.Value.GetType();
if(type == typeof(string) || type.IsPrimitive)
textBox.Text = e.Value.ToString();
else if(type == typeof(ComplexStruct))
FillStructControls((ComplexStruct)e.Value);
}

private void FillStructControls(ComplexStruct structure)
```

```
{
tbComplexStruct_IntVal.Text = structure.intVal.ToString();
tbComplexStruct_dintArr.Text = String.Format("{0:d}, {1:d}, {2:d}, {3:d}", structure.dintArr[0],
structure.dintArr[1], structure.dintArr[2], structure.dintArr[3]);
tbComplexStruct_boolVal.Text = structure.boolVal.ToString();
tbComplexStruct_ByteVal.Text = structure.byteVal.ToString();
tbComplexStruct_stringVal.Text = structure.stringVal;
tbComplexStruct_SimpleStruct1_lrealVal.Text = structure.simpleStruct1.lrealVal.ToString();
tbComplexStruct_SimpleStruct_dintVal.Text = structure.simpleStruct1.dintVal1.ToString();
}

private ComplexStruct GetStructFromControls()
{
ComplexStruct structure = new ComplexStruct();
String[] stringArr = tbComplexStruct_dintArr.Text.Split(new char[] {','});
structure.intVal = short.Parse(tbComplexStruct_IntVal.Text);
for(int i=0; i<stringArr.Length; i++)
structure.dintArr[i] = int.Parse(stringArr[i]);

structure.boolVal = Boolean.Parse(tbComplexStruct_boolVal.Text);
structure.byteVal = Byte.Parse(tbComplexStruct_ByteVal.Text);
structure.stringVal = tbComplexStruct_stringVal.Text;
structure.simpleStruct1.dintVal1 = int.Parse(tbComplexStruct_SimpleStruct_dintVal.Text);
structure.simpleStruct1.lrealVal = double.Parse(tbComplexStruct_SimpleStruct1_lrealVal.Text);
return structure;
}
}

// TwinCAT2 Pack = 1, TwinCAT 3 Pack = 0
[StructLayout(LayoutKind.Sequential, Pack=0)]
public class SimpleStruct
{
public double lrealVal;
public int dintVal1;
}

// TwinCAT2 Pack = 1, TwinCAT3 Pack = 0
[StructLayout(LayoutKind.Sequential, Pack=0)]
public class ComplexStruct
{
public short intVal;
//specifies how .NET should marshal the array
//SizeConst specifies the number of elements the array has.
[MarshalAs(UnmanagedType.ByValArray, SizeConst=4)]
public int[] dintArr = new int[4];
[MarshalAs(UnmanagedType.I1)]
public bool boolVal;
public byte byteVal;
//specifies how .NET should marshal the string
//SizeConst specifies the number of characters the string has.
//'(inclusive the terminating null ).
[MarshalAs(UnmanagedType.ByValTStr, SizeConst=6)]
public string stringVal = "";
public SimpleStruct simpleStruct1 =new SimpleStruct();
}
}
```

PLC program

```
TYPE TSimpleStruct :
STRUCT
lrealVal: LREAL := 1.23;
dintVal1: DINT := 120000;
END_STRUCT
END_TYPE

TYPE TComplexStruct :
STRUCT
intVal : INT:=1200;
dintArr: ARRAY[0..3] OF DINT:= 1,2,3,4;
boolVal: BOOL := FALSE;
byteVal: BYTE:=10;
stringVal : STRING(5) := 'hallo';
simpleStruct1: TSimpleStruct;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
(*primitive Types*)
Bool1:BOOL := FALSE;
int1:INT := 30000;
dint1:DINT:=125000;
usint1:USINT:=200;
real1:REAL:= 1.2;
lreal1:LREAL:=3.5;

(*string Types*)
str1:STRING := 'this is a test string';
str2:STRING(5) := 'hallo';

(*struct Types*)
complexStruct1 : TComplexStruct;
END_VAR
```

3.8 Detect state changes in TwinCAT router and PLC

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample08.zip

Task

Detect state changes in TwinCAT router and PLC.

Description

State changes in ADS devices can be detected effectivly by registering callback functions to the devices. These function are called on state changes of the ADS devices.

State changes in the TwinCAT router can be detected via the TcAdsClient.AmsRouterNotification in the TcAdsClient class . In order to detect state changes in the PLC an ADS Notification to the ADS status word has to be registered. Callback functions can be registered for both. These are called on state changes of the devices.

The following program monitors the state of the TwinCAT router and the PLC using the above techniques.

C# Program

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

namespace TwinCATAds_Sample08
{
public partial class Form1 : Form
{
private TcAdsClient _tcClient = null;
private AdsStream _adsStream = null;
private BinaryReader _binRead = null;
private int _notificationHandle = 0;

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

