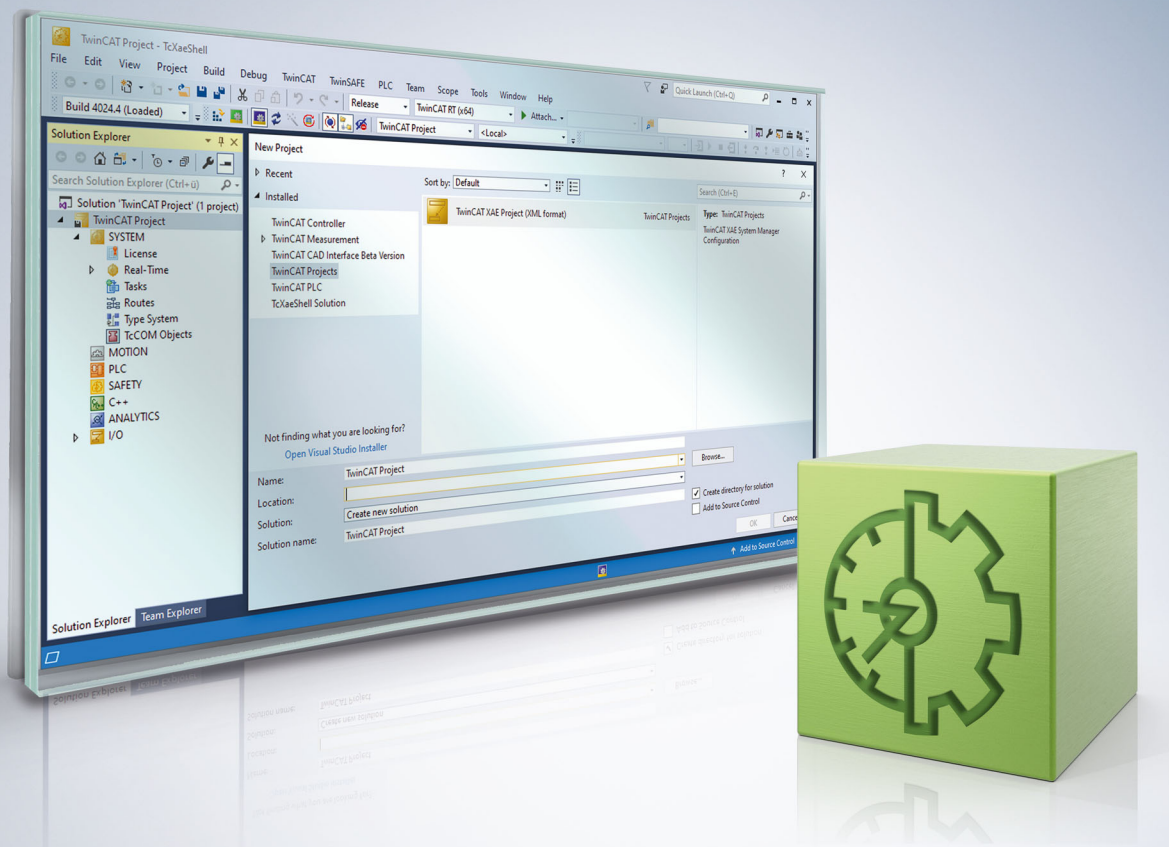
BECKHOFF New Automation Technology

Handbuch | DE

ADS-DLL .NET Samples

TwinCAT 3



31.08.2022 | Version: 1.2

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT®

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Integration

2.1 Einbinden in TwinCAT 3

Neues Projekt erstellen

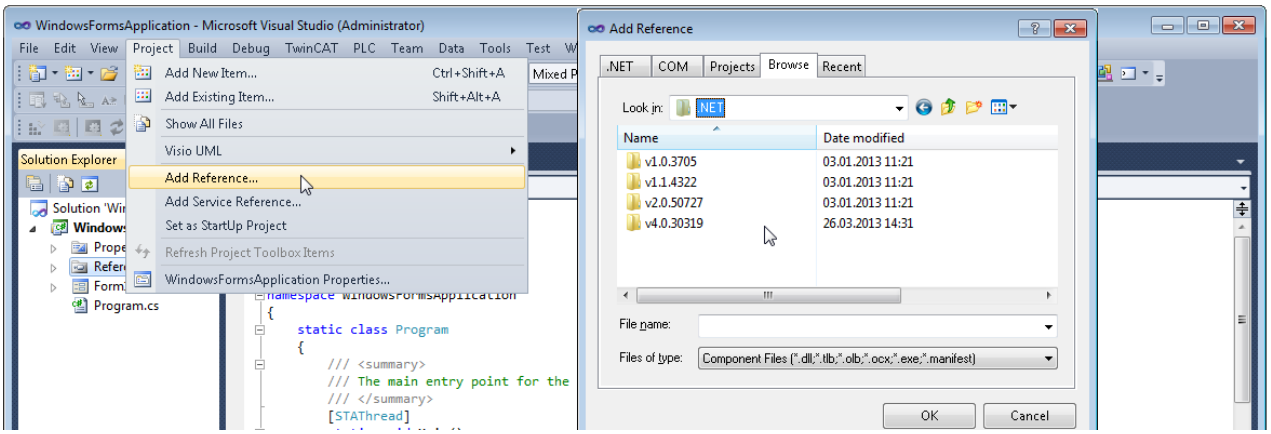
Erstellen Sie im Microsoft Visual Studio ein neues Projekt (Windows Forms Application).

Referenz hinzufügen

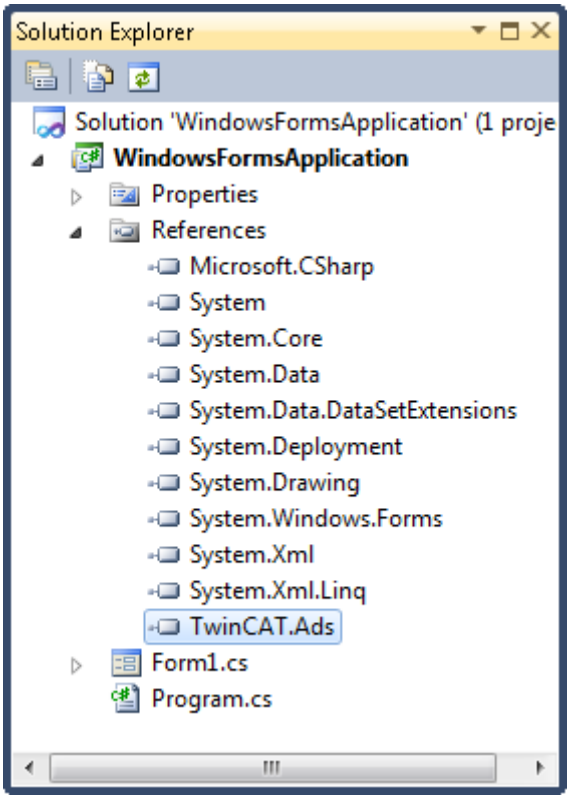
Um die Klassenbibliothek TwinCAT.Ads zu selektieren müssen Sie das Kommando **Add Reference...** des *Project* -Menüs auswählen. Dies öffnet den **Add Reference** Dialog.

