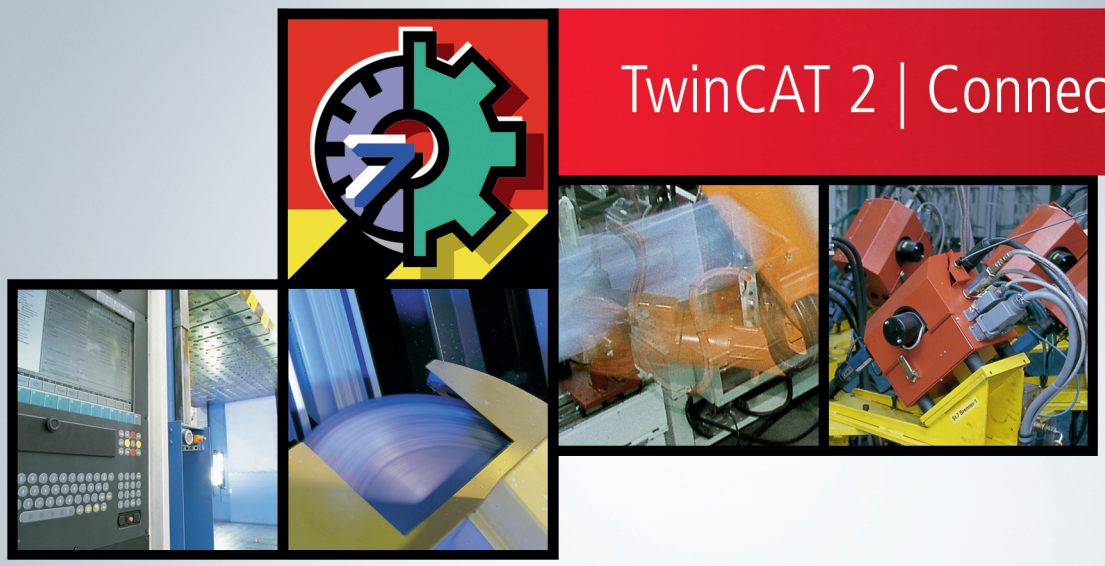
BECKHOFF New Automation Technology

Handbuch | DE

TX1000

TwinCAT 2 | ADS-OCX



TwinCAT 2 | Connectivity

06.10.2022 | Version: 1.1

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

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 Zugriff auf ADS-Geräte

Für den Zugriff auf die Daten eines ADS-Gerätes stehen mehrere Möglichkeiten zur Verfügung:

- synchron
- asynchron
- zyklisch

Je nach Applikationsumgebung (Kommunikationsmedium, Datenmenge, Datenübertragungsrate,...) hat jede dieser Möglichkeiten ihre Vorzüge, die weiter unten im Einzelnen erläutert werden. Um eine Variable in einem ADS-Gerät zu identifizieren, gibt es zwei weitere Varianten.

Per Adresse

Es wird eine Adresse angegeben. Die Adresse setzt sich zusammen aus der Index-Gruppe und dem Index-Offset. Die Adressaufteilung ist aus der entsprechenden Dokumentation des ADS-Gerätes zu entnehmen.

Per Variablenname

Beim Zugriff auf ADS-Geräte kann alternativ auch der Name einer ADS-Variablen angegeben werden.

Synchron [▶ 11]

Nach dem Aufruf der Schreib-/Lesemethode wird die Ausführung des Visual Basic-Programms so lange unterbrochen, bis die angeforderten Daten vorliegen. In den darauf folgenden Anweisungen kann mit den Daten sofort weiter gearbeitet werden. Der Vorteil dieser Zugriffsart ist, dass sehr wenig Programmieraufwand im Visual Basic-Programm durchgeführt werden muss.
Diese Zugriffsweise ist dann zu empfehlen, wenn sich das Visual Basic-Programm und das ADS-Gerät auf dem gleichen Rechner befinden oder über ein schnelles Netzwerk miteinander verbunden sind, so dass die Wartezeit sehr gering ist.
Beispiel: In einem Eingabefenster soll der Bediener verschiedene Parameter eingeben. Beim Betätigen eines Buttons, sollen die Daten in die SPS geschrieben werden. Da das Schreiben der Werte nicht zyklisch stattfindet, sondern von dem Verhalten des Bedieners abhängig ist, sollte in diesem Fall ein synchroner Schreibbefehl verwendet werden.

Asynchron [▶ 11]

Bei dem asynchronen Zugriff wird die Ausführung des Visual Basic-Programms nicht unterbrochen, sondern sofort mit der Abarbeitung des nächsten Befehls fortgefahren. Treffen die angeforderten Daten beim ADS-OCX ein, wird im Visual Basic-Programm eine Ereignisfunktion ausgelöst, in der als Parameter der Wert übergeben wird. Dadurch, dass das Visual Basic-Programm zu einem beliebigen Zeitpunkt seine Daten erhalten kann, ist dort ein größerer Programmieraufwand notwendig als bei der synchronen Zugriffsweise. Wenn der ADS-Server und das Visual Basic-Programm räumlich voneinander getrennt sind und das Datenübertragungsmedium sehr langsam ist, z.B. Modem oder ISDN, so ist die asynchrone Arbeitsweise sinnvoll.

Connect [▶ 11]

Sollen Werte kontinuierlich zu einem Visual Basic-Programm übertragen werden, so ist die zyklische Zugriffsmethode, auch 'per connect' genannt, die einfachste und effektivste Möglichkeit. Ein Methodenaufruf veranlasst, dass die Daten aus dem ADS-Gerät zyklisch oder bei Veränderung dem Visual Basic-Programm durch eine Ereignisfunktion übergeben werden.

Beispiel: In einem Anzeigefenster sollen die Positionen mehrerer Achsen angezeigt werden; alle 250ms. Benutzen Sie die Methode AdsReadVarConnectEx() [▶ 34] damit für jede Achsenposition alle 250ms das Ereignis AdsReadConnectUpdateEx() [▶ 54] ausgelöst wird. Dieses Prinzip kann noch weiter optimiert werden, so dass nur dann Werte übertragen werden, wenn sich die Position der Achse verändert (server on change)! Ein kleines Beispiel finden Sie unter 'Ereignisgesteuertes Lesen [▶ 76]'.

Rückgabewerte

Der Rückgabewert aller Methoden gibt Auskunft darüber, ob die durchgeführte Operation erfolgreich war oder ob ein Fehler aufgetreten ist. Allgemein gilt die Festlegung, dass der Rückgabewert 0 eine fehlerfreie Durchführung signalisiert. Eine genaue Auflistung der möglichen Rückgabewerte und ihre Bedeutungen finden Sie unter ADS-Fehlercodes.
Alternativ kann das ADS-OCX im Fehlerfall auch eine Exception auslösen. Dazu muss die Eigenschaft EnableErrorHandling [▶ 64] auf TRUE gesetzt werden. Über das Objekt *Err* kann die Fehlerursache ermittelt werden. Das Object *Err* wird in der Dokumentation von Visual Basic beschrieben.

Methodenübersicht

		per Adresse	per Variablenname
synchron	**Lesen**	AdsSyncReadReq() [▶ 26] AdsSyncReadBoolReq() [▶ 27] AdsSyncReadIntegerReq() [▶ 27] AdsSyncReadLongReq() [▶ 27] AdsSyncReadSingleReq() [▶ 27] AdsSyncReadDoubleReq() [▶ 27] AdsSyncReadStringReq() [▶ 27]	- AdsSyncReadBoolVarReq() [▶ 24]AdsSyncReadIntegerVarReq() [▶ 24]AdsSyncReadLongVarReq() [▶ 24]AdsSyncReadSingleVarReq() [▶ 24]AdsSyncReadDoubleVarReq() [▶ 24]AdsSyncReadStringVarReq() [▶ 24]
	Schreiben	AdsSyncWriteReq() [▶ 30] AdsSyncWriteBoolReq() [▶ 30] AdsSyncWriteIntegerReq() [▶ 30] AdsSyncWriteLongReq() [▶ 30] AdsSyncWriteSingleReq() [▶ 30] AdsSyncWriteDoubleReq() [▶ 30] AdsSyncWriteStringReq() [▶ 30]	- AdsSyncWriteBoolVarReq() [▶ 29] AdsSyncWriteIntegerVarReq() [▶ 29] AdsSyncWriteLongVarReq() [▶ 29] AdsSyncWriteSingleVarReq() [▶ 29] AdsSyncWriteDoubleVarReq() [▶ 29]AdsSyncWriteStringVarReq() [▶ 29]
asynchron	**Lesen**	AdsReadIntegerReq() [▶ 31] AdsReadLongReq() [▶ 31] AdsReadSingleReq() [▶ 31] AdsReadDoubleReq() [▶ 31] AdsReadStringReq() [▶ 31]	-
	Schreiben	AdsWriteIntegerReq() [▶ 33] AdsWriteLongReq() [▶ 33] AdsWriteSingleReq() [▶ 33] AdsWriteDoubleReq() [▶ 30] AdsWriteStringReq() [▶ 33]	-
connect	**Lesen**	AdsReadConnect() [▶ 41] AdsReadBoolConnect() [▶ 42] AdsReadIntegerConnect() [▶ 42] AdsReadLongConnect() [▶ 42] AdsReadSingleConnect() [▶ 42] AdsReadDoubleConnect() [▶ 42] AdsReadStringConnect() [▶ 42]	AdsReadVarConnectEx() [▶ 34]
	Schreiben	AdsWriteConnect() [▶ 49] AdsWriteBoolConnect() [▶ 50] AdsWriteIntegerConnect() [▶ 50] AdsWriteLongConnect() [▶ 50] AdsWriteSingleConnect() [▶ 50] AdsWriteDoubleConnect() [▶ 50]	AdsWriteVarConnect() [▶ 46] AdsWriteBoolVarConnect() [▶ 47] AdsWriteIntegerVarConnect() [▶ 47] AdsWriteLongVarConnect() [▶ 47] AdsWriteSingleVarConnect() [▶ 47] AdsWriteDoubleVarConnect() [▶ 47]

3 Manuelle Installation des ADS OCX

Das ADS OCX kann mit Regsvr32 eingefügt werden.

✓ Dazu ist der Pfad der zu registrierenden Datei anzugeben.

1. Wählen Sie **Start > Ausführen**
2. Geben Sie *Regsvr32 <Pfad zur AdsOcx-Datei>\AdsOcs.ocx* ein.

⇨ ADS OCX ist mit Regsvr32 eingefügt worden.

Ein Reboot des Rechners ist nicht erforderlich.

4 API

4.1 allgemein

4.1.1 AboutBox

Zeigt ein Informationsfenster mit der aktuellen Versionsnummer und dem Copyrightvermerk des ADS-OCX an.

```
object.AboutBox( )
```

Parameter

-

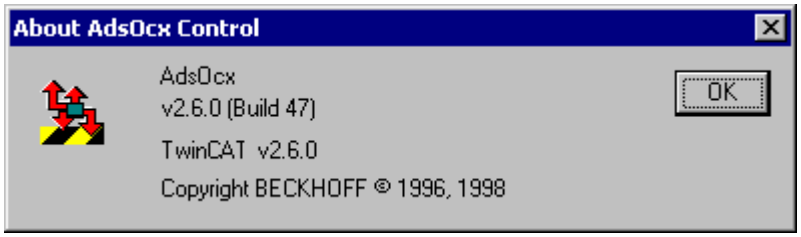
Rückgabewert

-

Bemerkungen

-

Beispiel



4.1.2 AdsAmsDisconnect

Über diese Methode wird das ADS-OCX vom TwinCAT-Router abgemeldet.

```
object.AdsAmsDisconnect() As Long
```

Parameter

-

Rückgabewert

-

Bemerkungen

Meldet sich der aktuelle Benutzer von Windows NT/2000/XP ab, so werden alle Anwendungen beendet. Enthält das Programm das ADS-OCX, welches mit dem Router verbunden ist, so muss das Programm sich vom TwinCAT-Router abmelden. Wird dieses nicht gemacht, so kann das Programm nicht komplett entladen werden; es ist nach dem erneuten Anmelden noch im NT-Task-Manager zu sehen.

Das Abmelden vom TwinCAT-Router wird durch die Methode *AdsAmsDisconnect()* realisiert. Diese sollte in dem Ereignis *Form_Unload()* aufgerufen werden.

Beispiel

```
Private Sub Form_Unload(Cancel As Integer)
  Call AdsOcx1.AdsAmsDisconnect
End Sub
```

4.1.3 AdsAmsPortEnabled

Über diese Methode kann festgestellt werden, ob der AMS-Port für die Kommunikation zur Verfügung steht.

```
object.AdsAmsPortEnabled() As Boolean
```

Parameter

-

Rückgabewert

-

Bemerkungen

-

Beispiel

Im folgenden Beispiel wird eine Funktion gezeigt, in der Meldungen in die Ereignisanzeige von Windows NT/2000/XP geschrieben werden. Dazu wird die Methode AdsLogFmtString() [▶ 19] benutzt. Da der Zugriff auf die Ereignisanzeige über den TwinCAT-Router stattfindet, sollte die Methode nur dann aufgerufen werden, wenn der AMS-Port aktiv ist.

```
'Meldungen über ADS in die Ereignisanzeige schreiben
Public Function LogMsg (MsgType As ADSLOGMSGTYPE, MsgStr As String)
  If (AdsOcx.AdsAmsPortEnabled = True) Then
    MsgStr = Left(MsgStr, 250)
    Call AdsOcx.AdsLogFmtString(MsgType, MsgStr, 0, 0, 0, 0)
  End If
End Function
```

4.1.4 AdsCreateVarHandle

Erzeugt einen eindeutigen Handle von einer ADS-Variablen.

```
object.AdsCreateVarHandle(
  varName As String,
  hVar As Long
) As Long
```

Parameter

varName

[in] Name der ADS-Variable

hVar

[out] Handle der ADS-Variable

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen zur SPS:

Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie dem Handbuch des PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion 'Funk1' die Variable 'SPSVar1' angesprochen werden, so muss als erster Parameter 'Funk1.SPSVar1' angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. '.SPSGlobVar'. Der Parameter 'varName' unterscheidet nicht zwischen Groß- und Kleinbuchstaben. Wenn in einer Form nur bestimmte SPS-Variablen benötigt werden, sollte der Handle erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden. Siehe auch die AdsDeleteVarHandle() [▶ 15] Methode.

Bemerkungen zur NC:

HINWEIS

Symbol-Download bei jeder Achse aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

Beispiel

-

4.1.5 AdsDeleteVarHandle

Gibt den Handle einer SPS-Variablen wieder frei.

```
object.AdsDeleteVarHandle(hVar As Long) As Long
```

Parameter

hVar

[in] Handle der ADS-Variable

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird eine ADS-Variable, die über einen Handle angesprochen wird, nicht mehr benötigt, sollte dieses über die Methode AdsDeleteVarHandle() wieder freigegeben werden. Wenn in einer Form nur bestimmte ADS-Variablen benötigt werden, sollte der Handle erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden. Siehe auch Methode AdsCreateVarHandle() [▶ 14].

Beispiel

-

4.1.6 AdsEnableLogNotification

Setzt den Filter für das Empfangen von Meldungen über den TwinCAT-Router.

```
object.AdsEnableLogNotification(
  nBasePort As Long,
  nPorts As Long,
  dwCtrlMask As Long
) As Long
```

Parameter

nBasePort

[in] Erste Portnummer, für die das Ereignis AdsLogNotification() [▶ 53] ausgelöst wird

nPorts

[in] Anzahl der Ports ab *nBasePort*, für die das Ereignis AdsLogNotification() [▶ 53] ausgelöst wird

dwCtrlMask

[in] Filtermaske, für die Art der Nachrichten die gemeldet werden sollen (siehe Datentyp ADSLOGMSGTYPE [▶ 66]

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

ADS-Geräte haben die Möglichkeit Meldungen über den TwinCAT-Router an andere ADS-Geräte zu schicken. Bevor ein ADS-Gerät mit Hilfe des ADS-OCX Meldungen empfangen kann, muss mit Hilfe der Methode AdsEnableLogNotification() ein Filter definiert werden. Hierdurch wird festgelegt, welche Meldungen signalisiert werden.

Zum einen definiert der Filter ein Bereich von Portnummern. Alle Meldungen von den ADS-Geräten, die in diesem Portnummern-Bereich liegen, werden durch das Ereignis AdsLogNotification() [▶ 53] gemeldet.

Der zweite Parameter, mit dem Meldungen gefiltert werden können, ist die Art der Meldung. Es wird unterschieden zwischen Hinweis, Warnung und Fehler (siehe ADSLOGMSGTYPE [▶ 66]). Durch eine ODER-Verknüpfung können auch verschiedene Meldearten empfangen werden.

Beispiel

Visual Basic Beispiel: 'Meldungen über den TwinCAT-Router senden/empfangen [▶ 82]'

4.1.7 AdsEnumSymbols

Mit dieser Methode kann die Liste der deklarierten Variablen aus einem ADS-Gerät ausgelesen werden.

```
object.AdsEnumSymbols(
  strSymbolName As String,
  nSymbolType As Long,
  cbSymbolSize As Long,
  strComment As String,
  nIndexGroup As Long,
  nIndexOffset As Long,
  bNextAs Boolean
) As Long
```

Parameter

strSymbolName

[out] Name der ADS-Variable

nSymbolType

[out] Datentyp der ADS-Variable (siehe Datentyp ADSDATATYPEID [▶ 65])

cbSymbolSize

[out] Datenlänge der ADS-Variable in Byte

strComment

[out] Kommentar hinter der ADS-Variablendeklaration

nIndexGroup

[out] Index-Gruppe der ADS-Variable

nIndexOffset

[out] Index-Offset der ADS-Variable

bNext

[in] TRUE für die erste ADS-Variable, FALSE für alle folgenden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Bei dem ersten Methodenaufruf von AdsEnumSymbols() müssen Sie den Parameter *bNext* auf FALSE setzen. Dadurch werden alle Informationen über die erste Variable ausgelesen. Bei jeden weiteren Aufruf von AdsEnumSymbols() muss der Parameter auf TRUE stehen. Dadurch werden die Informationen von der folgenden Variablen ausgelesen.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie dem Handbuch des PLC-Control entnehmen.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Beispiel

Visual Basic Beispiel: 'SPS-Variablendeklaration auslesen [▶ 78]'

4.1.8 AdsSetFirstDynSymbol

Mit dieser Methode kann die Liste der deklarierten Variablen aus einem ADS-Gerät ausgelesen werden.

```
object.AdsSetFirstDynSymbol(bForceReload As Boolean) As Long
```

Parameter

bForceReload

[in] TRUE wenn ein (neues) Laden der Symbolinformation vom Server gewünscht wird. Sind noch keine Symbolinformationen vorhanden, werden diese unabhängig von *bForceReload* geladen.

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Bei dem Methodenaufruf von AdsSetFirstDynSymbol() wird der interne "Zeiger" auf das akuelle Symbol, welches mit AdsGetNextDynSymbol [▶ 18]() geladen werden kann, auf den Anfang zurück gesetzt.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Beispiel

Visual Basic Beispiel: 'SPS-Variablendeklaration auslesen [▶ 78]'

4.1.9 AdsGetNextDynSymbol

Mit dieser Methode kann die Liste der deklarierten Variablen aus einem ADS-Gerät ausgelesen werden.

```
object.AdsGetNextDynSymbol(
  navType As ADSGETDYNSYMBOLTYPE,
  bstrName As String,
  bstrFullName As String,
  bstrType As String,
  bstrComment As String,
  adsType As Long,
  symbolSize As Long,
  nIndexGroup As Long,
  nIndexOffset As Long
) As Long
```

Parameter

navType

[in] Navigationsvorgabe im Symbolbaum (siehe Datentyp ADSGETDYNSYMBOLTYPE [▶ 67])

bstrName

[out] Name des Symbols (Kurzform ohne vorangestellte Namen des Parent)

bstrFullName

[out] Vollständiger Name des Symbols

bstrType

[out] Name des Datentyps des Symbols

strComment

[out] Kommentar hinter der ADS-Variablendeklaration

adsType

[out] Datentyp der ADS-Variable (siehe Datentyp ADSDATATYPEID [▶ 65])

symbolSize

[out] Bytelänge des Symbols

nIndexGroup

[out] Index-Gruppe der ADS-Variable

nIndexOffset

[out] Index-Offset der ADS-Variable

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Beim *navType* **ADSDYNSYM_GET_NEXT** wird durch den gesamten Symbolbaum navigiert. Hiermit können auf einfache Weise alle Symbole ausgelesen werden. Die drei anderen *navTypes* können zur gesteuerten Navigation durch den Symbolbaum benutzt werden.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Beispiel

Visual Basic Beispiel: 'SPS-Variablendeklaration auslesen [▶ 78]'

4.1.10 AdsLogFmtString

Setzt eine Meldung über den TwinCAT-Router ab.

```
object.AdsLogFmtString(
  nMsgType As ADSLOGMSGTYPE,
  strFmt As String,
  arg0 As Variant,
  arg1 As Variant,
  arg2 As Variant,
  arg3 As Variant
) As Long
```

Parameter

nMsgType

[in] Art der Meldung (siehe Datentyp ADSLOGMSGTYPE [▶ 66])

strFmt

[in] Meldetext der abgesetzt werden soll

arg0

[in] 1. Parameter im Meldetext

arg1

[in] 2. Parameter im Meldetext

arg2

[in] 3. Parameter im Meldetext

arg3

[in] 4. Parameter im Meldetext

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Die abgesetzte Meldung wird bei allen ADS-Geräten signalisiert, bei denen die Filterbedingungen erfüllt sind. Außerdem wird die abgesetzte Meldung in den Event Logger von Windows NT/2000/XP geschrieben.

Es gibt drei Arten von Meldungen: Hinweis, Warnung und Fehler. Die abgesetzte Meldung muss einer dieser drei Meldungsarten angehören. In dem Meldestring können bis zu vier numerischen Parametern angeben. Als Platzhalter können folgende Buchstaben benutzt werden:

Platzhalter	Bedeutung
%d	Platzhalter für eine Variable vom Typ Long/Integer
%f	Platzhalter für eine Variable vom Typ Single/Double
%x	Platzhalter für eine Variable vom Typ Hexadezimal
%X	Platzhalter für eine Variable vom Typ Hexadezimal

Dabei wird der erste Platzhalter mit dem ersten Parameter (arg0) belegt, der zweite Platzhalter mit dem zweiten Parameter (arg1), usw.

HINWEIS

Zu viele Meldungen in kurzer Zeit

Achten Sie darauf, dass nicht zu viele Meldungen in kurzer Zeit übertragen werden, da dieses das Gesamtsystem sonst beeinträchtigen könnte.

Meldungen protokollieren

Wollen Sie in Ihrem Programm Meldungen protokollieren (z. B. Störungen einer Maschine), so sollten Sie dafür den TwinCAT-Event Logger benutzen. Dieser ist deutlich leistungsfähiger als der Event Logger von Windows NT/2000/XP und auf die Anforderungen der Automatisierungstechnik abgestimmt.

Beispiel

Visual Basic Beispiel: 'Meldungen über den TwinCAT-Router senden/empfangen [▶ 82]'

4.1.11 AdsReadSymbolDesc

Mit der Methode AdsReadSymbolDesc() können Informationen einzelner Symbole (Variablen) von ADS-Geräten ermittelt werden.

```
object.AdsReadSymbolDesc(
  strSymbolName As String,
  nSymbolType As ADSDATATYPEID,
  cbSymbolSize As Long,
  strComment As String,
  nIndexGroup As Long,
  nIndexOffset As Long
) As Long
```

Parameter

strSymbolName

[in] Name der ADS-Variable, von der die Informationen ausgelesen werden sollen

nSymbolType

[out] Datentyp der ADS-Variable (siehe Datentyp ADSDATATYPID [▶ 65])

cbSymbolSize

[out] Datenlänge der ADS-Variable in Byte

strComment

[out] Kommentar hinter der ADS-Variablendeklaration

nIndexGroup

[out] Index-Gruppe der ADS-Variable

nIndexOffset

[out] Index-Offset der ADS-Variable

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wollen Sie die Informationen aller ADS-Variablen aus einem ADS-Gerät auslesen, so finden Sie dieses im Beispiel 'SPS-Variablendeklaration auslesen [▶ 78]'.

HINWEIS

Beim PLC-Control den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Beispiel

-

4.1.12 AdsReadSymbolInfo

Mit der Methode AdsReadSymbolInfo() können Informationen über die Symbole (Variablen) von ADS-Geräten ermittelt werden.

```
object.AdsReadSymbolInfo(
  pSymbolsAvailable As Long,
  pBufSizeNeeded As Long
) As Long
```

Parameter

pSymbolsAvailable

[out] Anzahl der Symbole im ADS-Gerät

pBufSizeNeeded

[out] Länge der Daten in Byte, in der die Symbolinformationen gespeichert werden sollen

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Bevor mit der Methode AdsEnumSymbols() [▶ 16] die Symbolliste auslesen werden kann, muß mit der Methode AdsReadSymbolInfo() die Anzahl der Symbole und die Größe der Symbolliste ermittelt werden.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muß bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Beispiel

Visual Basic Beispiel: 'SPS-Variablendeklaration auslesen [▶ 78]'

4.1.13 AdsSyncWriteControlReq

Ändert den Status eines ADS-Gerätes.

```
object.AdsSyncWriteControlReq(
  ADSSTATE As Long,
  deviceState As Long,
  length As Long,
  pData As Integer
) As Long
```

Parameter

ADSSTATE

[in] neuer Zustand des ADS-Gerätes (siehe Datentyp ADSSTATE [▶ 66])

deviceState

[in] reserviert

length

[in] Länge der Daten in Byte

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabeparameter

Siehe ADS-Fehlercodes

Bemerkungen

Außer dem Ändern des ADS-Status ist es zusätzlich noch möglich Daten zum ADS-Gerät zu schicken. Ob und wie diese Daten ausgewertet werden, ist von den einzelnen ADS-Geräten abhängig. Die mit TwinCAT ausgelieferten ADS-Geräte (SPS, NC/NCI, Nockenschaltwert, ...) werten diese Informationen nicht aus.

Beispiel

Visual Basic Beispiel: 'Statusänderung vom TwinCAT-Router und der SPS erkennen/verändern [▶ 80]'

4.1.14 AdsWriteControlReq

Ändert den ADS-Status und den Geräte-Status von dem ADS-Server.

```
object.AdsWriteControlReq(
  nInvokeId As Long,
  nAdsState As Long,
  nDeviceState As Long,
  cbLength As Long,
  pData As Integer
) As Long
```

Parameter

nInvokeId

[in] Auftragsnummer zur Identifizierung der Antwort

nAdsState

[in] neuer ADS-Status (siehe Datentyp ADSSTATE [▶ 66])

nDeviceState

[in] neuer Geräte-Status

cbLength

[in] Länge der Daten in Byte

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Außer dem Ändern von dem ADS-Status und dem Geräte-Status ist es zusätzlich noch möglich, Daten zum ADS-Server zu schicken, um weitere Informationen zu übertragen. Bei den aktuellen ADS-Geräten (PLC, NC, ...) werden diese Daten nicht weiter ausgewertet.
Jedes ADS-Gerät kann seinen aktuellen Zustand anderen ADS-Geräten mitteilen. Dabei wird zwischen den Status des Gerätes selbst (DeviceState) und dem Status der ADS-Schnittstelle von dem ADS-Gerät (AdsState) unterschieden. Die möglichen Zustände, die die ADS-Schnittstelle annehmen kann, ist durch die ADS-Spezifikation festgelegt.

Beispiel

-

4.1.15 ShowPropertyPages

Zeigt das Eigenschaftsfenster vom ADS-OCX an.

```
object.ShowPropertyPages( ) As Long
```

Parameter

-

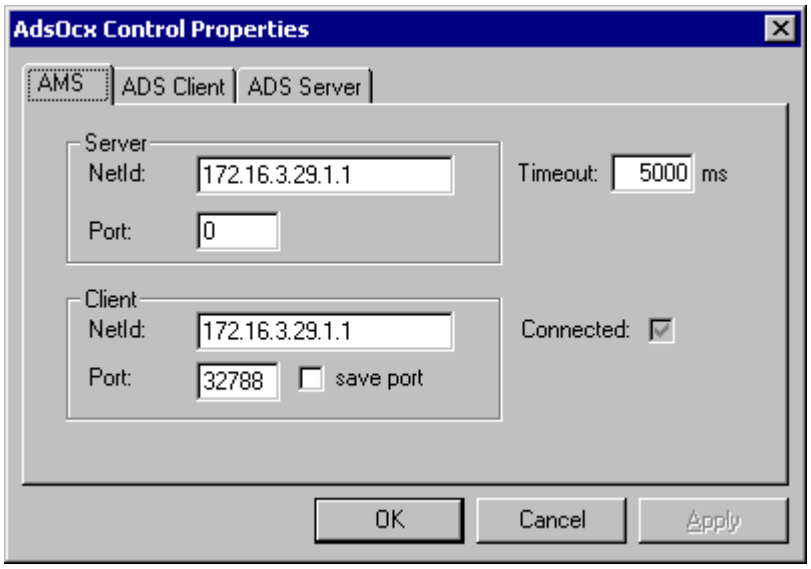
Rückgabewert

Siehe ADS-Fehlercodes

Anmerkungen

-

Beispiel



4.2 synchron

4.2.1 AdsSyncRead[Datatype]VarReq

AdsSyncReadBoolVarReq

AdsSyncReadIntegerVarReq

AdsSyncReadLongVarReq

AdsSyncReadSingleVarReq

AdsSyncReadDoubleVarReq

AdsSyncReadStringVarReq

Liest synchron Daten von einem ADS-Gerät an und schreibt diese in eine Visual Basic-Variable vom Typ Boolean, Integer, Long, Single, Double oder String.

```
object.AdsSyncRead[Datatype]VarReq(
  hVar As Long,
  cbLength As Long,
  pData As [Datatype]
) As Long
```

Parameter

hVar

[in] Handle der ADS-Variable (siehe Methode AdsCreateVarHandle() [▶ 14])

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

HINWEIS

VB-Variable wird auf "0" gesetzt

Im Falle eines Fehlers wird die VB-Variable (pData), deren Wert beschrieben werden sollte, auf "0" gesetzt.

Bemerkungen

Die Ausführung des Visual Basic-Programms wird so lange angehalten, bis die Daten vom ADS-Gerät vorliegen oder bis die Zeit in der Eigenschaft AdsAmsCommTimeout [▶ 60] überschritten ist.
Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

VB Beispiel

```
Dim hVar As Long
Dim VBVar As Single
'Handle der SPS-Variable holen
Call AdsOcx1.AdsCreateVarHandle("MAIN.PLCVar", hVar)
'Variable auslesen
Call AdsOcx1.AdsSyncReadSingleVarReq(hVar, 4&, VBVar)
'Variablen anzeigen
Label1.Caption = VBVar
'Handle wieder freigeben
Call AdsOcx1.AdsDeleteVarHandle(hVar)
```

```
Dim hVar As Long
Dim VBVar As String
'Handle der SPS-Variable holen
Call AdsOcx1.AdsCreateVarHandle("MAIN.PLCVar", hVar)
'Visual Basic initialisieren
VBVar = Space(10)
'Variable auslesen
Call AdsOcx1.AdsSyncReadStringVarReq(hVar, LenB(VBVar), VBVar)
'Variablen anzeigen
Label1.Caption = VBVar
'Handle wieder freigeben
Call AdsOcx1.AdsDeleteVarHandle(hVar)
```

Delphi Beispiel

```
procedure TForm1.Button1Click(Sender: TObject);
var     res1, res2, res3 :integer;
    //handles
    hBoolean, hSmallint, hLongint, hSingle, hDouble, hString : integer;
    //read buffer
    vWordBool : WordBool;
    vSmallint : Smallint;
    vLongint : Longint;
    vSingle : Single;
    vDouble : Double;
    vString : WideString;
begin
    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vBOOL', hBoolean );
```

```
    res2 :=  AdsOcx1.AdsSyncReadBoolVarReq( hBoolean, sizeof(vWordBool), vWordBool );
    res3 := AdsOcx1.AdsDeleteVarHandle( hBoolean );
    Label1.Caption := Format('res1: %d, res2: %d, res3: %d,  Value: %s', [res1, res2, res3, BoolToSt
r(vWordBool, TRUE)]);

    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vINT', hSmallint );
    res2 :=  AdsOcx1.AdsSyncReadIntegerVarReq( hSmallint, sizeof(vSmallint), vSmallint );
    res3 := AdsOcx1.AdsDeleteVarHandle( hSmallint );
    Label2.Caption := Format('res1: %d, res2: %d, res3: %d,  Value: %d', [res1, res2, res3, vSmallin
t]);

    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vDINT', hLongint );
    res2 :=  AdsOcx1.AdsSyncReadLongVarReq( hLongint, sizeof(vLongint), vLongint );
    res3 := AdsOcx1.AdsDeleteVarHandle( hLongint );
    Label3.Caption := Format('res1: %d, res2: %d, res3: %d,  Value: %d', [res1, res2, res3, vLongint
]);

    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vREAL', hSingle );
    res2 :=  AdsOcx1.AdsSyncReadSingleVarReq( hSingle, sizeof(vSingle), vSingle );
    res3 := AdsOcx1.AdsDeleteVarHandle( hSingle );
    Label4.Caption := Format('res1: %d, res2: %d, res3: %d,  Value: %f', [res1, res2, res3, vSingle]
);

    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vLREAL', hDouble );
    res2 :=  AdsOcx1.AdsSyncReadDoubleVarReq( hDouble, sizeof(vDouble), vDouble );
    res3 := AdsOcx1.AdsDeleteVarHandle( hDouble );
    Label5.Caption := Format('res1: %d, res2: %d, res3: %d,  Value: %f', [res1, res2, res3, vDouble]
);

    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vSTRING', hString );
    SetLength(vString,80{standard length of the PLC string variable});
    res2 :=  AdsOcx1.AdsSyncReadStringVarReq( hString, Length(vString)*2{byte length!}, vString );
    res3 := AdsOcx1.AdsDeleteVarHandle( hString );
    Label6.Caption := Format('res1: %d, res2: %d, res3: %d,  Length: %d, Value: %s', [res1, res2, re
s3, Length(vString), vString]);
end;
```

4.2.2 AdsSyncReadReq

Liest Daten von einem beliebigen Typ synchron aus einem ADS-Gerät.

```
object.AdsSyncReadReq(
  nIndexGroup As Long,
  nIndexOffset As Long,
  cbLength As Long,
  pData As YY
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

HINWEIS

VB-Variable wird auf "0" gesetzt

Im Falle eines Fehlers wird die VB-Variable (pData), deren Wert beschrieben werden sollte, auf "0" gesetzt.

Bemerkung

Die Ausführung des Visual Basic-Programms wird so lange angehalten, bis die Daten vom ADS-Gerät vorliegen oder bis die Zeit in der Eigenschaft AdsAmsCommTimeout [▶ 60] überschritten ist.
Die Visual Basic-Variable muss als Array deklariert werden. An die Methode wird das gesamte Array übergeben.
Es wird nicht der Variablentyp String unterstützt.

Beispiel

```
Dim VBVarInteger(0) As Integer
Dim VBVarLong(0) As Long
Dim VBVarSingle(0) As Single
Dim VBVarDouble(0) As Double
Dim VBVarByte(0) As Byte
Dim VBVarBool(0) As Boolean

'Variablen auslesen
Call AdsOcx1.AdsSyncReadReq(&H4020&, 0&, 2&, VBVarInteger)
Call AdsOcx1.AdsSyncReadReq(&H4020&, 2&, 4&, VBVarLong)
Call AdsOcx1.AdsSyncReadReq(&H4020&, 6&, 4&, VBVarSingle)
Call AdsOcx1.AdsSyncReadReq(&H4020&, 10&, 8&, VBVarDouble)
Call AdsOcx1.AdsSyncReadReq(&H4020&, 18&, 1&, VBVarByte)
Call AdsOcx1.AdsSyncReadReq(&H4021&, 152&, 2&, VBVarBool)

'Variablen anzeigen
lblInteger.Caption = VBVarInteger(0)
lblLong.Caption = VBVarLong(0)
lblSingle.Caption = VBVarSingle(0)
lblDouble.Caption = VBVarDouble(0)
lblByte.Caption = VBVarByte(0)
lblBool.Caption = VBVarBool(0)
```

4.2.3 AdsSyncRead[Datatype]Req

AdsSyncReadBoolReq

AdsSyncReadIntegerReq

AdsSyncReadLongReq

AdsSyncReadSingleReq

AdsSyncReadDoubleReq

AdsSyncReadStringReq

Liest synchron Daten von einem ADS-Gerät an und schreibt diese in eine Visual Basic-Variable vom Typ Boolean, Integer, Long, Single, Double oder String.

```
object.AdsSyncRead[Datatype]Req(
  nIndexGroup As Long,
  nIndexOffset As Long,
  cbLength As Long,
  pData As [Datatype]
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

HINWEIS

VB-Variable wird auf "0" gesetzt

Im Falle eines Fehlers wird die VB-Variable (pData), deren Wert beschrieben werden sollte, auf "0" gesetzt.

Bemerkungen

Die Ausführung des Visual Basic-Programms wird so lange angehalten, bis die Daten vom ADS-Gerät vorliegen oder bis die Zeit in der Eigenschaft AdsAmsCommTimeout [▶ 60] überschritten ist.
Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

VB Beispiel

```
Dim VBVar As Long
'Wert auslesen
Call AdsOcx1.AdsSyncReadLongReq(&H4020&, 0&, 8&, VBVar)
'Variablen anzeigen
Label1.Caption = VBVar
```

```
Dim VBVar As String
'Visual Basic Variable initialisieren
VBVar = Space(10)
"Wert aus Variable auslesen
Call AdsOcx1.AdsSyncReadStringReq(&H4020&, 0&, LenB(VBVar), VBVar)
'Variablen in Form anzeigen
Label1.Caption = VBVar
```

Delphi Beispiel

```
procedure TForm1.Button2Click(Sender: TObject);
var     res :integer;
    //read buffer
    vWordBool : WordBool;
    vSmallint : Smallint;
    vLongint : Longint;
    vSingle : Single;
    vDouble : Double;
    vString : WideString;
begin
    res :=  AdsOcx1.AdsSyncReadBoolReq( $4020, 0, sizeof(vWordBool), vWordBool );
    Label1.Caption := Format('res: %d,  Value: %s', [res, BoolToStr(vWordBool, TRUE)]);

    res :=  AdsOcx1.AdsSyncReadIntegerReq( $4020, 2, sizeof(vSmallint), vSmallint );
    Label2.Caption := Format('res: %d,  Value: %d', [res, vSmallint]);

    res :=  AdsOcx1.AdsSyncReadLongReq( $4020, 4, sizeof(vLongint), vLongint );
    Label3.Caption := Format('res: %d,  Value: %d', [res, vLongint]);

    res :=  AdsOcx1.AdsSyncReadSingleReq( $4020, 16, sizeof(vSingle), vSingle );
    Label4.Caption := Format('res: %d,  Value: %f', [res, vSingle]);

    res :=  AdsOcx1.AdsSyncReadDoubleReq( $4020, 32, sizeof(vDouble), vDouble );
    Label5.Caption := Format('res: %d,  Value: %f', [res, vDouble]);

    SetLength(vString,80{standard length of the PLC string variable});
```

```
    res :=  AdsOcx1.AdsSyncReadStringReq( $4020, 64, Length(vString)*2{byte length!}, vString );
    Label6.Caption := Format('res: %d,  Length: %d, Value: %s', [res, Length(vString), vString]);
end;
```

4.2.4 AdsSyncWrite[Datatype]VarReq

AdsSyncWriteBoolVarReq

AdsSyncWriteIntegerVarReq

AdsSyncWriteLongVarReq

AdsSyncWriteSingleVarReq

AdsSyncWriteDoubleVarReq

AdsSyncWriteStringVarReq

Fordert synchron Daten von einem ADS-Gerät an und schreibt diese in eine Visual Basic-Variable vom Typ Boolean, Integer, Long, Single, Double oder String.

```
object.AdsSyncWrite[Datatype]VarReq(
  hVar As Long,
  length As Long,
  pData As [Datatype]
) As Long
```

Parameter

hVar

[in] Handle der ADS-Variable (siehe Methode AdsCreateVarHandle() [▶ 14])

length

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Die Ausführung des Visual Basic-Programms wird so lange angehalten, bis die Daten vom ADS-Gerät vorliegen oder bis die Zeit in der Eigenschaft AdsAmsCommTimeout [▶ 60] überschritten ist.
Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

Beispiel

```
Dim hVar As Long
Dim VBVar As Double
Call AdsOcx1.AdsCreateVarHandle("MAIN.PLCVar", hVar)
VBVar = 3,1415
Call AdsOcx1.AdsSyncWriteDoubleVarReq(hVar, 8&, VBVar)
Call AdsOcx1.AdsDeleteVarHandle(hVar)
```

```
Dim hVar As Long
Dim VBVar As String
'Handle holen
Call AdsOcx1.AdsCreateVarHandle("MAIN.PLCVar", hVar)
VBVar = "TwinCAT"
Call AdsOcx1.AdsSyncWriteStringVarReq(hVar, LenB(VBVar), VBVar)
'Handle freigeben
Call AdsOcx1.AdsDeleteVarHandle(hVar)
```

4.2.5 AdsSyncWriteReq

Schreibt Daten von einen beliebigen Typ synchron in ein ADS-Gerät.

```
object.AdsSyncWriteReq(nIndexGroup As Long,
  nIndexOffset As Long,
  cbLength As Long,
  pData As YY
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Die Ausführung des Visual Basic-Programms wird so lange angehalten, bis das ADS-Gerät die Daten empfangen hat oder bis die Zeit in der Eigenschaft AdsAmsCommTimeout [▶ 60] überschritten ist.
Die Visual Basic-Variable muss als Array deklariert werden. An die Methode wird das gesamte Array übergeben.
Es wird nicht der Variablentyp String unterstützt.

Beispiel

```
Dim VBVarInteger(0) As Integer
Dim VBVarLong(0) As Long
Dim VBVarSingle(0) As Single
Dim VBVarDouble(0) As Double
Dim VBVarByte(0) As Byte
Dim VBVarBoolean(0) As Boolean

VBVarInteger(0) = 123
VBVarLong(0) = 456
VBVarSingle(0) = 3,1415
VBVarDouble(0) = 2,876
VBVarByte(0) = 7
VBVarBoolean(0) = False

'Werte in SPS schreiben
Call AdsOcx1.AdsSyncWriteReq(&H4020&, 0&, 2&, VBVarInteger)
Call AdsOcx1.AdsSyncWriteReq(&H4020&, 2&, 4&, VBVarLong)
Call AdsOcx1.AdsSyncWriteReq(&H4020&, 6&, 4&, VBVarSingle)
Call AdsOcx1.AdsSyncWriteReq(&H4020&, 10&, 8&, VBVarDouble)
Call AdsOcx1.AdsSyncWriteReq(&H4020&, 18&, 1&, VBVarByte)
Call AdsOcx1.AdsSyncWriteReq(&H4021&, 152&, 2&, VBVarBoolean)
```

4.2.6 AdsSyncWrite[Datatype]Req

AdsSyncWriteBoolReq

AdsSyncWriteIntegerReq

AdsSyncWriteLongReq

AdsSyncWriteSingleReq

AdsSyncWriteDoubleReq

AdsSyncWriteStringReq

Schreibt synchron Daten aus einer Visual Basic-Variable vom Typ Boolean, Integer, Long, Single, Double oder String in ein Datum eines ADS-Gerätes.