```
 }

private void Form1_Load(object sender, EventArgs e)
{
try
{
_tcClient = new TcAdsClient();


// Connect to local PLC - Runtime 1 - TwinCAT3 Port=851
_tcClient.Connect(851);

_adsStream = new AdsStream(2); /* stream for storing the ADS state of the PLC */
_binRead = new BinaryReader(_adsStream); /* reader for reading the state */

/* register callback function to detect state changes in the router */
_tcClient.AmsRouterNotification += new
AmsRouterNotificationEventHandler(AmsRouterNotificationCallback);

/* register an ADS notification an the ADS status word of the PLC */
_notificationHandle = _tcClient.AddDeviceNotification(
(int)AdsReservedIndexGroups.DeviceData, /* index group of the device state*/
(int)AdsReservedIndexOffsets.DeviceDataAdsState, /*index offset of the device state */
_adsStream, /* stream to store the state */
AdsTransMode.OnChange, /* transfer mode: transmit ste on change */
0, /* transmit changes immediately */
0,
null);

/* register callback function to react on notifications */
_tcClient.AdsNotification += new AdsNotificationEventHandler(OnAdsNotification);
}
catch (AdsErrorException ex)
{
MessageBox.Show(ex.Message);
}
}

/* callback function called on state changes of the router */
void OnAdsNotification(object sender, AdsNotificationEventArgs e)
{
if (e.NotificationHandle == _notificationHandle)
{
AdsState plcState = (AdsState)_binRead.ReadInt16(); /* state was written to the stream */
_plcLabelValue.Text = plcState.ToString();
}
}

/* Ccallback function called on state changes of the PLC */
void AmsRouterNotificationCallback(object sender, AmsRouterNotificationEventArgs e)
{
_routerLabelValue.Text = e.State.ToString();
}

private void _exitButton_Click(object sender, EventArgs e)
{
this.Close();
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
try
{
_tcClient.DeleteDeviceNotification(_notificationHandle);
_tcClient.Dispose();
}
catch(AdsErrorException ex)
{
MessageBox.Show(ex.Message);
}
}
}
}
```

3.9 ADS-Sum Command: Reading or writing several variables

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample09.zip

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.

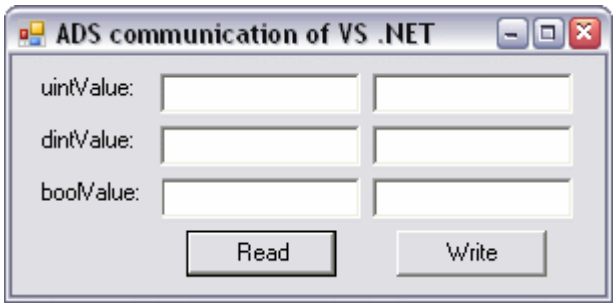


Fig. 1: TwinCAT.Ads_Sample8

In the beginning it's important, binding 'TwinCAT.Ads.dll' to your project! To do so, open `'Solution Explorer'` and choose `'Add References...'` via`'References'`. `'Browse'` for the DLL in the folder `'TwinCAT -> ADS API -> .NET'`.

C# program

1. Read variables

First, define two structures:

```
namespace AdsBlockRead
{
// Structure declaration for values
internal struct MyStruct
{
public ushort uintValue;
public int dintValue;
public bool boolValue;
}

// Structure declaration for handles
internal struct VariableInfo
{
public int indexGroup;
public int indexOffset;
public int length;
}
```

and declare some global variables.

```
public classForm1 : System.Windows.Forms.Form
{
[...]
```

```
privateTcAdsClient adsClient;
private string[] variableNames;
private int[] variableLengths;
VariableInfo[] variables;
```

On applications start an ADS connection to PLC is established and Handle parameters are written into the structure.

```
private void Form1_Load(object sender, System.EventArgs e)
{
try
{
// Connect to PLC
adsClient = newTcAdsClient();
adsClient.Connect(851);

// Fill structures with name and size of PLC variables
variableNames = new string[] { "MAIN.uintValue", "MAIN.dintValue", "MAIN.boolValue" };
variableLengths = new int[] { 2, 4, 1 };

// Write handle parameter into structure
variables = newVariableInfo[variableNames.Length];
for (int i = 0; i < variables.Length; i++)
{
variables[i].indexGroup = (int)AdsReservedIndexGroups.SymbolValueByHandle;
variables[i].indexOffset = adsClient.CreateVariableHandle(variableNames[i]);
variables[i].length = variableLengths[i];
}
}
catch (Exception err)
{
MessageBox.Show(err.Message);
adsClient = null;
}
}
```

After clicking 'Read' Button, `'BlockRead'` method returns an ADS Stream. Check for ADS return codes (Err) for error handling, before data is read out of the stream und stored in the text boxes.