Unter folgendem Pfad befindet sich die TwinCAT.ADS Reference für die unterschiedlichen .NET Laufzeiten:

C:\TwinCAT\AdsApi\.NET



Betätigen Sie im **Add Reference** Dialog den **Browse** Button und wählen Sie die Datei TwinCAT.Ads.dll aus. Im Solution Explorer können Sie überprüfen, ob die Komponente zu der Liste der Referenzen hinzugefügt worden ist:



Alle Typen der Klassenbibliothek gehören zum Namespace TwinCAT.Ads. Fügen Sie folgende Zeilen am Anfang des Quelltextes an:

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

Dies ermöglicht den Zugriff auf die in TwinCAT.Ads definierten Typen, ohne den Namen des Namespaces mit anzugeben. Die Klasse TcAdsClient ist der Kern der TwinCAT.Ads-Klassenbibliothek und ermöglicht dem Benutzer die Kommunikation mit einem Ads-Gerät. Dazu muss zunächst eine Instanz der Klasse erzeugt werden. Dann wird mit der Methode Connect eine Verbindung zum ADS-Gerät aufgebaut.

3 Beispiele ADS .NET

Tab. 1: TwinCAT ADS .NET

Beschreibung
Einbinden in TwinCAT3 (C#) [▶ 8]

DescriptionBeschreibung	Visual C#
Beispiel 1: Zugriff auf ein Array in der SPS [▶ 10]	Beispiel01.zip
Beispiel 2: Übertragen von Strukturen an die SPS [▶ 13]	Beispiel02.zip
Beispiel 3: Ereignisgesteuertes Lesen [▶ 16]	Beispiel03.zip
Beispiel 4: Lesen und Schreiben von String Variablen [▶ 19]	Beispiel04.zip
Beispiel 5: Lesen und Schreiben von DATE/TIME-Variablen [▶ 22]	Beispiel05.zip
Beispiel 6: SPS Variablen-Deklaration auslesen [▶ 24]	Beispiel06.zip
Beispiel 7: Lesen und Schreiben von SPS Variablen eines beliebigen Typs [▶ 30]	Beispiel07.zip
Beispiel 8: Statusänderung vom TwinCAT-Router und der SPS erkennen [▶ 37]	Beispiel08.zip
Beispiel 9: ADS-Summenkommando: Lesen oder Schreiben von mehreren Variablen [▶ 39]	Beispiel09.zip
Beispiel 10: Reserved	
Beispiel 11: Handle einer SPS Variablen löschen [▶ 43]	Beispiel11.zip
Beispiel 12: Merker synchron aus der SPS lesen [▶ 44]	Beispiel12.zip
Beispiel 13: Merker synchron in die SPS schreiben [▶ 45]	Beispiel13.zip
Beispiel 14: SPS starten/stoppen [▶ 45]	Beispiel14.zip
Beispiel 15: Zugriff per Variablenname [▶ 46]	Beispiel15.zip

3.1 Zugriff auf ein Array in der SPS

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample01.zip

Aufgabe

In der SPS befindet sich ein Array, welches von einer .NET Applikation mit einem Lesebefehl ausgelesen werden soll.

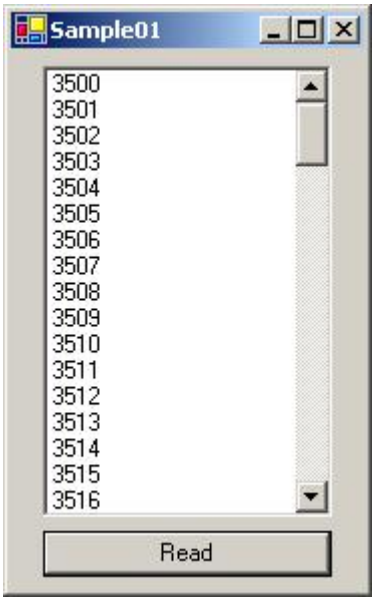
Beschreibung

In der SPS befindet sich ein Array mit 100 Elementen vom Typ Integer (2 Byte). Das Array wird in der SPS mit den Werten 3500 bis 3599 aufgefüllt.

In der Event-Methode Form1_Load wird eine neue Instanz der Klasse TcAdsClient kreiert. Dann wird die Methode TcAdsClient.Connect des TcAdsClient Objektes aufgerufen, um eine Verbindung zum Port 801 zu erstellen. Zum Schluss wird dann die Methode TcAdsClient.CreateVariableHandle aufgerufen, um sich ein Handle auf die SPS-Variable zu besorgen. Wenn das Programm beendet wird, wird das Handle in der Event-Methode Form1_Closing wieder freigegeben und die Dispose-Methode des TcAdsClient Objektes wird aufgerufen.

Wenn der Anwender den Button betätigt, wird das gesamte Array mit Hilfe der Methode TcAdsClient.Read aus der SPS in ein AdsStream Objekt eingelesen. Der Stream sollte genauso groß wie die Daten in der SPS sein. Da wir 100 INTs (je 2 Bytes) lesen wollen, muss der Stream mindestens eine Größe von 2 * 100 Bytes haben.

Die Klasse System.IO.BinaryReader ist notwendig, um die einzelnen Felder des Arrays aus dem Stream lesen. Dazu wird als erstes die Position des Streams auf 0 gesetzt. Dann können die einzelnen Felder des Arrays mit Hilfe der Methode BinaryReader.ReadInt16 in einer for-Schleife ausgelesen werden. Der Positionszeiger des Streams wird automatisch um die Anzahl der gelesenen Bytes erhöht.