```
object.AdsSyncWrite[Datatype]Req(indexGroup As Long,
  indexOffset As Long,
  length As Long,
  pData As [Datatype]
) As Long
```

Parameter

indexGroup

[in] Index-Gruppe der ADS-Variable

indexOffset

[in] Index-Offset der ADS-Variable

length

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Die Ausführung des Visual Basic-Programms wird so lange angehalten, bis das ADS-Gerät die Daten empfangen hat oder bis die Zeit in der Eigenschaft AdsAmsCommTimeout [▶ 60] überschritten ist.
Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

Beispiel

```
Dim VBVar As Boolean
VBVar = True
Call AdsOcx1.AdsSyncWriteBoolReq(&H4021&, 0&, 2&, VBVar)
```

```
Dim VBVar As String
VBVar = "TwinCAT"
Call AdsOcx1.AdsSyncWriteStringReq(&H4020&, 0&, LenB(VBVar), VBVar)
```

4.3 asynchron

4.3.1 AdsRead[Datatype]Req

AdsReadIntegerReq

AdsReadLongReq

AdsReadSingleReq

AdsReadDoubleReq

AdsReadStringReq

Setzt ein Read-Request für ein Datum von Typ Integer, Long, Single, Double oder String ab.

```
object.AdsRead[Datatype]Req(
  nInvokeId As Long,
  nIndexGroup As Long,
  nIndexOffset As Long,
  cbLength As Long
) As Long
```

Parameter

nInvokeId

[in] Auftragsnummer zur Identifizierung der Antwort

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Nachdem ein Read-Request an das ADS-Gerät abgesetzt wurde, wird mit der Abarbeitung des Visual Basic-Programms fortgefahren. Sobald die Daten vorliegen, löst das ADS-OCX die Ereignisfunktion AdsRead[Datatype]Conf() [▶ 56] aus, mit der die geforderten Daten übermittelt werden.
Beim Absetzen vom Read-Request muss eine Identifizierungsnummer mit angegeben werden, die später beim Aufruf der Ereignisfunktion wieder zurückgegeben wird. Dadurch ist eine Zuordnung zwischen Read-Request und der Ereignisfunktion möglich.
Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

Beispiel

```
Dim nInvokeId As Long
nInvokeId = 1
'Lesen von MW0 aus der SPS
Call AdsOcx1.AdsReadIntegerReq(nInvokeId, &H4020&, 0&, 4&)

Private Sub AdsOcx1_AdsReadIntegerConf(ByVal nInvokeId As Long, ByVal nResult As Long, ByVal cbLengt
h As Long, pData As Integer)
    If (nInvokeId = 1) And (nResult = 0) Then
    'Daten anzeigen
    Label1.Caption = pData
    End If
End Sub
```

```
Dim nInvokeId As Long
nInvokeId = 1
'Lesen aus SPS
Call AdsOcx1.AdsReadStringReq(nInvokeId, &H4020&, 0&, 20&)

Private Sub AdsOcx1_AdsReadStringConf(ByVal nInvokeId As Long, ByVal nResult As Long, ByVal cbLength
 As Long, ByVal pData As String)
    If (nInvokeId = 1) And (nResult = 0) Then
    'Daten anzeigen
```

```
    Label1.Caption = pData
    End If
End Sub
```

4.3.2 AdsWrite[Datatype]Req

AdsWriteIntegerReq

AdsWriteLongReq

AdsWriteSingleReq

AdsWriteDoubleReq

AdsWriteStringReq

Setzt ein Read-Request für ein Datum von Typ Integer, Long, Single, Double oder String ab.

```
object.AdsWrite[Datatype]Req(
  nInvokeId As Long,
  nIndexGroup As Long,
  nIndexOffset As Long,
  cbLength As Long,
  pData As [Datatype]
) As Long
```

Parameter

nInvokeId

[in] Auftragsnummer zur Identifizierung der Antwort

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Nachdem das Write-Request an das ADS-Gerät abgesetzt wurde, wird mit der Abarbeitung des Visual Basic-Programms fortgefahren. Sobald die Daten geschrieben wurden, löst das ADS-OCX die Ereignisfunktion AdsWriteConf() [▶ 59] aus.

Beim Absetzen vom Write-Request muß eine Identifizierungsnummer mit angegeben werden, die später beim Aufruf der Ereignisfunktion wieder zurückgegeben wird. Dadurch ist eine Zuordnung zwischen Write-Request und der Ereignisfunktion möglich.

Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

Beispiel

```
Dim VBVar As Integer
Dim nInvokeId As Long
VBVar = 100
nInvokeId = 1
Call AdsOcx1.AdsWriteIntegerReq(nInvokeId, &H4020&, 0&, 2&, VBVar)

Private Sub AdsOcx1_AdsWriteConf(ByVal nInvokeId As Long, ByVal nResult As Long)
    If (nResult <> 0) Then MsgBox ("Error AdsWriteConf " & nResult)
End Sub
```

```
Dim VBVar As String
Dim InvokeId As Long
VBVar = "TwinCAT"
InvokeId = 1
Call AdsOcx1.AdsWriteStringReq(InvokeId, &H4020&, 0&, LenB(VBVar), VBVar)

Private Sub AdsOcx1_AdsWriteConf(ByVal nInvokeId As Long, ByVal nResult As Long)
    If (nResult <> 0) Then MsgBox ("Error AdsWriteConf " & nResult)
End Sub
```

4.4 connect

4.4.1 AdsReadVarConnectEx

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen und einem Datum aus einem ADS-Gerät her.

```
object.AdsReadVarConnectEx(nIndexOffset As String,
  nRefreshType As ADSOCXTRANSMODE,
  nCycleTime As Long,
  phConnect    As Long
  hUser As Variant
) As Long
```

Parameter

adsVarName

[in] Name der ADS-Variable

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Lese-Zyklus in ms

phConnect

[out] enthält einen eindeutigen Handle für die aufgebaute Verbindung (dieses ist nicht der Handle der ADS-Variablen!).

hUser

[in] Optional: Dieser Wert wird beim Aufruf des Ereignis AdsReadConnectUpdateEx() [▶ 54] übergeben.

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte diese über die Methode AdsDisconnectEx() [▶ 40] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der Symbol-Download aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion *'Funk1'* die Variable *'SPSVar1'* angesprochen werden, so muss als erster Parameter *'Funk1.SPSVar1'* angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. *'.SPSGlobVar'*. Der Parameter *adsVarName* unterscheidet nicht zwischen Groß- und Kleinbuchstaben.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter Allgemein festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

HINWEIS

Unter Borland Delphi Parameter nicht richtig übergeben

Beim Aufruf der dazugehörigen Ereignisfunktion AdsReadConnectUpdateEx() werden die OleVariant-Parameter an die Delphi-Applikation nicht richtig übergeben. Die Methode AdsReadVarConnectEx2() mit der dazugehörigen Ereignissfunktion AdsReadConnectUpdateEx2() bietet die gleiche Funktionalität wie die Methode AdsReadVarConnectEx/AdsReadConnectUpdateEx. Bitte benutzen Sie diese Methode/Ereignis in Delphi-Applikationen. In Visual Basic-Applikationen können beide Methoden benutzt werden.

Beispiel

Visual Basic Beispiel: 'Ereignisgesteuertes Lesen [▶ 76]'

Sehen Sie dazu auch

- AdsReadVarConnectEx2 [▶ 35]
- AdsReadConnectUpdateEx2 [▶ 55]
- AdsReadConnectUpdateEx [▶ 54]

4.4.2 AdsReadVarConnectEx2

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen und einem Datum aus einem ADS-Gerät her.

```
object.AdsReadVarConnectEx2(nIndexOffset As String,
  nRefreshType As ADSOCXTRANSMODE,
  nCycleTime As Long,
  phConnect    As Long
  hUser As Variant
) As Long
```

Parameter

adsVarName

[in] Name der ADS-Variable

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Lese-Zyklus in ms

phConnect

[out] Enthält einen eindeutigen Handle für die aufgebaute Verbindung (dieses ist nicht der Handle der ADS-Variablen!).

hUser

[in] Optional: Dieser Wert wird beim Aufruf des Ereignis AdsReadConnectUpdateEx2() [▶ 55] übergeben.

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte diese über die Methode AdsDisconnectEx() [▶ 40] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der Symbol-Download aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion *'Funk1'* die Variable *'SPSVar1'* angesprochen werden, so muss als erster Parameter *'Funk1.SPSVar1'* angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. *'.SPSGlobVar'*. Der Parameter *adsVarName* unterscheidet nicht zwischen Groß- und Kleinbuchstaben.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter Allgemein festlegen. Das Feld Symbole erzeugen muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

HINWEIS

Unter Borland Delphi Parameter nicht richtig übergeben

Beim Aufruf der Ereignisfunktion AdsReadConnectUpdateEx() werden die OleVariant-Parameter an die Delphi-Applikation nicht richtig übergeben. Bitte benutzen Sie die Methode AdsReadVarConnectEx2 und das dazugehörige Ereignis in Delphi-Applikationen. In Visual Basic-Applikationen können beide Methoden/ Ereignisse benutzt werden.

Beispiel

Visual Basic: 'Ereignisgesteuertes Lesen [▶ 76]'

4.4.3 AdsReadVarConvertConnect

Ab TwinCAT 2.8 Build > 743 und höher.

Die Methode stellt eine feste Verbindung zu einer Variablen eines ADS-Gerätes her. Durch den *'usrConvertType'* Parameter kann festgelegt werden, welchen Datentyp (Format) die ankommenden Variablen-Daten in der Ereignisfunktion haben sollen. Die *'usrConvertType'* Parameterübergabe erfolgt "ByValue", d.h der übergebene Datentyp wird nur als "Vorlage" für die Konvertierung benutzt. Bei der Konvertierung wird jeweils die entsprechende Menge an Datenbytes in den vom Benutzer festgelegten Datentyp hineinkopiert.

```
object.AdsReadVarConvertConnect(nIndexOffset As String,
  nRefreshType As ADSOCXTRANSMODE,
  nCycleTime As Long,
  phConnectAs Long,
  usrConvertType As Variant,
  hUser As Variant
) As Long
```

Parameter

adsVarName

[in] Name der ADS-Variable

nRefreshType

[in] Art des Datenaustausches zwischen der VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Lese-Zyklus in ms

phConnect

[out] Enthält einen eindeutigen Handle für die aufgebaute Verbindung (dieses ist nicht der Handle der ADS-Variablen!).

usrConverType

[in] Datentyp in den die Ereignisdaten konvertiert werden sollen. Die folgende Tabelle enthält eine Liste der unterstützten VB-Datentypen die als Parameter übergeben werden können.

Visual Basic data type	Equal to C++ VARTYPE	Equal to PLC data type (memory use)
Byte	VT_UI1	BYTE (1Byte)
Integer	VT_I2	INT (2 Byte) and enums
Long	VT_I4	DINT (4 Byte)
Single	VT_R4	REAL (4 Byte)
Double	VT_R8	LREAL (8 Byte)
String*	VT_BSTR	STRING (Deklarierte Stringlänge + Null-Terminierung)
Boolean**	VT_BOOL	BOOL (1Byte)
Date***	VT_DATE	DT; DATE_AND_TIME (4Byte)
not supported in VB	VT_UI2	WORD; UINT (2Byte)
not supported in VB	VT_UI4	DWORD; UDINT (4Byte)
not supported in VB	VT_I1	SINT (1Byte)
Variant****	VT_VARIANT	-
Dim varArray() As <anything of above types>	VT_ARRAY | <anything of above types>	-

* Die Stringlänge muss auf die maximale Anzahl der Zeichen (inklusive der abschließenden NULL) gesetzt werden, die die Stringvariable aufnehmen soll. Aus den Ereignisdaten werden dann (VB-Stringlänge + 1Byte(für Nullterminierung)) Bytes in die Stringvariable hineinkopiert. Danach wird die Länge des Strings auf die tatsächliche Länge verkürzt. D.h. der String wird beim ersten Null-Character abgeschnitten. Bei passend gesetzter Stringlänge können auch Stringarrays aus der SPS gelesen werden. Z. B.:

```
VAR_GLOBAL
     plcStringArr    : ARRAY[ 1..2 ] OF STRING(30);
END_VAR
```

in VB:

```
Dim  vbStringArr( 1 To 2) As String
vbStringArr(1) = String( 31, "#")
vbStringArr(2) = String( 31, "#")
call AdsOcx1.AdsReadVarConvertConnect(".plcStringArr", ADSTRANS_SERVERONCHA, 300, hConnect, vbString
Arr )
```

** Bei der Konvertierung wird jeweils ein Byte Ereignisdaten in einen 2-Byte OleVariant-Datentyp konvertiert. Es gilt: TRUE bei Data <> 0 und FALSE bei Data = 0;

*** Der OleVariant-Datentyp *Date* kann dafür benutzt werden, um z. B. die SPS-Variable vom Typ DATE_AND_TIME in eine VB-Applikation zu lesen. Bei der Konvertierung werden die lokalen Einstellungen des PC's berücksichtigt. Andere SPS-Datentypen wie z. B. TIME oder TOD werden nicht unterstützt, weil sie nicht passend konvertiert werden können.

**** Die Variant-Variable muss mit einem Datentyp initialisiert werden. VT_EMPTY oder VT_NULL sind z. B. nicht erlaubt.

hUser

[in] Optional: Dieser Wert wird beim Aufruf des Ereignis AdsReadConvertConnectUpdate() [▶ 55] übergeben.

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte diese über die Methode AdsDisconnectEx() [▶ 40] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der Symbol-Download aktiviert ist. Nähere Informationen können Sie dem Handbuch des PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion *'Funk1'* die Variable *'SPSVar1'* angesprochen werden, so muss als erster Parameter *'Funk1.SPSVar1'* angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. *'.SPSGlobVar'*. Der Parameter *adsVarName* unterscheidet nicht zwischen Groß- und Kleinbuchstaben.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter Allgemein festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

Beispiel

Visual Basic Beispiel: Ereignisgesteuertes Lesen (mit Konvertierung in einen anderen Typ) [▶ 86]

4.4.4 AdsRead[Datatype]VarConnect

AdsReadBoolVarConnect

AdsReadIntegerVarConnect

AdsReadLongVarConnect

AdsReadSingleVarConnect

AdsReadDoubleVarConnect

AdsReadStringVarConnect

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen vom Typ Boolean, Integer, Long, Single, Double oder String und einem Datum aus einem ADS-Gerät her.

```
object.AdsRead[Datatype]VarConnect(
  nIndexOffset As String,
  cbLength As Long,
  nRefreshType As Integer,
  nCycleTime As Integer,
  pData As [Datatype]
) As Long
```

Parameter

adsVarName

[in] Name der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Lese-Zyklus in ms

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkung

Bei Änderung der SPS-Variablen wird das Ereignis AdsReadConnectUpdate() [▶ 53] ausgelöst.

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte diese über die Methode AdsRead[Datentyp]Disconnect() [▶ 44] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

Es wird pro SPS-Variabel nur ein Handle erzeugt, d.h. beim Verbinden mehrerer Variablen auf eine SPS-Variabel wird bei Änderungen entsprechend mehrmals das Ereignis AdsReadConnectUpdate() [▶ 53] mit dem selben Handle aufgerufen.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der 'Symbol-Download' aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion 'Funk1' die Variable 'SPSVar1' angesprochen werden, so muss als erster Parameter 'Funk1.SPSVar1' angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. '.SPSGlobVar'. Der Parameter 'adsVarName' unterscheidet nicht zwischen Groß- und Kleinbuchstaben.

Wenn eine Variable aus der SPS mit einer Visual Basic-Variablen verbunden wird, müssen Sie als Länge 2 angeben, da Visual Basic Boolean-Variablen intern mit 2 Byte verwaltet.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter 'Allgemein' festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

Diese Methode wurde durch AdsReadVarConnectEx() [▶ 34] ersetzt. Benutzen Sie in Zunkunft AdsReadVarConnectEx() da AdsReadBoolVarConnect() nicht weiter geflegt wird und nur noch aus Kompatibilitätsgründen enthalten sein wird.

Beispiel

-

4.4.5 AdsDisconnectEx

Beendet eine feste Verbindung zwischen einer Visual Basic-Variablen und einem Datum aus einem ADS-Gerät.

```
object.AdsDisconnectEx(hConnectAs Long) As Long
```

Parameter

hConnect

[in] Handle der Verbindung zwischen der Visual Basic-Variablen und der ADS-Variablen

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte die Verbindung über die Methode AdsDisconnectEx() beendet werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden. Siehe auch Methode AdsReadVarConnectEx() [▶ 34].

Beispiel

Visual Basic Beispiel: 'Ereignisgesteuertes Lesen [▶ 76]'

4.4.6 AdsReadConnect

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen und einem Datum aus einem ADS-Gerät her.

```
object.AdsReadConnect(
  nIndexGroupAs Long,
  nIndexOffset As Long,
  cbLength As Long,
  nRefreshType As ADSOCXTRANSMODE,
  nCycleTime As Integer,
  pData As Variant
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Lese-Zyklus in ms

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte dieses über die Methode AdsReadDisconnect() [▶ 42] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden. Der Variablentyp String wird von der Methode AdsReadConnect() nicht unterstützt.

Beispiel

```
Dim VBVarInteger(0) As Integer
Dim VBVarSingle(0) As Single
Dim VBVarBoolean(0) As Boolean

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
  'Verbindung zu den Variablen in der SPS herstellen
  Call AdsOcx1.AdsReadConnect(&H4020&, 0&, 2&, ADSTRANS_SERVERONCHA, 55, VBVarInteger)
  Call AdsOcx1.AdsReadConnect(&H4020&, 2&, 4&, ADSTRANS_SERVERONCHA, 55, VBVarSingle)
  Call AdsOcx1.AdsReadConnect(&H4021&, 48&, 2&, ADSTRANS_SERVERONCHA, 55, VBVarBoolean)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
  'Verbindungen zu den Variablen in der SPS beenden
  Call AdsOcx1.AdsReadDisconnect(VBVarInteger)
  Call AdsOcx1.AdsReadDisconnect(VBVarSingle)
  Call AdsOcx1.AdsReadDisconnect(VBVarBoolean)
End Sub

'wird nach Änderung einer SPS-Variablen vom ADS-OCX aufgerufen
Private Sub AdsOcx1_AdsReadConnectUpdate(ByVal nIndexGroup As Long, ByVal nIndexOffset As Long)
  If (nIndexGroup = &H4020&) Then
    Select Case nIndexOffset
      Case 0: lblInteger.Caption = VBVarInteger(0)
      Case 2: lblSingle.Caption = VBVarSingle(0)
    End Select
  End If
  If (nIndexGroup = &H4021&) Then
    Select Case nIndexOffset
      Case 48: Shape1.BackColor = IIf(VBVarBoolean(0) = True, &HFF00&, &H8000&)
    End Select
  End If
End Sub
```

4.4.7 AdsReadDisconnect

Beendet eine feste Verbindung zwischen einer Visual Basic-Variablen und einem Datum aus einem ADS-Gerät.

```
object.AdsReadDisconnect(pData As Variant) As Long
```

Parameter

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte die Verbindung über die Methode AdsReadDisconnect() beendet werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden. Siehe auch Methode AdsReadConnect() [▶ 41].

Beispiel

-

4.4.8 AdsRead[Datatype]Connect

AdsReadBoolConnect

AdsReadIntegerConnect

AdsReadLongConnect

AdsReadSingleConnect

AdsReadDoubleConnect

AdsReadStringConnect

Stellt eine zyklische Verbindung zwischen einer Visual Basic-Variablen vom Typ Boolean, Integer, Long, Single, Double oder String und einem Datum aus einem ADS-Gerät her.

```
object.AdsRead[Datatype]Connect(
  nIndexGroupAs Long,
  nIndexOffset As Long,
  cbLength As Long,
  nRefreshType As Integer,
  nCycleTime As Integer,
  pData As [Datatype]
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Lesezyklus in ms

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu der ADS-Variable nicht mehr benötigt, sollte dieses über die Methode AdsRead[Datatype]Disconnect() [▶ 44] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.
Anmerkung zum Datentyp String: Bei der Länge der Daten ist zu beachten, dass sich diese auf die Länge der Variable im Visual Basic Programm bezieht. Da Visual Basic ein Zeichen mit 2 Byte darstellt, muss die Länge der Variable mit LenB() ermittelt werden, nicht mit Len().

Beispiel

```
Dim VBVar As Integer

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
    'Verbindung zwischen Merkerwort 0 der SPS und VBVar herstellen
```

```
    Call AdsOcx1.AdsReadIntegerConnect(&H4020&, 0&, 2&, 1, 110, VBVar)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
    'Verbindung zwischen den Variablen trennen
    Call AdsOcx1.AdsReadIntegerDisconnect(VBVar)
End Sub

'wird nach jedem Lesen vom ADS-OCX aufgerufen
Private Sub AdsOcx1_AdsReadConnectUpdate(ByVal nIndexGroup As Long, ByVal nIndexOffset As Long)
    'Variablen am Bildschirm anzeigen
    Label1.Caption = VBVar
End Sub
```

```
Dim VBVar As String

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
    'Visual Basic Variable initialisieren
    VBVar = Space(10)
    'Verbindung zur Variable in der SPS herstellen
    Call AdsOcx1.AdsReadStringConnect(&H4020&, 0&, LenB(VBVar), 4, 110, VBVar)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
    'Verbindung zur Variable in SPS beenden
    Call AdsOcx1.AdsReadStringDisconnect(VBVar)
End Sub

'wird bei Veränderung der SPS-Variablen vom ADS-OCX aufgerufen
Private Sub AdsOcx1_AdsReadConnectUpdate(ByVal nIndexGroup As Long, ByVal nIndexOffset As Long)
    If (nIndexGroup = &H4020) And (nIndexOffset = 0) Then
    'Variablen in Form anzeigen
    Label1.Caption = VBVar
    End If
End Sub
```

4.4.9 AdsRead[Datatype]Disconnect

AdsReadBoolDisconnect

AdsReadIntegerDisconnect

AdsReadLongDisconnect

AdsReadSingleDisconnect

AdsReadDoubleDisconnect

AdsReadStringDisconnect

Beendet eine feste Verbindung zwischen einer Visual Basic-Variablen vom Typ Boolean, Integer, Long, Single, Double oder String und einem Datum aus einem ADS-Gerät.

```
object.AdsRead[Datatype]Disconnect(
  pData As [Datatype]
) As Long
```

Parameter

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird der Wert einer ADS-Variable nicht mehr benötigt, sollte die Verbindung mit der Methode AdsReadBoolDisconnect() beendet werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden. Siehe auch Methode AdsRead[Datatype]VarConnect() [▶ 39] und AdsRead[Datatype]Connect() [▶ 42].

Beispiel

-

4.4.10 AdsWriteDisconnect

Beendet eine feste Verbindung zwischen einer Visual Basic-Variablen und dem Datum eines ADS-Gerätes.

```
object.AdsWriteDisconnect(pData As Variant) As Long
```

Parameter

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung der ADS-Variable nicht mehr benötigt, sollte diese über die Methode AdsWriteDisconnect() wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

Beispiel

-

4.4.11 AdsWrite[Datatype]Disconnect

AdsWriteBoolDisconnect

AdsWriteIntegerDisconnect

AdsWriteLongDisconnect

AdsWriteSingleDisconnect

AdsWriteDoubleDisconnect

Beendet eine feste Verbindung zwischen einer Visual Basic-Variablen vom Typ Boolean, Integer, Long, Single oder Double und einem Datum aus einem ADS-Gerät.

```
object.AdsWrite[Datatype]Disconnect(pData As [Datatype]) As Long
```

Parameter

pData

[in] Visual Basic-Variable, in der die Daten der ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung der ADS-Variable nicht mehr benötigt, sollte diese über die Methode AdsWrite[*Datatype*]Disconnect() wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

Beispiel

-

4.4.12 AdsWriteVarConnect

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen und einer Variablen von einem ADS-Gerät her.

```
object.AdsWriteVarConnect(
  adsVarName As String,
  cbLength As Long,
  nRefreshType As ADSOCXTRANSMODE,
  nCycleTime As Integer,
  pData As Variant
) As Long
```

Parameter

adsVarName

[in] Name der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66]). Die Methode *AdsWriteVarConnect* unterstützt (sinnvollerweise) nur den **ADSTRANS_CLIENTCYCLE** Modus. Der Wert der Visual Basic-Variablen wird dabei zyklisch zum ADS-Gerät geschrieben.

nCycleTime

[in] Schreib-Zyklus in ms

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte diese über die Methode AdsWriteDisconnect() [▶ 45] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.
Der Variablentyp String wird nicht unterstützt.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der Symbol-Download aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion *'Funk1'* die Variable *'SPSVar1'* angesprochen werden, so muss als erster Parameter *'Funk1.SPSVar1'* angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. *'.SPSGlobVar'*. Der Parameter *adsVarName* unterscheidet nicht zwischen Groß- und Kleinbuchstaben.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter Allgemein festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

Beispiel

```
Dim VBVarInteger(0) As Integer
Dim VBVarSingle(0) As Single
Dim VBVarBoolean(0) As Boolean

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
  'Verbindung zu den Variablen in der SPS herstellen
  Call AdsOcx1.AdsWriteVarConnect("MAIN.PLCVarInteger", 2&, 1, 110, VBVarInteger)
  Call AdsOcx1.AdsWriteVarConnect("MAIN.PLCVarSingle", 4&, 1, 110, VBVarSingle)
  Call AdsOcx1.AdsWriteVarConnect("MAIN.PLCVarBoolean", 2&, 1, 110, VBVarBoolean)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
  'Verbindung zu den Variablen in der SPS beenden
  Call AdsOcx1.AdsWriteDisconnect(VBVarInteger)
  Call AdsOcx1.AdsWriteDisconnect(VBVarSingle)
  Call AdsOcx1.AdsWriteDisconnect(VBVarBoolean)
End Sub

'wird vom Bediener aufgerufen
Private Sub cmd_write_Click()
  VBVarInteger(0) = CInt(txt_int.Text)
  VBVarSingle(0) = CSng(txt_single.Text)
  VBVarBoolean(0) = IIf(chk_boolean.Value = 1, True, False)
End Sub
```

4.4.13 AdsWrite[Datatype]VarConnect

AdsWriteBoolVarConnect

AdsWriteIntegerVarConnect

AdsWriteLongVarConnect

AdsWriteSingleVarConnect

AdsWriteDoubleVarConnect

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen vom Typ Boolean, Integer, Long, Single oder Double und einem ADS-Gerät her.

```
object.AdsWrite[Datatype]VarConnect(
  adsVarName As String,
  cbLength As Long,
  nRefreshType As Integer,
```

```
  nCycleTime As Integer,
  pData As [Datatype]
) As Long
```

Parameter

adsVarName

[in] Name der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Schreib-Zyklus in ms

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung zu einer ADS-Variablen nicht mehr benötigt, sollte diese über die Methode AdsWrite[Datatype]Disconnect() [▶ 45] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

HINWEIS

Bei der SPS den Symbol-Download aktivieren

Achten Sie beim PLC-Control darauf, dass unter Projekt / Optionen / TwinCAT der Symbol-Download aktiviert ist. Nähere Informationen können Sie aus dem Handbuch vom PLC-Control entnehmen.

Der erste Parameter der Methode setzt sich aus dem POE-Namen und der SPS-Variablen zusammen, die adressiert werden soll. Soll z. B. aus der Funktion *'Funk1'* die Variable *'SPSVar1'* angesprochen werden, so muss als erster Parameter *'Funk1.SPSVar1'* angegeben werden. Beim Zugriff auf globale Variablen wird der POE-Name weggelassen, also z. B. *'.SPSGlobVar'*. Der Parameter *adsVarName* unterscheidet nicht zwischen Groß- und Kleinbuchstaben.

HINWEIS

Bei der NC bei jeder Achse den Symbol-Download aktivieren

Im System-Manager muss bei jeder Achse der Symbol-Download aktiviert werden. Dieses können Sie bei dem Konfigurations-Dialog der Achse unter Allgemein festlegen. Das Feld 'Symbole erzeugen' muss gekennzeichnet sein. Siehe Handbuch System Manager.

Die Symbolnamen der einzelnen Parameter der NC sind fest vorgegeben und können aus der NC-Dokumentation entnommen werden.

Beispiel

```
Dim VBVar As Integer

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
    'Verbindung zur Variable in der SPS herstellen
```

```
    Call AdsOcx1.AdsWriteIntegerVarConnect("MAIN.PLCVar", 2&, 1, 110, VBVar)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
    'Verbindung zur Variable in SPS beenden
    Call AdsOcx1.AdsWriteIntegerDisconnect(VBVar)
End Sub

'wird vom Bediener aufgerufen
Private Sub cmd_write_Click()
    VBVar = CInt(Text1.Text)
End Sub
```

4.4.14 AdsWriteConnect

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen und einem ADS-Gerät her.

```
object.AdsWriteConnect(
  nIndexGroup As Long,
  nIndexOffset As Long,
  cbLength As Long,
  nRefreshType As ADSOCXTRANSMODE,
  nCycleTime As Integer,
  pData As Variant
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Schreib-Zyklus in ms

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung der ADS-Variable nicht mehr benötigt, sollte diese über die Methode AdsWriteDisconnect() [▶ 45] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.
Der Variablentyp String wird nicht unterstützt.

Beispiel

```
Dim VBVarInteger(0) As Integer
Dim VBVarSingle(0) As Single
Dim VBVarBoolean(0) As Boolean

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
  'Verbindung zu den Variablen in der SPS herstellen
   Call AdsOcx1.AdsWriteConnect(&H4020&, 0&, 2&, ADSTRANS_CLIENTCYCLE, 55, VBVarInteger)
   Call AdsOcx1.AdsWriteConnect(&H4020&, 2&, 4&, ADSTRANS_CLIENTCYCLE, 55, VBVarSingle)
   Call AdsOcx1.AdsWriteConnect(&H4021&, 48&, 2&, ADSTRANS_CLIENTCYCLE, 55, VBVarBoolean)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
  'Verbindung zu den Variablen in der SPS beenden
  Call AdsOcx1.AdsWriteDisconnect(VBVarInteger)
  Call AdsOcx1.AdsWriteDisconnect(VBVarSingle)
  Call AdsOcx1.AdsWriteDisconnect(VBVarBoolean)
End Sub

'wird vom Bediener aufgerufen
Private Sub cmd_write_Click()
  VBVarInteger(0) = CInt(txt_int.Text)
  VBVarSingle(0) = CSng(txt_single.Text)
  VBVarBoolean(0) = IIf(chk_boolean.Value = 1, True, False)
End Sub
```

4.4.15 AdsWrite[Datatype]Connect

AdsWriteBoolConnect

AdsWriteIntegerConnect

AdsWriteLongConnect

AdsWriteSingleConnect

AdsWriteDoubleConnect

Stellt eine feste Verbindung zwischen einer Visual Basic-Variablen vom Typ Boolean, Integer, Long, Single oder Double und einem Datum aus einem ADS-Gerät her.

```
object.AdsWrite[Datatype]Connect(
  nIndexGroupAs Long,
  nIndexOffset As Long,
  cbLength As Long,
  nRefreshType As Integer,
  nCycleTime As Integer,
  pData As [Datatype]
) As Long
```

Parameter

nIndexGroup

[in] Index-Gruppe der ADS-Variable

nIndexOffset

[in] Index-Offset der ADS-Variable

cbLength

[in] Länge der Daten in Byte (siehe VB-Variablenlängen [▶ 71])

nRefreshType

[in] Art des Datenaustausches zwischen VB-Variable und ADS-Variable (siehe Datentyp ADSOCXTRANSMODE [▶ 66])

nCycleTime

[in] Schreib-Zyklus in ms

pData

[in] Visual Basic-Variable, aus der die Daten in die ADS-Variable geschrieben werden

Rückgabewert

Siehe ADS-Fehlercodes

Bemerkungen

Wird die Verbindung der ADS-Variable nicht mehr benötigt, sollte diese über die Methode AdsWrite[Datatype]Disconnect() [▶ 45] wieder freigegeben werden. Wenn in einer Form nur bestimmte Werte benötigt werden, sollte die Verbindung erst beim Laden der Form erzeugt werden und beim Schließen der Form wieder freigegeben werden.

Beispiel

```
Dim VBVar As Integer

'wird beim Starten des Programms aufgerufen ---
Private Sub Form_Load()
    'Verbindung zur Variable herstellen
    Call AdsOcx1.AdsWriteIntegerConnect(&H4020&, 0&, 2&, 1, 110, VBVar)
End Sub

'wird beim Beenden des Programms aufgerufen
Private Sub Form_Unload(Cancel As Integer)
    'Verbindung zu den Variablen in SPS beenden
    Call AdsOcx1.AdsWriteIntegerDisconnect(VBVar)
End Sub

'wird durch den Bediener aufgerufen
Private Sub Cmd_write_Click()
    VBVar = CInt(Text1.Text)
End Sub
```

4.5 Ereignisse

4.5.1 AdsAmsConnectTimeout

Dieses Ereignis wird aufgerufen, sobald bei einer "per Connect" verbundenen Variable eine Zeitüberschreitung auftritt.

```
object_AdsAmsConnectTimeout(
  nIndexGroup As Long,
  nIndexOffset As Long
)
```

Parameter

nIndexGroup

[out] Index-Gruppe der ADS-Variable, bei der die Zeitüberschreitung aufgetreten ist.

nIndexOffset

[out] Index-Offset der ADS-Variable, bei der die Zeitüberschreitung aufgetreten ist.

Bemerkungen

Das Ereignis AdsAmsConnectTimeout wird nur aufgerufen, wenn die Art des Datenaustausches zwischen VB-Variable und ADS-Variable vom Client (ADSTRANS_CLIENTCYCLE [▶ 66]) gesteuert wird.

Beispiel

-

4.5.2 AdsAmsTimeout

Dieses Ereignis wird aufgerufen, sobald bei einer asynchronen Lese- oder Schreibanfrage (read/write request) eine Zeitüberschreitung auftritt.

```
object_AdsAmsTimeout(nInvokeId As Long)
```

Parameter

nInvokeId

[out] Identifikationsnummer der Anfrage (request), bei der die Zeitüberschreitung aufgetreten ist.

Bemerkungen

-

Beispiel

-

4.5.3 AdsConnectError

Falls im Server ein Fehler bei einer "per Connect" verbundenen Variable auftritt, wird dieses Ereignis aufgerufen.

```
object_AdsAmsConnectTimeout(
  nIndexGroup As Long,
  nIndexOffset As Long,
  errorCode As Long
)
```

Parameter

nIndexGroup

[out] Index-Gruppe der ADS-Variable, bei der der Fehler aufgetreten ist.

nIndexOffset

[out] Index-Offset der ADS-Variable, bei der der Fehler aufgetreten ist.

errorCode

[out] Fehlerstatus; siehe ADS-Fehlercodes

Bemerkungen

-

Beispiel

-

4.5.4 AdsLogNotification

Dieses Ereignis wird aufgerufen, sobald ein ADS-Gerät eine Nachricht abgesetzt hat und die zuvor definierten Filterbedingungen erfüllt sind.

```
object_AdsLogNotification(
  dateTime As Date,
  nMs As Long,
  dwMsgCtrl As Long,
  nServerPort As Long,
  szDeviceName As String,
  szLogMsgAs String
)
```

Parameter

dateTime

[out] Datum und Uhrzeit, als die Meldung vom ADS-Gerät abgesetzt wurde

nMs

[out] Millisekunden, als die Meldung vom ADS-Gerät abgesetzt wurde

dwMsgCtrl

[out] Filtermaske für die Art der Nachrichten die gemeldet werden sollen (siehe DatentypADSLOGMSGTYPE [▶ 66])

nServerPort

[out] Portnummer des ADS-Gerätes, welches die Meldung abgesetzt hat

szDeviceName

[out] Name des ADS-Gerätes, welches die Meldung abgesetzt hat

szLogMsg

[out] Nachricht, die vom ADS-Gerät abgesetzt wurde

Bemerkungen

Sobald ein ADS-Gerät eine Nachricht abgesetzt hat und die Filterbedingungen, die durch die Methode AdsEnableLogNotificaion() [▶ 15] definiert wurden erfüllt sind, wird das Ereignis AdsLogNotification() ausgelöst. Anhand der übergebenen Parameter, kann die Meldung weiter ausgewertet werden.

Beispiel

Visual Basic Beispiel: 'Meldungen über den TwinCAT-Router senden/empfangen [▶ 82]'

4.5.5 AdsReadConnectUpdate

Dieses Ereignis wird aufgerufen, wenn die Methode AdsReadYY(Var)Connect() aufgerufen wurde und der Wert aus dem ADS-Gerät gelesen wurde bzw. sich geändert hat.

```
object_AdsReadConnectUpdate(
  nIndexGroup As Long,
  nIndexOffset As Long
)
```

Parameter

nIndexGroup

[out] Datum und Uhrzeit, als die Meldung vom ADS-Gerät abgesetzt wurde

nIndexOffset

[out] Millisekunden, als die Meldung vom ADS-Gerät abgesetzt wurde

Bemerkungen

Bei dem Ereignis AdsReadConnectUpdate() ist es nicht notwendig, dass der Wert mit übertragen wird, da das ADS-OCX im Hintergrund die Visual Basic Variable aktualisiert. Um Schreibzugriffe auf Anzeigeobjekte in der Form zu optimieren, sollte in der Ereignisfunktion abgefragt werden, welche Variable sich geändert hat und nur das Element auf der Form aktualisiert werden, welches den Wert anzeigt. Wurde eine VB-Variable per VarConnect mit einer ADS-Variablen verbunden, so wird bei dem Ereignis AdsReadConnectUpdate() im Parameter *nIndexOffset* der Handle der Variablen übergeben, in den Parameter *nIndexGroup* wird in diesen Fall der konstante Wert &HF005 übertragen. Damit Sie den *nIndexOffset* auswerten können, müssen Sie zuvor mit der Methode AdsCreateVarHandle() [▶ 14] den Handle der ADS-Variablen holen. Dieses kann z. B. im Load-Ereignis der Form geschehen. Im Ereignis AdsReadConnectUpdate() wird dann abgefragt, welcher Variablenhandle in dem Parameter *nIndexOffset* übergeben wurde.
Wurde die Verbindung nicht mit dem Variablennamen angelegt, sondern mit der Variablenadresse, so wird in den Parametern *nIndexGroup* und *nIndexOffset* die Adresse der Variablen übertragen, die sich geändert hat.
Wenn die Verbindung zwischen VB-Variable und ADS-Variable beendet wird, sollte auch der Handle mit der Methode AdsDeleteVarHandle() [▶ 15] wieder freigegeben werden.