```
private void button1_Click(object sender, System.EventArgs e)
{
if (adsClient == null)
return;

try
{
// Get the ADS return codes and examine for errorsBinaryReader reader =
newBinaryReader(BlockRead(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));
}

// Read the data from the ADS streamMyStruct myStruct;
myStruct.uintValue = reader.ReadUInt16();
myStruct.dintValue = reader.ReadInt32();
myStruct.boolValue = reader.ReadBoolean();

// Write data from the structure into the text boxes
tbUint.Text = myStruct.uintValue.ToString();
tbDint.Text = myStruct.dintValue.ToString();
tbBool.Text = myStruct.boolValue.ToString();
}
catch (Exception err)
{
MessageBox.Show(err.Message);
}
}
```

'BlockRead' methode takes an object of type `'VariableInfo[]'`, where the handle parameters are strored. After reserving memory for the values to be read an write, an ADS stream is written with the parameters delivered by `'VariableInfo[]'`. At the end the sum command transfers the commands and an ADS stream is returned.

```
privateAdsStream BlockRead(VariableInfo[] variables)
{
// Allocate memoryint rdLength = variables.Length * 4;
int wrLength = variables.Length * 12;

// Write data for handles into the ADS StreamBinaryWriter writer =
newBinaryWriter(newAdsStream(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 PLCAdsStream rdStream = newAdsStream(rdLength);
adsClient.ReadWrite(0xF080, variables.Length, rdStream, (AdsStream)writer.BaseStream);

// Return the ADS error codesreturn rdStream;
}
```

Sum Command's Parameters consists of IndexGroup (0xF080) - sum command call, `IndexOffset (variables.Length)` - number of sub commands, `rdDataStream (rdStream)` - memory, taking the read values, `wrDataStream (writer.BaseStream)` - memory, containing values to be written.

sub commands in `wrDataStream`

Fig. 2: TwinCAT.Ads_Sample9

response in rdDataStream

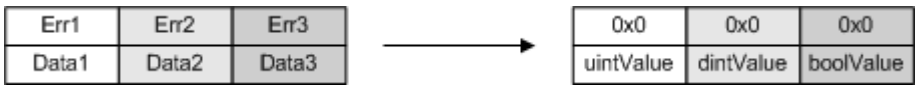


Fig. 3: TwinCAT.Ads_Sample10

2. writing variables After clicking 'Write' button, `'BlockRead2'` methode returns an ADS stream. Following is a check of ADS return codes (Err) for error handlings.

```
private void button2_Click(object sender, EventArgs e)
{
if (adsClient == null)
return;

try
{
// Get the ADS return codes and examine for errorsBinaryReader reader =
newBinaryReader(BlockRead2(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));
}
}
catch (Exception err)
```

```
{
MessageBox.Show(err.Message);
}
}
```

'BlockRead2' methode takes an object of type `'VariableInfo[]'`, where handle parameters are stored. After reserving memory for the values to be read an write, `'MyStruct'` object is filled with the values stored in the textboxes. Next, an ADS stream containing parameters from `'VariableInfo[]'` objects an the values of `'MyStruct'` object is written. At the end sum command transfers the commands and an ADS Stream is returned.

```
privateAdsStream BlockRead2(VariableInfo[] variables)
{
// Allocate memoryint rdLength = variables.Length * 4;
int wrLength = variables.Length * 12 + 7;

BinaryWriter writer = newBinaryWriter(new AdsStream(wrLength));
MyStruct myStruct;
myStruct.uintValue = ushort.Parse(tbUint2.Text);
myStruct.dintValue = int.Parse(tbDint2.Text);
myStruct.boolValue = bool.Parse(tbBool2.Text);

// Write data for handles into the ADS streamfor (int i = 0; i < variables.Length; i++)
{
writer.Write(variables[i].indexGroup);
writer.Write(variables[i].indexOffset);
writer.Write(variables[i].length);
}

// Write data to send to PLC behind the structure
writer.Write(myStruct.uintValue);
writer.Write(myStruct.dintValue);
writer.Write(myStruct.boolValue);