C# Programm

```
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 Programm

```
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 Übertragen von Strukturen an die SPS

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample02.zip

Aufgabe

Von einer .NET Applikation soll eine Struktur in die SPS geschrieben werden. Die Elemente der Struktur haben verschiedene Datentypen.

Beschreibung

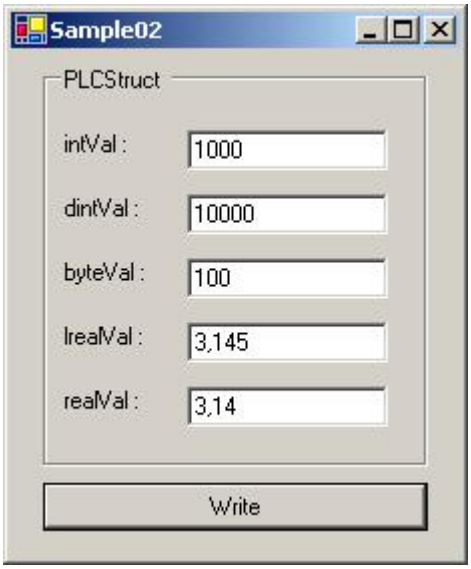
Die Struktur, die in die SPS geschrieben werden soll:

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

In C# würde die Struktur wie folgt aussehen:

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

Anstatt diese Struktur direkt zu verwenden, wird wie in Sample01 ein AdsStream verwendet. Das AdsStream Objekt wird mit einer Grösse von 19 Bytes initialisiert. Dies entspricht der Grösse des Datentyps PLCStruct in der SPS. Ein BinaryWriter wird verwendet, um die Daten der Editierfelder in den Stream zu schreiben. Nachdem die Position des Streams auf 0 gesetzt wird, werden die einzelnen Felder der Struktur in den Stream geschrieben. Als letztes wird der gesamte Stream mit Hilfe der TcAdsClient.Write in die SPS geschrieben.



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 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 Programm

```
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 Ereignisgesteuertes Lesen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample03.zip

Aufgabe

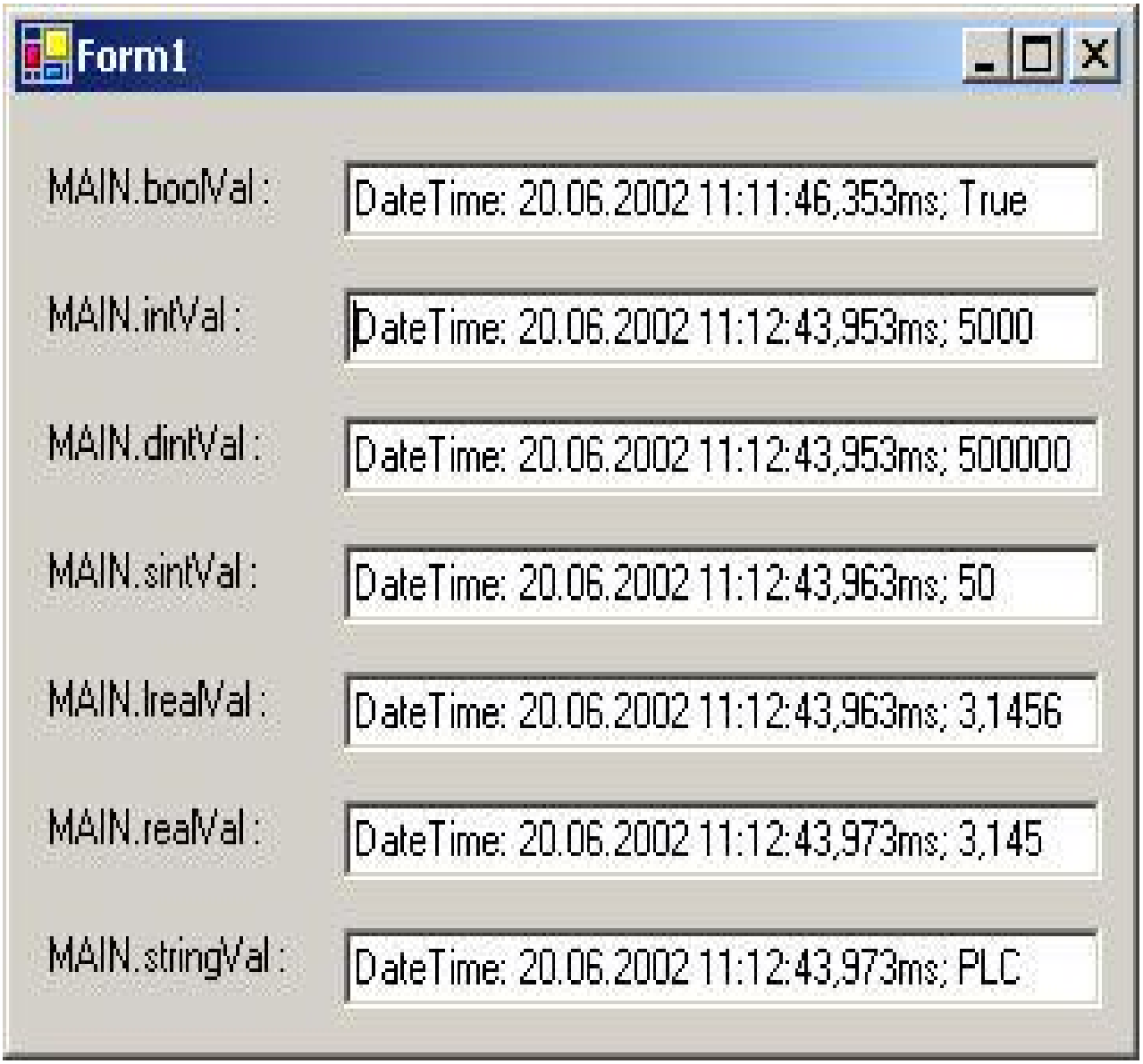
In der SPS befinden sich 7 globale Variablen. Jede SPS-Variable ist von einem anderen Datentyp. Die Werte der Variablen sollen auf möglichst effektive Weise ausgelesen und der Wert mit Zeitstempel auf einer Form in Visual Basic dargestellt werden.

Beschreibung

In dem Load-Ereignis der Form wird mit der Methode TcAdsClient.AddDeviceNotification() eine Verbindung zu jeder Variablen in der SPS erzeugt. Das Handle dieser Verbindung wird in einem Array gespeichert. Der Parameter TransMode spezifiziert die Art des Datenaustausches. In diesem Fall wird mit AdsTransMode.OnChange festgelegt, dass die SPS-Variable nur übermittelt wird falls sich der Wert in der SPS geändert hat(siehe AdsTransMode). Der Parameter cycleTime bestimmt, wie oft die PLC checken soll, ob sich die entsprechende Varaiable geändert hat. MaxDelay ermöglicht es die Notifications für einen Zeitraum zu sammeln und nach Ablauf des Intervall im Block zu verschicken.

Wenn die SPS-Variable sich geändert hat, wird das TcAdsClient.AdsNotification() gefeuert. Der Parameter e der Eventhandler-Methode ist vom Typ AdsNotificationEventArgs und enthält die Zeit, das Handle, den Wert und das Textfeld, das die Daten anzeigen soll. In dem Closing-Ereignis werden die Verbindungen wieder mit der Methode TcAdsClient.DeleteDeviceNotification() freigegeben. Dieses sollten Sie umbedingt beachten, da jede Verbindung die mit TcAdsClient.AddDeviceNotification() hergestellt wurde Resourcen verbraucht. Setzen Sie außerdem die CycleTime auf angemessene Werte, da zu viele Schreib / Leseoperationen das System so stark auslasten kann, dass die Bedieneroberfläche stark verlangsamt wird.

Hinweis: Im Callback (OnNotification()) dürfen keine zeitintensiven Aktionen ausgeführt werden.



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 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 Programm