Beispiel

-

4.5.6 AdsReadConnectUpdateEx

Dieses Ereignis wird aufgerufen, wenn die Methode AdsReadVarConnectEx() [▶ 34] aufgerufen wurde und der Wert aus dem ADS-Gerät gelesen wurde bzw. sich geändert hat.

```
object_AdsReadConnectUpdateEx(
  ByVal dateTime As Date,
  ByVal nMs As Long,
  ByVal hConnect As Long,
  ByVal data As Variant,
  Optional ByVal hUser  As Variant
)
```

Parameter

dateTime

[out] Zeitstempel

nHs

[out] Millisekunden vom Zeitstempel

hConnect

[out] Handle der Verbindung; wird durch die Methode AdsReadVarConnectEx() [▶ 34] erzeugt

data

[out] Wert aus dem ADS-Gerät

hUser

[out] frei benutzbarer Wert; wird bei dem Aufruf der Methode AdsReadVarConnectEx() [▶ 34] übergeben

Bemerkungen

-

Beispiel

Visual Basic Beispiel: 'Ereignisgesteuertes Lesen [▶ 76]'

4.5.7 AdsReadConnectUpdateEx2

Dieses Ereignis wird aufgerufen, wenn die Methode AdsReadVarConnectEx2() [▶ 35] aufgerufen wurde und der Wert aus dem ADS-Gerät gelesen wurde bzw. sich geändert hat.

```
object_AdsReadConnectUpdateEx2(
  ByVal dateTime As Date,
  ByVal nMs As Long,
  ByVal hConnect As Long,
  ByRef data As Variant,
  Optional ByRef hUser As Variant
)
```

Parameter

dateTime

[out] Zeitstempel

nHs

[out] Millisekunden vom Zeitstempel

hConnect

[out] Handle der Verbindung; wird durch die Methode AdsReadVarConnectEx2() [▶ 35] erzeugt

data

[out] Wert aus dem ADS-Gerät

hUser

[out] frei benutzbarer Wert; wird bei dem Aufruf der Methode AdsReadVarConnectEx2() [▶ 35] übergeben

Bemerkungen

Die Parameter *data* und *hUser* müssen **ByRef** übergeben werden (notwendig für den Einsatz unter Borland Delphi).

Beispiel

Visual Basic: 'Ereignisgesteuertes Lesen [▶ 76]'

4.5.8 AdsReadConvertConnectUpdate

Ab TwinCAT 2.8 Build > 743 und höher.

Dieses Ereignis wird aufgerufen, wenn die Methode AdsReadVarConvertConnect() [▶ 37] aufgerufen wurde und der Wert aus dem ADS-Gerät gelesen wurde bzw. sich geändert hat.

```
object_AdsReadConvertConnectUpdate(
    ByVal dateTime As Date,
    ByVal nMs As Long,
    ByVal hConnect As Long,
    ByRef data As Variant,
    Optional ByRef hUser As Variant
)
```

Parameter

dateTime

[out] Zeitstempel.

nHs

[out] Millisekunden vom Zeitstempel.

hConnect

[out] Handle der Verbindung; wird durch die Methode AdsReadVarConvertConnect() [▶ 37] erzeugt.

data

[out] Wert aus dem ADS-Gerät. Der Datentyp der Variant-Variablen wurde als Parameter beim Aufruf von AdsReadVarConvertConnect() [▶ 37] festgelegt.

hUser

[out] frei benutzbarer Wert; wird bei dem Aufruf der Methode AdsReadVarConvertConnect() [▶ 37] übergeben.

Beispiel

Visual Basic: Ereignisgesteuertes Lesen (mit Konvertierung in einen anderen Typ) [▶ 86]

4.5.9 AdsRead[Datatype]Conf

AdsReadIntegerConf

AdsReadLongConf

AdsReadSingleConf

AdsReadDoubleConf

AdsReadStringConf

Liefert das Ergebnis nach einen Aufruf der Methode AdsRead[*Datatype*]Req().

```
object_AdsRead[Datatype]Conf(
  nInvokeId As Long,
  nResult As Long,
  cbLength As Long,
  pData As [Datatype]
)
```

Parameter

nInvokeId

[out] Auftragsnummer zur Identifizierung der Antwort

nResult

[out] Fehlerstatus; siehe ADS-Fehlercodes

cbLength

[out] Länge der Daten in Byte

pData

[out] ausgelesende Daten aus dem ADS-Gerät

Bemerkungen

Nachdem ein Read-Request an das ADS-Gerät abgesetzt wurde, wird mit der Abarbeitung des Visual Basic-Programms fortgefahren. Sobald die Daten vorliegen, löst das ADS-OCX die Ereignisfunktion AdsRead[*Datatype*]Conf() aus, mit der die geforderten Daten übermittelt werden.
Beim Absetzen vom Read-Request muss eine Identifizierungsnummer mit angegeben werden, die später beim Aufruf der Ereignisfunktion wieder zurückgegeben wird. Dadurch ist eine Zuordnung zwischen Read-Request und der Ereignisfunktion möglich. Siehe auch AdsRead[Datatype]Req() [▶ 31].

Beispiel

-

4.5.10 AdsRouterRemove

Wird in der Systemsteuerung von Windows NT/2000 der TwinCAT-Router komplett aus dem Betriebssystem entfernt, so wird dieses Ereignis ausgelöst.

```
object_AdsRouterRemove()
```

Parameter

-

Bemerkungen

-

Beispiel

Visual Basic: 'Statusänderung vom TwinCAT-Router und der SPS erkennen/verändern [▶ 80]'

4.5.11 AdsRouterShutdown

Wird der TwinCAT-Router gestoppt, so wird dieses Ereignis ausgelöst.

```
object_AdsRouterShutdown()
```

Parameter

-

Bemerkungen

-

Beispiel

Visual Basic: 'Statusänderung vom TwinCAT-Router und der SPS erkennen/verändern [▶ 80]'

4.5.12 AdsRouterStart

Wird der TwinCAT-Router gestartet, so wird dieses Ereignis ausgelöst.

```
object_AdsRouterStart()
```

Parameter

-

Bemerkungen

-

Beispiel

Visual Basic: 'Statusänderung vom TwinCAT-Router und der SPS erkennen/verändern [▶ 80]'

4.5.13 AdsServerStateChanged

Diese Ereignisfunktion wird aufgerufen, wenn sich der Zustand des ADS-Gerätes geändert hat.

```
object_AdsServerStateChanged(
  nAdsState As ADSSTATE,
  nDeviceState As Long
)
```

Parameter

nAdsState

[out] neuer Zustand des ADS-Gerätes (siehe Datentyp ADSSTATE [▶ 66])

nDeviceState

[out] (wird derzeit nicht unterstützt)

Bemerkungen

-

Beispiel

Visual Basic: 'Statusänderung vom TwinCAT-Router und der SPS erkennen/verändern [▶ 80]'

Sehen Sie dazu auch

ADSSTATE [▶ 66]

4.5.14 AdsServerSymChanged

Diese Ereignisfunktion wird ausgelöst, wenn Sie die Symboltabelle im ADS-Gerät geändert hat.

```
object_AdsServerSymChanged()
```

Parameter

-

Bemerkungen

Jedes ADS-Gerät legt seine Symbolnamen in eine interne Tabelle ab. Dabei wird jedem Symbol ein Handle zugeordnet, der mit der Methode AdsCreateVarHandle() [▶ 14] ausgelesen werden kann. Ändert sich die Symboltabelle, z. B. weil sich die Anzahl der Variablen geändert hat, wird dieses Ereignis ausgelöst.

Beispiel

Visual Basic: 'Statusänderung vom TwinCAT-Router und der SPS erkennen/verändern [▶ 80]'

4.5.15 AdsWriteConf

Bestätigt ein Schreibt-Request.

```
object_AdsWriteConf(
  nInvokeId As Long,
  nResult As Long
)
```

Parameter

nInvokeId

[out] Auftragsnummer zur Identifizierung der Antwort

nResult

[out] Fehlerstatus; siehe ADS-Fehlercodes

Bemerkungen

Nachdem ein WriteRequest an das ADS-Gerät abgesetzt wurde, wird mit der Abarbeitung des Visual Basic-Programms fortgefahren. Sobald die Daten in das Gerät geschrieben wurden, löst das ADS-OCX die Ereignisfunktion AdsWriteConf() aus.
Beim Absetzen vom Write-Request muss eine Identifizierungsnummer mit angegeben werden, die später beim Aufruf der Ereignisfunktion wieder zurückgegeben wird. Dadurch ist eine Zuordnung zwischen Write-Request und der Ereignisfunktion möglich. Siehe auch AdsWrite[Datatype]Req() [▶ 33].

Beispiel

-

4.6 Eigenschaften

4.6.1 AdsAmsClientNetId

In dieser Eigenschaft ist die NetId des Rechners abgelegt, auf dem das Visual Basic-Programm mit dem ADS-OCX ausgeführt wird.

```
object.AdsAmsClientNetId As String
```

Bemerkung

Diese Eigenschaft ist nur lesbar und kann weder innerhalb der Visual Basic-Entwicklungsumgebung noch während der Laufzeit des Programms geändert werden.

Über das TwinCAT System-Control kann die NetId eingestellt werden.

4.6.2 AdsAmsClientPort

Die Client-Portnummer ist die Portnummer, unter der andere ADS-Geräte das Visual Basic Programm ansprechen können.

```
object.AdsAmsClientPort As Long
```

Bemerkung

Wenn Sie selbst keine Portnummer vorgeben, vergibt das ADS-OCX automatisch eine Portnummer, die immer größer als 32767 ist. Beachten Sie, dass sich diese Portnummer jedesmal beim Starten ändert.

Wenn Ihr Visual Basic-Programm eine feste Portnummer erhalten soll, müssen Sie im Programm die Eigenschaft AdsAmsClientPort mit der gewünschten Portnummer setzen. Dieser Wert muss dann zwischen 16000 und 32000 liegen.

Sie können auch während der Entwicklungsphase über das Eigenschaftfenster des ADS-OCX die Eigenschaft AdsAmsClientPort setzen. Anschließend muss die SavePort auf TRUE gesetzt werden. Dadurch wird dem ADS-OCX angeordnet, die Portnummer nicht zu ändern.

Siehe auch Eigenschaft AdsAmsSaveClientPort [▶ 60].

4.6.3 AdsAmsCommTimeout

In AdsAmsCommTimeOut wird eine Zeit in Millisekunden angegeben, in der eine Antwort des Kommunikationspartners erwartet wird.

```
object.AdsAmsCommTimeout As Long
```

Bemerkung

Zulässige Werte: 1 bis 2147483647 Millisekunden. Negative Werte und der Wert Null werden bei einer Zuweisung nicht übernommen. Default: 5000 Millisekunden.

4.6.4 AdsAmsConnected

AdsAmsConnected kann dazu verwendet werden, den Status der Verbindung zwischen ADS-OCX und TwinCAT ADS-Routers festzustellen

```
object.AdsAmsConnected As Boolean
```

Bemerkungen

Wenn die Verbindung zu dem TwinCAT ADS-Router besteht, ist der Wert der Eigenschaft "TRUE", andernfalls ist der Wert "FALSE".

Diese Eigenschaft kann nur gelesen werden.

4.6.5 AdsAmsSaveClientPort

Verhindert, dass das ADS-OCX dynamisch den Port des Clients vergibt.

```
object.AdsAmsSaveClientPort As Boolean
```

Bemerkung

Wenn Sie selbst keine Portnummer vorgeben, vergibt das ADS-OCX automatisch eine Portnummer, die immer größer als 32767 ist. Beachten Sie, dass sich diese Portnummer jedesmal beim Starten ändert.

Wenn Ihr Visual Basic-Programm eine feste Portnummer erhalten soll, müssen Sie im Programm die Eigenschaft AdsAmsClientPort mit der gewünschten Portnummer setzen. Dieser Wert muss dann zwischen 16000 und 32000 liegen.

Sie können auch während der Entwicklungsphase über das Eigenschaftfenster des ADS-OCX die Eigenschaft AdsAmsClientPort setzen. Anschließend muss die SavePort auf TRUE gesetzt werden. Dadurch wird dem ADS-OCX angeordnet, die Portnummer nicht zu ändern.

Siehe auch Eigenschaft AdsAmsClientPort [▶ 59].

4.6.6 AdsAmsServerNetId

In dieser Eigenschaft ist die NetId des Rechners abgelegt, auf dem das Visual Basic-Programm mit dem ADS-OCX zugreifen soll.

```
object.AdsAmsServerNetId As String
```

Bemerkung

ADS-Geräte können sich auf verschiedenen Rechnern innerhalb eines Netzwerkes befinden. Jeder Rechner muss eine eindeutige NetId innerhalb dieses Netzwerks besitzen.

Geben Sie in dieser Eigenschaft die NetId des Rechners ein, auf dem sich das ADS-Gerät befindet, mit welchem Sie kommunizieren wollen. Falls sich in dieser Eigenschaft ein Leerstring befindet, werden die ADS-Geräte des lokalen Rechners angesprochen. Wenn sich z. B. Ihr Visual Basic-Programm immer auf den gleichen Rechner befindet wie die SPS, so lassen Sie diese Eigenschaft leer. Dadurch können Sie das Visual Basic-Programm leichter auf andere Rechner einsetzen, auch wenn diese Rechner unterschiedliche NetIds haben.

Über das TwinCAT System-Control kann festgestellt werden, welche NetId eingestellt ist.

4.6.7 AdsAmsServerPort

Beinhaltet die Portnummer des ADS-Gerätes, welches mit dem ADS-OCX angesprochen werden soll.

```
object.AdsAmsServerPort As Long
```

Bemerkung

Die Portnummern der einzelnen ADS-Gerätes sind aus der entsprechenden Dokumentation zu entnehmen. Hier eine Tabelle mit den wichtigsten ADS-Geräten:

ADS-Gerät	Portnummer
NC / NCI	501
SPS Laufzeitsystem 1	801
SPS Laufzeitsystem 2	811
SPS Laufzeitsystem 3	821
SPS Laufzeitsystem 4	831
Nockenschaltwerk	901

4.6.8 AdsClientAdsState

Mit Hilfe dieser Eigenschaft kann anderen Kommunikationspartner mitgeteilt werden, in welchem Zustand sich gerade das ADS-Gerät befindet.

```
object.AdsClientAdsState As String
```

Bemerkung

Welche Zustände ein ADS-Gerät unterstützen, ist aus der Dokumentation des ADS-Gerätes zu entnehmen.

4.6.9 AdsClientBuild

Enthält den Buildlevel des ADS-Gerätes.

```
object.AdsClientBuild As Integer
```

Bemerkung

Jedes ADS-Gerät besitzt Eigenschaften, aus denen die Versionsnummer und die Typbezeichnung des ADS-Gerätes ausgelesen werden kann. Die Versionsnummer besteht aus:

- Version (siehe Eigenschaft AdsClientVersion [▶ 62])
- Revision (siehe Eigenschaft AdsClientRevision [▶ 62])
- Build (siehe Eigenschaft AdsClientBuild [▶ 61])

Sie sollten beim Erstellen von ADS-Geräten immer dafür sorgen, dass diese Eigenschaften gesetzt werden, damit das ADS-Gerät von anderen Teilnehmern identifiziert werden kann.

4.6.10 AdsClientRevision

Enthält den Revisionlevel des ADS-Gerätes.

```
object.AdsClientRevision As Integer
```

Bemerkung

Jedes ADS-Gerät besitzt Eigenschaften, aus denen die Versionsnummer und die Typbezeichnung des ADS-Gerätes ausgelesen werden kann. Die Versionsnummer besteht aus:

- Version (siehe Eigenschaft AdsClientVersion [▶ 62])
- Revision (siehe Eigenschaft AdsClientRevision [▶ 62])
- Build (siehe Eigenschaft AdsClientBuild [▶ 61])

Sie sollten beim Erstellen von ADS-Geräten immer dafür sorgen, dass diese Eigenschaften gesetzt werden, damit das ADS-Gerät von anderen Teilnehmern identifiziert werden kann.

4.6.11 AdsClientType

Enthält den Typbezeichnung des ADS-Gerätes.

```
object.AdsClientType As String
```

Bemerkung

Die Typbezeichnung besteht aus einer beliebigen Zeichenkette. Sie sollten beim Erstellen von ADS-Geräten immer dafür sorgen, dass diese Eigenschaften gesetzt werden, damit dieses ADS-Gerät von anderen Teilnehmern identifiziert werden kann.

4.6.12 AdsClientVersion

Enthält die Versionsnummer des ADS-Gerätes.

```
object.AdsClientVersion As Integer
```

Bemerkung

Jedes ADS-Gerät besitzt Eigenschaften, aus denen die Versionsnummer und die Typbezeichnung des ADS-Gerätes ausgelesen werden kann. Die Versionsnummer besteht aus:

- Version (siehe Eigenschaft AdsClientVersion [▶ 62])
- Revision (siehe Eigenschaft AdsClientRevision [▶ 62])

- Build (siehe Eigenschaft AdsClientBuild [▶ 61])

Sie sollten beim Erstellen von ADS-Geräten immer dafür sorgen, dass diese Eigenschaften gesetzt werden, damit das ADS-Gerät von anderen Teilnehmern identifiziert werden kann.

4.6.13 AdsServerAdsState

Anhand dieser Eigenschaften kann abgefragt werden, in welchem Zustand sich gerade das angesprochene ADS-Gerät befindet.

```
object.AdsServerAdsState As String
```

Bemerkung

Welche Zustände das ADS-Gerät durch diese Eigenschaft signalisert, ist aus der Dokumentation des ADS-Gerätes zu entnehmen.
Dieses Property ist nur lesbar.

4.6.14 AdsServerBuild

Anhand dieser Eigenschaften kann abgefragt werden, welche Version das angesprochenen ADS-Gerät hat

```
object.AdsServerBuild As Integer
```

Bemerkung

Jedes ADS-Gerät besitzt Eigenschaften, aus denen die Versionsnummer und die Typbezeichnung des ADS-Gerätes ausgelesen werden kann. Die Versionsnummer besteht aus:

· Version (siehe Eigenschaft AdsServerVersion [▶ 64])

· Revision (siehe Eigenschaft AdsServerRevision [▶ 63])

· Build (siehe Eigenschaft AdsServerBuild [▶ 63])

4.6.15 AdsServerRevision

Anhand dieser Eigenschaften kann abgefragt werden, welche Revisionlevel das angesprochenen ADS-Gerät hat.

```
object.AdsServerRevision As Integer
```

Bemerkung

Jedes ADS-Gerät besitzt Eigenschaften, aus denen die Versionsnummer und die Typbezeichnung des ADS-Gerätes ausgelesen werden kann. Die Versionsnummer besteht aus:

· Version (siehe Eigenschaft AdsServerVersion [▶ 64])

· Revision (siehe Eigenschaft AdsServerRevision [▶ 63])

· Build (siehe Eigenschaft AdsServerBuild [▶ 63])

4.6.16 AdsServerType

Anhand dieser Eigenschaften kann abgefragt werden, von welchen Typ das angesprochene ADS-Gerät ist.

```
object.AdsServerType As String
```

Bemerkung

In der unteren Tabelle sind die Typbezeichnungen der wichtigsten ADS-Geräte aufgelistet:

ADS-Gerät	Bezeichnung
IO	I/O Server
SPS	PLC Server
NC / NCI	NC-ADS-Server

4.6.17 AdsServerVersion

Anhand dieser Eigenschaften kann abgefragt werden, welche Versionsnummer das angesprochenen ADS-Gerät hat

```
object.AdsServerVersion As Integer
```

Bemerkung

Jedes ADS-Gerät besitzt Eigenschaften, aus denen die Versionsnummer und die Typbezeichnung des ADS-Gerätes ausgelesen werden kann. Die Versionsnummer besteht aus:

- Version (siehe Eigenschaft AdsServerVersion [▶ 64])
- Revision (siehe Eigenschaft AdsServerRevision [▶ 63])
- Build (siehe Eigenschaft AdsServerBuild [▶ 63])

4.6.18 EnableErrorHandling

Schaltet das Exceptionhandling ein.

```
object.EnableErrorHandling As Boolean
```

Bemerkung

Ist diese Eigenschaft TRUE und tritt innerhalb einer Methode ein Fehler auf, so wird eine Exception ausgelöst. Mit der Anweisung *On Error Goto* können Sie an einer definierten Sprungmarke die Exception abfangen und über das Object *Err* die Ursache ermitteln.

4.6.19 Index

Index innerhalb eines Controlarrays.

```
object.Index As Integer
```

Bemerkung

Es ist möglich, ein Array von mehreren ADS-OCXe anzulegen. Dazu bekommt jedes ADS-OCX, welches zu dem Array gehören soll, den gleichen Namen. Unterschieden werden die einzelnen ADS-OCXe durch die Index-Eigenschaft. Der Index beginnt standardmäßig bei 0. Angesprochen werden die einzelnen Objekte durch den Arrayname mit nachfolgendem Index in runden Klammern, z. B. *AdsOcxName(1)*.

4.6.20 Name

Eindeutiger Name des Controls.

```
object.Name As String
```

Bemerkung

Der Standardname für neu hinzugefügte ADS-OCXe ist der Objekttyp (AdsOcx) plus einer eindeutigen Integerzahl. Zum Beispiel hat das erste ADS-OCX den Namen *AdsOcx1*, das Zweite *AdsOcx2* und das Dritte *AdsOcx3*.
Der Name muss mit einem Buchstaben anfangen und kann maximal 40 Zeichen lang sein. Innerhalb des Namens sind Unterstriche (_) und Zahlen erlaubt. Es sollten keine Namen von globalen Systemobjekten (Clipboard, Screen oder App) benutzt werden, da diese anschließend nicht mehr angesprochen werden können.

4.6.21 Object

Mit Hilfe der Object-Eigenschaft eines OLE-Containers können Sie auch die Eigenschaften und Methoden des verknüpften oder eingebetteten Objekts verwenden.

```
object.Object As Object
```

Bemerkung

Diese Eigenschaft wird in Verbindung mit OLE (Object Linking and Embedding) benutzt. Für weitere Informationen siehe Programmierhandbuch von Visual Basic.

4.6.22 Parent

Gibt eine Form, ein Objekt oder eine Collection zurück, in der das ADS-OCX enthalten ist.

```
object.Parent As Object
```

Bemerkung

Um z. B. den Namen des Containers zu ermitteln, müssen Sie folgende Anweisung eingeben:
AdsOcx1.Parent.Name

4.6.23 Tag

Enthält einen frei benutzbaren String.

```
object.Tag As String
```

Bemerkung

In dieser Eigenschaft können beliebige Daten abgelegt werden. Diese werden weder von Visual Basic noch von dem ADS-OCX weiter ausgewertet.

4.7 Enums

4.7.1 ADSDATATYPEID

```
ADST_BIT         = 33 (&H21)
ADST_INT8        = 16 (&H10)
ADST_INT16       = 2
ADST_INT32       = 3
ADST_INT64       = 20 (&H14)
ADST_UINT8       = 17 (&H11)
ADST_UINT16      = 18 (&H12)
ADST_UINT32      = 19 (&H13)
ADST_UINT64      = 21 (&H15)
ADST_REAL32      = 4
ADST_REAL64      = 5
```

```
ADST_REAL80      = 32 (&H20)
ADST_BIGTYPE     = 65 (&H41)
ADST_VOID        = 0
```

4.7.2 ADSLOGMSGTYPE

```
ADSLOG_MSGTYPE_HINT     = 1
ADSLOG_MSGTYPE_WARN     = 2
ADSLOG_MSGTYPE_ERROR    = 4
```

4.7.3 ADSOCXTRANSMODE

```
ADSTRANS_CLIENTCYCLE      = 1
ADSTRANS_SERVERCYCLE      = 3
ADSTRANS_SERVERONCHA      = 4
```

Beschreibung

Parameter	Beschreibung
ADSTRANS_CLIENTCYCLE	Das ADS-OCX führt zyklisch einen Schreib-/ Lesebefehl aus. Die Zykluszeit wird auf das Vielfache von 55 gerundet. Die kleinste Zeit beträgt 55ms. Der Timer, der das Lesen / Schreiben anstößt, läuft im Usermode von Windows NT/2000/XP, deshalb ist das Zeitverhalten stark von der Auslastung des Systems abhängig.
ADSTRANS_SERVERCYCLE (nur beim Lesen)	Das angesprochene ADS-Gerät schreibt zyklisch die Daten zum ADS-OCX. Die kleinste mögliche Zeit, ist die Zykluszeit des ADS-Gerätes; bei der SPS ist dieses die Zykluszeit der Task. Die Zykluszeit kann in 1ms-Schritten ausgewertet werden. Geben Sie als Zykluszeit 0ms an, so werden in jedem Zyklus der Task des ADS-Gerätes die Daten zum ADS-OCX geschickt.
ADSTRANS_SERVERONCHA (nur beim Lesen)	Das angesprochene ADS-Gerät schreibt nur dann die Daten zum ADS-OCX, wenn sich diese geändert haben. Die Abtastung beim ADS-Gerät erfolgt in der Zeit, die durch die Zykluszeit angegeben wird. Die Zykluszeit kann in 1ms-Schritten ausgewertet werden. Geben Sie als Zykluszeit 0ms an, so wird jede Änderung der Variablen zum ADS-OCX geschickt. Durch eine größere Zykluszeit kann die Anzahl der Datenübertragungen zum ADS-OCX reduziert werden.

Die größte Zykluszeit beträgt 32767ms.

HINWEIS

Zu viele Schreib / Leseoperationen

Durch zu viele Schreib- / Leseoperationen kann das System so stark ausgelastet werden, dass die Bedieneroberfläche stark verlangsamt wird.

- Setzen Sie die Zykluszeit auf möglichst angemessene Werte und beenden Sie jedes Mal die Verbindungen, wenn diese nicht mehr benötigt werden.

4.7.4 ADSSTATE

```
ADSSTATE_INVALID     = 0
ADSSTATE_IDLE        = 1
ADSSTATE_RESET       = 2
ADSSTATE_INIT        = 3
ADSSTATE_START       = 4
```

```
ADSSTATE_RUN          = 5
ADSSTATE_STOP         = 6
ADSSTATE_SAVECFG      = 7
ADSSTATE_LOADCFG      = 8
ADSSTATE_POWERFAILURE     = 9
ADSSTATE_POWERGOOD        = 10 (&H0A)
ADSSTATE_ERROR        = 11 (&H0B)
ADSSTATE_SHUTDOWN     = 12 (&H0C)
ADSSTATE_SUSPEND      = 13 (&H0D)
ADSSTATE_RESUME       = 14 (&H0E)
ADSSTATE_CONFIG       = 15 (&H0F)'system is in config mode
ADSSTATE_RECONFIG     = 16 (&H10)'system should restart in config mode
ADSSTATE_MAXSTATES        = 17 (&H11)
```

4.7.5 ADSGETDYNSYMBOLTYPE

```
ADSDYNSYM_GET_NEXT      = 1     ' liefert nächstes Symbol (versucht erst ADSDYNSYM_GET_CHILD, dann
ADSDYNSYM_GET_SIBLING, dann ADSDYNSYM_GET_PARENT)
ADSDYNSYM_GET_SIBLING   = 2     ' liefert nächstes Symbol auf derselben Ebene
ADSDYNSYM_GET_CHILD     = 3     ' liefert Child Symbol
ADSDYNSYM_GET_PARENT    = 4     ' liefert das nächste Symbol auf der Ebene des Parents
```

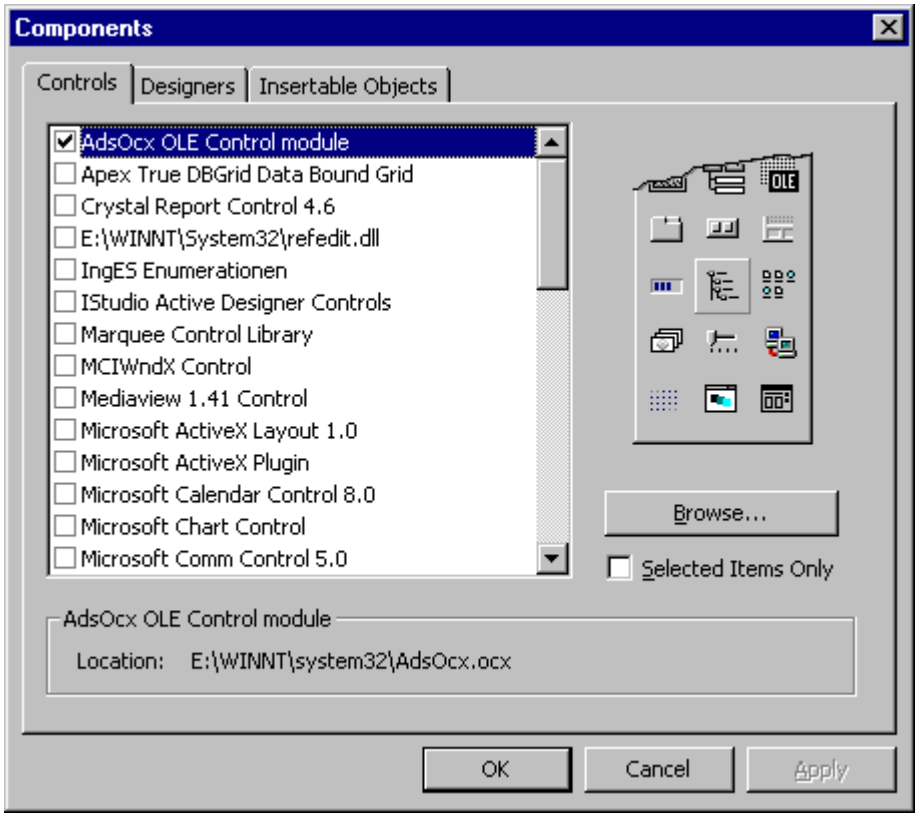
5 Beispiele

5.1 Visual Basic - Beispiele

5.1.1 Einbinden in Visual Basic

ADS-OCX auswählen

Um das ADS-OCX auszuwählen, müssen Sie in Visual Basic unter dem Menüpunkt *Projekt* den Befehl *Komponenten...* auswählen und den Eintrag *AdsOcx OLE Control module* markieren.



Anschließend erscheint das ADS-OCX in der Toolbox von Visual Basic (rechts unten).

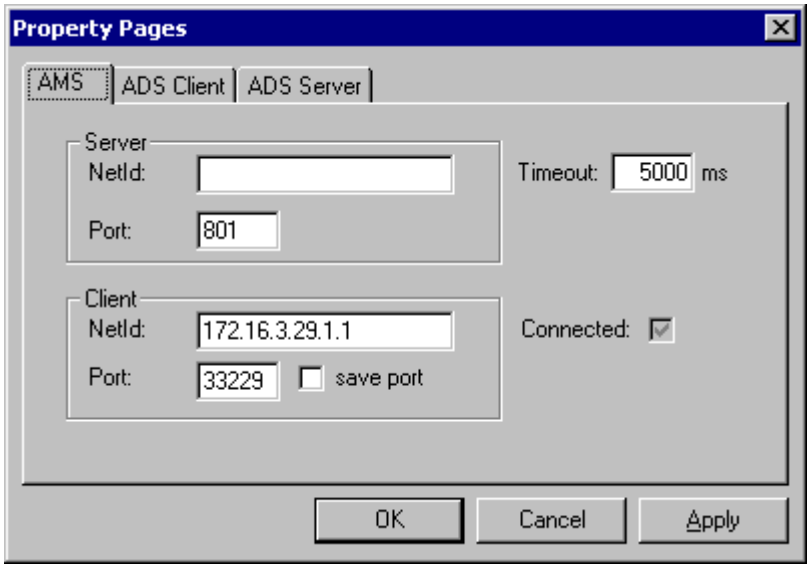
Eigenschaften definieren

Bevor Sie das ADS-OCX nutzen können, müssen Sie dieses auf eine Form ziehen und die Eigenschaften einstellen. Die allgemeinen Eigenschaften können über die *Eigenschaften Seiten* des ADS-OCX oder in der Visual Basic *Eigenschaften-Liste* konfiguriert bzw. ausgelesen werden. Sehr übersichtlich ist die Darstellung

in der *Eigenschaften Seite*, da hier die Eigenschaften nach Gruppen sortiert dargestellt werden. Damit das Visual Basic-Programm besser lesbar ist, fangen (fast) alle Eigenschaften, Methoden und Ereignisse mit *Ads* an.

AMS-Eigenschaften

Diese Seite beschreibt den Kommunikationskanal zwischen dem ADS-OCX und dem ADS-Gerät, welches angesprochen werden soll. Es wird an dieser Stelle auch von ADS-Client und ADS-Server gesprochen. Der ADS-Client ist das Programm, welches beim ADS-Server Informationen oder Dienste anfordert. In unserem Fall ist das Visual Basic-Programm, mit dem ADS-OCX der ADS-Client. In den meisten Beispielen ist der TwinCAT PLC-Server der ADS-Server.



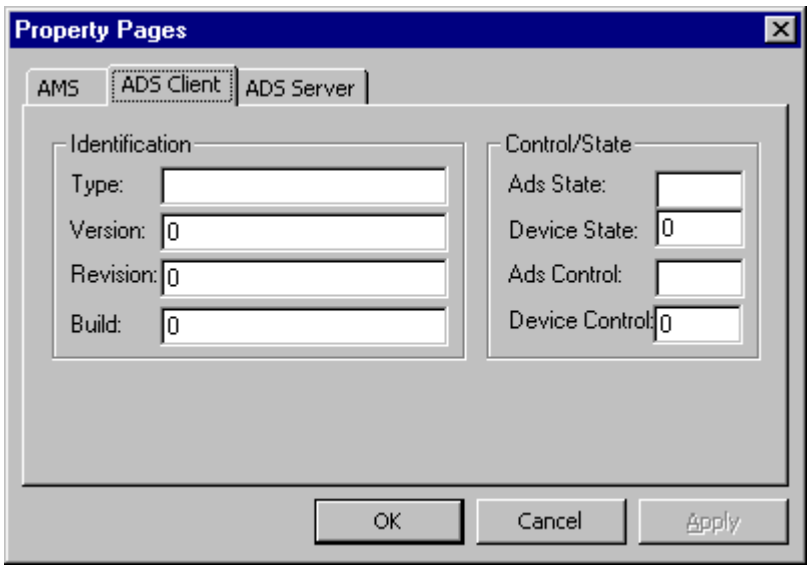
Jedes ADS-Gerät besitzt innerhalb von TwinCAT eine eindeutige Adresse. Diese Adresse setzt sich zusammen aus einer NetId und der Portnummer. Die NetId muss für jedes TwinCAT System innerhalb eines Netzwerks eindeutig sein. Welche NetId ein Rechner besitzt, kann durch das System-Control im Dialog *TwinCAT System Eigenschaften* auf der Seite *AMS Router* abgelesen werden. Die NetId besteht aus 6 Ziffern, die durch einen Punkt voneinander getrennt sind. Geben Sie keine NetId an, wenn Sie lokale ADS-Geräte ansprechen wollen. Neben der NetId wird jedes ADS-Gerät zusätzlich durch eine Portnummer adressiert. Jede Portnummern darf auf einem TwinCAT-Rechner nur einmal vorkommen. Weitere Informationen finden Sie auch unter TwinCAT ADS.

Wollen Sie z. B. auf dem lokalen Rechner das Laufzeitsystem 1 der SPS ansprechen, so lassen Sie das Feld *NetId* vom Server leer und tragen die Portnummer 801 unter *Port* vom Server ein. Die Felder unter *Client* können bei Ihnen andere Werte enthalten als oben abgebildet.

Wenn Sie mit dem ADS-OCX mehrere ADS-Geräte ansprechen wollen (z. B. SPS und NC), dann sollten Sie für jedes dieser ADS-Geräte ein ADS-OCX verwenden. Sie sollten nicht während der Laufzeit die Portnummer oder die NetId ändern.

ADS Client-Eigenschaften

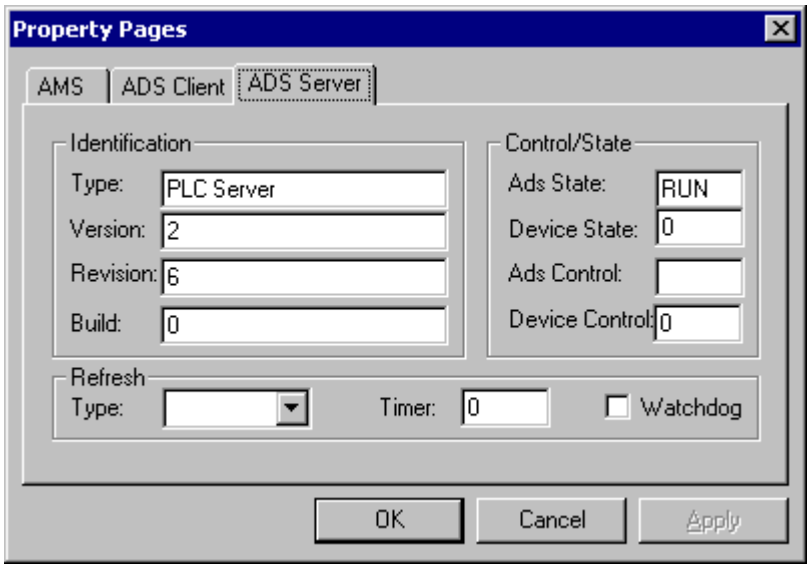
Diese Seite zeigt Typ und Version des ADS-Gerätes mit dem ADS-OCX, sowie den aktuellen Status der ADS-Schnittstelle innerhalb des ADS-OCX.



Die Eigenschaftsgruppe *Identification* zeigt Typ, Version, Revision und Build des ADS-OCX an. Der andere Kommunikationspartner hat die Möglichkeit diese Werte abzufragen, um weitere Informationen über das ADS-Gerät zu bekommen. Diese Felder können von der Applikation freigesetzt werden.
In der Eigenschaftgruppe *Control/State* wird der aktuelle Status der ADS-Schnittstelle innerhalb des ADS-OCX angezeigt.

ADS Server-Eigenschaften

Diese Seite zeigt Typ und Version des ADS-Gerätes, welches vom ADS-OCX angesprochen werden soll. Außerdem wird auch vom ADS-Gerät der aktuelle Status angezeigt.



Die Eigenschaftsgruppe *Identification* zeigt Typ, Version, Revision und Build des ADS-Gerätes an, welches durch das ADS-OCX angesprochen wird. Diese Eigenschaften sind nur lesbar.
In der Eigenschaftsgruppe *Control/State* wird der aktuelle Status der ADS-Schnittstelle innerhalb des ADS-Gerätes angezeigt.
Die Eigenschaftsgruppe *Refresh* ist reserviert und wird noch nicht unterstützt.

Bei dem oben abgebildeten Dialog wird vom ADS-OCX die SPS angesprochen. Diese befindet sich im RUN-Zustand und hat die Version 2.6.0.

Das ADS-OCX ist jetzt so konfiguriert, dass es die SPS ansprechen kann.

5.1.2 Visual Basic 6.0 Variablenlängen

VB-Variablentyp	Variablenlänge in Byte
Boolean	2
Integer	2
Long	4
Single	4
Double	8
String	Anzahl der Zeichen * 2
Byte	1

Größen von Arrays

Die Länge eines Arrays berechnet sich aus der Anzahl der einzelnen Array-Elemente, multipliziert mit der Länge des Variablentyps.

Beispiel

Soll ein Array von 5 Long Elementen ausgelesen werden, beträgt die Länge 20 Byte (5 Elemente * 4 Byte).

Bei einem String mit 25 Zeichen beträgt die Länge 50 Byte (25 Zeichen * 2 Byte).

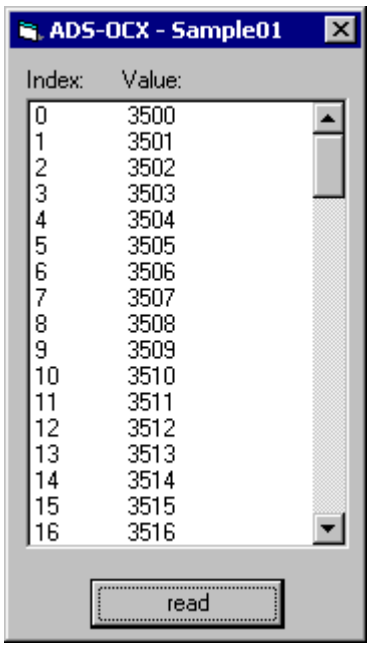
5.1.3 Zugriff auf ein Array in der SPS

Aufgabe

In der SPS befindet sich ein Array, welches von Visual Basic 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 Load-Ereignisfunktion vom Visual Basic-Programm wird als erstes der Handle der SPS-Variablen geholt. Beim Beenden des Programms wird dieser in der Unload-Ereignisfunktion wieder freigegeben.
Betätigt der Benutzer den Button auf der Form, so wird mit der Methode AdsSyncRead[Datatype]VarReq() [▶ 24] das komplette Array aus der SPS in die Visual Basic Variable *Data* eingelesen.
Die Variable *Data* muss den gleichen Aufbau haben, wie die entsprechende Variable in der SPS; 100 Elemente vom Typ Integer (2 Byte). Die Längenangabe bei dem Methodenaufruf ist 200, da die Länge der angeforderten Daten 200 Byte (100 Elemente mit je 2 Byte) beträgt.
In der folgenden FOR-Schleife wird das Array aus der SPS in einem Listbox-Control angezeigt.