// Sum command to write the data into the PLCAdsStream rdStream = newAdsStream(rdLength);
adsClient.ReadWrite(0xF081, variables.Length, rdStream, (AdsStream)writer.BaseStream);

// Return the ADS error codesreturn rdStream;
}
```

Sum Command's Parameters consists of IndexGroup (0xF081) - sum command call, `IndexOffset (variables.Length)` - number of sub commands, `rdDataStream (rdStream)` - memory, taking the read values, `wrDataStream (writer.BaseStream)` - memory, containing values to be written.

sub commands in `wrDataStream`

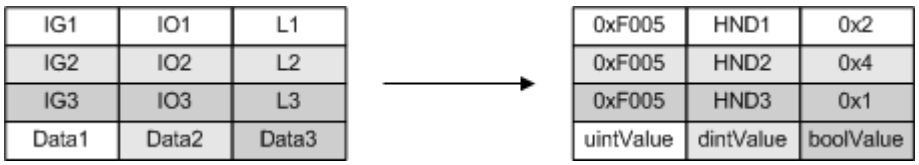


Fig. 4: TwinCAT.Ads_Sample11

response in rdDataStream

Fig. 5: TwinCAT.Ads_Sample12

3.10 Free Sample

Download

Language / IDE	Extract the sample program
Visual C#	-

3.11 Delete a handle of a PLC variable

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample11.zip

Description

This Sample shows how to delete a handle of a PLC variable:

C# program

```
static void Main(string[] args)
{
//Create a new instance of class TcAdsClient
TcAdsClient tcClient = new TcAdsClient();
int iHandle = 0;

try
{
// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
tcClient.Connect(851);

//Get the handle of the PLC variable "PLCVar"
iHandle = tcClient.CreateVariableHandle("MAIN.PLCVar");

//Release the specific handle of "PLCVar"
tcClient.DeleteVariableHandle(iHandle);
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
Console.ReadKey();
}
finally
{
tcClient.Dispose();
}
}
```

3.12 Read flag synchronously from the PLC

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample12.zip

Description

In this example program the value in flag double word 0 in the PLC is read and displayed on the screen:

C# program

```
static void Main(string[] args)
{
//Create a new instance of class TcAdsClient
TcAdsClient tcClient = new TcAdsClient();

try
{
// Connect to local PLC - Runtime 1 - TwinCAT 2 Port=801, TwinCAT 3 Port=851
tcClient.Connect(801);

//Specify IndexGroup, IndexOffset and read SPSVar
int iFlag = (int)tcClient.ReadAny(0x4020, 0x0, typeof(Int32));

Console.WriteLine("" + iFlag);
Console.ReadKey();
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
Console.ReadKey();
}
finally
{
tcClient.Dispose();
}
}
```

3.13 Write flag synchronously into the PLC

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample13.zip

Description

In this example program, the value that the user has entered is written into flag double word 0:

C# program

```
static void Main(string[] args)
{
//Create a new instance of class TcAdsClient
TcAdsClient tcClient = new TcAdsClient();

try
{
// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
tcClient.Connect(851);

//Specify IndexGroup, IndexOffset and write SPSVar
int iNewValue = 0;
tcClient.WriteAny(0x4020, 0x0, iNewValue);
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
Console.ReadKey();
}
finally
```

```
{
tcClient.Dispose();
}
}
```

3.14 Start/stop PLC

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample14.zip

Description

The following program starts or stops run-time system 1 in the PLC:

C# program

```
static void Main(string[] args)
{
//Create a new instance of class TcAdsClient
TcAdsClient tcClient = new TcAdsClient();

try
{
// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
tcClient.Connect(851);

Console.WriteLine(" PLC Run\t[R]");
Console.WriteLine(" PLC Stop\t[S]");
Console.WriteLine("\r\nPlease choose \"Run\" or \"Stop\" and confirm with enter..");
string sInput = Console.ReadLine().ToLower();