```
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 Lesen und Schreiben von String-Variablen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample04.zip

Aufgabe

Eine .NET Applikation soll einen String von der SPS lesen und einen String in die SPS schreiben.

Beschreibung

Im Load-Ereignis wird eine neue Instanz der Klasse TcAdsClient erzeugt. Dann wird die Methode TcAdsClient.Connect des TcAdsClient Objektes aufgerufen, um eine Verbindung mit dem Port 801 herzustellen. Zum Schluss wird dann die Methode TcAdsClient.CreateVariableHandle aufgerufen, um sich ein Handle auf die SPS-Variable zu besorgen. Wenn das Programm beendet wird, wird das Handle in der Event-Methode Form1_Closing wieder freigegeben und die Dispose-Methode des TcAdsClient Objektes wird aufgerufen.

Wenn der Anwender den "Read"-Button betätigt, wird die String-Variable MAIN.text mit Hilfe der Methode TcAdsClient.Read aus der SPS gelesen und in dem Editierfeld angezeigt. Wenn der Anwender den "Write"-Button betätigt, wird der Text der im Editierfeld steht in die SPS-Variable MAIN.text geschrieben.

Mit Hilfe der Klassen AdsBinaryReader and AdsBinaryWriter können SPS-Strings aus einem AdsStream gelesen und in einen AdsStream geschrieben werden(siehe kommentierten Abschnitt des Beispiel-Programmes). Diese Klassen sind direkt von den Klassen BinaryReader und BinaryWriter abgeleitet. Um einen String aus dem Stream zu lesen wird die Methode AdsBinaryReader.ReadPlcString aufgerufen. Um einen String in den Stream zu schreiben wird die Methode AdsBinaryWriter.WritePLCString aufgerufen. Die Länge vom AdsStream muss der Länge des Strings in der SPS entsprechen (ohne Null-Terminierung). Die Länge der String-Variablen wird der Methode als Parameter übergeben und muss der Länge des Strings in der SPS entsprechen (ohne Null-Terminierung).

C# Programm

```
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 Programm

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

3.5 Lesen und Schreiben von DATE/TIME-Variablen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample05.zip

Aufgabe

Es soll eine .NET Applikation geschrieben werden die DATE/TIME-Variablen liest bzw. beschreibt.

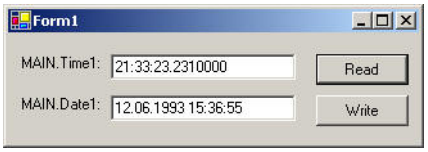
Beschreibung

Die SPS enthält die Variable MAIN.Time1 vom Typ TIME und die Variable MAIN.Date1 vom Typ DT.

Im Load-Ereignis des Forms wird eine neue Instanz der Klasse TcAdsClient erzeugt. Dann wird die Methode TcAdsClient.Connect des TcAdsClient Objektes aufgerufen, um eine Verbindung mit dem Port 801 herzustellen. Zum Schluss wird dann die Methode TcAdsClient.CreateVariableHandle aufgerufen, um sich ein Handle auf die SPS-Variable zu besorgen. Wenn das Programm beendet wird, wird das Handle in der Event-Methode Form1_Closing wieder freigegeben und die Dispose-Methode des TcAdsClient Objektes wird aufgerufen.

Wenn der Anwender den "Read"-Button betätigt, werden die Variablen MAIN.Time1 und MAIN.Date1 mit Hilfe der Methode TcAdsClient.Read aus dem AdsStream gelesen. Die Klasse AdsBinaryReader, die von BinaryReader abgeleitet ist, wird dazu eingesetzt , eine Zeit bzw. ein Datum aus der SPS zu lesen. Die Methode AdsBinaryReader.ReadPlcTIME liest die TIME-Variable aus dem Stream und konvertiert diese in den .NET Datentypen TimeSpan. Die Methode AdsBinaryReader.ReadPlcDATE liest die DT-Variable aus dem AdsStream und konvertiert diese in den .NET Typen DataTime.

Wenn der Anwender den "Write"-Button betätigt, werden die SPS-Variablen MAIN.Time1 und MAIN.Date1 beschrieben. Die Klasse AdsBinaryWriter, die von BinaryWriter abgeleitet ist, kann eingesetzt werden, um eine Zeit bzw. ein Datum in diese zu schreiben. Die Methode AdsBinaryWriter.WritePlcType schreibt die .NET Typen TimeSpan und DateTime im SPS-Format in den Stream.



C# Programm