Visual Basic 6 Programm

```
Dim hVar As Long
Dim Data(100) As Integer

'--- wird beim Starten aufgerufen ---
Private Sub Form_Load()
  '--- Exception freigeben --- AdsOcx1.EnableErrorHandling = True
  Call AdsOcx1.AdsCreateVarHandle("Main.PLCVar", hVar)
End Sub

'--- wird beim Beenden aufgerufen ---
Private Sub Form_Unload(Cancel As Integer)
  Call AdsOcx1.AdsDeleteVarHandle(hVar)
End Sub

'--- wird vom Bediener aufgerufen ---
Private Sub cmd_read_Click()
  Dim intIndex As Integer
  '--- Array komplett auslesen ---
  Call AdsOcx1.AdsSyncReadIntegerVarReq(hVar, 200, Data(0))
  '--- Array Elemente in Form anzeigen ---
  lstArray.Clear
  For intIndex = 0 To 99
    lstArray.AddItem (CStr(intIndex) & Chr(vbKeyTab) & _ CStr(Data(intIndex)))
  Next
End Sub
```

SPS-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
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463158027.exe

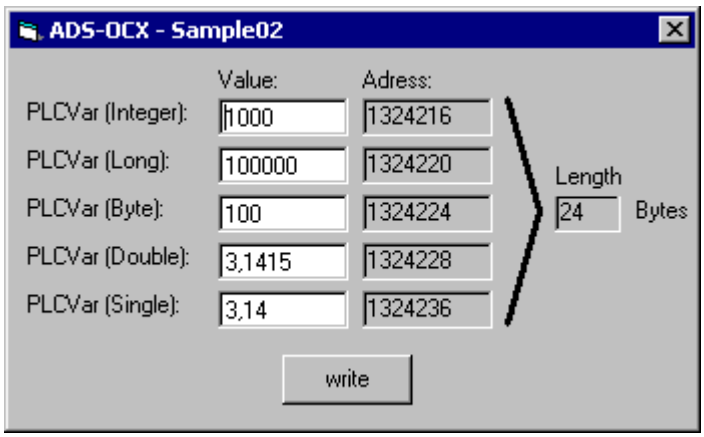
5.1.4 Übertragen von Strukturen an die SPS

Aufgabe

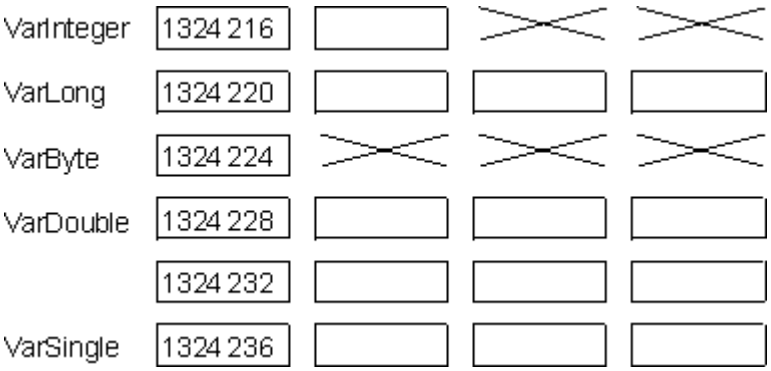
Von Visual Basic soll eine Struktur in die SPS geschrieben werden. Die Elemente der Struktur haben verschiedene Datentypen.

Beschreibung

Damit die CPU unter Windows NT/2000 schneller auf Variablen zugreifen kann, werden diese von Visual Basic (und auch von anderen Programmiersprachen) im Hauptspeicher nach bestimmten Richtlinien ausgerichtet. Dieses Ausrichten der Variablen wird Alignment (engl. Ausrichten) bezeichnet. Dadurch kann es vorkommen, dass innerhalb einer Struktur 'Speicherlücken' entstehen. Da Visual Basic und IEC1131-3 unterschiedliche Richtlinien für das Alignment besitzen, müssen diese durch Dummyvariablen aufgefüllt werden.
Leider lässt sich unter Visual Basic keine Allgemeinregel für das Alignment definieren. Es besteht aber die Möglichkeit mit zwei Visual Basic Funktionen die Speicherbelegung einer Struktur zu analysieren. Es sind die Funktionen *VarPtr()* und *LenB()*.
VarPtr() gibt die Adresse einer Variablen zurück, *LenB()* die Länge in Byte, die eine Variable (oder eine ganze Struktur) belegt. Das untere Beispiel zeigt den Speicheraufbau der Struktur in einer Form an. Anhand dieser Informationen kann ermittelt werden, welche 'Speicherlücken' die Struktur besitzt. In dem Beipielprogramm werden diese durch die Variablen *VarDummyX* aufgefüllt.
Die Funktion *VarPtr()* steht erst ab Visual Basic 5 zur Verfügung.



In der folgenden Skizze wird die Speicheraufteilung nochmals grafisch dargestellt:



Ein Rechteck bedeutet, dass die Variable an dieser Stelle ein Byte belegt. Ein Kreuz stellt an dieser Stelle ein Byte da, welches von keiner Variablen belegt wird. Die Kreuze wurden in dem Beispielprogramm durch Dummyvariablen aufgefüllt.

Strukturdeklaration in Visual Basic

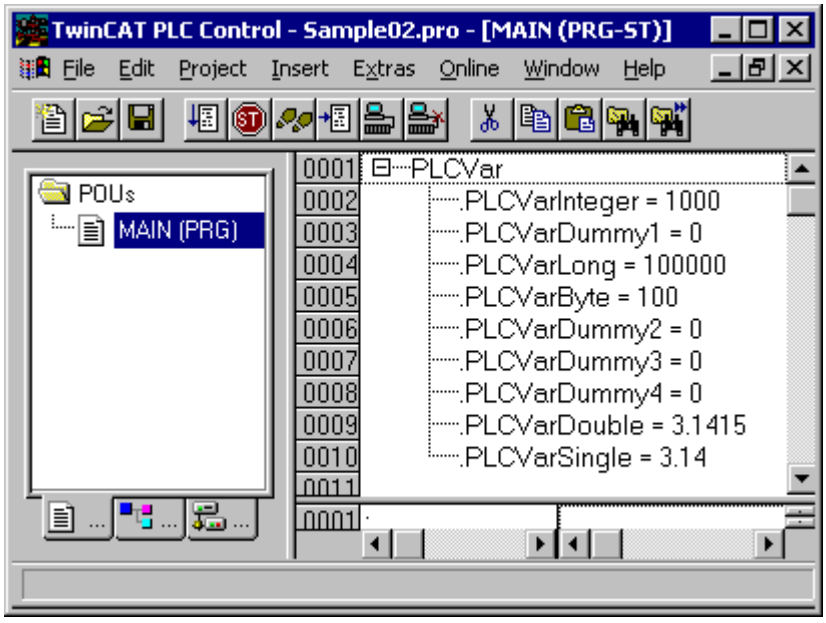
```
Type VBStruct
    VarInteger   As Integer
    VarDummy1    As Integer
    VarLong      As Long
    VarByte      As Byte
```

```
    VarDummy2    As Byte
    VarDummy3    As Byte
    VarDummy4    As Byte
    VarDouble    As Double
    VarSingle    As Single
End Type
```

Strukturdeklaration in der SPS

Nachdem die Struktur im Visual Basic-Programm dem Alignment angepasst wurde, muss die Struktur im SPS-Programm ebenfalls ergänzt werden:

```
TYPE PLCStruct
STRUCT
    PLCVarInteger : INT;
    PLCVarDummy1  : INT;
    PLCVarLong    : DINT;
    PLCVarByte    : SINT;
    PLCVarDummy2  : SINT;
    PLCVarDummy3  : SINT;
    PLCVarDummy4  : SINT;
    PLCVarDouble  : LREAL;
    PLCVarSingle  : REAL;
END_STRUCT
END_TYPE
```



Visual Basic 6 Programm

```
Dim hVar As Long
Dim VBVar As VBStruct

'--- wird beim Starten aufgerufen ---
Private Sub Form_Load()
      '--- Exception freigeben --- AdsOcx1.EnableErrorHandling = True
     Call AdsOcx1.AdsCreateVarHandle("Main.PLCVar", hVar)
      '--- Adressen der Variablen anzeigen ---
     lblInteger.Caption = VarPtr(VBVar.VarInteger)
    lblLong.Caption = VarPtr(VBVar.VarLong)
    lblByte.Caption = VarPtr(VBVar.VarByte)
    lblDouble.Caption = VarPtr(VBVar.VarDouble)
    lblSingle.Caption = VarPtr(VBVar.VarSingle)
      '--- Länge der Struktur anzeigen ---
     lblVarLength.Caption = LenB(VBVar)
End Sub

'--- wird beim Beenden aufgerufen ---
Private Sub Form_Unload(Cancel As Integer)
    Call AdsOcx1.AdsDeleteVarHandle(hVar)
End Sub

'--- wird vom Bediener aufgerufen ---
Private Sub cmd_write_Click()
```

```
    Dim intIndex As Integer
      '--- Struktur auffüllen ---
     VBVar.VarInteger = CInt(txtInteger.Text)
    VBVar.VarLong = CLng(txtLong.Text)
    VBVar.VarByte = CByte(txtByte.Text)
    VBVar.VarDouble = CDbl(txtDouble.Text)
    VBVar.VarSingle = CSng(txtSingle.Text)
      '--- Struktur in SPS schreiben ---
     Call AdsOcx1.AdsSyncWriteIntegerVarReq(hVar, LenB(VBVar), VBVar.VarInteger)
End Sub
```

SPS-Programm

```
PROGRAM MAIN
VAR
    PLCVar : PLCStruct;
END_VAR
```

Optimierungen

Durch eine geschickte Anordnung der VBStruct-Membervariablen in der VB-Applikation kann das hinzufügen der Dummy-Bytes vermieden werden. Folgende Regel muss dabei beachtet werden:

- Ordnen Sie die Membervariablen in der VB-Struktur nach der belegten Speichergröße: Zuerst die größten und zum Schluss die kleinsten Datentypen.
- Die letzten Bytes können (müssen aber nicht) auf volle 4 Bytes aufgefüllt werden.

Optimierte Strukturdeklaration in Visual Basic

```
Type VBStruct
    VarDouble As Double     ' 8 bytes
 VarSingle As Single     '+4 bytes
    VarLong As Long     '+4 byte
    VarInteger As Integer   '+2 bytes
    VarByte As Byte     '+1 byte       '+1 hidden padding byte in memory
                '=20 bytes (LenB result)
End Type
```

Optimierte Strukturdeklaration in der SPS

```
TYPE PLCStruct
STRUCT
    PLCVarDouble : LREAL;
    PLCVarSingle : REAL;
    PLCVarLong : DINT;
    PLCVarInteger : INT;
    PLCVarByte : SINT;
END_STRUCT
END_TYPE
```

Unsere optimierte VB Struktur beginnt jetzt mit einem Double, entsprechend muss das VB-Programm geändert werden:

```
'--- wird vom Bediener aufgerufen ---
Private Sub cmd_write_Click()
...
    '--- Struktur in SPS schreiben ---
 call AdsOcx1.AdsSyncWriteDoubleVarReq(hVar, Len(VBVar), VBVar.VarDouble)
End Sub
```

Neben dem geänderten Methodennamen muss die zu schreibende Datenlänge mit der Len-Funktion und nicht mit LenB ermittelt werden. Wenn Sie LenB benutzen, werden die Daten nicht in die SPS geschrieben. Die Ursache liegt darin, daß LenB eine Länge = 20 Bytes zurückliefert (inklusive eines padding bytes im VB Speicher), unsere Struktur in der SPS aber nur 19 Bytes lang ist.

Len vs. LenB

- With user-defined types, **Len** returns the size as it will be written to the file.
- With user-defined types, **LenB** returns the in-memory size, including any padding between elements.

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463159435.exe

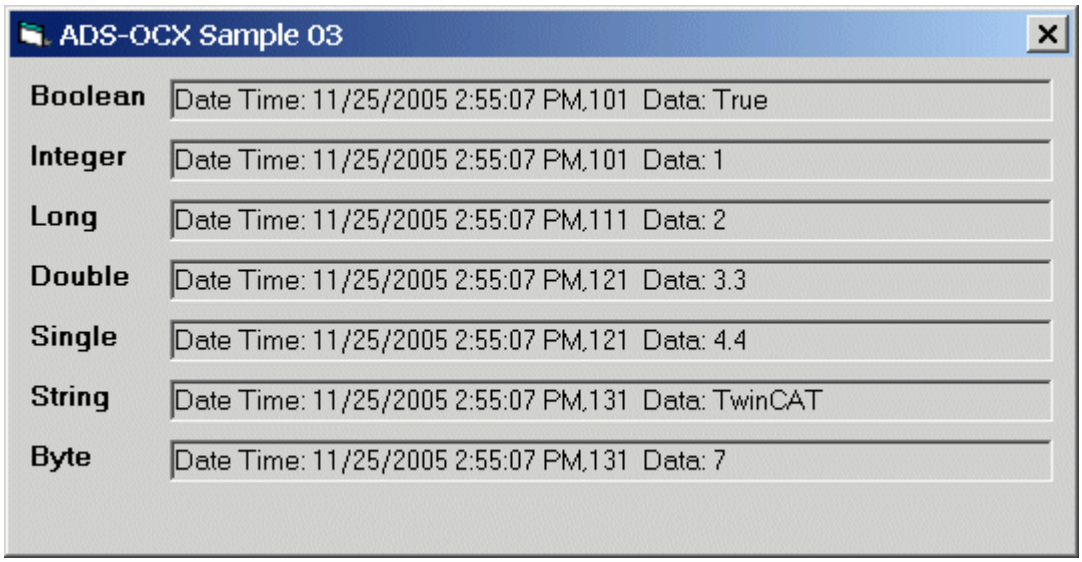
5.1.5 Ereignisgesteuertes Lesen

Aufgabe

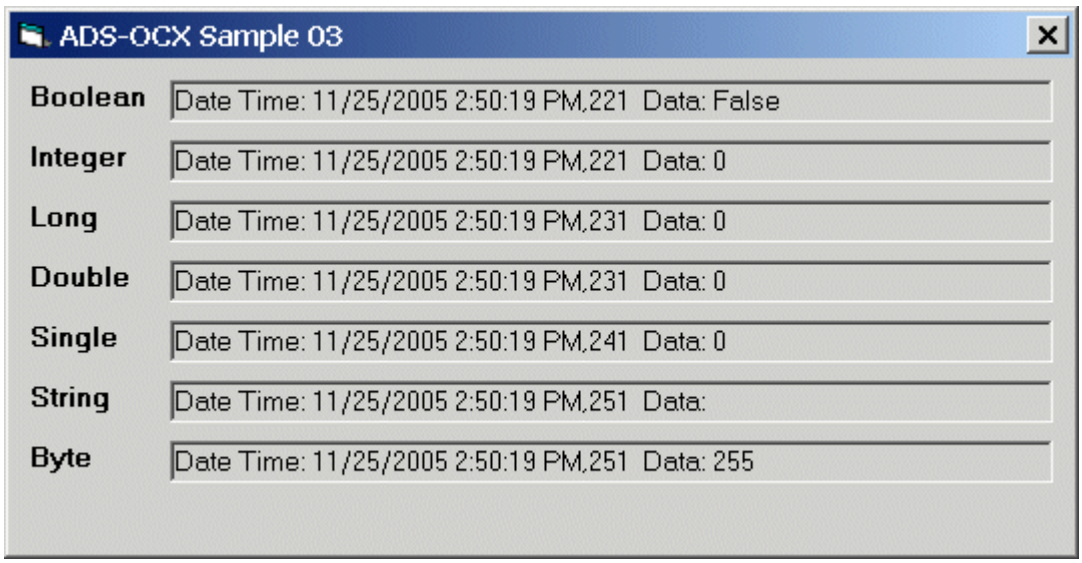
In der SPS befinden sich 7 globale Variablen. Jede SPS-Variable ist von einem anderen Datentyp. Die Werte der Variablen sollen auf möglichst effektive Weise ausgelesen und der Wert mit Zeitstempel auf einer Form in Visual Basic dargestellt werden.

Beschreibung

In dem Load-Ereignis der Form wird mit der Methode AdsReadVarConnectEx() [▶ 34] eine Verbindung zu jeder SPS-Variablen hergestellt. Der Handle dieser Verbindung wird in einen globalen Array abgelegt.
Der zweite Parameter der Methode AdsReadVarConnectEx() gibt die Art des Datenaustausches an. Hier wurde ADSTRANS_SERVERONCHA gewählt. Dadurch wird der Wert der SPS-Variable nur dann übertragen, wenn dieser sich in der SPS geändert hat (siehe Datentyp ADSOCXTRANSMODE [▶ 66]). Über den dritten Parameter wird angegeben das alle 100ms die SPS überprüft, ob sich die entsprechende Variable geändert hat.



Bei einer Änderung der SPS-Variablen wird das Ereignis AdsReadConnectUpdateEx() [▶ 54] aufgerufen. Als Parameter wird der Zeitstempel, der Handle, der Wert und eine Referenz auf das Control, in dem der Wert angezeigt werden soll übergeben.
In dem Unload-Ereignis werden die Verbindungen mit der Methode AdsDisconnectEx() [▶ 40] wieder aufgelöst. Dieses sollten Sie unbedingt beachten, da jede Verbindung, die mit AdsReadVarConnectEx() hergestellt wurde, Ressourcen 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.



Visual Basic 6 Programm

```
Option Explicit

Dim hConnect(0 to 6) As Long

Private Sub Form_Load()
    Dim nErr As Long

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarBoolean", ADSTRANS_SERVERONCHA, 100, hConnect(0), lbl
Boolean)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarBoolean: " & nErr)

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarInteger", ADSTRANS_SERVERONCHA, 100, hConnect(1), lbl
Integer)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarInteger: " & nErr)

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarLong", ADSTRANS_SERVERONCHA, 100, hConnect(2), lblLon
g)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarLong: " & nErr)

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarDouble", ADSTRANS_SERVERONCHA, 100, hConnect(3), lblD
ouble)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarDouble: " & nErr)

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarSingle", ADSTRANS_SERVERONCHA, 100, hConnect(4), lblS
ingle)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarSingle: " & nErr)

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarString", ADSTRANS_SERVERONCHA, 100, hConnect(5), lblS
tring)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarString: " & nErr)

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarByte", ADSTRANS_SERVERONCHA, 100, hConnect(6), lblByt
e)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarByte: " & nErr)
End Sub

Private Sub AdsOcx1_AdsReadConnectUpdateEx(ByVal dateTime As Date,
                        ByVal nMs As Long,
                        ByVal hConnect As Long,
                        ByVal data As Variant,
                        Optional ByVal hUser As Variant)
    hUser.Caption = ("Date Time: " & dateTime & "," & nMs & "  Data: " & data)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Dim nIndex As Long
    For nIndex = 0 To 6
    Call AdsOcx1.AdsDisconnectEx(hConnect(nIndex))
    Next
End Sub
```

SPS-Programm

```
VAR_GLOBAL
    PLCVarBoolean : BOOL;
    PLCVarInteger       : INT;
    PLCVarLong  : DINT;
    PLCVarDouble    : LREAL;
    PLCVarSingle    : REAL;
    PLCVarString    : STRING(10);
    PLCVarByte  : BYTE;
END_VAR

PROGRAM MAIN
VAR
    ;
END_VAR
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463160843.exe

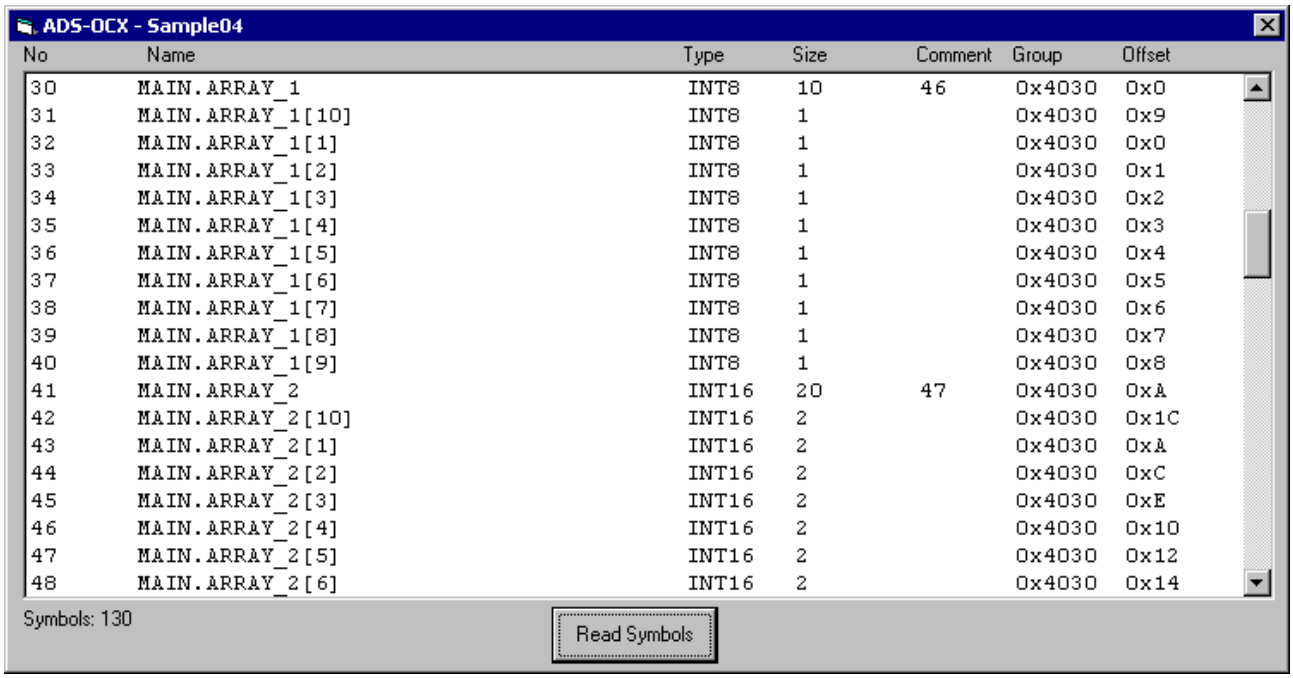
5.1.6 SPS-Variablendeklaration auslesen

Aufgabe

Aus der SPS sollen alle Informationen der Variablen ausgelesen werden (Symbol-Upload).

Beschreibung

Durch betätigen des Buttons auf der Form wird die Ereignisfunktion *cmdReadSymbols_Click()* aufgerufen. Die Methode AdsReadSymbolInfo() [▶ 21] liefert die Anzahl der Variablen (Symbole) und die Länge der Daten, in der die Symbole gespeichert sind. Bei dem ersten Aufruf der Methode AdsEnumSymbols() [▶ 16] muss der Parameter *bNext* auf FALSE gesetzt werden. Dadurch werden alle Informationen des ersten Symbols ausgelesen. Bei jedem weiteren Aufruf wird *bNext* auf TRUE gesetzt. Durch die FOR-Schleife wird AdsEnumSymbols() so oft aufgerufen, wie sich Symbole in der SPS befinden.



Visual Basic 6 Programm

```
Option Explicit

'--- wird beim Starten des Programms aufgerufen ---
Private Sub Form_Load()
    '--- Exception freigeben ---
```

```
    AdsOcx1.EnableErrorHandling = True
    '--- Anzeigeliste- und Felder löschen ---
    lstSymbols.Clear
    lblSymbols.Caption = "Symbols: "
End Sub

'--- wird durch den Bediener aufgerufen ---
Private Sub cmdReadSymbols_Click()
    Dim nSymbolsAvailable As Long
    Dim cbBufSizeNeeded As Long
    Dim strSymbolName As String
    Dim strComment As String
    Dim nSymbolType As Long
    Dim cbSymbolSize As Long
    Dim nIndexOffset As Long
    Dim nIndexGroup As Long
    Dim intIndex As Long

    '--- Anzeigeliste löschen ---
    lstSymbols.Clear

    Call AdsOcx1.AdsReadSymbolInfo(nSymbolsAvailable, cbBufSizeNeeded)
    lblSymbols.Caption = "Symbols: " & nSymbolsAvailable

    '--- erstes Symbol laden und anzeigen ---
    Call AdsOcx1.AdsEnumSymbols(strSymbolName, nSymbolType, cbSymbolSize, _
                strComment, nIndexGroup, nIndexOffset, False)
    lstSymbols.AddItem ("0" & vbTab & Format(strSymbolName, "!
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@") & vbTab & _
            GetSymbolType(nSymbolType) & vbTab & cbSymbolSize & vbTab & strComment & vbTab & _
            "0x" & Hex(nIndexGroup) & vbTab & "0x" & Hex(nIndexOffset))

    '--- die restlichen Symbole laden und anzeigen ---
    For intIndex = 1 To nSymbolsAvailable - 1
    Call AdsOcx1.AdsEnumSymbols(strSymbolName, nSymbolType, cbSymbolSize, _
                    strComment, nIndexGroup, nIndexOffset, True)
    lstSymbols.AddItem (intIndex & vbTab & Format(strSymbolName, "!
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@") & vbTab & _
                GetSymbolType(nSymbolType) & vbTab & cbSymbolSize & vbTab & strComment & vbTab & _
                "0x" & Hex(nIndexGroup) & vbTab & "0x" & Hex(nIndexOffset))
    Next intIndex
End Sub

Private Function GetSymbolType(VarType As ADSDATATYPEID) As String
    Select Case VarType
    Case ADST_BIT:     GetSymbolType = "BIT"
    Case ADST_INT8:    GetSymbolType = "INT8"
    Case ADST_INT16:   GetSymbolType = "INT16"
    Case ADST_INT32:   GetSymbolType = "INT32"
    Case ADST_INT64:   GetSymbolType = "INT64"
    Case ADST_UINT8:   GetSymbolType = "UINT8"
    Case ADST_UINT16:  GetSymbolType = "UINT16"
    Case ADST_UINT32:  GetSymbolType = "UINT32"
    Case ADST_UINT64:  GetSymbolType = "UINT64"
    Case ADST_REAL32:  GetSymbolType = "REAL32"
    Case ADST_REAL64:  GetSymbolType = "REAL64"
    Case ADST_REAL80:  GetSymbolType = "REAL80"
    Case ADST_BIGTYPE: GetSymbolType = "BIGTYPE"
    Case ADST_VOID:    GetSymbolType = "VOID"
    End Select
End Function
```

SPS-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 *)
END_VAR
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463802251.exe

5.1.7 Status vom Router und der SPS erkennen/verändern

Aufgabe

Das ADS-OCX bietet Möglichkeiten, Statusänderungen vom TwinCAT-Router und den ADS-Geräten abzufangen. Dazu stehen die Ereignisse AdsRouterRemove() [▶ 57], AdsRouterShutdown() [▶ 57], AdsRouterStart() [▶ 57], AdsServerStateChanged() [▶ 58] und AdsServerSymChanged() [▶ 58] zur Verfügung.

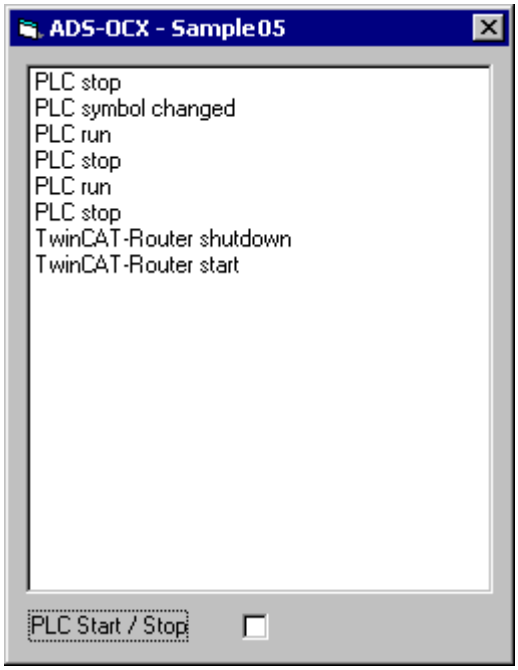
Beschreibung

Wird beim Stoppen des TwinCAT-Routers das Ereignis AdsRouterShutdown() aufgerufen, so kann das betreffende Programm entsprechend darauf reagieren. Beim Starten des TwinCAT-Router wird das Ereignis AdsRouterStart() aufgerufen, in dem dann z. B. mit der Methode AdsReadVarConnectEx() [▶ 34] die Verbindungen zu den ADS-Variablen wieder hergestellt werden kann. Wird in der Systemsteuerung von Windows NT/2000/XP der TwinCAT-Router komplett aus dem Betriebssystem entfernt, so wird das Ereignis AdsRouterRemove() aufgerufen.
Neben Statusänderungen des TwinCAT-Routers können auch Zustandsänderungen in ADS-Geräten abgefangen werden. Besondere Bedeutung hat dieses bei der SPS. Durch das Ereignis

AdsServerStateChanged() kann festgestellt werden, ob die SPS gestartet oder gestoppt wurde. Mit dem Ereignis AdsServerSymChanged() werden Änderungen an der Symboltabelle mitgeteilt. Dieses geschieht z. B. wenn das SPS-Programm komplett neu übersetzt wird und anschließend in die SPS übertragen wird.

So wie der Status eines ADS-Gerätes abgefragt werden kann, so kann dieser auch geändert werden. Die Methode AdsSyncWriteControlReq() [▶ 22] bietet diese Möglichkeit. Die SPS kann die ADS-Zustände STOP und RUN annehmen. Durch ein Check-Button in dem unteren Beispiel-Programm, kann der Bediener zwischen diesen beiden Zuständen umschalten.

Bei jeder Statusänderung des TwinCAT-Routers erfolgt ein entsprechender Eintrag in die Listbox.



HINWEIS

Änderung der Symboltabelle

Wird eine Änderung der Symboltabelle erkannt, so kann es sein, dass eine Variable, die mit AdsReadVarConnectEx() angesprochen wird, gelöscht oder umbenannt wurde. Beim Auftreten des Ereignisses AdsServerSymChanged() sollten alle Connects und Handles gelöscht und anschließend neu angelegt werden.

Visual Basic 6 Programm

```
Option Explicit

'--- wird beim Starten aufgerufen ---
Private Sub Form_Load()
    Call lstEvent.Clear
    AdsOcx1.EnableErrorHandling = True
End Sub

'--- wird aufgerufen, wenn sich der Status des ADS-Gerätes ändert ---
Private Sub AdsOcx1_AdsServerStateChanged(ByVal nAdsState As ADSOCXLib.ADSSTATE, ByVal nDeviceState
As Long)
    Select Case nAdsState
    Case ADSSTATE_INVALID:      lstEvent.AddItem ("PLC invalid")
    Case ADSSTATE_IDLE:     lstEvent.AddItem ("PLC idle")
    Case ADSSTATE_RESET:    lstEvent.AddItem ("PLC reset")
    Case ADSSTATE_INIT:     lstEvent.AddItem ("PLC init")
    Case ADSSTATE_START:    lstEvent.AddItem ("PLC start")
    Case ADSSTATE_RUN:      lstEvent.AddItem ("PLC run")
                    chkRunStop.Value = 1
    Case ADSSTATE_STOP:     lstEvent.AddItem ("PLC stop")
                    chkRunStop.Value = 0
    Case ADSSTATE_SAVECFG:      lstEvent.AddItem ("PLC savecfg")
    Case ADSSTATE_LOADCFG:      lstEvent.AddItem ("PLC loadcfg")
    Case ADSSTATE_POWERFAILURE: lstEvent.AddItem ("PLC powerfailure")
    Case ADSSTATE_POWERGOOD:    lstEvent.AddItem ("PLC powergood")
    Case ADSSTATE_ERROR:    lstEvent.AddItem ("PLC error")
    End Select
```

```
End Sub

'--- wird bei Änderung der Symboltabelle aufgerufen ---
Private Sub AdsOcx1_AdsServerSymChanged()
    lstEvent.AddItem ("PLC symbol changed")
End Sub

'--- wird beim Entfernen des TwinCAT-Routers aufgerufen ---
Private Sub AdsOcx1_AdsRouterRemove()
    lstEvent.AddItem ("TwinCAT-Router remove")
End Sub

'--- wird beim Stoppen des TwinCAT-Routers aufgerufen ---
Private Sub AdsOcx1_AdsRouterShutdown()
    lstEvent.AddItem ("TwinCAT-Router shutdown")
End Sub

'--- wird beim Starten des TwinCAT-Routers aufgerufen ---
Private Sub AdsOcx1_AdsRouterStart()
    lstEvent.AddItem ("TwinCAT-Router start")
End Sub

'--- wird vom Bediener aufgerufen ---
Private Sub chkRunStop_Click()
    Dim nState As ADSOCXLib.ADSSTATE
    Dim nRet As Integer
    nState = IIf(chkRunStop.Value = 0, ADSSTATE_STOP, ADSSTATE_RUN)
    Call AdsOcx1.AdsSyncWriteControlReq(nState, 0&, 0&, nRet)
End Sub
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463803659.exe

5.1.8 Meldungen über den Router senden/empfangen

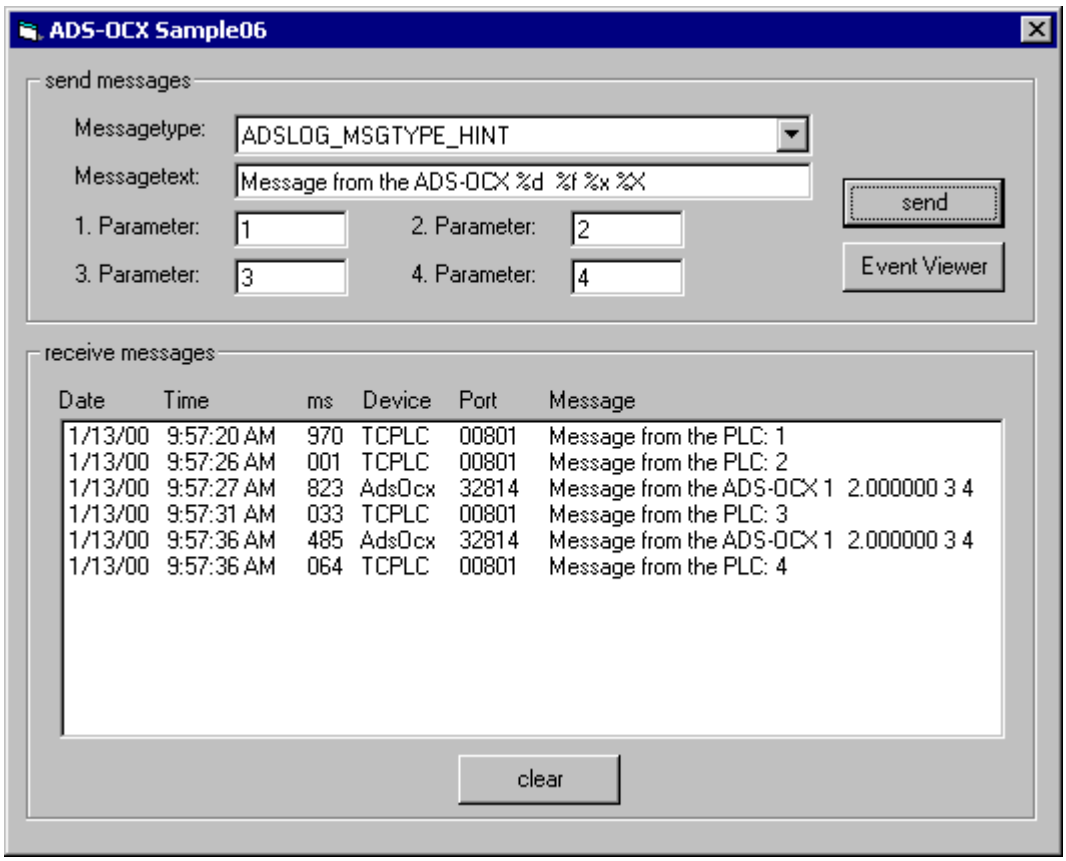
Aufgabe

ADS-Geräte können über den TwinCAT-Router Meldungen zu anderen ADS-Geräten schicken. Diese können dort empfangen und ausgewertet werden. Ebenso ist es möglich, Meldungen in den Windows NT/2000/XP Event Logger zu schreiben.
Das folgende Visual Basic-Programm empfängt Meldungen von der SPS und zeigt diese auf dem Bildschirm an. Ebenso kann von dem Visual Basic-Programm aus, Meldungen in den Windows NT/2000/XP Event Logger geschrieben werden.

Beschreibung

Um Meldungen empfangen zu können, muss als erstes mit der Methode AdsEnableLogNotification() [▶ 15] ein Filter definiert werden. Hierbei wird der Bereich der Portnummern angegeben, von dessen ADS-Geräten Meldungen empfangen werden sollen. Als zweiten Parameter wird noch die Meldeart angegeben (Fehler, Hinweis oder Warnung). Mit dem OR-Operator können auch mehrere Meldearten kombiniert werden. Jedesmal wenn eine Meldung von einem ADS-Gerät abgeschickt wurde und die Filterbedingungen erfüllt sind, wird das Ereignis AdsLogNotification() [▶ 53] aufgerufen. Über die Parameter kann die Quelle, die Art, der Zeitpunkt und die Meldung an sich ermittelt werden.
Soll eine Meldung mit Hilfe des ADS-OCX abgeschickt werden, so wird hierzu die Methode AdsLogFmtString() [▶ 19] benutzt. Der erste Parameter enthält den Meldetyp (Fehler, Hinweis oder Warnung). Der Meldetext kann bis zu vier Platzhalter für numerische Werte enthalten. Dadurch können z. B. Werte übertragen werden, die erst zur Laufzeit des Programms bekannt sind. Alle Meldungen, die mit dem ADS-OCX abgeschickt werden, werden automatisch in den Event Logger von Windows NT/2000/XP geschrieben.

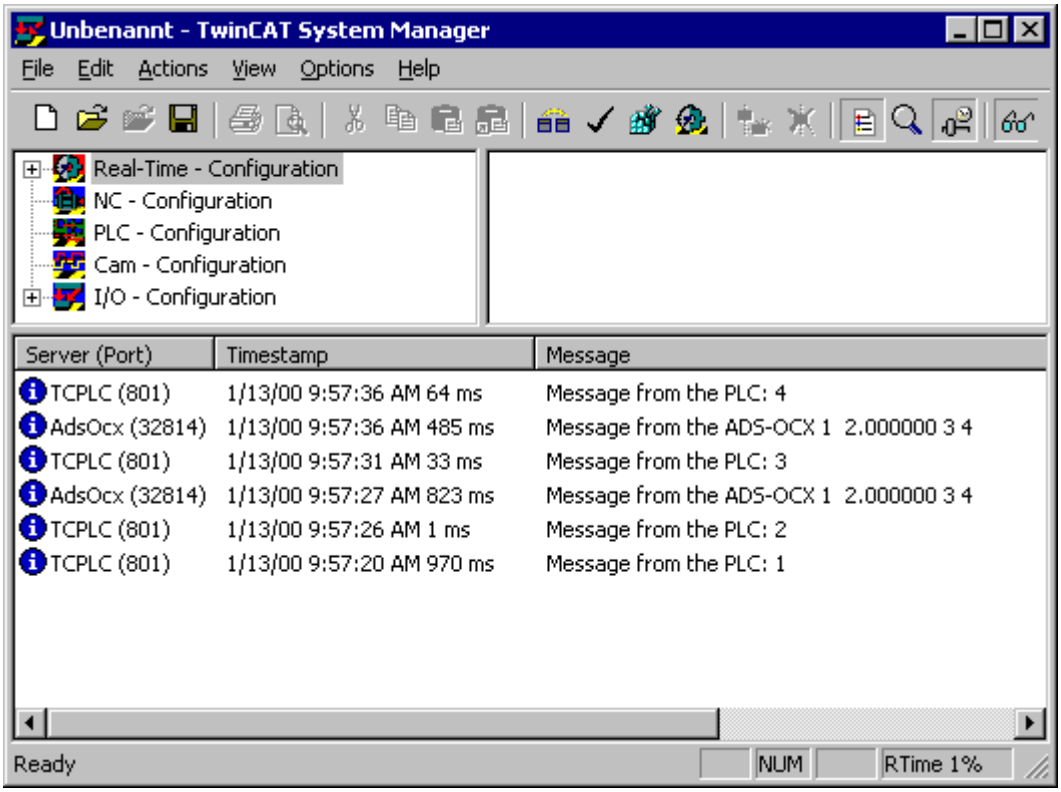
Von dem Visual Basic-Programm kann der Event Viewer von Windows NT/2000/XP aufgerufen werden. Hiermit lassen sich alle Meldungen betrachten, die im Event Logger enthalten sind. Event Logger und Event Viewer werden mit Windows NT/2000/XP standardmäßig ausgeliefert. Nähere Informationen entnehmen Sie bitte der Windows NT/2000/XP Dokumentation.



Das SPS-Programm setzt zyklisch alle 5 Sekunden einen Hinweis ab. Dieser Hinweis wird zusätzlich noch in den Windows NT/2000/XP Event Logger geschrieben. Die SPS-Funktion ADSLOGDINT() ist in der Dokumentation zur SPS beschrieben.

Meldungen online beobachten

Alle Meldungen, die über den TwinCAT-Router abgeschickt werden, können im System Manager online beobachtet werden. Hierzu muss im Menü Ansicht die Logger Ausgabe aktiv sein.



Windows NT/2000/XP beinhaltet API-Funktionen mit denen die gespeicherten Meldungen aus dem Event Logger wieder ausgelesen werden können. Visual Basic hat leider (noch) keine Komponenten, um auf die gespeicherten Meldungen im Event Logger zugreifen zu können. In der Zeitschrift basico*pro* 6/98 Seite 56ff aus dem Steingräberverlag (http://www.basicpro.de) ist ein Artikel veröffentlicht worden, der die Anwendung der API-Funktionen unter Visual Basic demonstriert.

Anwendung

Bei der Entwicklung und Fehlersuche von Applikationen hat sich diese Möglichkeit als sehr hilfreich erwiesen. So kann z. B. ein SPS-Programm oder ein Visual Basic-Programm bestimmte interne Programmzustände anzeigen (im System Manager) und speichern (Windows NT/2000 Event Logger). Für diese Art der Programmverfolgung ist es nicht notwendig die Entwicklungsumgebung (z. B. auf einem Maschinenrechner) zu installieren.

HINWEIS

Zu viele Meldungen in kurzer Zeit

Achten Sie darauf, dass nicht zu viele Meldungen in kurzer Zeit übertragen werden, da dieses das Gesamtsystem sonst beeinträchtigen könnte.

Meldungen protokollieren

Wollen Sie in Ihrem Programm Meldungen protokollieren (z. B. Störungen einer Maschine), so sollten Sie dafür den TwinCAT-Event Logger benutzen. Dieser ist deutlich leistungsfähiger als der Event Logger von Windows NT/2000/XP und auf die Anforderungen der Automatisierungstechnik abgestimmt.