//Process user input and apply chosen state
do{
switch (sInput)
{
case "r": tcClient.WriteControl(new StateInfo(AdsState.Run, tcClient.ReadState().DeviceState));
break;
case "s": tcClient.WriteControl(new StateInfo(AdsState.Stop, tcClient.ReadState().DeviceState));
break;
default: Console.WriteLine("Please choose \"Run\" or \"Stop\" and confirm with enter.."); sInput =
Console.ReadLine().ToLower(); break;
}
} while (sInput != "r" && sInput != "s");
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
Console.ReadKey();
}
finally
{
tcClient.Dispose();
}
}
```

3.15 Access by variable name

Download

Requirements

Language / IDE	Extract the sample program
C# / Visual Studio	Sample15.zip

Description

The following program accesses a PLC variable that does not have an address. Access must therefore be made by the variable name. Once the PLC variable in the example program exceeds 10 it is reset to 0:

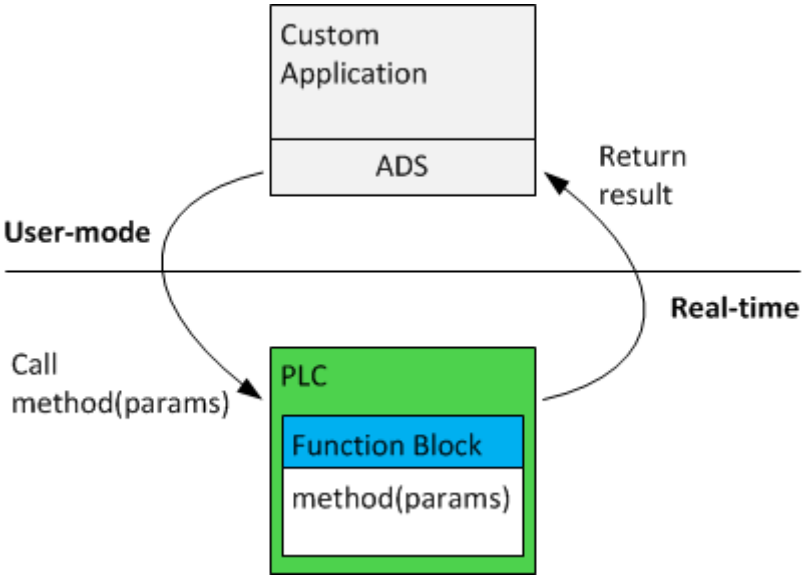
C# program

```
static void Main(string[] args)
{
//Create a new instance of class TcAdsClient
TcAdsClient tcClient = new TcAdsClient();
AdsStream dataStream = new AdsStream(4);
AdsBinaryReader binReader = new AdsBinaryReader(dataStream);

int iHandle = 0;
int iValue = 0;

try
{
// Connect to local PLC - Runtime 1 - TwinCAT 3 Port=851
tcClient.Connect(851);

//Get the handle of the PLC variable "PLCVar"
iHandle = tcClient.CreateVariableHandle("MAIN.PLCVar");

Console.WriteLine("Press enter to continue and any other key to abort..");

do
{
//Use the handle to read PLCVar
tcClient.Read(iHandle, dataStream);
iValue = binReader.ReadInt32();
dataStream.Position = 0;

Console.WriteLine("Current value is: " + iValue);

if (iValue >= 10)
{
//Reset PLC variable to zero
tcClient.WriteAny(iHandle, 0);
}

} while (Console.ReadKey().Key.Equals(ConsoleKey.Enter));
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
Console.ReadKey();
}
finally
{
//Delete variable handle
tcClient.DeleteVariableHandle(iHandle);
tcClient.Dispose();
}
}
```

3.16 PLC method call

Download: https://infosys.beckhoff.com/content/1033/tc3_adssamples_net/Resources/7860265995/.zip

The following program shows how a PLC method can be called from a .NET program via ADS. The following ficure shows that the method parameters ("params") can be transferred directly via ADS and the result ("result") is returned after execution.



The methods are called after the execution of the PLC. The remaining computing time of the cycle is available.

Procedure

The following steps must be observed if you want to offer and call a method in the PLC:

1. Add a method for a block.
2. the method for the ADS access is offered via the following attribute:

```
{attribute 'TcRpcEnable'}
```

1. Create a handle for the method for the following symbol with the syntax:

```
Function block_Name#Methods_Name
```

The method can then be called using the following ADS Read/Write command:

ADS Read/Write Parameter:

Index group: ADSIGRP_SYM_VALBYHND (0xF005)

Offset: Methods Handle (hMethod)

ReadData: ADS data with the return value of the method

WriteData: ADS data with the transfer

Constraints

If you call methods using ADS, you must note the following points:

- CPU time: If the methods require more processing time than is available, real-time overruns can occur.
- Breakpoints: Are not supported within the methods and can generate exceptions.
- Pointer: If a pointer points to an array of elements, the attribute TcRpcLengthIs can be used to specify the length.

```
        len : INT;
    {attribute 'TcRpcLengthIs' := 'len'}
    ptr : POINTER TO BYTE;
```

Requirements

Runtime Environment	Target System
TwinCAT v3.1.0 Build 4016	PC or CX (x86, ARM)

More Information:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com