```
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 Programm

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

3.6 SPS Variablen-Deklaration auslesen

Download

Voraussetzungen

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

Aufgabe

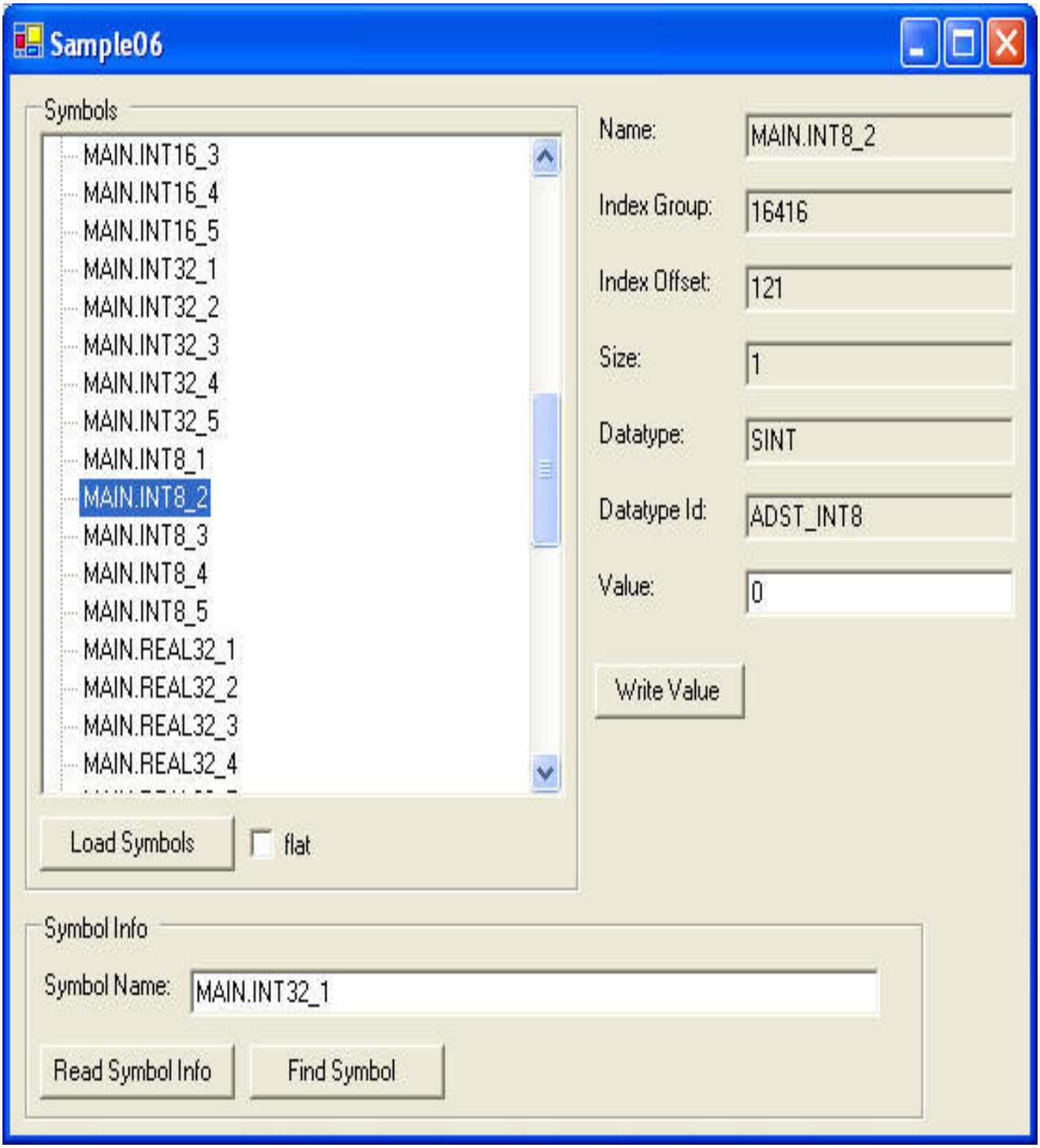
Alle Variablen, die in der SPS deklariert sind, sollen in einer Baumansicht dargestellt.

Beschreibung

In der Methode Form1_Load wird mit TcAdsClient.CreateSymbolInfoLoader eine Instanz der Klasse TcAdsSymbolInfoLoader erzeugt. Diese Klasse ist verantwortlich für das laden der Symbol-Information von der SPS. Durch das Betätigen des Buttons *Load Symbols* werden die Symbole mit Hilfe der Methode TcAdsSymbolInfoLoader.GetFirstSymbol von der SPS geladen. Als Rückgabewert wird das erste geladen Symbol in Form eines TcAdsSymbolInfo-Objektes zurückgeliefert. Die Methoden TcAdsSymbolInfo.NextSymbol und TcAdsSymbolInfo.FirstSubSymbol werden dann verwendet, um über alle Symbole zu itererieren und damit die Symbols-Baumansicht hierarchisch zu füllen. Wenn die Checkbox *flat*gecheckt ist werden die Symbole als flache Liste dargestellt. Dazu wird einfach mit foreach über die Symbole von TcAdsSymbolInfoLoader enumeriert. Dabei werden auch die Sub-Symbole dargestellt.

Beim Selektieren eines TreeView-Items werden die Editierfelder mit der Symbol-Information gefüllt. Zusätzlich wird der Wert des Symbols mit TcAdClient.ReadSymbol ausgelesen und im Editierfeld *Value* ausgegeben. Durch Betätigen des *Write Value* Buttons wird die Methode TcAdsCient.WriteSymbol verwendet, um einen Wert in die SPS-Variable zu schreiben.

Um die Symbol-Information einer bestimmten Variablen auszulesen, kann entweder *Read Symbol Info* oder *Find Symbol* verwendet werden. Die erste Variante verwendet die Methode TcAdsClient.ReadSymbolInfo. Hierbei wird ein ADS-Aufruf verwendet, um die Symbol-Information für dieses Symbol von der SPS zu lesen. Die zweite Variante verwendet die Methode TcAdsSymbolInfoLoader.FindSymbol, die das Symbol in der Liste der bereits geladenen Symbole sucht. Wenn vorher noch keine Symbole geladen wurden, werden alle Symbole von der SPS geladen.