Visual Basic 6 Programm

```
Option Explicit

'--- wird beim Starten des Programms aufgerufen ---
Private Sub Form_Load()
    cboMessageType.ListIndex = 0
    AdsOcx1.EnableErrorHandling = True
    '--- Meldungen abfangen ---
    Call AdsOcx1.AdsEnableLogNotification(1, 65535, ADSLOG_MSGTYPE_HINT Or ADSLOG_MSGTYPE_ERROR Or A
DSLOG_MSGTYPE_WARN)
End Sub

'--- wird beim eintreffen einer Nachricht vom AdsOCX aufgreufen ---
Private Sub AdsOcx1_AdsLogNotification(ByVal dateTime As Date, ByVal nMs As Long, _
                      ByVal dwMsgCtrl As Long, ByVal nServerPort As Long, _
                      ByVal szDeviceName As String, ByVal szLogMsg As String)
    '--- Meldung anzeigen ---
    lstMessages.AddItem Format(DateValue(dateTime), "!@@@@@@@@@@") & _
            Format(TimeValue(dateTime), "!@@@@@@@@@@@@@@@@") & _
            Format(nMs, "000   ") & _
            Format(szDeviceName, "!@@@@@@@@@@") & _
            Format(nServerPort, "00000     ") & _
            szLogMsg
End Sub

'--- Meldung absetzen ---
Private Sub cmdSend_Click()
    Dim Para1 As Long
    Dim Para2 As Double
    Dim Para3 As Integer
    Dim Para4 As Integer
    '--- Parameter setzen ---
    Para1 = CLng(txt1Para.Text)
    Para2 = CDbl(txt2Para.Text)
    Para3 = CInt(txt3Para.Text)
    Para4 = CInt(txt4Para.Text)
    '--- Meldung absetzen ---
    Call AdsOcx1.AdsLogFmtString(cboMessageType.ItemData(cboMessageType.ListIndex), _
                txtMessage.Text, Para1, Para2, Para3, Para4)
End Sub

'--- Ereignisanzeige von Windows NT/2000 anzeigen ---
Private Sub cmdEventViewer_Click()
    Call Shell("eventvwr.exe", vbNormalFocus)
End Sub
```

```
'--- List löschen ---
Private Sub cmdClearList_Click()
    Call lstMessages.Clear
End Sub
```

SPS Programm

```
PROGRAM MAIN
VAR
    PLCVarInteger AT %MW0  : INT;
    TP_1  : TP;
    TOGGEL  : BOOL;
    AdsLogResult  : DINT;
END_VAR

TOGGEL := NOT TOGGEL;
TP_1( IN := TOGGEL, PT := t#5s);
IF (TP_1.Q = 0) THEN
    IF (TOGGEL = 0) THEN
    PLCVarInteger := PLCVarInteger + 1;
    AdsLogResult := ADSLOGDINT(ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_LOG , 'Message from the PLC: %d
', PLCVarInteger);
    END_IF
END_IF
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463805067.exe

5.1.9 Handle einer SPS-Variablen löschen

In diesem Beispiel wird gezeigt, wie das Handle einer SPS-Variablen gelöscht werden kann:

Visual Basic 6 Programm

```
Dim handle As Long

'--- Is called at the start ---
Private Sub Form_Load()
    txtHandle.Text = handle
End Sub

' --- Is called when "Get Handle" is pressed ---
Private Sub btnGetHandle_Click()
    Call AdsOcx1.AdsCreateVarHandle("MAIN.PLCVar", handle)
    txtHandle.Text = handle
End Sub

' --- Is called when "Release Handle" is pressed ---
Private Sub btnReleaseHandle_Click()
    Call AdsOcx1.AdsDeleteVarHandle(handle)
    handle = 0
    txtHandle.Text = handle
End Sub
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463806475.exe

5.1.10 Ereignisgesteuertes Lesen (mit Konvertierung in einen anderen Typ)

Ab TwinCAT 2.8 Build > 743 und höher

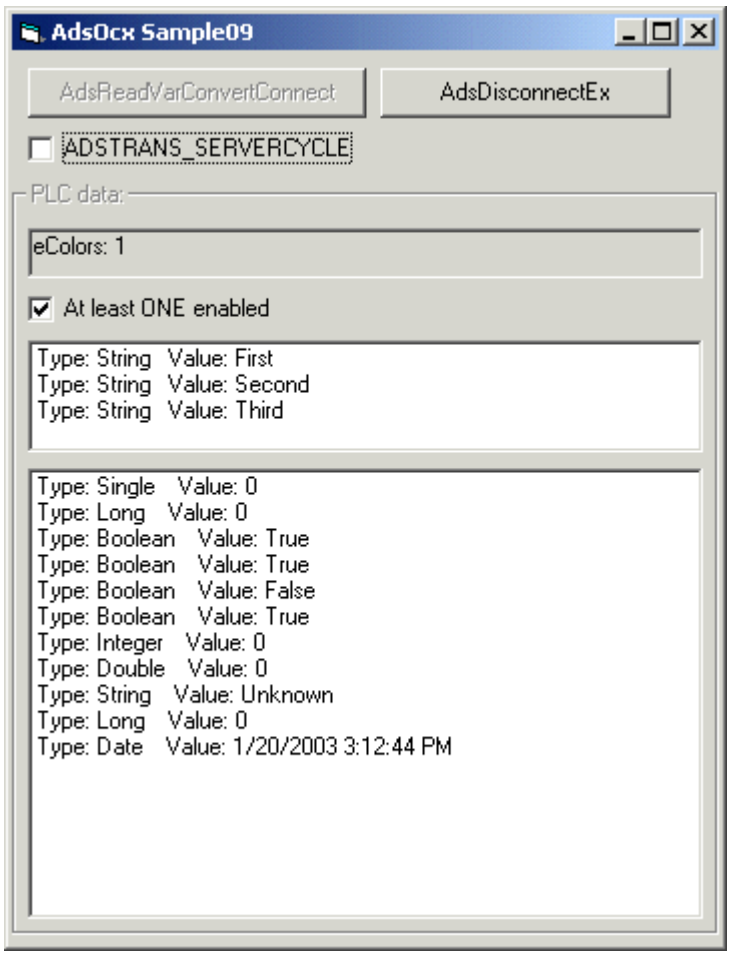
Aufgabe

In der SPS befinden sich 4 Variablen vom unterschiedlichen Typ. Die Variablen sollen auf möglichst effektive Weise ausgelesen und die Werte auf einer Visual Basic Form dargestellt werden. Mit einer Checkbox kann zwischen zwei Verbindungsmodi (ADSTRANS_SERVERCYCLE oder ADSTRANS_SERVERONCHA) umgeschaltet werden. Mit zwei Buttons kann die Verbindung zu den SPS-Variablen aufgebaut oder abgebaut werden.

Bei den SPS-Variablen handelt es sich um strukturierte Datentypen. Diese werden z. B. von der SPS als ein Datenblock an den AdsOcx-Client gesendet. Das AdsOcx kann aber an die VB-Ereignisroutine nur Variablen vom bestimmten Datentyp als Parameter übergeben, darunter den Variant-Typ. Mit der Methode AdsReadVarConvertConnect [▶ 37] kann der Typ der Variant-Variablen in der VB-Ereignisroutine vom Benutzer vorher festgelegt werden. Die Ereignisdaten werden dann vom AdsOcx in die Variant-Variable kopiert und so an die VB-Ereignisroutine übergeben. Ein Variant-Array kann auch eine komplexe Struktur in der SPS abbilden. Wieviele Daten in die einzelnen Variant-Elemente kopiert werden, wird durch den Typ der einzelnen Elemente festgelegt. Einige Ausnahmen z. B. bei Strings (die Stringlänge muss vorher entsprechend gesetzt werden) und boolschen Variablen (aus 1 Byte Daten wird ein 2 Byte VB-Boolean) sind dabei zu beachten.

Folgende SPS-Variablen sollen auf der Form angezeigt werden:

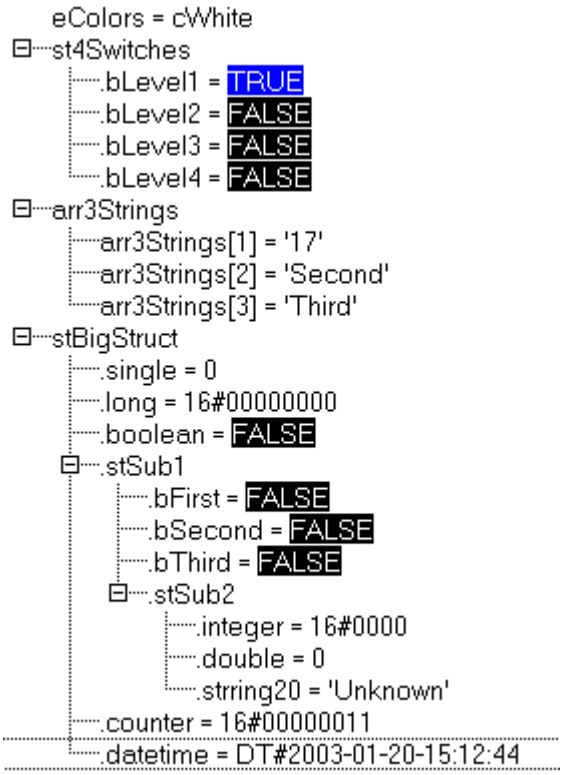
- Der Wert eines Aufzählungstyps (Enum) soll in eine Long-Variable eingelesen und im Label angezeigt werden.
- Der Wert eines Strukturierten-Datentyps (Struktur mit 4-Booleans) soll in eine Long-Variable eingelesen und in einer CheckBox dargestellt werden. Die CheckBox soll ausgewählt werden, wenn eine der boolschen Variablen in der SPS den Wert TRUE besitzt.
- Der Wert eines Stringarrays soll in einer ListBox angezeigt werden.
- Der Wert eines Strukturierten-Datentyps soll in ein Variantarray eingelesen und in einer weiteren ListBox angezeigt werden.



Die SPS-Applikation

```
VAR_GLOBAL
    eColors         : E_Colors := cWhite;
    st4Switches     : ST_4Switches;
    arr3Strings     : ARRAY[1..3] OF STRING :=1('First'), 1('Second'),1('Third');
    stBigStruct     : ST_BigStruct;
END_VAR
```

Online-Ansicht der SPS-Daten:



die Definition des Aufzählungstyps:

```
TYPE E_Colors :
(
    cUnknown,
    cWhite := 1,
    cBlue := 2,
    cRed := 3,
    cBlack
);
END_TYPE
```

Die Definition der Struktur mit 4 boolschen Variablen:

```
TYPE ST_4Switches :
STRUCT
    bLevel1 : BOOL;
    bLevel2 : BOOL;
    bLevel3 : BOOL;
    bLevel4 : BOOL;
END_STRUCT
END_TYPE
```

Die Definition des Strukturierten-Datentyps:

```
TYPE ST_BigStruct :
STRUCT
    single : REAL;
    long : DINT;
    boolean : BOOL;
    stSub1 : ST_Sub1;
    counter : DINT;
    datetime : DT := DT#2003-01-20-15:12:44;
END_STRUCT
END_TYPE
```

Dieser besitzt wiederum 2 Unterstrukturen:

```
TYPE ST_Sub1 :
STRUCT
    bFirst : BOOL;
    bSecond : BOOL;
    bThird : BOOL;
    stSub2 : ST_Sub2;
END_STRUCT
END_TYPE
```

```
TYPE ST_Sub2 :
STRUCT
    integer : INT;
    double : LREAL;
    string20 : STRING(20) := 'Unknown';
END_STRUCT
END_TYPE
```

Visual Basic 6 Programm

```
Option Explicit
Dim adsErr As Long
Dim hConnect_EnumVar As Long
Dim hConnect_4Switches As Long
Dim hConnect_StringArray As Long
Dim hConnect_BigStruct As Long
```

Beim Laden der Form wird eine Verbindung zum ersten SPS-Laufzeitsystem aufgebaut:

```
Private Sub Form_Load()
    AdsOcx1.AdsAmsServerNetId = AdsOcx1.AdsAmsClientNetId
    AdsOcx1.AdsAmsServerPort = 801
    AdsOcx1.EnableErrorHandling = True
End Sub
```

Bei einem Mausklick auf den *AdsReadVarConvertConnect* - Button wird eine Verbindung zu den SPS-Variablen aufgebaut. Beim Erfolg liefert die Methode AdsReadVarConvertConnectein Handle zurück. Nur über dieses Handle wird die Verbindung identifiziert und kann später abgebaut werden.

1. Der Aufzählungstyp in der SPS belegt nur 2 Byte Speicher. Diese 2 Byte werden in eine Long-Variable (4 Byte) eingelesen und in der Ereignisfunktion zurückgeliefert. Man könnte aber genauso gut den VB Integer-Datentyp verwenden.

2. Die 4 boolschen Werte der Strukturvariablen belegen in der SPS 4 einzelne Bytes SPS-Speicher. Diese werden in eine Long-Variable eingelesen und in der Ereignisfunktion als eine Long-Variable zurückgeliefert.

3. Die Strings in dem Array belegen in der SPS insgesamt 243 Byte Speicher (Definierte Stringlänge + 1 Byte für die Nullterminierung) *3. Die Länge der einzelnen VB-Strings muss der Länge der SPS-Strings entsprechen, um die einzelnen Strings trennen zu können. Besitzt der String die Länge Null, werden keine Ereignisdaten in eine Stringvariable hineinkopiert.

4. Die Strukturvariable kann in ein eindimensionales Variant-Array eingelesen werden. Die einzelnen Arrayelemente können vom unterschiedlichen Typ sein. Vor dem Aufbau der Verbindung müssen aber die einzelnen Arrayelemente mit entsprechendem Typ initialisiert werden.

```
Private Sub cmdConnect_Click()
    Dim adsTransMode As ADSOCXTRANSMODE
    adsTransMode = IIf(chkTransMode.Value = vbChecked, ADSTRANS_SERVERCYCLE, ADSTRANS_SERVERONCHA)

    'Connects to enum var
    Dim convertedEnumVar As Long
    adsErr = AdsOcx1.AdsReadVarConvertConnect(".eColors", adsTransMode, 300, hConnect_EnumVar, conve
rtedEnumVar, lblEnum)

    'Connects to struct with 4 boolean variables
    Dim converted4Switches As Long
    adsErr = AdsOcx1.AdsReadVarConvertConnect(".st4Switches", adsTransMode, 300, hConnect_4Switches,
 converted4Switches, chk4Switches)

    'Connects to array of strings
    Dim convertedStringArray(1 To 3) As String
    Dim i As Integer
    For i = LBound(convertedStringArray) To UBound(convertedStringArray)
        convertedStringArray(i) = String(81, "#")
    Next i
    adsErr = AdsOcx1.AdsReadVarConvertConnect(".arr3Strings", adsTransMode, 300, hConnect_StringArra
y, convertedStringArray, lstStringArray)


    'Connects to struct variable
    Dim convertedBigStruct(1 To 11) As Variant
    convertedBigStruct(1) = CSng(0)         'stBigStruct.single
    convertedBigStruct(2) = CLng(0)     'stBigStruct.long
    convertedBigStruct(3) = CBool(False)    'stBigStruct.boolean
```

```
    convertedBigStruct(4) = CBool(False)    'stBigStruct.stSub1.bFirst
    convertedBigStruct(5) = CBool(False)    'stBigStruct.stSub1.bSecond
    convertedBigStruct(6) = CBool(False)    'stBigStruct.stSub1.bThird
    convertedBigStruct(7) = CInt(0)     'stBigStruct.stSub1.stSub2.integer
    convertedBigStruct(8) = CDbl(0)     'stBigStruct.stSub1.stSub2.double
    convertedBigStruct(9) = CStr(String(21, "*"))'stBigStruct.stSub1.stSub2.string20
    convertedBigStruct(10) = CLng(0)        'stBigStruct.counter
    convertedBigStruct(11) = CDate(0)           'stBigStruct.datetime
    adsErr = AdsOcx1.AdsReadVarConvertConnect(".stBigStruct", adsTransMode, 300, hConnect_BigStruct,
 convertedBigStruct, lstBigStruct)

    cmdConnect.Enabled = False
    cmdDisconnect.Enabled = True
End Sub
```

Bei einem Mausklick auf AdsDisconnectEx [▶ 40] - Button werden die Verbindungen zu den SPS-Variablen getrennt:

```
Private Sub cmdDisconnect_Click()
    adsErr = AdsOcx1.AdsDisconnectEx(hConnect_EnumVar)
    adsErr = AdsOcx1.AdsDisconnectEx(hConnect_4Switches)
    adsErr = AdsOcx1.AdsDisconnectEx(hConnect_StringArray)
    adsErr = AdsOcx1.AdsDisconnectEx(hConnect_BigStruct)

    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
End Sub
```

Die Ereignisroutine AdsReadConvertConnectUpdate [▶ 55]. Diese Ereignisroutine wird zyklisch (wenn ADSTRANS_SERVERCYCLE ausgewählt) oder immer nur dann aufgerufen, wenn sich der Wert der SPS-Variablen geändert hat (bei ADSTRANS_SERVERONCHA). Der hUser Parameter kann dafür benutzt werden, um die Ereignisdaten dem passenden Control (Label, CheckBox, ListBox) zuordnen zu können.

```
Private Sub AdsOcx1_AdsReadConvertConnectUpdate(ByVal dateTime As Date, ByVal nMs As Long, ByVal hCo
nnect As Long, data As Variant, Optional hUser As Variant)
    Dim i As Integer
    If TypeOf hUser Is CheckBox Then

        chk4Switches.Value = IIf(data = 0, vbUnchecked, vbChecked)
        chk4Switches.Caption = IIf(data = 0, "ALL disbled", "At least ONE enabled")

    ElseIf TypeOf hUser Is ListBox Then

        If hUser Is lstStringArray Then
            Call lstStringArray.Clear
            For i = LBound(data) To UBound(data)
                Call lstStringArray.AddItem("Type: " & TypeName(data(i)) & " Value: " & data(i))
            Next i
        ElseIf hUser Is lstBigStruct Then
            Call lstBigStruct.Clear
            For i = LBound(data) To UBound(data)
                Call lstBigStruct.AddItem("Type: " & TypeName(data(i)) & " Value: " & data(i))
            Next i
        End If

    Else 'lblEnum
        Dim objLabel As Label
        Set objLabel = hUser
        objLabel.Caption = "eColors: " & data
    End If
End Sub
```

Sprache / IDE	Beispielprogramm auspacken
Visual Basic 6	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12463807883.exe

5.2 Delphi - Beispiele

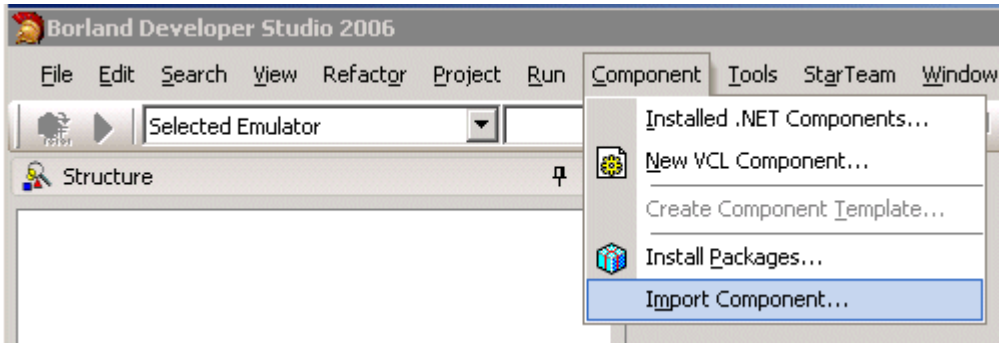
5.2.1 Integration in Delphi

5.2.1.1 Einbinden in Borland Developer Studio 2006 (VCL for Delphi Win32)

Diese Anleitung kann auch benutzt werden, um das ADS-OCX in **Borland Delphi 2005** einzubinden. Die Unterschiede zu **"Borland Delphi 2006" oder "Delphi XE2"** sind nur geringfügig.

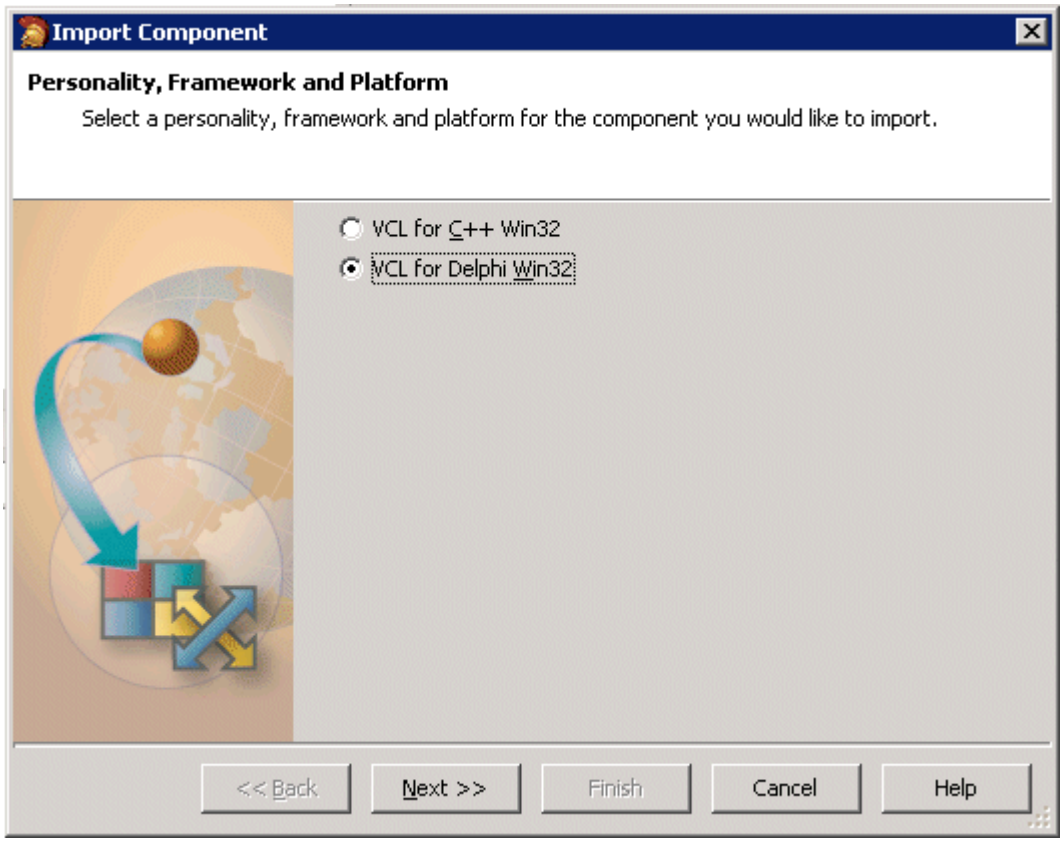
Schritt 1

Zuerst muss ein Delphi-Unit aus dem ActiveX-Steuerelement abgeleitet werden. Dazu wählen Sie unter dem Menüpunkt *"Komponente" >"Komponente importieren..."*



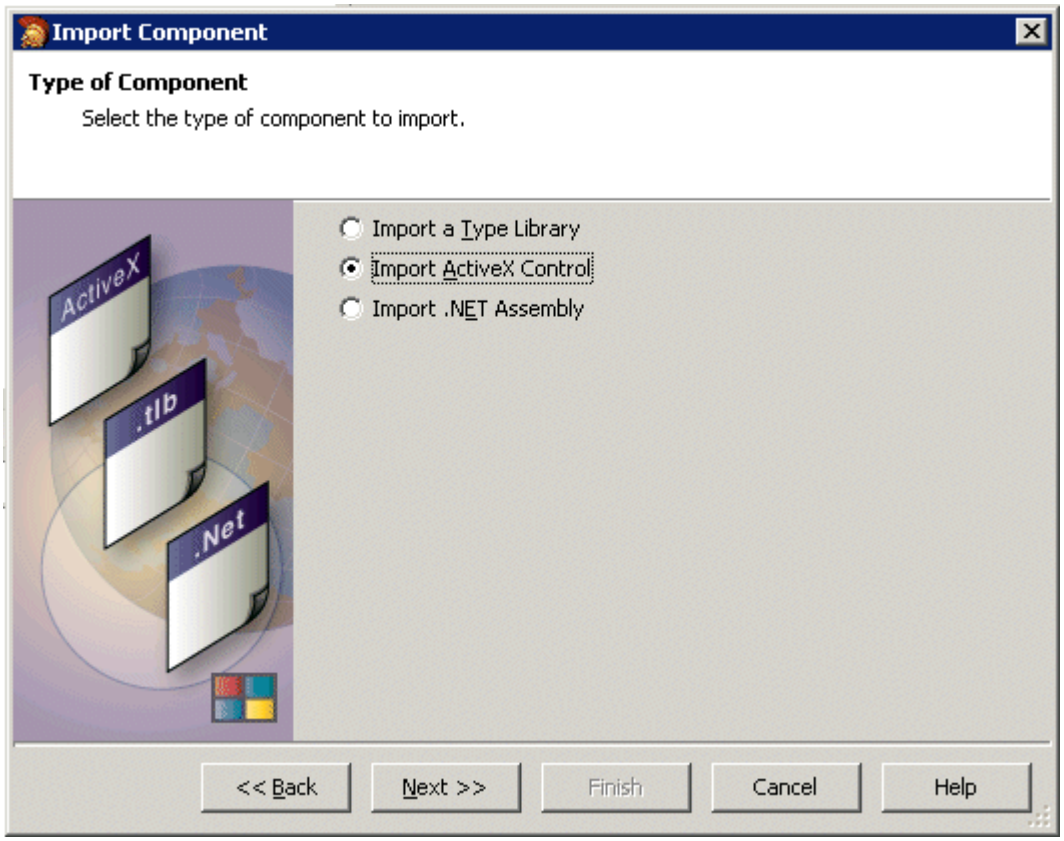
Schritt 2

Es öffnet sich der Komponenten-Wizard. Nach der Anwahl von *"VCL for Delphi Win32"* bestätigen Sie mit *"Weiter>>"*.



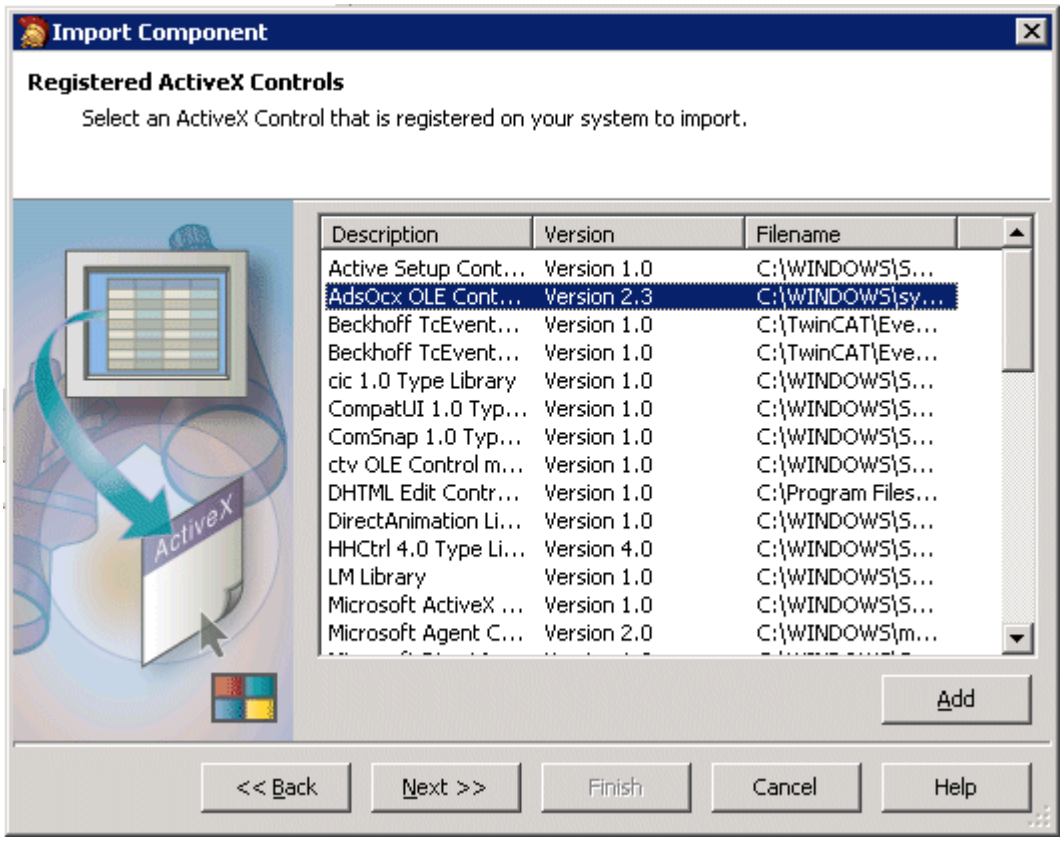
Schritt 3

Im nächsten Dialog wählen Sie *"ActiveX-Steuerelement importieren"* und klicken auf *"Weiter>>"*.



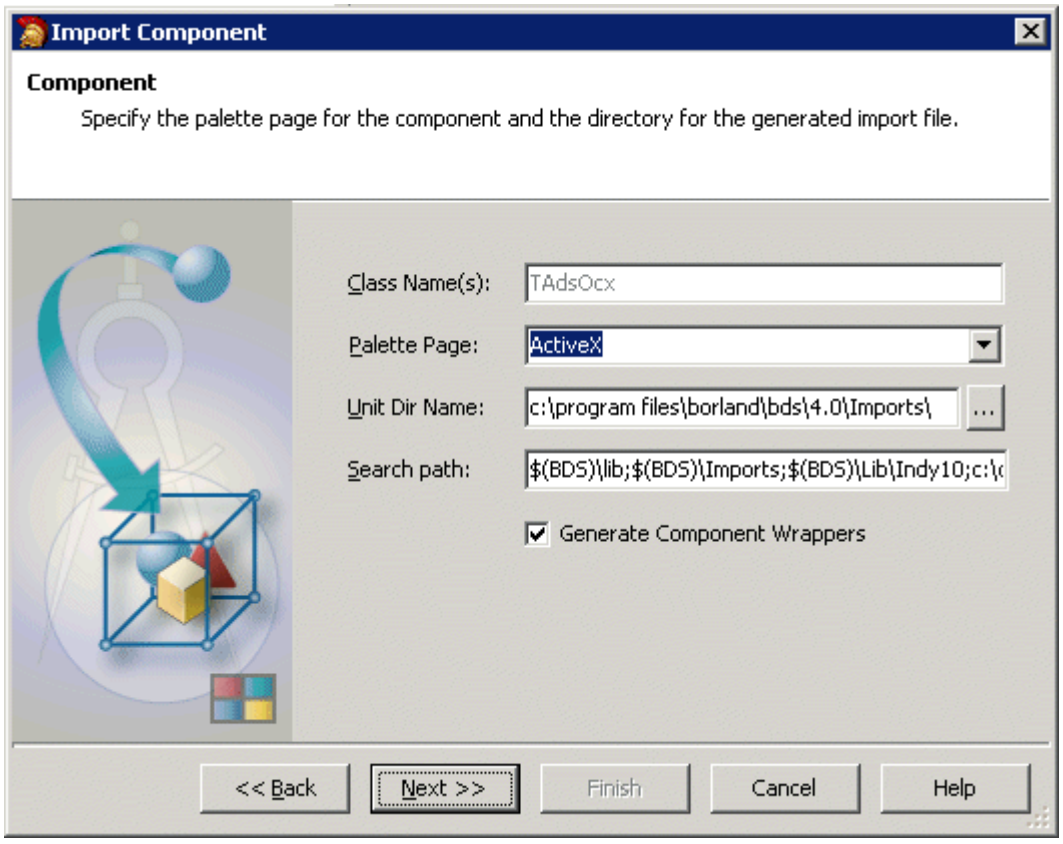
Schritt 4

Aus der Liste der registrierten ActiveX-Steuerelemente wählen Sie nun die entsprechende Komponente aus (AdsOcx OLE Control Module). Falls die Komponente in der Liste der registrierten Elemente nicht erscheint, muss sie über den Button *"Hinzufügen"* registriert und eingebunden werden. Anschließend klicken Sie auf *"Weiter"*.



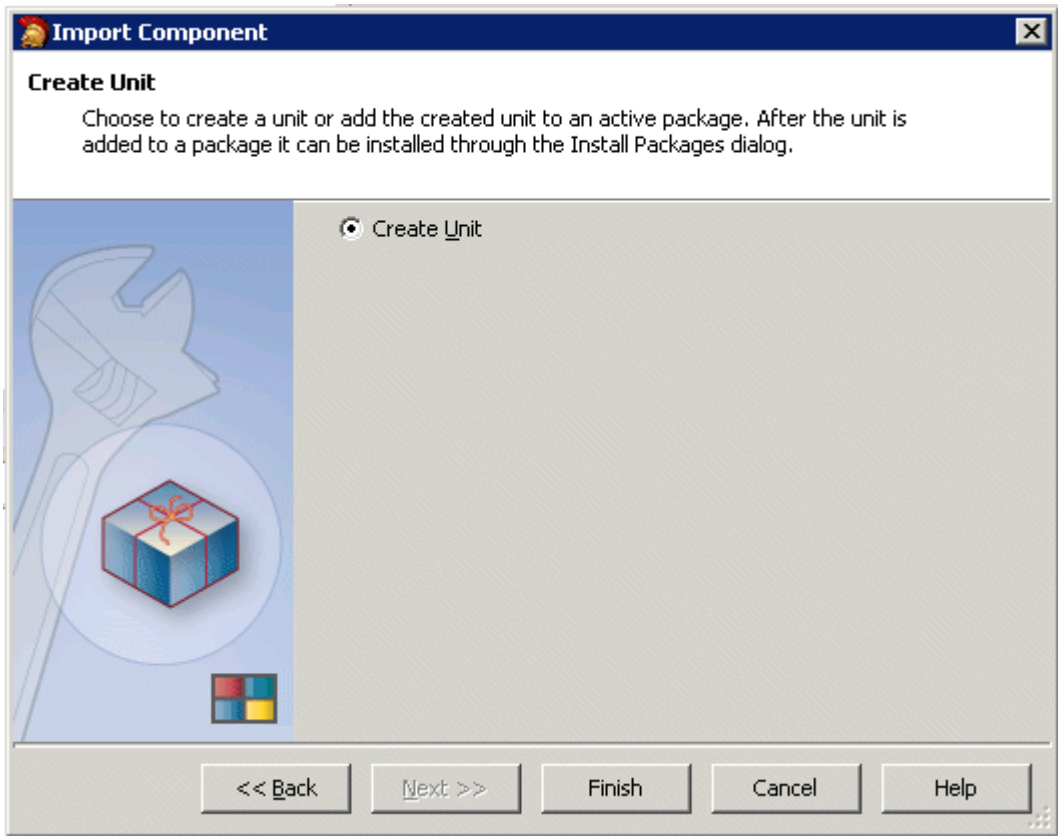
Schritt 5

Im nächsten Wizard-Fenster wählen Sie die VCL Palettenseite und das Verzeichnis für das neu erstellte Unit (Standardeinstellung: *C:\program files\borland\bds\4.0\Imports*).
Bestätigen Sie mit "Weiter".

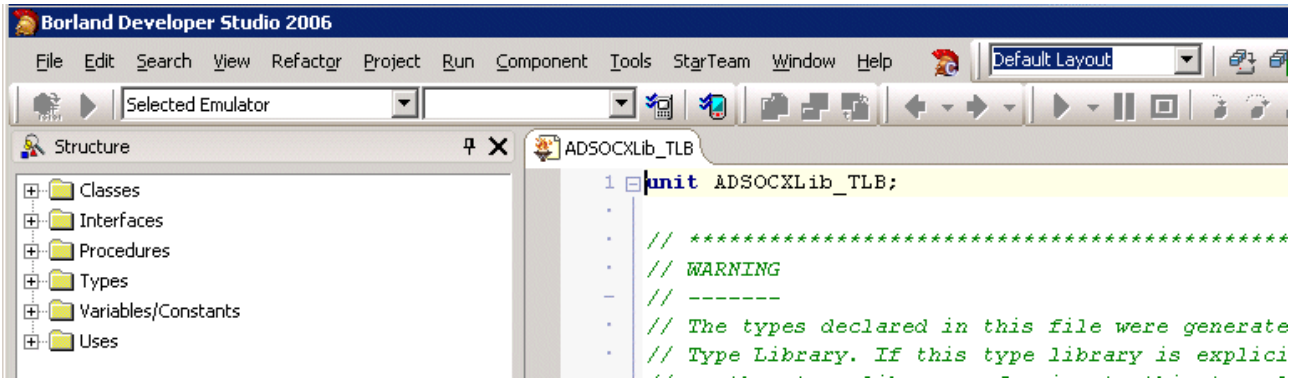


Schritt 6

Anschließend wird ein Unit für die ActiveX-Komponente generiert. Dazu müssen Sie mit "Beenden" bestätigen.

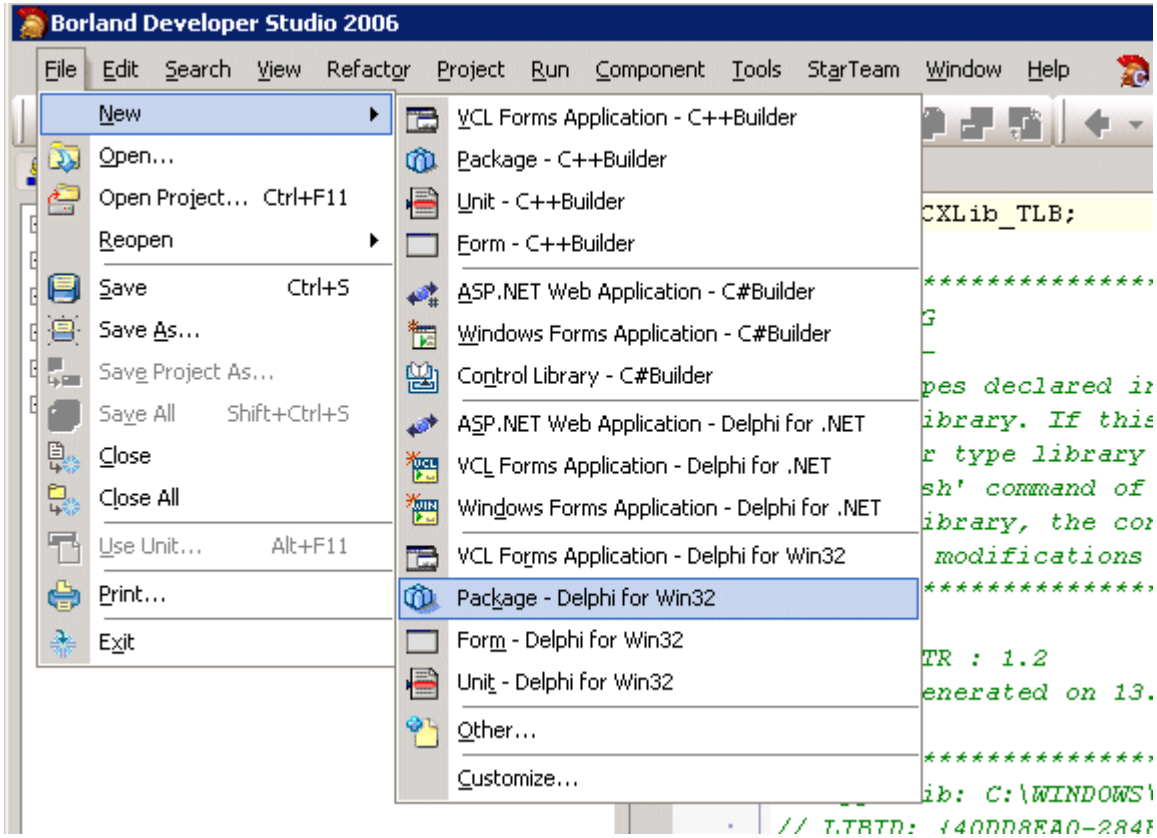


Das generierte Unit wird zur Kontrolle automatisch geöffnet:



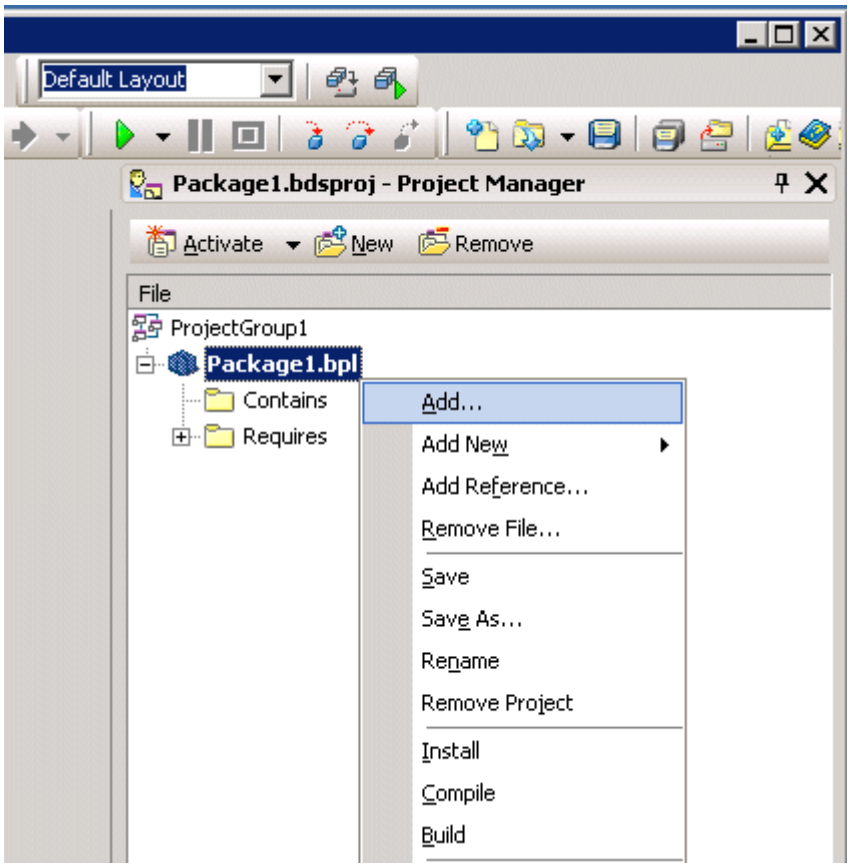
Schritt 7

Im nächsten Schritt muss ein neues Package erzeugt werden. Dazu klicken Sie auf *"Datei > Neu > Package"* im Hauptmenü.