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 Programm

```
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 Lesen und Schreiben von SPS Variablen eines beliebigen Typs (ReadAny, WriteAny)

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample07.zip

Aufgabe

Auslesen bzw. Beschreiben von Variablen eines beliebigen Typs mit Hilfe der Methode ReadAny bzw. WriteAny.

Beschreibung

ReadAny

In der Event-Methode btnRead_Click wird die Methode TcAdsClient.ReadAny verwendet um die Variablen per Handle auszulesen:

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

ReadAnyReadAny

Der Methode wird im Parameter **type** der Typ der zu lesenden Variable übergeben. Wenn die Methode erfolgreich gewesen ist, werden die gelesenen Daten als Objekt zurückgeliefert. Der Typ des Objektes entspricht dem Parameter **type**. Da bei einigen Datentypen(Arrays und Strings) zusätzliche Informationen notwendig sind gibt es eine Überladung der Methode ReadAny, die den zusätzlichen Parameter **args** enthält. Bei Strings z.B. müsste ein Integer-Array der Länge 1 übergeben werden. Eine vollständige Liste der unterstützten Typen finden Sie in der Dokumentation der überladenen Methode.

Beispiel:

Eine SPS-Variable vom Type ARRAY[0..3] OF DINT soll gelesen werden:

```
int hArr;
int[] arr;

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

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

WriteAny

In der Event-Methode btnWrite_Click wird die Methode TcAdsClient.WriteAny verwendet um die Variablen per Handle zu beschreiben:

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

WriteAnyWriteAny

Der Parameter value enthält das Objekt, das in die SPS geschrieben werden soll. Eine vollständige Liste der unterstützten Typen von **value** finden Sie in der Dokumentation der überladenen Methode.

Beispiel:

Eine SPS-Variable vom Typ ARRAY[0..3] OF DINT soll beschrieben werden:

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

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

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

Lesen und Schreiben von Strukturen:
(nicht möglich mit dem Compact Framework)

Beim Lesen und beschreiben von Strukturen und Klassen, muss darauf geachtet werden, dass das Speicherlayout der Struktur dem Layout in der SPS entspricht. Das Layout kann mit dem Attribut StructLayoutAttribute festgelegt werden. Dabei muss als LayoutKind LayoutKind.Sequential ausgewählt werden und Pack auf 1 gesetzt werden. Die Klasse SimpleStruct ist deshalb wie folgt definiert worden:

```
[StructLayout(LayoutKind.Sequential, Pack=0)] // TwinCAT2->Pack=1 TwinCAT3->Pack=0
public class SimpleStruct
{
public double lrealVal;
public int dintVal1;
}
```

Wenn Arrays, Strings oder boolsche Werte in einer Struktur/Klasse definiert sind, muss man festlegen, wie diese gemarshallt werden. Dies geschieht mit Hilfe des MarshalAs Attributs. Da Arrays und Strings unter .Net normalerweise keine feste Länge haben, muss mit dem Property SizeConst die Grösse des Arrays bzw. des Strings festgelegt werden. Das Marshallen von Mehrdimensionale Arrays oder Arrays von

Strukturen ist mit dem Marhaler des .NET Frameworks 1.1 nicht möglich. Mehrdimensionale Arrays müssten auf eindimensionale Arrays abgebildet werden. Arrays von Strukturen müssten in der .NET Struktur aufgelöst werden.

In dem Beispiel wird das MarshalAsAttribute in der Klasse ComplexStruct verwendet:

```
[StructLayout(LayoutKind.Sequential, Pack=0)] // TwinCAT2->Pack=1 TwinCAT3->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();
}
```

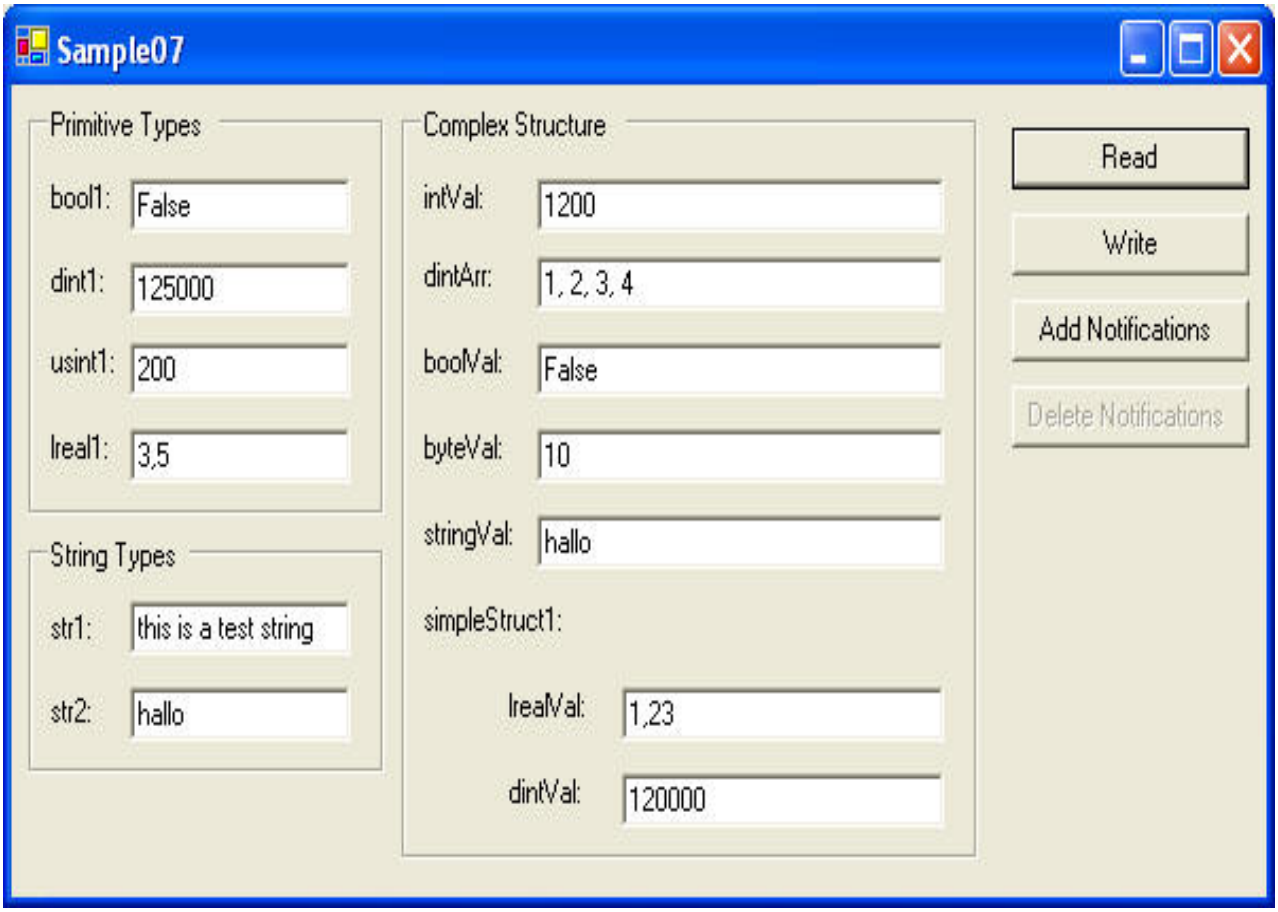
Anmelden von ADS Notifications

In der Event-Methode btnAddNotifications_Click wird die Methode AddDeviceNotificationEx dazu verwendet, Notifications für die SPS Variable anzumelden. Bei Änderung des Wertes der Variable wird das Event AdsNotificationEx ausgelöst. Der Unterschied zu dem Event AdsNotifcation, besteht darin das der Wert einer Variablen in einem Objekt gespeichert ist, anstatt in einem AdsStream. Der Methode AddNotificationEx muss dazu der Typ dieses Objektes übergeben werden:

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

Als User-Objekt wurde hier die Textbox, die den Wert anzeigen soll, übergeben. Wenn das Event ausgelöst wird, wird die Event-Methode **adsClient_AdsNotificationEx** aufgerufen. Dazu wird das Event in Form_Load angemeldet:

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



C# Programm

```
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 Programm

```
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 Statusänderung vom TwinCAT-Router und der SPS erkennen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample08.zip

Aufgabe

Erkennen von Statusänderungen des Routers und der SPS.

Beschreibung

Statusänderungen von ADS-Geräten können effektiv über Callback-Funktionen erkannt werden. Diese werden aufgerufen, wenn sich der Status eines ADS-Gerätes ändert.

Zur Erkennung von Statusänderungen des Routers enthält die Klasse TcAdsClient das Event TcAdsClient.AmsRouterNotification. Zur Erkennung von Statusänderungen der SPS kann eine ADS Notification auf den Status der SPS angemeldet werden. Sowohl für das Router-Event als auch für die SPS-Notification können Callback-Funktionen registriert werden, die bei den entsprechenden Statusänderungen aufgerufen werden.

Im folgenden Beispielprogramm werden die Statusänderungen des Routers und der SPS nach dem obigen Verfahren überwacht.

C# Programm

```
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-Summenkommando: Lesen oder Schreiben von mehreren Variablen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample09.zip

Dieses Beispiel zeigt, wie man unter Zuhilfenahme des ADS-Summenkommandos, viele Variablen lesen oder schreiben kann. Aufgebaut als TcAdsClient.ReadWrite dient es als Behälter, in dem die Unterkommandos in einem ADS-Stream transportiert werden.

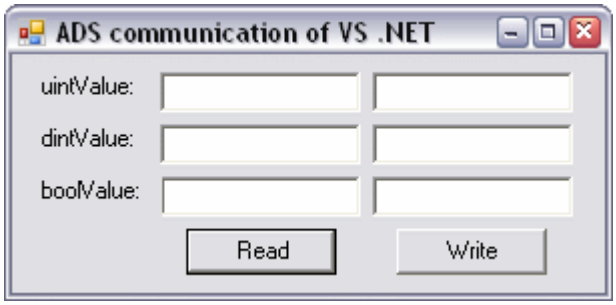


Abb. 1: TwinCAT.Ads_Sample8

Wichtig ist vor Beginn, dass die 'TwinCAT.Ads.dll' eingebunden wird! Hierfür einfach den *'Solution Explorer'* öffnen und *'Add References...'* unter *'References'* auswählen. Auf den Reiter *'Browse'* klicken und die DLL aus dem *'TwinCAT -> ADS API -> .NET'* Verzeichnis hinzufügen.

C# Programm

1. Variablen lesen

Zu Beginn werden zwei Strukturen definiert

```
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;
}
```

und ein paar Variablen global deklariert.

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

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

Beim Start der Anwendung wird eine Verbindung mit der PLC hergestellt und Handle Parameter in die Struktur geschrieben.

```
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;
}
}
```

Nach einem Klick auf den 'Read' Button, gibt die *'BlockRead'* Methode einen ADS Stream zurück. Es folgt das Prüfen der ADS Return Codes (Err), bevor die restlichen Daten aus dem Stream gelesen und in die Textboxen geschrieben werden.

```
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);
}
}
```

Die 'BlockRead' Methode nimmt ein Objekt vom Typ *'VariableInfo[]'* entgegen, in dem die Handle Parameter enthalten sind. Nach der Reservierung von Speicher, für die zu lesenden und schreibenden Werte, wird ein ADS Stream mit den Parametern vom *'VariableInfo[]'* Objekt beschrieben. Am Ende überträgt das Summenkommando die Subkommandos und ein ADS Stream wird zurückgegeben.

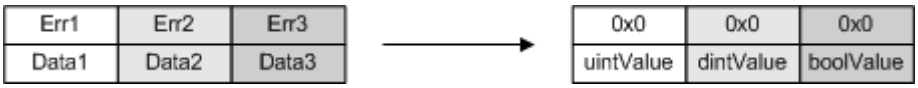
```
privateAdsStream BlockRead(VariableInfo[] variables)
{
// Allocate memoryint rdLength = variables.Length * 4;
int wrLength = variables.Length * 12;

// 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;
}
```

Die Parameter für das Summenkommando bestehen aus IndexGroup (0xF080) - Aufruf des Summenkommandos, *IndexOffset (variables.Length)* - Anzahl der Unterkommandos, *rdDataStream (rdStream)* - Speicher, der gelesene Daten entgegen nimmt, *wrDataStream (writer.BaseStream)* - Speicher, der zu sendende Daten enthält.

Subkommandos in *wrDataStream*

Abb. 2: TwinCAT.Ads_Sample9

Response in rdDataStream



2. Variablen schreiben Nach einem Klick auf den 'Write' Button, gibt die *'BlockRead2'* Methode einen ADS Stream zurück. Es folgt das Prüfen der ADS Return Codes (Err).

```
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);
}
}
```

Die 'BlockRead2' Methode nimmt ein Objekt vom Typ *'VariableInfo[]'* entgegen, in dem die Handle Parameter enthalten sind. Nach der Reservierung von Speicher, für die zu lesenden und schreibenden Werte, wird das *'MyStruct'* Objekt mit den Werten der Textboxen gefüllt. Anschließend wird ein ADS Stream mit den Parametern vom *'VariableInfo[]'* Objekt und den Werten vom *'MyStruct'* Objekt beschrieben. Am Ende überträgt das Summenkommando die Subkommandos und ein ADS Stream wird zurück gegeben.