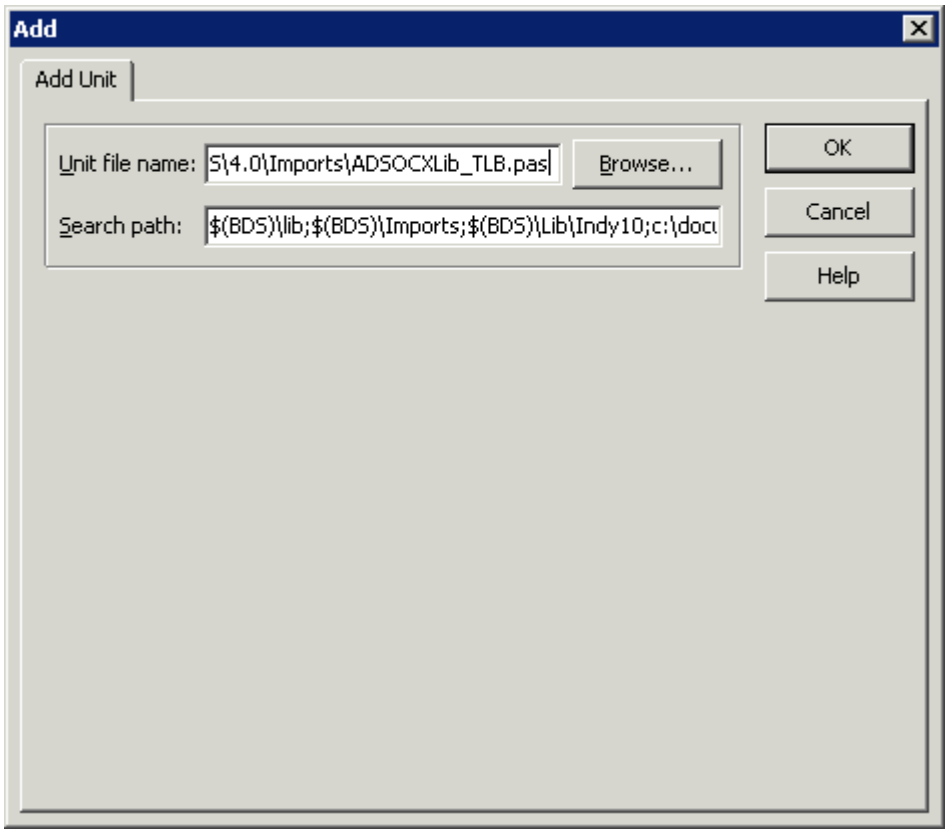
Schritt 8

In das neu angelegte Package muss nun das zuvor generierte Unit eingefügt werden.
Wählen Sie mit einem Klick der rechten Maustaste im Projektmanager den Eintrag *"Package1.bpl"* und im sich öffnenden Kontextmenü *"Hinzufügen"*.



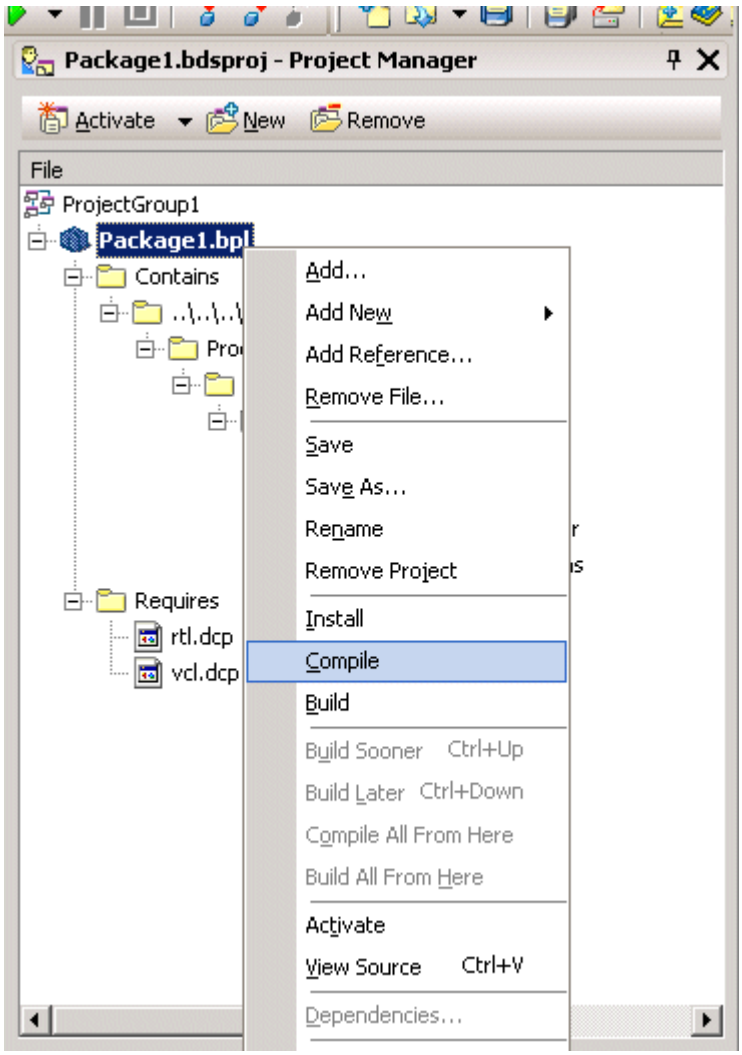
Schritt 9

Im Fenster *"Hinzufügen"* geben Sie den Speicherort der Unit an, der vorher für die ActiveX-Komponente generiert wurde (Standardeinstellung: *C:\program files\borland\bds\4.0\Imports\ADSOCXLib_TLB.pas*).



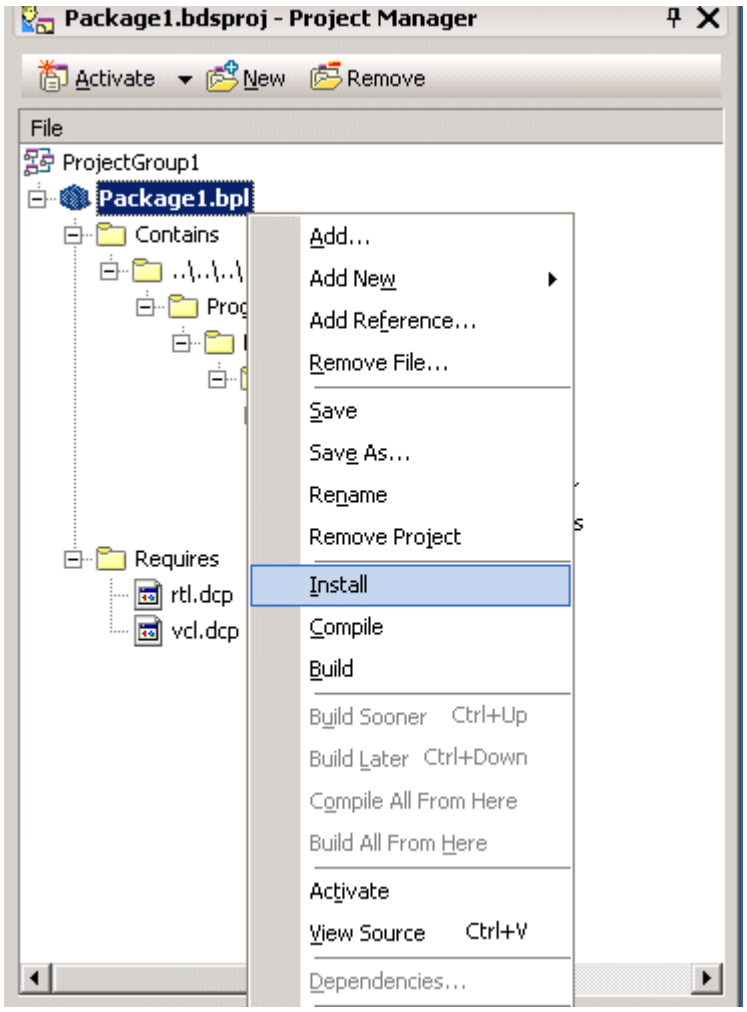
Schritt 10

Um das Package kompilieren zu können, gehen Sie mit der rechten Maustaste auf *"Package1.bpl"* und wählen im Kontextmenü *"Compilieren"*.

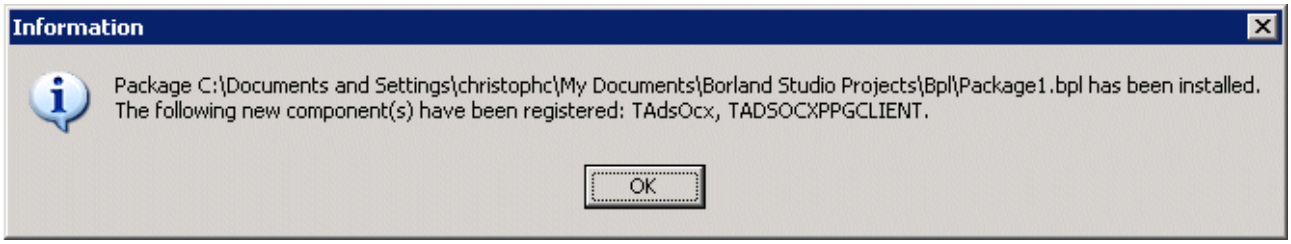


Schritt 11

Nach dem Compilieren des neuen Package wählen Sie im Kontextmenü *"Installieren"*.

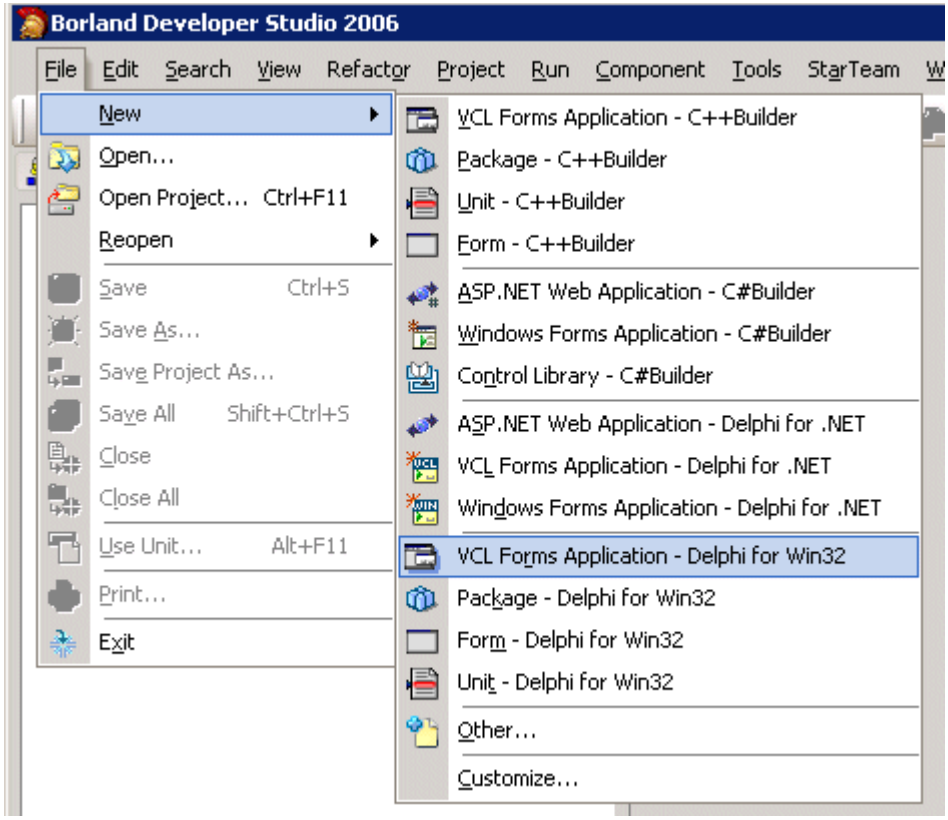


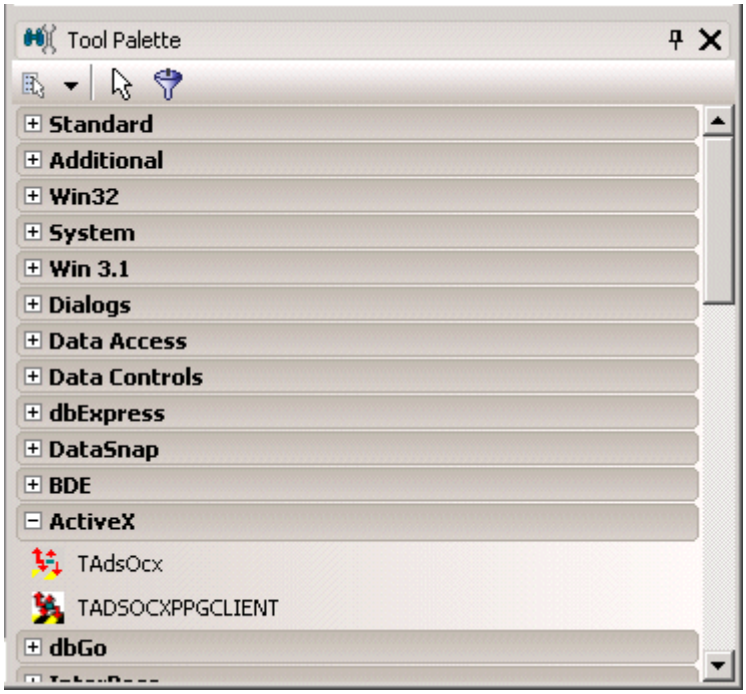
Die Installation ist damit beendet. Die folgende Meldung erscheint:



Nach der Installation

Die ActiveX-Komponente erscheint in der festgelegten Kategorie, wenn Sie z. B. eine neue *"VCL Forms Application - Delphi for Win32"* erstellen.





5.2.1.2 Einbinden in Delphi 3,4,5,6,7, ... (classic)

Bitte beachten Sie die Hinweise über die Einschränkungen und Limitierungen [▶ 106] beim Einsatz vom AdsOcx in Delphi-Anwendungen.

ActiveX-Controls können auf zweifache Weise in Delphi eingebunden werden:

1. Einbinden über den Import des ActiveX-Controls [▶ 100]

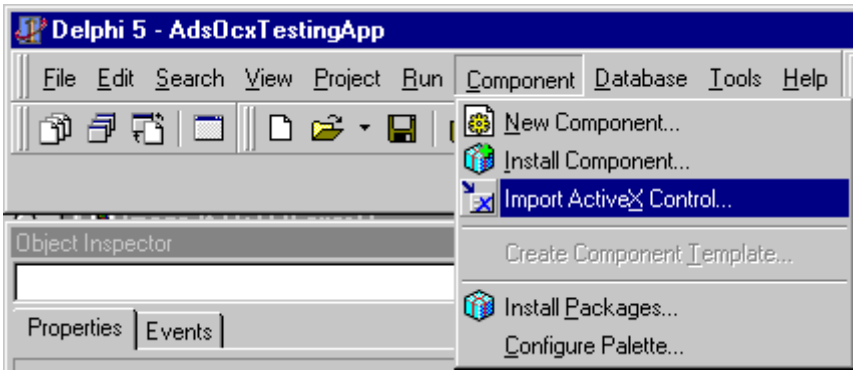
2. Einbinden über den Import der Typ-Bibliothek des ActiveX-Controls [▶ 102]

Bei den älteren Versionen von Delphi müssen Sie noch:

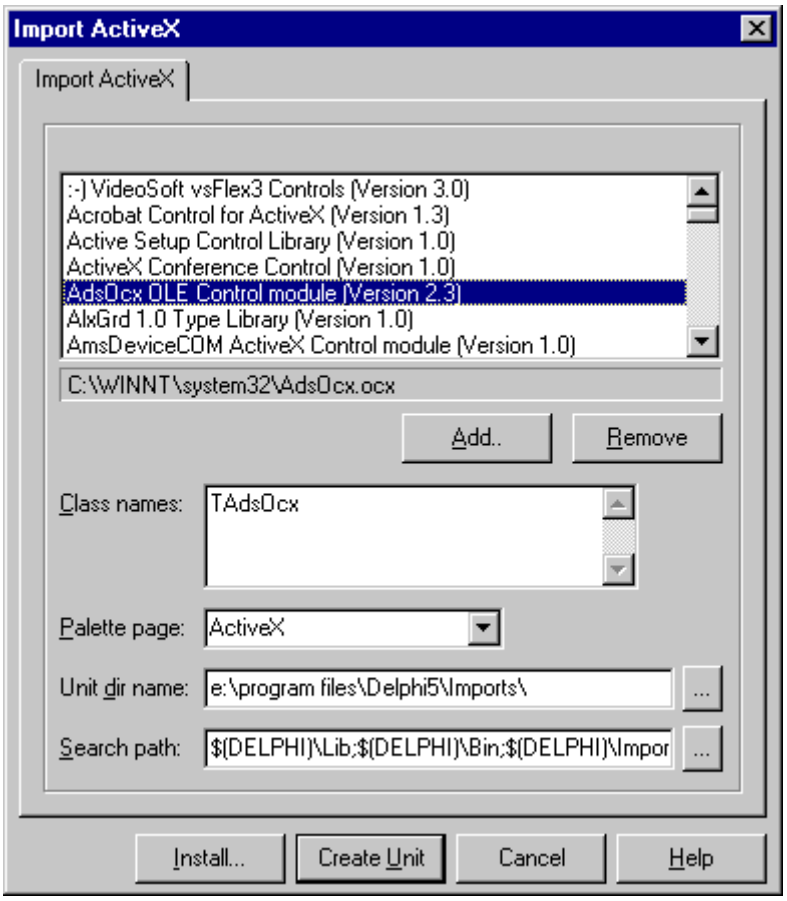
3. AdsOcx über die generierte Typ-Bibliothek in die Komponentenpalette installieren [▶ 104]

1. Einbinden über den Import des ActiveX-Controls

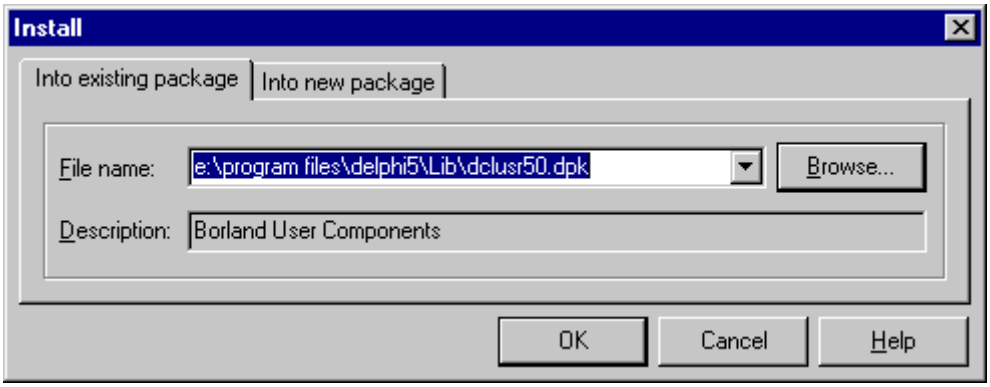
1.1 Über den Menübefehl *Component->Import ActiveX Control* das Import ActiveX-Dialogfenster aufrufen.



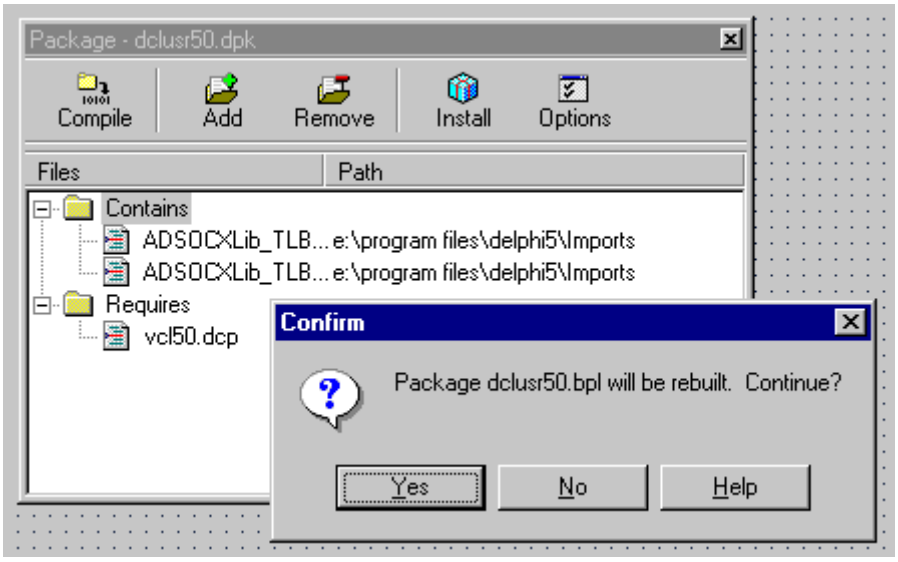
1.2 In dem Dialogfenster markieren Sie aus der Liste der ActiveX-Controls das *AdsOcx OLE Control module* und bestätigen mit einem Mausklick auf *Install....* Befindet sich das AdsOcx-Control nicht in der Liste, dann können Sie es über den Befehl *Add...* in die Liste hinzufügen. Das AdsOcx befindet sich standardmäßig in dem *.../WinNT/System32* - Ordner.



1.3 In dem Install-Dialogfenster mit OK bestätigen.

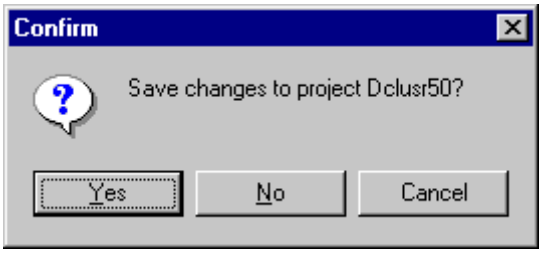


1.4 Das Package mit Benutzerdefinierten-Komponenten muss neu kompiliert werden. Bestätigen Sie mit *Yes*.

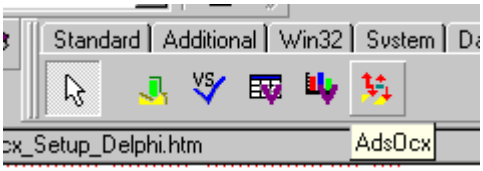


1.5 Bei Erfolg wird die AdsOcx-Komponente registriert. Bestätigen Sie mit *OK*.

1.6 Schließen Sie den Package-Editor und speichern Sie die Änderungen mit *Yes*.



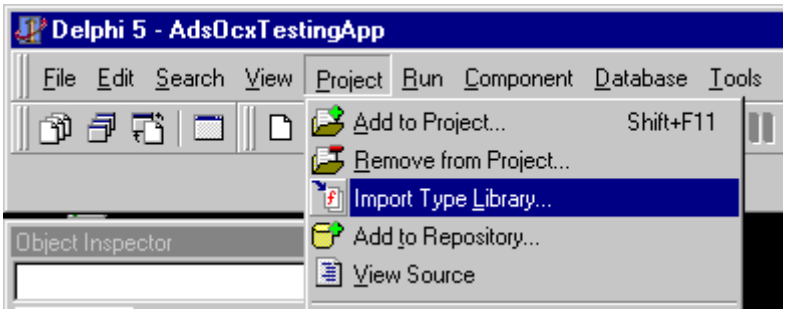
Ab jetzt können Sie die AdsOcx-Komponente aus der ActiveX-Komponentenpalette in einem neuen Projekt benutzen.



2. Einbinden über den Import der Typ-Bibliothek des ActiveX-Controls

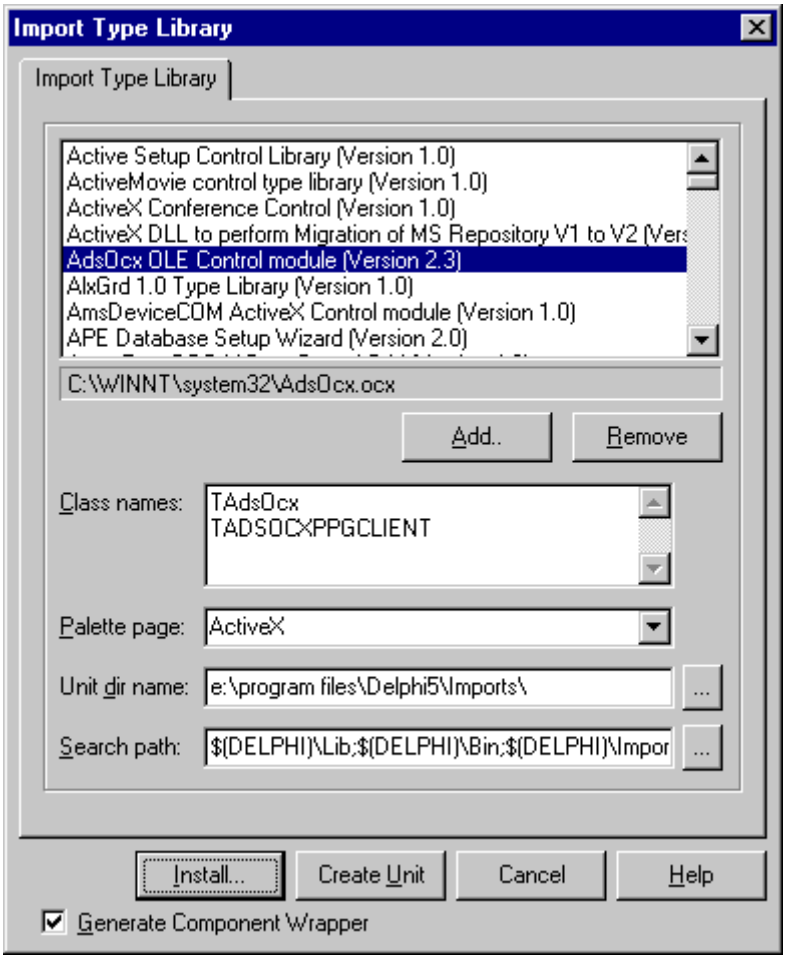
2.1 Um das AdsOcx in der Komponenten-Palette von Delphi einbinden zu können, muss zuerst eine Typ-Bibliothek (mit den Prototypen der Funktionen, Prozeduren und Datentypdefinitionen des ActiveX-Controls) generiert werden.

Die Typ-Bibliothek kann über das Menü *Project -> Import Type Library* generiert werden.

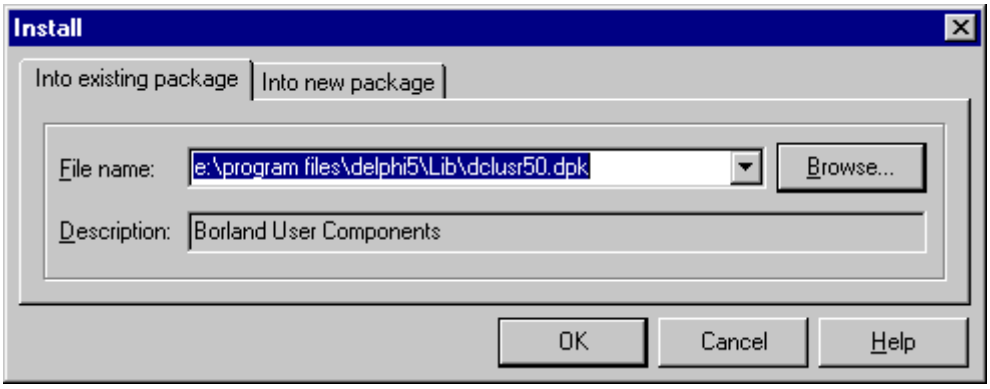


2.2 In dem danach folgendem Dialogfenster markieren Sie aus der Liste der ActiveX-Controls das *AdsOcx OLE Control module* und bestätigen mit *Install...*.

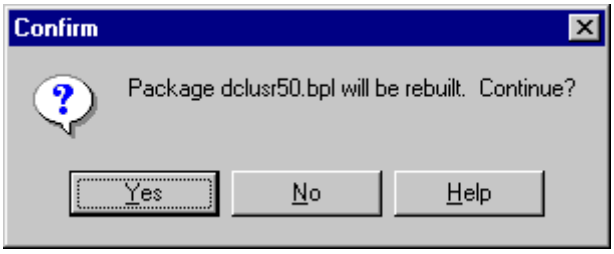
Befindet sich das AdsOcx nicht in der Auswahlliste, dann können Sie es über den Befehl *Add...* in die Liste hinzufügen. Das AdsOcx befindet sich standardmäßig in dem Ordner *.../WinNT/System32* und wird bei der TwinCAT-Installation dorthin kopiert.



2.3 In den neueren Versionen von Delphi (z. B. Delphi 5.0) wird die importierte Typ-Bibliothek sofort in die Komponentenpalette hinzugefügt. Bei den älteren Versionen wird nur die Typ-Bibliothek (z. B. in dem Ordner .../Delphi 3/Imports) generiert und Sie müssen das AdsOcx über die generierte Typ-Bibliothek in die Komponentenpalette installieren [▶ 104].. Haben Sie eine neuere Version, bestätigen Sie in dem nachfolgendem Dialog mit *OK*.

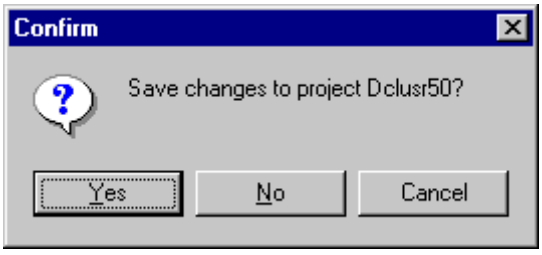


2.4 Das Package mit Benutzerdefinierten-Komponenten muss neu kompiliert werden. Bestätigen Sie mit *Yes*.

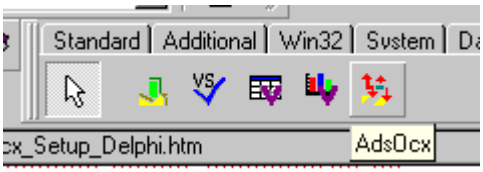


2.5 Beim Erfolg wird die AdsOcx-Komponente registriert. Bestätigen Sie mit *OK*.

2.6 Schließen Sie den Package-Editor und speichern Sie die Änderungen mit *Yes*.

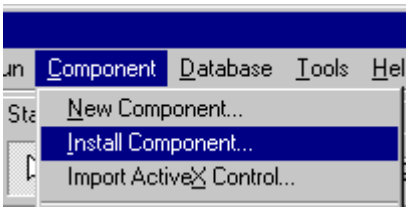


Ab jetzt können Sie die AdsOcx-Komponente aus der ActiveX-Komponentenpalette in einem neuen Projekt benutzen.

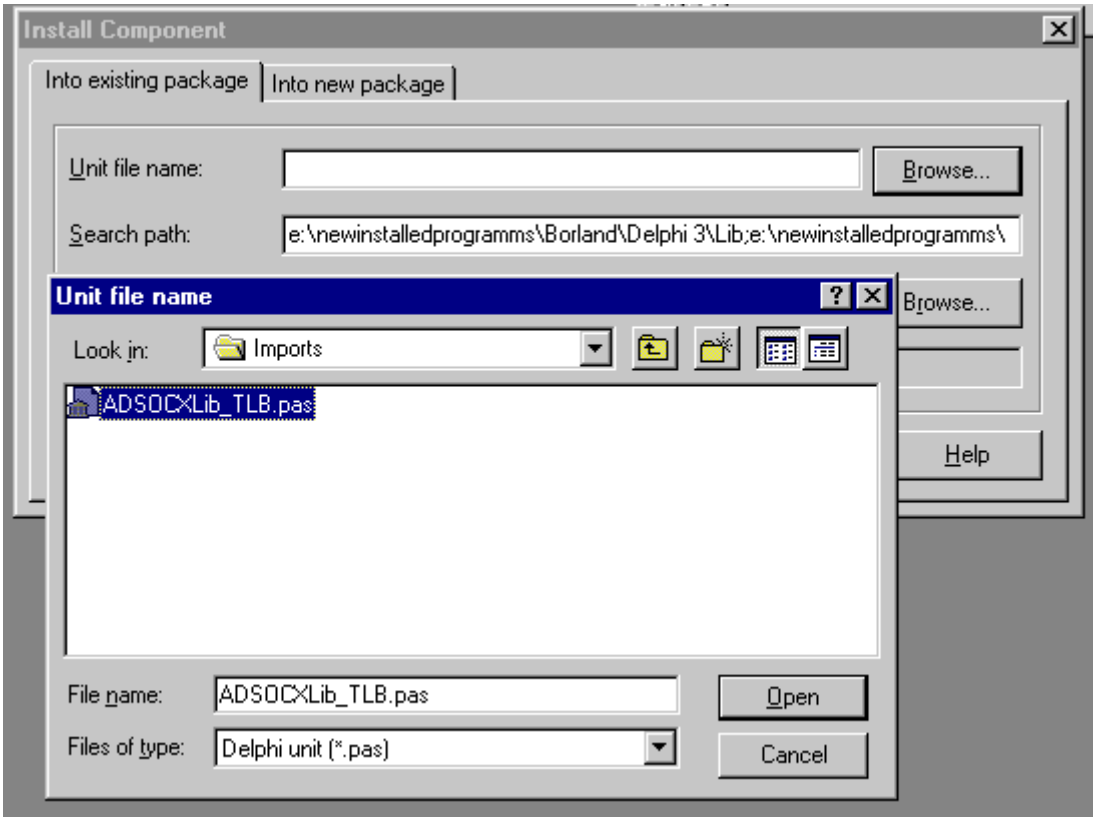


3. AdsOcx über die generierte Typ-Bibliothek in die Komponentenpalette installieren

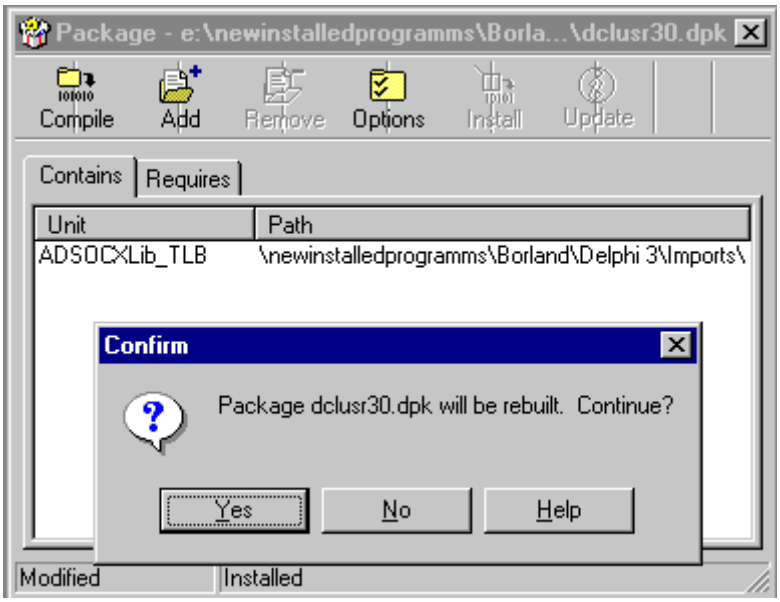
3.1 Nachdem die Typ-Bibliothek generiert wurde, kann aus der dabei erzeugten Pascal-Datei (standardmäßig wird die Datei ADSOCXLib_TLB.pas erzeugt) das AdsOcx als neue Komponente in die Komponenten-Palette hinzugefügt werden. Dazu müssen Sie den Menübefehl: *Component->Install Component...* auswählen.



3.2 In dem Dialogfenster muss über den Befehl *Browse...* die vorher erstellte Typ-Bibliothek ausgewählt werden. Die generierten Typ-Bibliotheken befinden sich standardmäßig in dem *.../Delphi 3/Imports/* Ordner. Wählen Sie die Typ-Bibliothek aus und bestätigen mit *Open*.

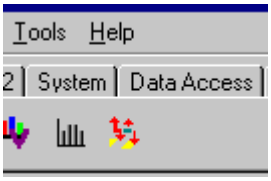


3.3 Die Komponenten-Palette muss anschließend neu Compiliert werden. Bestätigen Sie mit *Yes*.



3.4 Nach einer erfogreichen Compilierung wird das ActiveX-Control registriert. Bestätigen Sie mit *OK*.

3.5 Die Änderungen in dem Komponenten-Package müssen beim Schließen gespeichert werden. Das AdsOcx-ActiveX-Control kann jetzt von der Komponenten-Palette auf die Form gezogen und ähnlich wie alle anderen Delphi-Komponenten benutzt werden.



5.2.1.3 ADS-OCX Einschränkungen in Delphi-Anwendungen

Delphi's Memory Manager

In der AdsOcx Applikation müssen Sie sicherstellen daß die Systemvariable: **IsMultiThread** auf jeden Fall auf True gesetzt ist. Der Memory Manager ist "thread-safe" nur wenn diese Variable gesetzt ist. Nur dann werden auch die Zugriffe auf gemeinsame Resourcen verriegelt. Oft setzt der Memory Manager von Delphi diese Variable nicht wenn eine eingebundene DLL oder ein Control eigene Threads startet.

Fügen Sie folgende Zeile in die initialization section ihrer Applikation:

```
Initialization
    IsMultiThread := True;// Setting this system variable makes Delphi's memory manager thread-safe
```

Methoden/Eigenschaften

Folgende Eigenschaften, Methoden und Ereignisse verursachen Fehler in Delphi-Applikationen und dürfen nicht benutzt werden. Wie aus der Tabelle zu sehen ist, sollte die neueste Version von Delphi benutzt werden oder es muss auf einige Funktionalitäten verzichtet werden. Zu den aufgeführten Delphi-Versionen gibt es verschiedene Updates, die möglicherweise einige Fehler beheben.

	Fehlerbeschreibung	Workaround	Delphi-Version			
Eigenschaften			**3.0**	**4.0**	**5.0**	**6.0**
AdsClientType	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsClientAdsState	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsClientAdsControl	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	

	Fehlerbeschreibung	Workaround	Delphi-Version			
AdsServerAdsControl	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsServerAdsState	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsServerType	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsServerLastMessage	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsAmsClientNetId	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
AdsAmsServerNetId	Beim Zugriff auf die Eigenschaft entsteht **Memory-Leak.** Speicher für den zurückgelieferten String wird nicht korrekt zurückgegeben.	n/a	Bug	?	Fixed	
Methoden						
Alle Methoden	Die Funktionen der generierten Typbibliothek ADSOCXLib_TLB liefern undefinierte Rückgabeparameter.	Bitte installieren Sie den Delpphi 6 **Update Pack 2** und binden Sie das ADSOCX neu ein.	-	-	**kein Bug**	Bug

	Fehlerbeschreibung	Workaround	Delphi-Version			
AdsSyncReadReq *AdsSyncWriteReq*	Mit diesen Methoden können Variablen vom beliebigen Typ in die SPS übertragen oder aus der SPS gelesen werden. Die OleVariant-Parameter werden aber bei der *AdsSyncReadReq*-Methode per Wert und nicht per Reference übergeben. D.h. während des Aufrufs kann der Wert des *data*-Parameters von der Methode nicht geändert werden. Während des Aufrufs werden die SPS-Variablen zwar in eine entsprechende OleVariant-Variable hineinkopiert, die Variable ist aber nur eine Kopie der Variablen aus der aktuellen Parameterliste. Die Methoden-Prototypen für das ADS-OCX werden von der Delphi-Entwicklungsumgebung beim Einbinden des ADS-OCX automatisch generiert und können nicht beinflußt werden.	**Benutzen Sie die "aufgelösten" Methoden** um auf die SPS-Variablen synchron zuzugreifen (z. B. AdsSyncReadIntegerReq() u.s.w.).	Bug	?	Bug	
AdsReadVarConnectEx	Ähnlich wie bei den AdsSync-Methoden, werden die OleVariant-Parameter in den Ereignisfunktionen per Wert und nicht per Referenz übergeben.	Benutzen Sie die Methode **AdsReadVarConnectEx2**	Bug	?	Bug	
Ereignisse						
AdsReadConnectUpdateEx	Beim Aufrufen der Ereignisfunktion wird eine **Zugriffsverletzung** generiert.	Benutzen Sie die Ereignisfunktion **AdsReadConnectUpdateEx2**	Bug	?	Bug	

5.2.1.4 ADS-OCX-Applikation zurücksetzen

Nach einem Programmfehler kann oft nicht, wie sonst üblich, über die Bedienfunktion "Start -> Programm Reset" die Applikation beendet werden. Folgende Meldung des Debuggers ist die Folge:

Die Ursache liegt darin, dass bei der Benutzung des ADS-OCX eine Client-Server Verbindung zum TwinCAT Router erzeugt wird, die beim Beenden der Applikation abgebaut werden, muss. Die Delphi-Applikation kann über den Menübefehl "Programm Reset" nicht beendet werden, weil zu diesem Zeitpunkt eine Verbindung zum TwinCAT Router besteht. In der Applikation wird die Verbindung durch Zuweisung der AdsAmsNetId und der Portnummer erzeugt.

Es gibt folgende Möglichkeiten, die Applikation zu beenden, ohne den Rechner neu starten zu müssen:

- Zuerst den Laufzeitfehler mit OK bestätigen, dann TwinCAT System Stop über die Taskleiste aufrufen und dann Delphi-Applikation zurücksetzen. Es werden dabei bestehende Verbindungen zu den Clients abgebaut. Nachteil: Das TwinCAT System und die SPS muss danach neu gestartet werden;
- Zuerst den Laufzeitfehler mit OK bestätigen und Router-Cleanup über die Taskleiste aufrufen und dann Delphi-Applikation zurücksetzen;
- Exception-Handling benutzen. Über die Methode AdsAmsDisconnect() kann die Verbindung zum Router explizit abgebaut werden;

  try n:=8; Switch[n].Tag:=0; // Dieser Index ist unzulässig except on EAccessViolation do begin AdsOcxSPS.AdsAmsDisconnect(); Application.Terminate(); end; end;

5.2.2 Synchron/Asynchron/Connected auf SPS-Variablen zugreifen

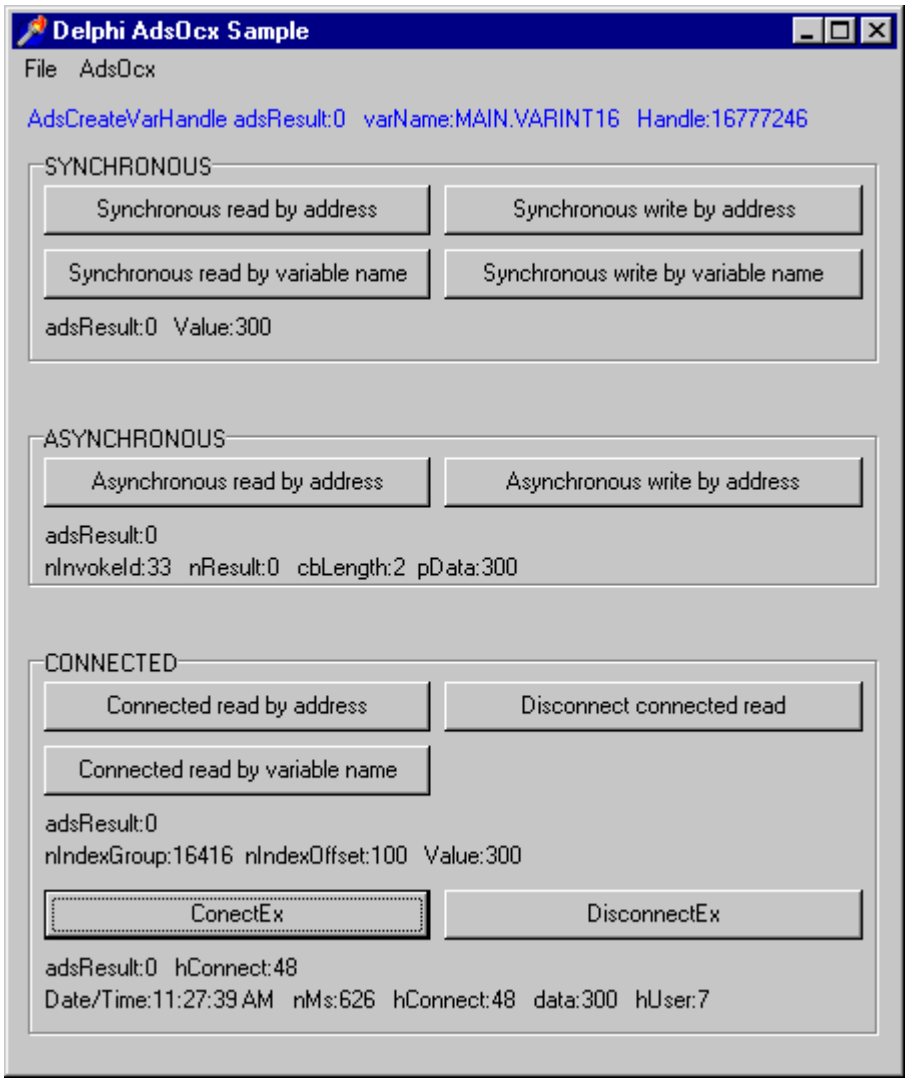
Systemvoraussetzungen:

- Delphi 5.0 oder höher;
- TwinCAT v2.9 oder höher;

Aufgabe

Das Beispielprogramm zeigt, wie Methoden und Events des AdsOcx in einer Delphi-Anwendung benutzt werden können. Dabei werden die unterschiedlichsten Zugriffsarten (Synchron/Asynchron/Connected) auf die SPS-Variablen vorgestellt. In dem SPS-Programm wurde eine Integer-Variable auf der Adresse 100 im Merkerbereich der Prozessdaten der SPS definiert. Auf die SPS-Variable soll über die unterschiedlichsten Zugriffsarten schreibend oder lesend aus der Delphi-Applikation zugegriffen werden.

Beschreibung



Auf die SPS-Variablen ist über das AdsOcx ein Synchroner-, Asynchroner- oder Connected-Zugriff möglich. Bei einem synchronen Zugriff wird die Applikation so lange angehalten, bis die angeforderten Daten vorliegen. Bei einem asynchronen Zugriff wird eine Anforderung (Request) an die SPS gesendet und die Ausführung der Windows-Applikation fortgesetzt. In der Windows-Applikation wird dann eine Callback-Funktion aufgerufen, wenn die angeforderten Daten vorliegen. Bei dem Connected-Zugriff kann in der Windows-Applikation eine Event-Funktion aufgerufen werden, wenn sich der Wert der SPS-Variablen geändert hat.

Delphi 5 Programm

In der Event-Funktion *OnFormCreate* wird über die Methode AdsCreateVarHandle [▶ 14] ein Handle für den Sysmbolnamen der SPS-Variablen angefordert. Das Handle wird dann in der Beispielapplikation für den schreibenden oder lesenden Zugriff auf die SPS-Variable benutzt. In der Event-Funktion *OnDestroy* wird das Handle beim Beenden der Applikation mit der Methode AdsDeleteVarHandle [▶ 15] freigegeben.

```
var
  Form1          : TForm1;
  varName        :WideString;     {PLC variable symbol name}
  varValue       :Smallint;       {PLC variable value}
  varHandle      :integer;        {PLC variable handle}
  hConnect       :integer;        {PLC variable connection handle}
  adsResult      :integer;        {Ads result}

implementation

{$R *.DFM}
```

```
procedure TForm1.OnFormCreate(Sender: TObject);
begin
     AdsOcx1.AdsAmsServerNetId := AdsOcx1.AdsAmsClientNetId;    {Sets PLC server network adress}
     AdsOcx1.AdsAmsServerPort := 801;               {Sets the PLC run time system}
     varName := 'MAIN.VARINT16';
     varValue := 0;
     varHandle := 0;
     hConnect := 0;
     adsResult := AdsOcx1.AdsCreateVarHandle( varName, varHandle ); {creates variable handle}

     if adsResult = 0 then
    LabelVarHandle.Font.Color := clBlue
     else
    LabelVarHandle.Font.Color := clRed;

     LabelVarHandle.Caption := Format( 'AdsCreateVarHandle adsResult:%d   varName:%s   Handle:%d',
[adsResult, varName, varHandle] );
end;

procedure TForm1.OnFormDestroy(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsDeleteVarHandle( varHandle );
end;
```

Synchroner Zugriff

Bei einem Mausklick auf einen der Buttons in der Gruppe SYNCHRONOUS wird der Wert der SPS-Variablen synchron gelesen bzw. geschrieben und als Text auf der Form ausgegeben. Es kann auf zweifache Weise auf die SPS-Variable zugegriffen werden: Über den Variablen-Namen oder über die Variablen-Adresse.

Zugriff über die Variablen-Adresse

```
procedure TForm1.OnSyncReadByAddrClick(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsSyncReadIntegerReq( $00004020, 100, 2, varValue );
     LabelSyncRetData.Caption:=Format( 'adsResult:%d   Value:%d',[adsResult, varValue] );
end;

procedure TForm1.OnSyncWriteByAddrClick(Sender: TObject);
begin
     varValue := 100;
     adsResult := AdsOcx1.AdsSyncWriteIntegerReq( $00004020, 100, 2, varValue );
     LabelSyncRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;
```

Zugriff über den Variablen-Namen

Bei einem Zugriff über den Variablen-Namen wird das entsprechende Handle der SPS-Variablen als Parameter in der Methode AdsSyncReadIntegerVarReq [▶ 24] bzw. AdsSyncWriteIntegerVarReq [▶ 29] benutzt. Das Handle der SPS-Variablen wurde in der *OnCreate* Event-Funktion beim Applikationsstart angefordert.

```
procedure TForm1.OnSyncReadByNameClick(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsSyncReadIntegerVarReq( varHandle, 2, varValue );
     LabelSyncRetData.Caption:=Format( 'adsResult:%d   Value:%d', [adsResult, varValue] );
end;

procedure TForm1.OnSyncWriteByNameClick(Sender: TObject);
begin
     varValue := 200;
     adsResult := AdsOcx1.AdsSyncWriteIntegerVarReq( varHandle, 2, varValue );
     LabelSyncRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;
```

Asynchroner Zugriff

Über die Methoden AdsReadIntegerReq [▶ 31] und AdsWriteIntegerReq [▶ 33] kann auf die SPS-Variable asynchron zugegriffen werden.

```
procedure TForm1.OnAsyncReadByAddrClick(Sender: TObject);
var  varInvokeId       :integer;
begin
     varInvokeId := 33;
     adsResult := AdsOcx1.AdsReadIntegerReq( varInvokeId, $00004020, 100, 2 );
     LabelAsyncRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;

procedure TForm1.OnAsyncWriteByAddrClick(Sender: TObject);
var  varInvokeId       :integer;
begin
     varInvokeId := 44;
     varValue := 300;
     adsResult := AdsOcx1.AdsWriteIntegerReq( varInvokeId, $00004020, 100, 2, varValue );
     LabelAsyncRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;
```

Die Ausführung der Delphi-Applikation wird nach einem asynchronen Zugriff fortgesetzt und beim Vorliegen der Rückgabeparameter eine Event-Funktion in der Windows-Applikation aufgerufen. In unserem Beispiel wird beim Lesen der SPS-Variablen die Event-Funktion AdsReadIntegerConf [▶ 56] und beim Schreiben der SPS-Variablen die Event-Funktion AdsWriteConf [▶ 59] aufgerufen.

```
procedure TForm1.AdsOcx1AdsReadIntegerConf(Sender: TObject; nInvokeId,
  nResult, cbLength: Integer; var pData: Smallint);
begin
     LabelAsyncEventData.Caption :=Format('nInvokeId:%d   nResult:%d   cbLength:%d  pData:%d',
               [nInvokeId, nResult, cbLength, pData]);
end;

procedure TForm1.AdsOcx1AdsWriteConf(Sender: TObject; nInvokeId,
  nResult: Integer);
begin
     LabelAsyncEventData.Caption :=Format('nInvokeId:%d   nResult:%d', [nInvokeId, nResult]);
end;
```

Connected Zugriff

Bei dem Connected-Zugriff wird eine "Verbindung" zu der SPS-Variablen aufgebaut. In Abhängigkeit von den Parametern (ADSTRANS_SERVERCYCLE oder ADSTRANS_SERVERONCHA) werden die Event-Funktionen zyklisch, oder bei einer Änderung der SPS-Variablen aufgerufen.

In der Beispielapplikation wird bei einem Mausklick auf den Button *Connected read by address* die Methode AdsReadIntegerConnect [▶ 42] aufgerufen und bei einem Mausklick auf den Button *Connected read by variable name* die Methode AdsReadIntegerVarConnect [▶ 39] aufgerufen.

```
procedure TForm1.OnConReadByAddrClick(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsReadIntegerConnect( $00004020, 100, 2, ADSTRANS_SERVERCYCLE, 220, varVa
lue );
     LabelConRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;

procedure TForm1.OnConReadByNameClick(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsReadIntegerVarConnect( varName, 2, ADSTRANS_SERVERCYCLE, 220, varValue
);
     LabelConRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;
```

Bei Erfolg wird in der Delphi-Applikation die Event-Funktion AdsReadConnectUpdate [▶ 53] aufgerufen, unabhängig davon welche der beiden Methoden für den Verbindungsaufbau benutzt wurde.

```
procedure TForm1.AdsOcx1AdsReadConnectUpdate(Sender: TObject; nIndexGroup,
  nIndexOffset: Integer);
begin
     LabelConEventData.Caption := Format('nIndexGroup:%d  nIndexOffset:%d   Value:%d',
               [nIndexGroup, nIndexOffset, varValue]);
end;
```

Über die Methode AdsReadIntegerDisconnect [▶ 44] kann die Verbindung zu der SPS-Variablen abgebaut werden.

```
procedure TForm1.OnDisconnectReadClick(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsReadIntegerDisconnect( varValue );
     LabelConRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;
```

ConnectEx-Methoden (Connected-Zugriff mit einem Benutzer-Handle)

Mit der Hilfe der ConnectEx-Methoden kann ähnlich wie bei den Connect-Methoden ein Connected-Zugriff zu den SPS-Variablen aufgebaut werden. Die ConnectEx-Methoden haben den Vorteil, dass bei dem Verbindungs-Aufbau ein benutzerdefiniertes Handle als Parameter in der Connect-Methode übergeben werden kann. Dieses Handle kann dann in der Event-Funktion ausgewertet und dazu benutzt werden, um die SPS-Variablen zu identifizieren, für die die Event-Funktion aufgerufen wurde.

Bei einem Mausklick auf den Button *ConnectEx* wird in der *OnConnectExClick*-Routine die Methode AdsReadVarConnectEx2 [▶ 35] aufgerufen.

```
procedure TForm1.OnConectExClick(Sender: TObject);
var  hUser :integer;
begin
     {disconnect old connection}
     if hConnect <> 0 then
     begin
    adsResult := AdsOcx1.AdsDisconnectEx( hConnect );
    if adsResult = 0 then
        hConnect := 0;
     end;

     hUser := 7;    {create user handle}

     adsResult := AdsOcx1.AdsReadVarConnectEx2( varName, ADSTRANS_SERVERCYCLE, 220, hConnect, hUser
);
     LabelConExRetData.Caption:=Format( 'adsResult:%d   hConnect:%d', [adsResult, hConnect] );
end;
```

Wurde die Verbindung erfolgreich aufgebaut, dann werden in der Event-Funktion AdsReadConnectUpdateEx2 [▶ 55] die Parameter als Text auf der Form ausgegeben.

```
procedure TForm1.AdsOcx1AdsReadConnectUpdateEx2(Sender: TObject;
  dateTime: TDateTime; nMs, hConnect: Integer; var data,
  hUser: OleVariant);
begin
     LabelConExEventData.Caption :=Format('Date/Time:%s   nMs:%d   hConnect:%d   data:%d   hUser:
%d',
                [ TimeToStr(dateTime), nMs, hConnect, integer(data), integer(hUser)]);
end;
```

Bei einem Mausklick auf den Button *DisconnectEx* wird die Methode AdsDisconnectEx [▶ 40] aufgerufen und die Verbindung zu der SPS-Variablen abgebaut.

```
procedure TForm1.OnDisconnectExClick(Sender: TObject);
begin
     adsResult := AdsOcx1.AdsDisconnectEx( hConnect );
     if adsResult = 0 then
     hConnect := 0;

     LabelConExRetData.Caption:=Format( 'adsResult:%d', [adsResult] );
end;
```

Bemerkung:

Bei der Einbindug des ADS-OCX in Delphi-Applikationen wurde festgestellt, dass von der Entwicklungsumgebung von Delphi fehlerhafte Prototypen (genauer: fehlerhafte Parameterübergabe bei den OleVariant-Typen) der Event-Funktion AdsReadConnectUpdateEx [▶ 54] generiert wurde. Aus diesem Grund wurde das ADS-OCX um eine neue Methode AdsReadVarConnectEx2 und eine dazugehörende Event-Funktion AdsReadConnectUpdateEx2 ergänzt. In der neuen Event-Funktion werden die OleVariant-Parameter per Referenz statt per Wert übergeben.

Other

```
procedure TForm1.Exit1Click(Sender: TObject);
begin
    Close();
end;

procedure TForm1.Properties1Click(Sender: TObject);
begin
    AdsOcx1.BrowseProperties();
end;

procedure TForm1.About1Click(Sender: TObject);
begin
    AdsOcx1.AboutBox();
end;

Initialization
    IsMultiThread := True;// Setting this system variable makes Delphi's memory manager thread-safe
```

SPS Programm

```
PROGRAM MAIN
VAR
    VARINT16    AT%MB100:INT;
END_VAR
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466730763.exe
Delphi 5 oder höher (classic)	

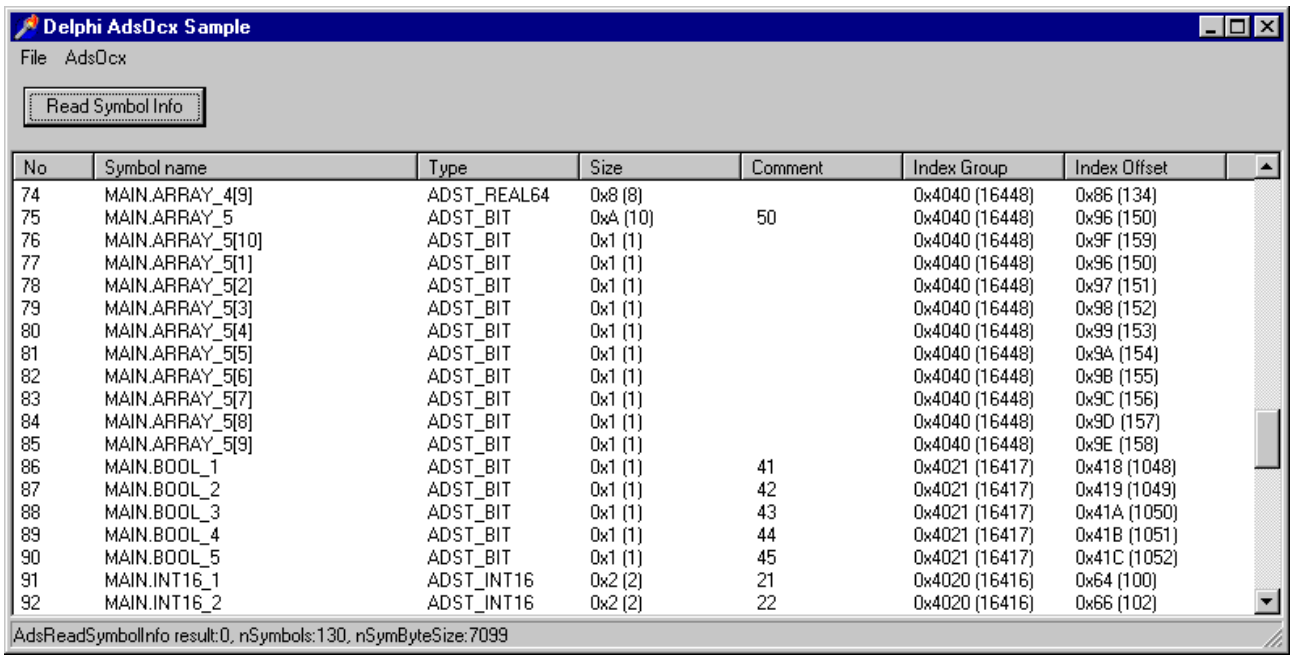
5.2.3 Liste der deklarierten Variablen eines ADS-Gerätes auslesen

Systemvoraussetzungen:

- Delphi 5.0 oder höher;
- TwinCAT v2.9 oder höher;

Aufgabe

Das Beispielprogramm zeigt, wie mit den Methoden: AdsReadSymbolInfo [▶ 21] und AdsEnumSymbols [▶ 16] die Liste der deklarierten Variablen eines ADS-Gerätes ausgelesen werden kann. Bei einem Mausklick auf den *Read Symbol Info*-Button sollen die Symbolinformationen des ersten SPS-Laufzeitsystems (Port 801) oder einer zusätzlichen Task im TwinCAT System Manager (Port 301) gelesen und in einer Tabelle dargestellt werden.



No	Symbol name	Type	Size	Comment	Index Group	Index Offset
74	MAIN.ARRAY_4[9]	ADST_REAL64	0x8 (8)		0x4040 (16448)	0x86 (134)
75	MAIN.ARRAY_5	ADST_BIT	0xA (10)	50	0x4040 (16448)	0x96 (150)
76	MAIN.ARRAY_5[10]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x9F (159)
77	MAIN.ARRAY_5[1]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x96 (150)
78	MAIN.ARRAY_5[2]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x97 (151)
79	MAIN.ARRAY_5[3]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x98 (152)
80	MAIN.ARRAY_5[4]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x99 (153)
81	MAIN.ARRAY_5[5]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x9A (154)
82	MAIN.ARRAY_5[6]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x9B (155)
83	MAIN.ARRAY_5[7]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x9C (156)
84	MAIN.ARRAY_5[8]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x9D (157)
85	MAIN.ARRAY_5[9]	ADST_BIT	0x1 (1)		0x4040 (16448)	0x9E (158)
86	MAIN.BOOL_1	ADST_BIT	0x1 (1)	41	0x4021 (16417)	0x418 (1048)
87	MAIN.BOOL_2	ADST_BIT	0x1 (1)	42	0x4021 (16417)	0x419 (1049)
88	MAIN.BOOL_3	ADST_BIT	0x1 (1)	43	0x4021 (16417)	0x41A (1050)
89	MAIN.BOOL_4	ADST_BIT	0x1 (1)	44	0x4021 (16417)	0x41B (1051)
90	MAIN.BOOL_5	ADST_BIT	0x1 (1)	45	0x4021 (16417)	0x41C (1052)
91	MAIN.INT16_1	ADST_INT16	0x2 (2)	21	0x4020 (16416)	0x64 (100)
92	MAIN.INT16_2	ADST_INT16	0x2 (2)	22	0x4020 (16416)	0x66 (102)

Beschreibung

Symbolkonfiguration für das SPS-Laufzeitsystem

Um auf die Symbolinformationen eines SPS-Laufzeitsystems zugreifen zu können, muss die Symbolgenerierung für die SPS-Variablen oder Strukturen aktiviert werden und die Symbolinformationen während des Projekt-Downloads in das SPS-Laufzeitsystem geladen werden. In der TwinCAT PLC Control können die notwendigen Einstellungen für den Symbol-Download über den Optionsdialog der Kategorie TwinCAT vorgenommen werden. Das erste Laufzeitsystem der SPS wird über die Portnummer 801 angesprochen.

Symbolkonfiguration der zusätzlichen Task im TwinCAT System Manager

Im TwinCAT System Manager kann eine zusätzliche Task eingefügt und konfiguriert werden. Die Variablen der zusätzlichen Task können mit anderen Variablen (z. B. mit den SPS-Variablen oder IO-Variablen eines Busklemmen-Controllers) verknüpft werden. Um auf die Symbolinformationen der zusätzlichen Task zugreifen zu können, muss die Checkbox für die Symbolgenerierung in dem Konfigurationsdialog der Task-Einstellungen aktiviert werden. Das zusätzliche Task wird über die Portnummer 301angesprochen.

Delphi 5 Programm

In der *OnFormCreate*-Eventfunktion wird die Verbindung zu dem ersten Laufzeitsystem der SPS (Port 801) auf dem lokalen PC hergestellt. Gleichzeitig wird die ListView-Komponente und die benötigten Variablen initialisiert. Bei einem Mausklick auf den *Read Symbol Info*-Button wird die Methode *ReadSymInfoButtonClick* aufgerufen. In dieser Methode wird zuerst über den Methodenaufruf: AdsReadSymbolInfo die Anzahl der verfügbaren Symbole ermittelt und dann in einer For-Schleife die Symbolinformationen für jede einzelne Symbolvariable gelesen. Die Werte werden dann über eine Hilfs-Prozedur *AddListViewItem* in die ListeView-Komponente eingefügt. Die Methode AdsEnumSymbols besitzt einen booleschen Flag: *bNext*. Ist dieser Flag auf FALSE gesetzt, dann werden die Symbolinformationen des ersten Symbols und bei bNext=TRUE aller weiteren Symbole gelesen. Um die Symbolinformationen der zusätzlichen Task im TwinCAT System Manager lesen zu können, muss die *AdsAmsServerPort* -Eigenschaft der AdsOcx-Komponente auf 301 gesetzt werden. Die Portnummer können Sie zur Laufzeit über die Eigenschaften-Seite der AdsOcx-Komponente setzen. Die Eigenschaften-Seite kann in der Beispielapplikation über das Menü: *AdsOcx->Properties* aufgerufen werden.

```
unit SampleUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
```

```
  StdCtrls, OleCtrls, ADSOCXLib_TLB, ExtCtrls, ComCtrls, Menus;

type
  TForm1 = class(TForm)
    AdsOcx1: TAdsOcx;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Exit1: TMenuItem;
    AdsOcx2: TMenuItem;
    Properties1: TMenuItem;
    About1: TMenuItem;
    ReadSymInfoButton: TButton;
    ListView1: TListView;
    StatusBar1: TStatusBar;
    procedure OnFormCreate(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Properties1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure ReadSymInfoButtonClick(Sender: TObject);
  private
    { Private declarations }
    procedure CreateColumns(Sender: TObject);
    procedure AddListViewItem(Sender: TObject; strSymbolName, strComment :WideString; nSymbolType, c
bSymbolSize , nIndexGroup, nIndexOffset : integer);
  public
    { Public declarations }
  end;

var
  Form1         : TForm1;
  adsResult     : integer;       {Ads result}
  nSymbols      : integer;
  nSymByteSize      : integer;

implementation

{$R *.DFM}

procedure TForm1.CreateColumns(Sender: TObject);
var ListColumn :TListColumn;
begin
     ListView1.ViewStyle := vsReport;
     ListView1.Align := alBottom;
     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 50;
     ListColumn.Caption := 'No';
     ListColumn.Alignment := taLeftJustify;

     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 200;
     ListColumn.Caption := 'Symbol name';
     ListColumn.Alignment := taLeftJustify;

     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 100;
     ListColumn.Caption := 'Type';
     ListColumn.Alignment := taLeftJustify;

     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 100;
     ListColumn.Caption := 'Size';
     ListColumn.Alignment := taLeftJustify;

     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 100;
     ListColumn.Caption := 'Comment';
     ListColumn.Alignment := taLeftJustify;

     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 100;
     ListColumn.Caption := 'Index Group';
     ListColumn.Alignment := taLeftJustify;

     ListColumn := ListView1.Columns.Add();
     ListColumn.Width := 100;
     ListColumn.Caption := 'Index Offset';
     ListColumn.Alignment := taLeftJustify;
end;
```

```
procedure TForm1.OnFormCreate(Sender: TObject);
begin
     nSymbols := 0;
     nSymByteSize := 0;
     StatusBar1.SimplePanel := true;

     AdsOcx1.AdsAmsServerNetId := AdsOcx1.AdsAmsClientNetId;    {Sets PLC server network adress}
     AdsOcx1.AdsAmsServerPort := 801;               {Sets the PLC run time system}
     StatusBar1.SimpleText := AdsOcx1.AdsServerAdsState;

     CreateColumns(Sender);
end;



procedure TForm1.ReadSymInfoButtonClick(Sender: TObject);
var
   strSymbolName : WideString;
   nSymbolType : Integer;
   cbSymbolSize : Integer;
   strComment : WideString;
   nIndexGroup : Integer;
   nIndexOffset : Integer;

   bNext : WordBool;
   nSymNo      :Integer;
begin
     ListView1.Items.Clear();      {clear old items}

     adsResult := AdsOcx1.AdsReadSymbolInfo( nSymbols, nSymByteSize );
     StatusBar1.SimpleText := Format('AdsReadSymbolInfo result:%d, nSymbols:%d, nSymByteSize:
%d', [adsResult, nSymbols, nSymByteSize]);

     if ( ( adsResult = 0 ) And ( nSymbols > 0 ) ) then
     begin

      bNext := false;      {read first symbol info}
      adsResult := AdsOcx1.AdsEnumSymbols( strSymbolName, nSymbolType, cbSymbolSize, strComment, nIn
dexGroup, nIndexOffset, bNext);
      AddListViewItem(Sender, strSymbolName, strComment, nSymbolType, cbSymbolSize, nIndexGroup, nIn
dexOffset);

      if adsResult > 0 then
         StatusBar1.SimpleText := Format('AdsEnumSymbols result:%d', [adsResult]);

      for nSymNo := 1 to nSymbols-1 do
      begin
           bNext := true;
           adsResult := AdsOcx1.AdsEnumSymbols( strSymbolName, nSymbolType, cbSymbolSize, strComment
, nIndexGroup, nIndexOffset, bNext);
           AddListViewItem(Sender, strSymbolName, strComment, nSymbolType, cbSymbolSize, nIndexGroup
, nIndexOffset);

           if (adsResult > 0) then
          StatusBar1.SimpleText := Format('AdsEnumSymbols result:%d', [adsResult]);
     end;
     end;
end;

procedure TForm1.AddListViewItem(Sender: TObject; strSymbolName, strComment :WideString; nSymbolType
, cbSymbolSize , nIndexGroup, nIndexOffset : integer);
var   ListItem     :TListItem;
      strAdsType   :String;
begin
     ListItem := ListView1.Items.Add();
     ListItem.Caption :=  Format('%d',[ListView1.Items.Count]);

     ListItem.SubItems.Add(strSymbolName);

     case  nSymbolType of
       0:  strAdsType := 'ADST_VOID';
       16:  strAdsType := 'ADST_INT8';
       17:  strAdsType := 'ADST_UINT8';
       2:  strAdsType := 'ADST_INT16';
       18:  strAdsType := 'ADST_UINT16';
       3:  strAdsType := 'ADST_INT32';
       19:  strAdsType := 'ADST_UINT32';
       20:  strAdsType := 'ADST_INT64';
       21:  strAdsType := 'ADST_UINT64';
       4:  strAdsType := 'ADST_REAL32';
```

```
       5:  strAdsType := 'ADST_REAL64';
       65:  strAdsType := 'ADST_BIGTYPE';
       30:  strAdsType := 'ADST_STRING';
       31:  strAdsType := 'ADST_WSTRING';
       32:  strAdsType := 'ADST_REAL80';
       33:  strAdsType := 'ADST_BIT';
       34:  strAdsType := 'ADST_MAXTYPES';
     end;

     ListItem.SubItems.Add(Format('%s',[strAdsType]));
     ListItem.SubItems.Add(Format('0x%x (%d)',[cbSymbolSize, cbSymbolSize]));
     ListItem.SubItems.Add(strComment);
     ListItem.SubItems.Add(Format('0x%x (%d)',[nIndexGroup, nIndexGroup]));
     ListItem.SubItems.Add(Format('0x%x (%d)',[nIndexOffset, nIndexOffset]));
end;


procedure TForm1.Exit1Click(Sender: TObject);
begin
    Close();
end;

procedure TForm1.Properties1Click(Sender: TObject);
begin
    AdsOcx1.BrowseProperties();
    StatusBar1.SimpleText := AdsOcx1.AdsServerAdsState;
end;

procedure TForm1.About1Click(Sender: TObject);
begin
    AdsOcx1.AboutBox();
end;

Initialization
    IsMultiThread := True;// Setting this system variable makes Delphi's memory manager thread-safe


end.
```

SPS 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 *)
END_VAR
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466732171.exe
Delphi 5 oder höher (classic)	

5.2.4 Array in die SPS schreiben oder aus der SPS lesen

Systemvoraussetzungen:

- Delphi 6.0 oder höher;
- TwinCAT v2.10 oder höher;

Benutzen Sie die für bestimmte Datentypen "aufgelösten" Methoden. Wenn Sie z. B. ein Array in der SPS vom Typ INT lesen wollen, können, abhängig von der Zugriffsart, z. B. folgende Methoden benutzt werden:

AdsSyncReadIntegerVarReq(hVar : Integer, length : Integer, var pData : Smallint)

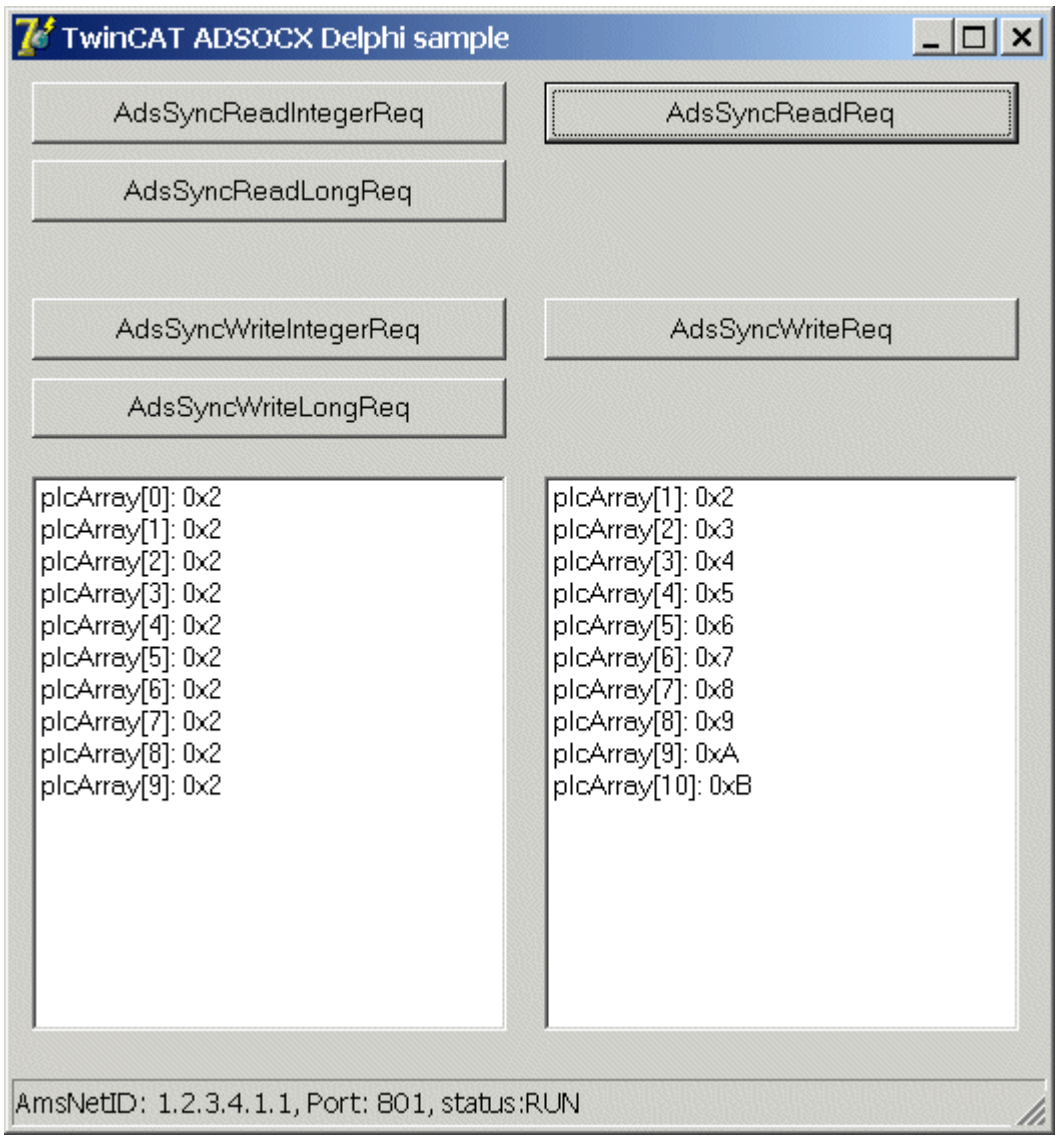
AdsSyncReadIntegerReq(indexGroup : Integer, indexOffset : Integer, length : Integer, var pData : Smallint)

AdsSyncWriteIntegerVarReq(hVar : Integer, length : Integer, var pData : Smallint)

AdsSyncWriteIntegerReq(indexGroup : Integer, indexOffset : Integer, length : Integer, var pData : Smallint)

AdsReadIntegerReq(nInvokeId : Integer, nIndexGroup : Integer, nIndexOffset : Integer, cbLength : Integer)

AdsWriteIntegerReq(nInvokeId : Integer, nIndexGroup : Integer, nIndexOffset : Integer, cbLength : Integer, var pData : Smallint)



Auf String-Arrays kann auf diese Weise nicht zugegriffen werden. Die Länge der Daten, die gelesen oder geschrieben werden sollen, ergibt sich aus der Anzahl der zu lesenden oder zu schreibenden Elemente, multipliziert mit der Byte-Größe eines Elementes. Diese Länge muss in dem *length*- oder *cbLength*-Parameter übergeben werden. Über den Parameter *pData* wird das erste Element des Delphi-Arrays übergeben.

Beispiel:

```
PROGRAM MAIN
VAR
    varIntArray :ARRAY[1..9] OF INT:=9(1);
END_VAR
```

Delphi 6 Programm:

Array aus der SPS lesen:

```
procedure TForm1.SyncReadArrayVarButtonClick(Sender: TObject);
var i, hVar, AdsResult:integer;
    varIntArray       : ARRAY[1..9] OF Smallint;
begin
    AdsResult := AdsOcx1.AdsCreateVarHandle( 'MAIN.VARINTARRAY', hVar );
    if AdsResult = 0 then
    begin
    AdsResult := AdsOcx1.AdsSyncReadIntegerVarReq( hVar, sizeof(varIntArray), varIntArray[1] );
    if AdsResult = 0 then
    begin
        ListBox1.Clear();
        for i:=1 to 9 do
        ListBox1.Items.Add( Format('varIntArray[%d] = %d', [i, varIntArray[i]] ) );
    end
```

```
    else Label1.Caption := Format('AdsSyncReadIntegerVarReq error:%d', [AdsResult] );
    AdsOcx1.AdsDeleteVarHandle( hVar );
    end
    else Label1.Caption := Format('AdsCreateVarHandle error:%d', [AdsResult] );
end;
```

Array in die SPS schreiben:

```
procedure TForm1.SyncWriteArrayVarButtonClick(Sender: TObject);
var i, hVar, AdsResult:integer;
    varIntArray       : ARRAY[1..9] OF Smallint;
begin
    for i:=1 to 9 do
    varIntArray[i] := i;

    AdsResult := AdsOcx1.AdsCreateVarHandle( 'MAIN.VARINTARRAY', hVar );
    if AdsResult = 0 then
    begin
    AdsResult := AdsOcx1.AdsSyncWriteIntegerVarReq( hVar, sizeof(varIntArray), varIntArray[1] );
    if AdsResult > 0 then
        Label1.Caption := Format('AdsSyncWriteIntegerVarReq error:%d', [AdsResult] );
    AdsOcx1.AdsDeleteVarHandle( hVar );
    end
    else Label1.Caption := Format('AdsCreateVarHandle error:%d', [AdsResult] );
end;
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466733579.exe
Delphi 6 oder höher (classic)	

5.2.5 ADS-OCX-Property-Seite aufrufen

Systemvoraussetzungen:

- Delphi 7.0 oder höher;
- TwinCAT v2.9 oder höher;

Beschreibung

Unter Delphi kann die ADS-OCX-Property-Seite auf folgende Weise aufgerufen werden:

```
procedure TForm1.btnShowPropertyPageClick(Sender: TObject);
begin
    AdsOcx1.BrowseProperties();
end;
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466734987.exe
Delphi 7 oder höher (classic)	

5.2.6 Arbeiten mit Handles von SPS-Variablen

Systemvoraussetzungen:

- Delphi 7.0 oder höher;
- TwinCAT v2.9 oder höher;

Alle benötigten Handles können ein Mal beim Start der Applikation angefordert werden und beim Beenden der Applikation freigegeben werden. Ständiges Anfordern und Freigeben der Handles verursacht unnötige Belastung des Systems.

Die bereits angeforderten Handles werden beim TwinCAT Restart ungültig und müssen neu angefordert werden, ebenso nach jedem 'Rebuild All' in der SPS. Bei 'Rebuild All' wird in das Laufzeitsystem ein komplett neues Programm geladen, dadurch werden alle bereits angeforderten Handles ungültig und und von TwinCAT automatisch freigegeben. Die nicht mehr benötigten Handles müssen immer freigegeben werden. Dies kann aber nur dann erfolgen, wenn das TwinCAT System noch läuft. Wurde das TwinCAT System bereits gestoppt, dann werden alle Handles automatisch freigegeben.

Mit dem ersten Laufzeitsystem auf dem lokalen PC verbinden und das Handle der SPS-Variablen holen:

```
procedure TForm1.FormCreate(Sender: TObject);
var adsResult : Integer;
begin
   AdsOcx1.AdsAmsServerNetId := AdsOcx1.AdsAmsClientNetId;
   AdsOcx1.AdsAmsServerPort := 801;
   adsResult := AdsOcx1.AdsCreateVarHandle( 'MAIN.VARINTARRAY', hVar );
   if adsResult <> 0 then
      ShowMessage( Format( 'AdsCreateVarHandle() error:%d', [adsResult] ) );
end;
```

Beim Beenden der Applikation Handle freigeben:

```
procedure TForm1.FormDestroy(Sender: TObject);
var adsResult : Integer;
begin
   adsResult := AdsOcx1.AdsDeleteVarHandle( hVar );
   if AdsResult <> 0 then
      ShowMessage( Format( 'AdsDeleteVarHandle() error:%d', [adsResult] ) );
   hVar := 0;
end;
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466736395.exe
Delphi 7 oder höher (classic)	

5.2.7 String in die SPS schreiben oder aus der SPS lesen

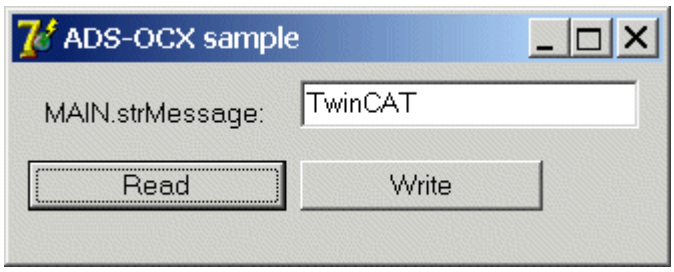
Systemvoraussetzungen:

- Delphi 7.0 oder höher;
- TwinCAT v2.11 Build 2034 oder höher;

Aufgabe

Ein String soll in die SPS geschrieben oder aus der SPS gelesen werden.