```
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;
}
```

Die Parameter für das Summenkommando bestehen aus IndexGroup (0xF081) - Aufruf des Summenkommandos, *IndexOffset (variables.Length)* - Anzahl der Unterkommandos, *rdDataStream (rdStream)* - Speicher, der gelesene Daten entgegen nimmt, *wrDataStream (writer.BaseStream)* - Speicher, der zu sendende Daten enthält.

Subkommandos in *wrDataStream*

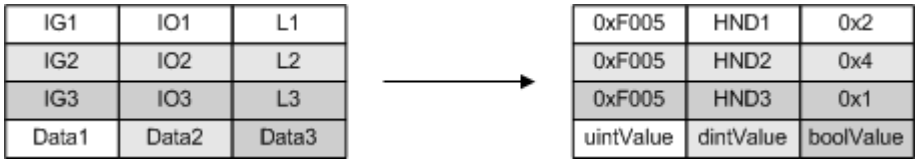


Abb. 3: TwinCAT.Ads_Sample11

Response in rdDataStream

3.10 Free Sample

Download

Sprache / IDE	Beispielprogram auspacken
Visual C#	-

3.11 Handle einer SPS Variablen löschen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample11.zip

Beschreibung

In diesem Beispiel wird das Handle einer SPS Variablen gelöscht:

C# Programm

```
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 Merker synchron aus der SPS lesen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample12.zip

Beschreibung

Bei diesem Beispielprogramm wird der Wert aus dem Merkerdoppelwort 0 der SPS ausgelesen und am Bildschirm angezeigt:

C# Programm

```
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 Merker synchron in die SPS schreiben

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample13.zip

Beschreibung

Bei diesem Beispielprogramm wird der Wert, den der Bediener eingegeben hat, in das Merkerdoppelwort 0 geschrieben:

C# Programm

```
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 SPS starten/stoppen

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample14.zip

Beschreibung

Mit dem folgenden Programm wird das Laufzeitsystem 1 der SPS gestartet, bzw. gestoppt:

C# Programm

```
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 Zugriff per Variablenname

Download

Voraussetzungen

Sprache / IDE	Beispielprogram auspacken
C# / Visual Studio	Sample15.zip

Beschreibung

Das folgende Programm greift auf eine SPS-Variable zu, die keine Adresse besitzt. Der Zugriff muss deshalb per Variablenname erfolgen. Die SPS-Variable wird vom Beispielprogramm wieder auf 0 zurückgesetzt, wenn diese den Wert 10 überschritten hat.

C# Programm

```
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 SPS Methoden Aufruf

Download: https://infosys.beckhoff.com/content/1031/tc3_adssamples_net/Resources/7860265995.zip

Das folgende Programm zeigt wie eine SPS Methode aus einem .NET-Programm aufgerufen werden kann via ADS. Die Abbildung 1 verdeutlicht, dass direkt über ADS die Methodenparameter („params“) übertragen werden können und das Ergebnis („result“) nach der Ausführung zurückgegeben wird.

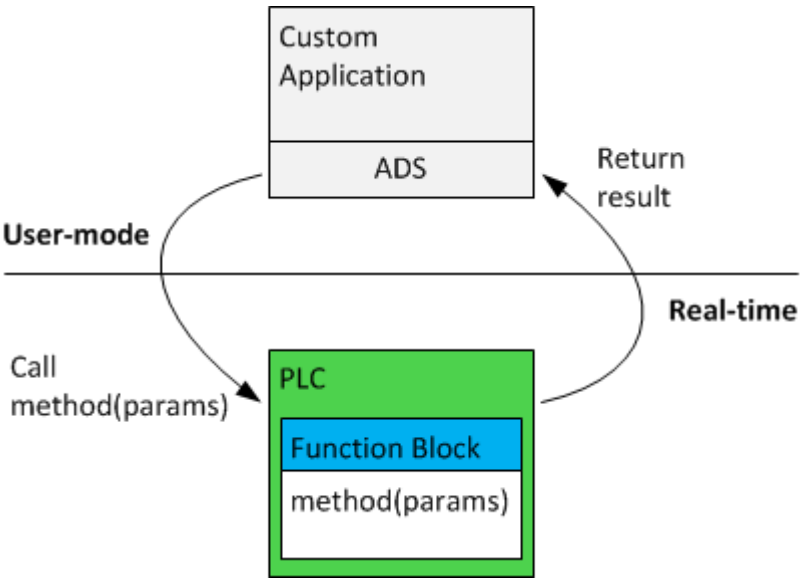


Abb. 4: Abbildung 1: ADS Methodenaufruf

Die Methoden werden nach der Ausführung der SPS aufgerufen. Es steht Ihr die verbleibende Rechenzeit des Zyklus zur Verfügung.

Ablauf

Folgende Schritte müssen beachtet werden, wenn Sie eine Methode in der SPS anbieten und aufrufen wollen:

1. Fügen Sie eine Methode für einen Baustein hinzu.
2. Über folgendes Attribut wird die Methode für den ADS Zugriff angeboten:

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

1. Erzeugen Sie einen Handle für die Methode für folgendes Symbol mit der Syntax:

```
Funktionsblock_Name#Methoden_Name
```

1. Anschließend kann die Methode über folgenden ADS Read/Write Befehl aufgerufen werden:

ADS Read/Write Parameter:

Indexgroup: ADSIGRP_SYM_VALBYHND (0xF005)

Offset: Methoden Handle (hMethod)

ReadData: ADS Daten mit dem Rückgabewert der Methode

WriteData: ADS Daten mit den Übergabeparametern

Einschränkungen

Wenn Sie per ADS Methoden aufrufen, müssen Sie folgende Punkte beachten:

1. CPU-Zeit: Wenn die Methoden mehr Rechenzeit benötigen als bereitsteht, können Echtzeitüberschreitungen auftreten.
2. Breakpoints: Werden innerhalb der Methoden nicht unterstützt und können Exceptions erzeugen.
3. Pointer: Wenn ein Pointer auf einen Array von Elementen zeigt kann mit dem Attribut TcRpcLengthIs die Länge angegeben werden.

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

Voraussetzungen

Laufzeitumgebung	Zielplattform
TwinCAT v3.1.0 Build 4016	PC or CX (x86, ARM)

Mehr Informationen:
www.beckhoff.de/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de