Beschreibung



Damit ein String in die SPS geschrieben oder oder aus der SPS gelesen werden kann, benötigt man die Länge des SPS-Strings. Die tatsächliche Länge eines SPS-Strings kann mit dem SPS-Operator SIZEOF ermittelt werden. In der SPS werden die Strings mit einer Null terminiert und die tatsächliche String-Länge errechnet sich aus der definierten Länge plus 1. Wurde bei der Stringdefinition keine Länge angegeben, dann besitzt der String eine tatsächliche Länge von 81 Zeichen inklusive der abschließenden Null.

SPS-Programm

```
PROGRAM MAIN
VAR
    strColor    :STRING(10)     :='Blue';
    strState    :STRING(20)     :='STOP';
    strMessage  :STRING         :='TwinCAT ADS-OCX';
END_VAR
```

strColor hat die Länge 11 Zeichen;

strState hat die Länge 21 Zeichen;

strMessage hat die Länge 81 Zeichen;

Delphi 7 Programm

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, OleCtrls, ADSOCXLib_TLB, Grids, ValEdit, ComCtrls;

type
  TForm1 = class(TForm)
    btnWrite: TButton;
    AdsOcx1: TAdsOcx;
    Label1: TLabel;
    Edit1: TEdit;
    btnRead: TButton;
    procedure btnReadClick(Sender: TObject);
    procedure btnWriteClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
    adsResult   : Integer;// Ads return code
    hVar    : Integer;// PLC variable handle
    varString   : WideString;// PLC variable value
  public
    { Public declarations }
  end;

var
   Form1: TForm1;

implementation

{$R *.dfm}
```

String aus der SPS lesen

```
procedure TForm1.btnReadClick(Sender: TObject);
begin
   SetLength(varString, 7 );//Realocate string space to a given length// Read string from PLC
   adsResult := AdsOcx1.AdsSyncReadStringVarReq( hVar, Length(varString) * 2, varString );
   if  adsResult = 0 then
      edit1.Text := varString
   else ShowMessage( Format( 'AdsSyncReadStringVarReq() error:%d', [adsResult] ) );
end;
```

In dem oberen Beispiel werden 7 Zeichen eines SPS-Strings in Delphi eingelesen. Die dynamischen String-Typen haben nach der Initialisierung die Länge Null. Damit das ADS-OCX den SPS-String in die Delphi-Stringvariable hineinkopieren kann, muss vorher die Delphi-Stringvariable auf die gewünschte Länge lokiert werden. Eine WideString-Variable in Delphi benötigt zwei Bytes pro Zeichen. Die Length-Funktion liefert die lokierte Anzahl der Zeichen im String. Der *Length*-Parameter im Methodenaufruf benötigt aber die Byte-Länge, deshalb wird die mit Length-Funktion ermittelte Länge verdoppelt.

String in die SPS schreiben

```
procedure TForm1.btnWriteClick(Sender: TObject);
begin
   varString := Edit1.Text;
   // Write string to the PLC
   adsResult := AdsOcx1.AdsSyncWriteStringVarReq( hVar, Length(varString)*2, varString );
   if adsResult <> 0 then
      ShowMessage( Format( 'AdsSyncWriteStringVarReq() error:%d', [adsResult] ) );
end;
```

Verbindung zur SPS aufbauen, variable handle holen

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Connection Setup
  AdsOcx1.AdsAmsServerNetId := AdsOcx1.AdsAmsServerNetId;
  AdsOcx1.AdsAmsServerPort := 801;

  // Create variable handle
  adsResult := AdsOcx1.AdsCreateVarHandle( 'MAIN.STRMESSAGE', hVar );
  if adsResult <> 0 then
     ShowMessage( Format( 'AdsCreateVarHandle() error:%d', [adsResult] ) );
end;
```

Resourcen (variable handle) freigeben

```
procedure TForm1.FormDestroy(Sender: TObject);
var adsResult : Integer;
begin
   // Delete variable handle
   adsResult := AdsOcx1.AdsDeleteVarHandle( hVar );
   if AdsResult <> 0 then
      ShowMessage( Format( 'AdsDeleteVarHandle() error:%d', [adsResult] ) );
   hVar := 0;
end;
```

```
Initialization
   IsMultiThread := True;// Setting this system variable makes Delphi's memory manager thread-safe

end.
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466737803.exe
Delphi 7 oder höher (classic)	

Dokumente hierzu

ads-ocxsample06.exe (Resources/exe/12466737803.exe)

5.2.8 Mehrere boolsche Variablen mit einem Zugriff in ein Array einlesen

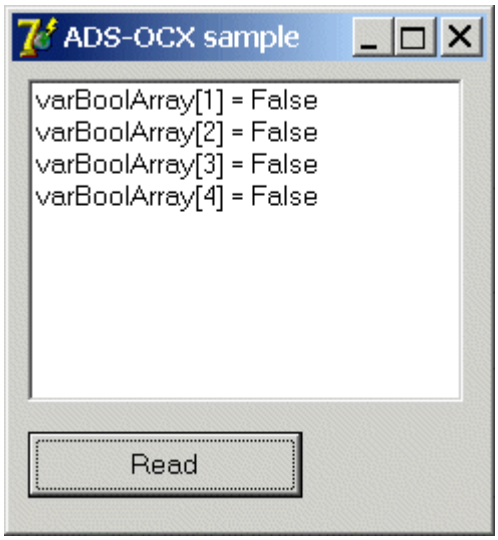
Systemvoraussetzungen:

- Delphi 7.0 oder höher;
- TwinCAT v2.11 Build 2034 oder höher;

Aufgabe

Mehrere boolesche SPS-Variablen können mit einem Zugriff in Delphi-Applikation eingelesen werden, wenn diese Variablen auf Adressen lokiert wurden, die hintereinander im Speicher liegen. Wichtig ist aber, dass die erste Variable auf eine Byteadresse lokiert wurde.

Beschreibung



SPS-Programm

```
PROGRAM MAIN
VAR
    varBoolean AT%MB6 : ARRAY[1..4] OF BOOL;
END_VAR
```

Delphi 7 Programm

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, OleCtrls, ADSOCXLib_TLB, StdCtrls;

type
  TForm1 = class(TForm)
    btnRead: TButton;
    AdsOcx1: TAdsOcx;
    ListBox1: TListBox;
    procedure btnReadClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  varBoolArray : ARRAY[1..4] OF WordBool;

implementation
{$R *.dfm}

procedure TForm1.btnReadClick(Sender: TObject);
var     i, hVar, AdsResult:integer;
begin
    // Create variable handle
    AdsResult := AdsOcx1.AdsCreateVarHandle( 'MAIN.VARBOOLEAN', hVar );
    if AdsResult = 0 then
    begin
        // Read data
        AdsResult := AdsOcx1.AdsSyncReadBoolVarReq( hVar, sizeof(varBoolArray), varBoolArray[1] );
        if AdsResult = 0 then
        begin
            // Clear list view and show data
            ListBox1.Clear();
            for i:=1 to 4 do
                ListBox1.Items.Add( Format('varBoolArray[%d] = %s', [i, BoolToStr(varBoolArray[i], t
rue) ] ) );
        end
        else ShowMessage( Format( 'AdsSyncReadBooleanVarReq() error:%d', [AdsResult] ) );
```

```
        // Release variable handle
        AdsResult := AdsOcx1.AdsDeleteVarHandle( hVar );
        if AdsResult <> 0 then
            ShowMessage( Format( 'AdsDeleteVarHandle() error:%d', [AdsResult] ) );
    end
    else ShowMessage( Format( 'AdsCreateVarHandle() error:%d', [AdsResult] ) );
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   // Connection Setup
   AdsOcx1.AdsAmsServerNetId := AdsOcx1.AdsAmsServerNetId;
   AdsOcx1.AdsAmsServerPort := 801;
end;

Initialization
   IsMultiThread := True;// Setting this system variable makes Delphi's memory manager thread-safe

end.
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466739211.exe
Delphi 7 oder höher (classic)	

5.2.9 Übertragen von Strukturen zur/von der SPS

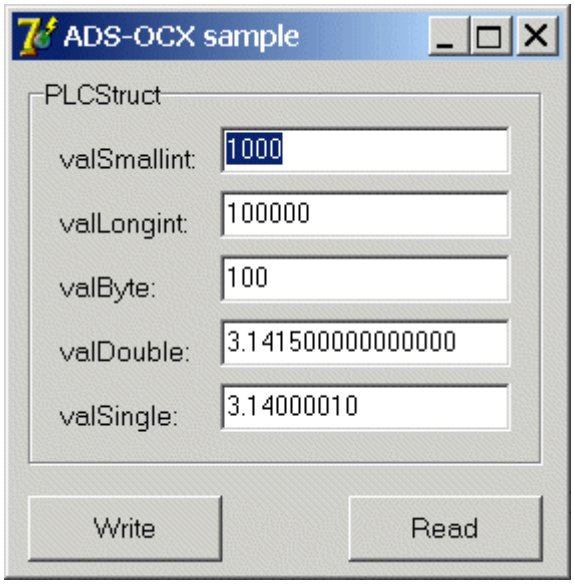
Systemvoraussetzungen:

- Delphi 7.0 oder höher;
- TwinCAT v2.11 Build 2034 oder höher;

Aufgabe

Von der Delphi-Applikation soll eine Struktur in die SPS geschrieben oder gelesen werden. Die Elemente der Struktur haben verschiedene Datentypen.

Beschreibung



Strukturdeklaration in der SPS

```
TYPE PLCStruct
STRUCT
    valSmallint     : INT;
    valLongint      : DINT;
    valByte         : BYTE;
    valDouble       : LREAL;
```

```
    valSingle       : REAL;
END_STRUCT
END_TYPE
```

SPS-Programm

```
PROGRAM MAIN
VAR
    PLCVar : PLCStruct;
END_VAR
```

```
;
```

Strukturdeklaration in Delphi

```
Type VBStruct
    TPLCStruct = packed record      // packed == force 1 byte alignment
    valSmallint     : Smallint;     // 2 bytes
    valLongint      : Longint;      // 4 bytes
    valByte         : Byte;         // 1 byte
    valDouble       : Double;       // 8 bytes
    valSingle       : Single;       // 4 bytes// = 19 bytes in memory
End;
```

Delphi 7 Programm

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, OleCtrls, ADSOCXLib_TLB, StdCtrls;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    AdsOcx1: TAdsOcx;
    btnWrite: TButton;
    btnRead: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    editSmallint: TEdit;
    editLongint: TEdit;
    editByte: TEdit;
    editDouble: TEdit;
    editSingle: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnWriteClick(Sender: TObject);
    procedure btnReadClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
   TPLCStruct = packed record // packed == force 1 byte alignment
      valSmallint : Smallint; // 2 bytes
      valLongint  : Longint;  // 4 bytes
      valByte     : Byte;     // 1 byte
      valDouble   : Double;   // 8 bytes
      valSingle   : Single;   // 4 bytes// = 19 bytes in memory
End;

var
  Form1: TForm1;
  hVar : Integer;

  // Create instance and initialize delphi structure members
  PLCStruct : TPLCStruct = ( valSmallint : 1000;
               valLongint : 100000;
               valByte : 100;
               valDouble : 3.1415;
               valSingle : 3.14 );
implementation
{$R *.dfm}
```

```
//--- Is called a the start ---
procedure TForm1.FormCreate(Sender: TObject);
var text : String;
begin
   //--- Enable exception ---
   AdsOcx1.EnableErrorHandling := True;
   //--- Set connection ---
   AdsOcx1.AdsAmsServerPort := 801;
   AdsOcx1.AdsAmsServerNetId := AdsOcx1.AdsAmsClientNetId;
   //--- Get PLC variable handle by variable name
   AdsOcx1.AdsCreateVarHandle('Main.PLCVar', hVar);
   //--- View init values ---
   Str( PLCStruct.valSmallint, text );
   editSmallint.Text := text;
   Str( PLCStruct.valLongint, text );
   editLongint.Text := text;
   Str( PLCStruct.valByte, text );
   editByte.Text := text;
   Str( PLCStruct.valDouble : 0 : 15, text );
   editDouble.Text := text;
   Str( PLCStruct.valSingle : 0 : 8, text );
   editSingle.Text := text;
end;

//--- Is called at the end ---
procedure TForm1.FormDestroy(Sender: TObject);
begin
   //--- Release PLC variable handle ---
   AdsOcx1.AdsDeleteVarHandle(hVar);
end;

//--- Is called by the user  ---
procedure TForm1.btnWriteClick(Sender: TObject);
var code : Integer;
begin
   //--- Fill structure ---
   Val( editSmallint.Text, PLCStruct.valSmallint, code );
   Val( editLongint.Text, PLCStruct.valLongint, code );
   Val( editByte.Text, PLCStruct.valByte, code );
   Val( editDouble.Text, PLCStruct.valDouble, code );
   Val( editSingle.Text, PLCStruct.valSingle, code );
   //--- Write structure to the PLC ---
   AdsOcx1.AdsSyncWriteIntegerVarReq( hVar, sizeof(PLCStruct), PLCStruct.valSmallint );
end;

//--- Is called by the user ---
procedure TForm1.btnReadClick(Sender: TObject);
var text : String;
begin
   //--- Read structure from the PLC ---
   AdsOcx1.AdsSyncReadIntegerVarReq( hVar, sizeof(PLCStruct), PLCStruct.valSmallint );
   //--- View read structure data ---
   Str( PLCStruct.valSmallint, text );
   editSmallint.Text := text;
   Str( PLCStruct.valLongint, text );
   editLongint.Text := text;
   Str( PLCStruct.valByte, text );
   editByte.Text := text;
   Str( PLCStruct.valDouble : 0 : 15, text );
   editDouble.Text := text;
   Str( PLCStruct.valSingle : 0 : 8, text );
   editSingle.Text := text;
end;

Initialization
   IsMultiThread := True;// Setting this system variable makes Delphi's memory manager thread-safe

end.
```

Sprache / IDE	Beispielprogramm auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/12466740619.exe
Delphi 7 oder höher (classic)	

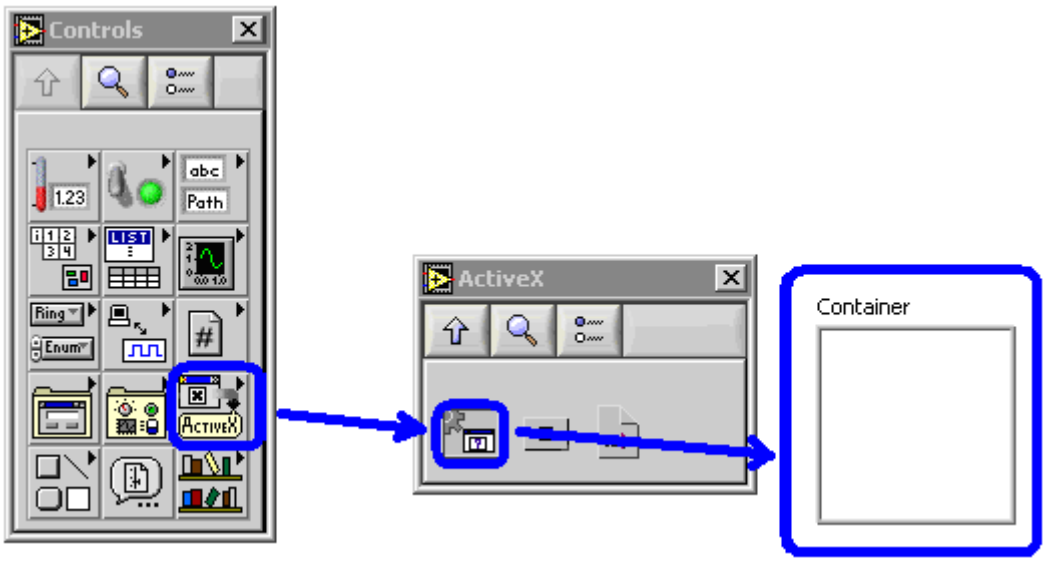
5.3 LabVIEW™ - Beispiele

5.3.1 Einbinden in LabVIEW™

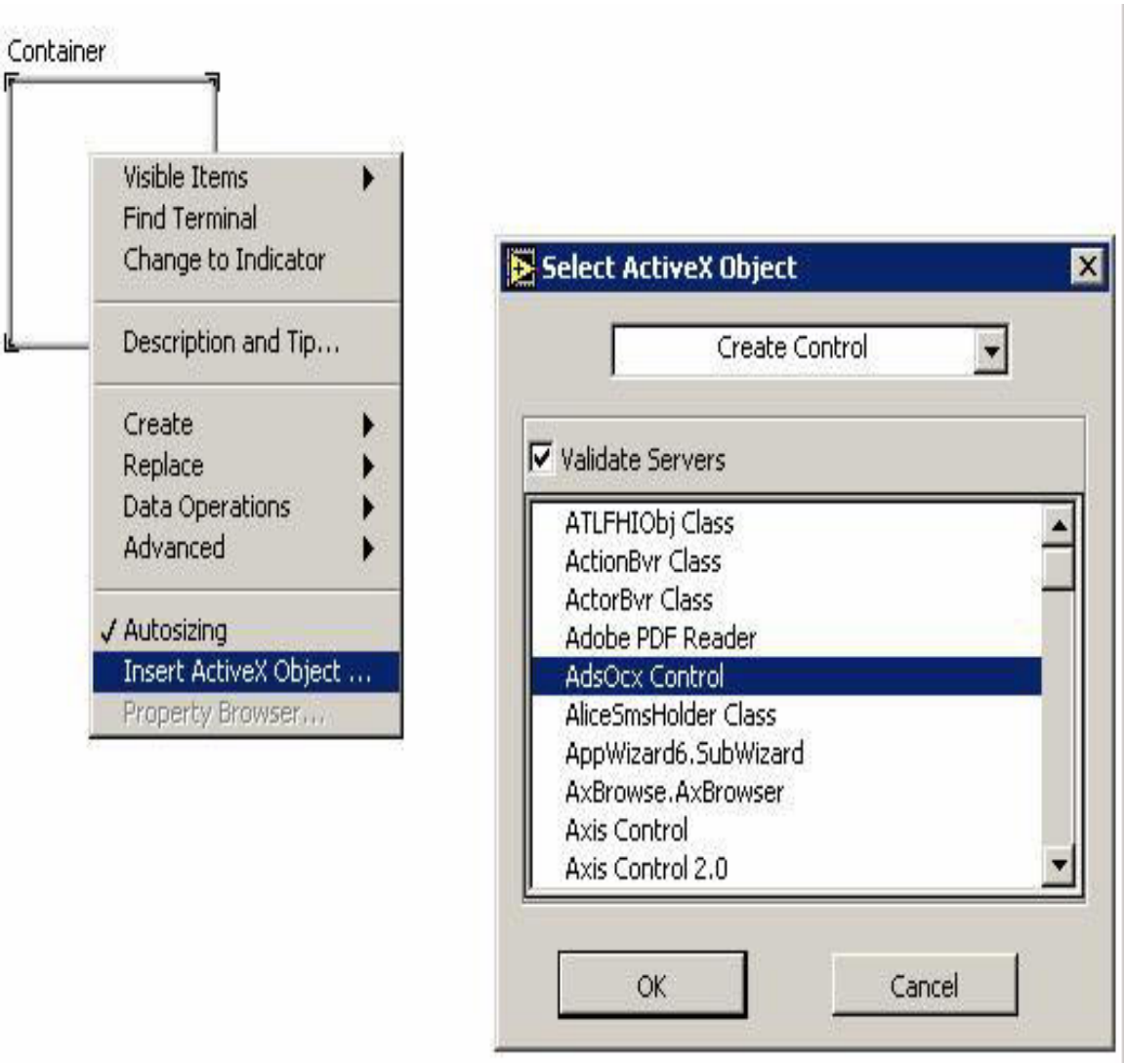
Nutzen Sie das TwinCAT 3 Interface for LabVIEW™

Wenn Sie eine ADS-Kommunikation zwischen LabVIEW™ und der TwinCAT 3 Laufzeit aufbauen möchten, nutzen Sie in jedem Fall das umfänglich supportete und dokumentierte Produkt TwinCAT 3 Interface for LabVIEW™, siehe TF3710. Die im Folgenden dargestellte händische Einbindung von kostenfreien ADS-Komponenten stellt lediglich Anwendungsbeispiele dar. Diese unterliegen nicht dem Beckhoff-Support.

1. ActiveX Container erstellen



2. ActiveX Object einfügen



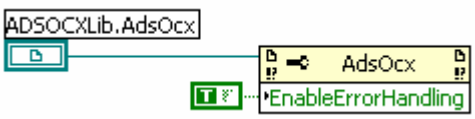
3. AdsOcx Element in LabVIEW™

   Panel:

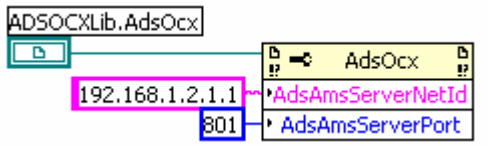
   Diagramm:

5.3.2 Anwendungsbeispiele mit AdsOcx Eigenschaften

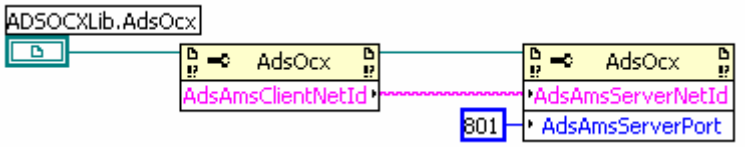
1. EnableErrorHandling auf True setzen



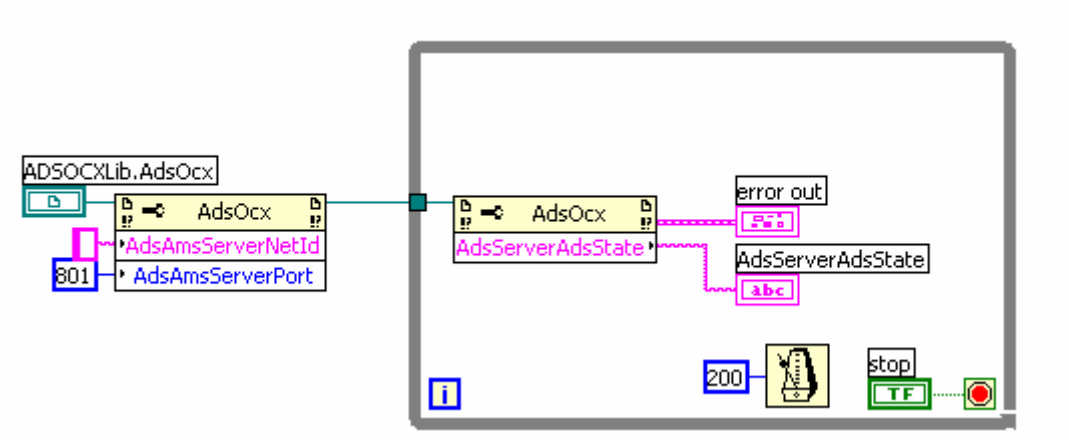
2. AdsAmsServerNetId und AdsAmsServerPort auf feste Werte setzen.



3. Zugriff auf lokale PLC durch auslesen und setzen der AmsNetId



4. Überwachung des Zustands des ADS-Gerätes (s. https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967690891.zip)



5.3.3 Synchrone Methoden, Lesen per Adresse

AdsSyncReadBoolReq,

AdsSyncReadIntegerReq,

AdsSyncReadLongReq,

AdsSyncReadSingleReq,

AdsSyncReadDoubleReq,

AdsSyncReadStringReq

Beispiel: **AdsSyncReadBoolReq**

PLC-Deklaration:

```
    TCtoLV_boolVal AT%MX0.0: BOOL;
```

LabVIEW™ (s. https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967692299.zip):

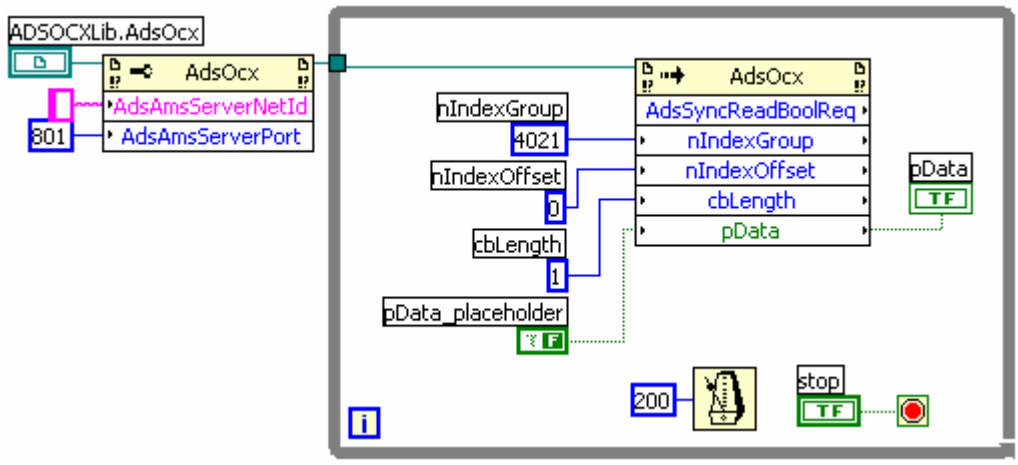


Abb. 1: TcAdsO35

5.3.4 Synchrone Methoden, Lesen per Name

AdsSyncReadBoolVarReq

AdsSyncReadIntegerVarReq

AdsSyncReadLongVarReq

AdsSyncReadSingleVarReq

AdsSyncReadDoubleVarReq

AdsSyncReadStringVarReq

Beispiel: **AdsSyncReadBoolVarReq**

PLC-Deklaration:

```
    TCtoLV_boolVal AT%MX0.0: BOOL;
```

LabVIEW™: (s. https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967693707.zip)

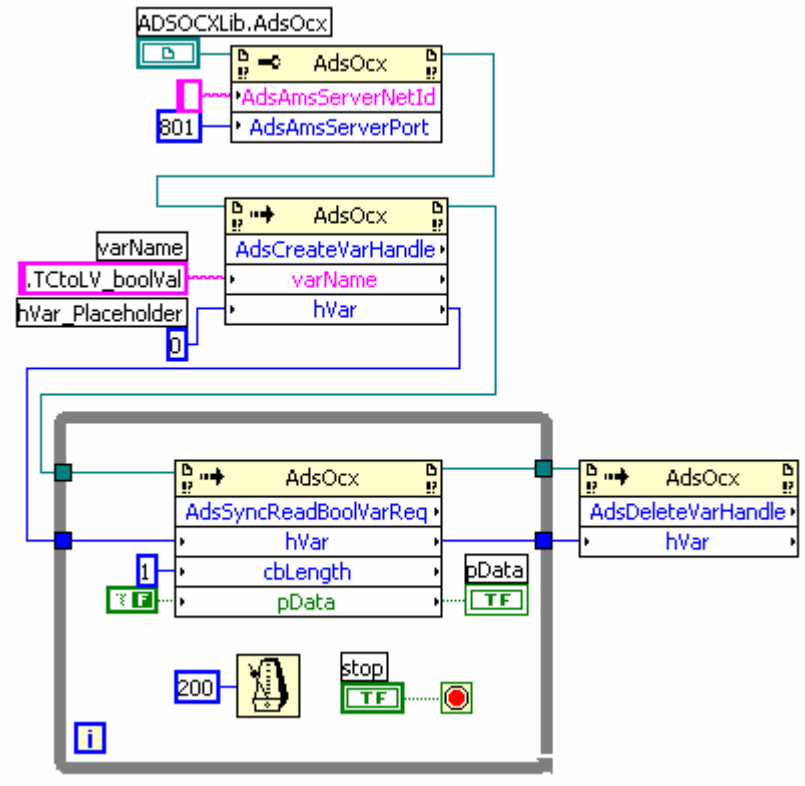


Abb. 2: TcAdsO36

5.3.5 Synchrone Methoden, Schreiben per Adresse

AdsSyncWriteBoolReq

AdsSyncWriteIntegerReq

AdsSyncWriteLongReq

AdsSyncWriteSingleReq

AdsSyncWriteDoubleReq

AdsSyncWriteStringReq

Beispiel: **AdsSyncWriteBoolReq**

PLC-Deklaration:

```
    LVtoTC_boolVal AT%MX500.0: BOOL;
```

LabVIEW™: (s. https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967695115.zip)

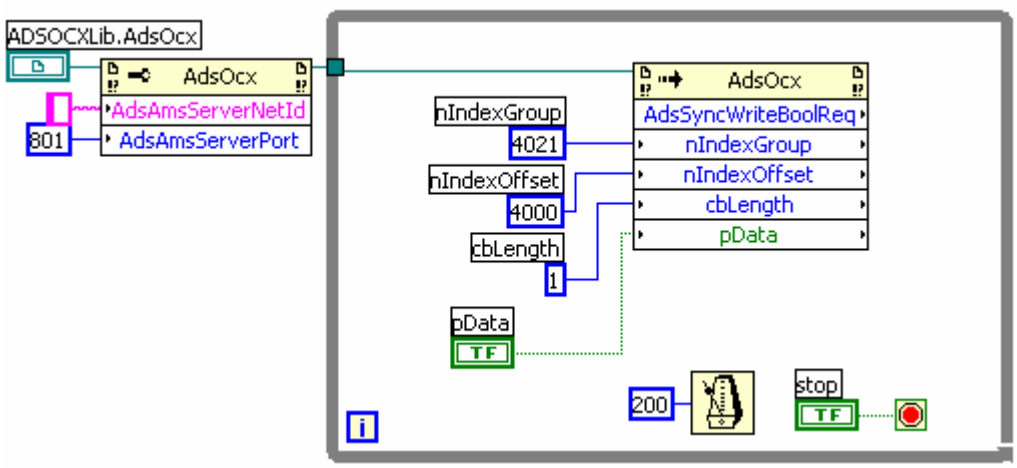


Abb. 3: TcAdsO37

5.3.6 Synchrone Methoden, Schreiben per Name

AdsSyncWriteBoolVarReq

AdsSyncWriteIntegerVarReq

AdsSyncWriteLongVarReq

AdsSyncWriteSingleVarReq

AdsSyncWriteDoubleVarReq

AdsSyncWriteStringVarReq

Beispiel: **AdsSyncWriteBoolVarReq**

PLC-Deklaration:

```
    LVtoTC_boolVal AT%MX500.0: BOOL;
```

LabVIEW™: (s. https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967696523.zip)

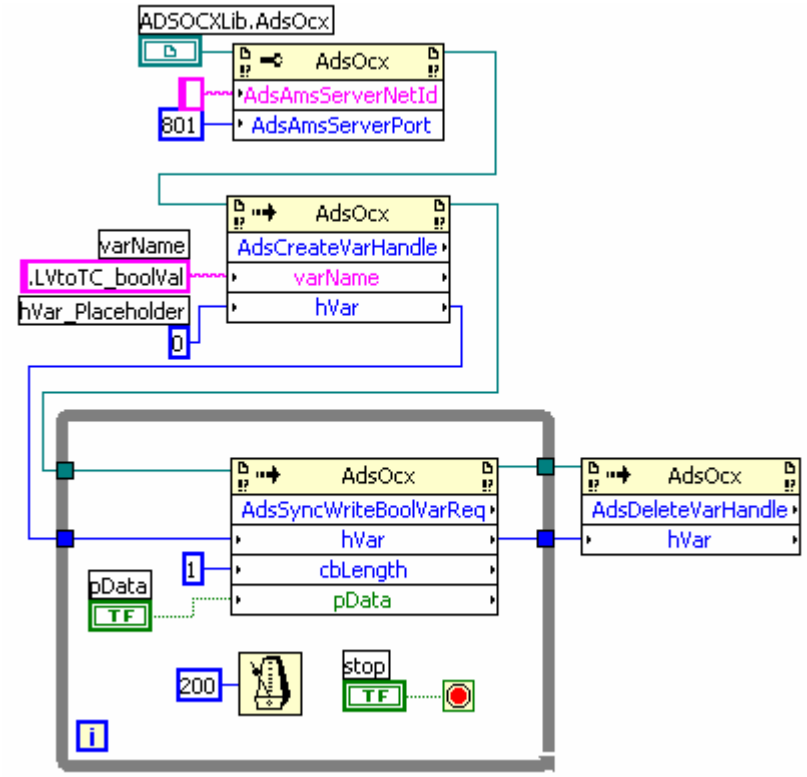


Abb. 4: TcAdsO38

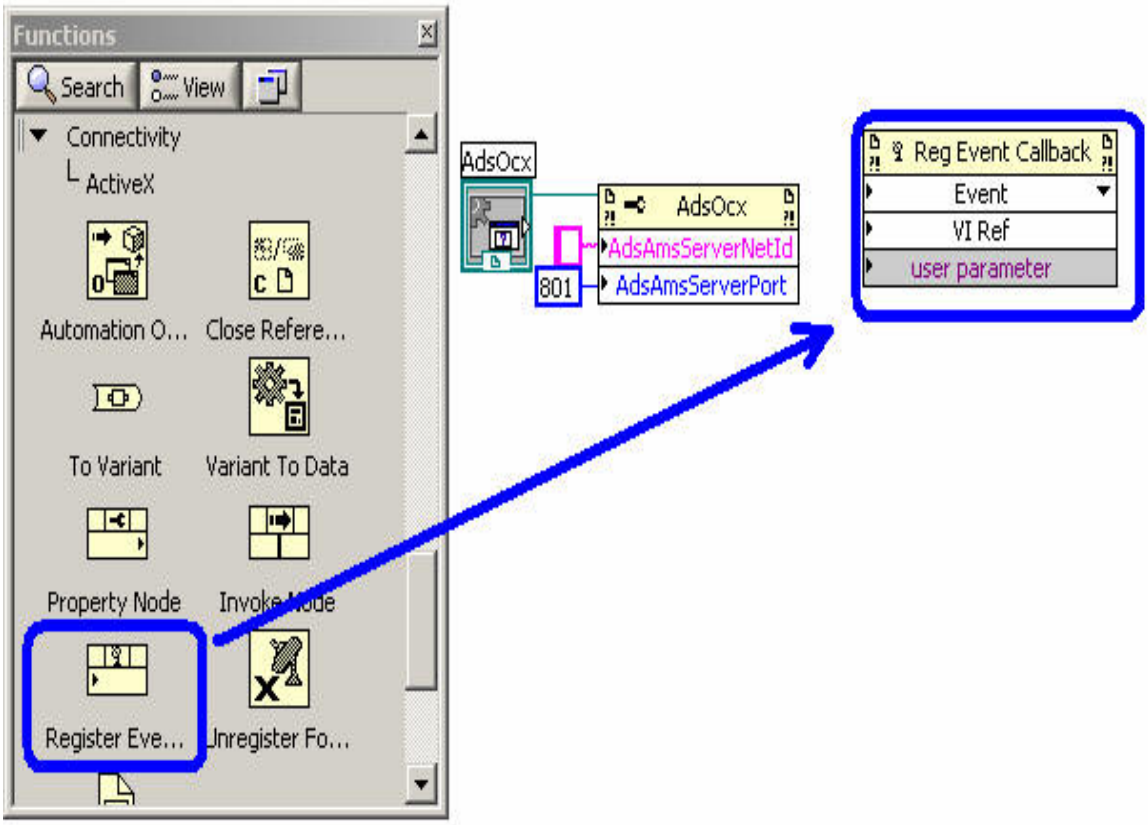
Dokumente hierzu

sample_dll_005_adsinforead.zip (Resources/zip/11967685259.zip)

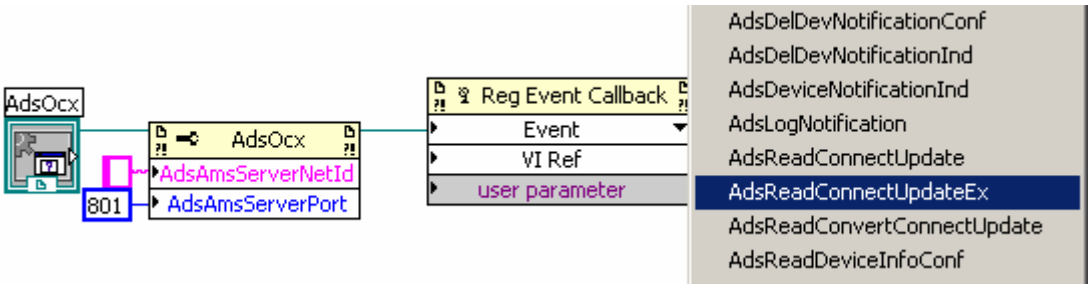
5.3.7 Ereignisgesteuertes Lesen, Registrieren eines Callback-vi

Beispieldateien: https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967697931.zip,

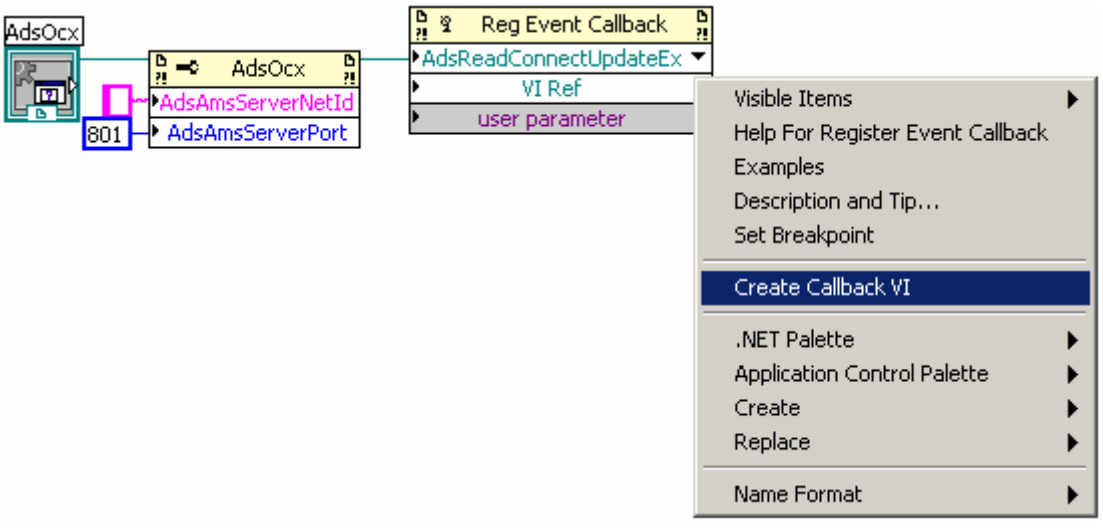
1. Um eine asynchrone Methode zu verwenden, wird ein Callback-Vi registriert, das vom AdsOcx aufgerufen wird.



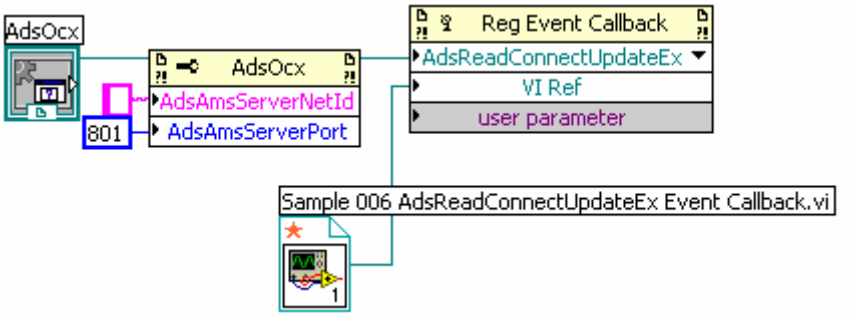
2. Das Event Element wird mir der AdsOcx Referenz verbunden, und das aufzurufende Ereignis ausgewählt.



3. Das Callback-Vi muss eine ganz bestimmte Parameterstruktur aufweisen. Man kann das Callback-Vi von LabVIEW™ erstellen lassen.



4. Das Event-Callback-vi sollte anschließend unter einem eindeutigen Namen abgespeichert werden.



5.3.8 Ereignisgesteuertes Lesen, einfache Datentypen

Methode: **AdsReadVarConnectEx**

Beispiel:

https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967699339.zip

PLC-Deklaration:

```
    TCtoLV_boolVal AT%MX0.0: BOOL;
```

Für das Event **AdsReadConnectUpdateEx** wird ein Callback-Vi registriert.

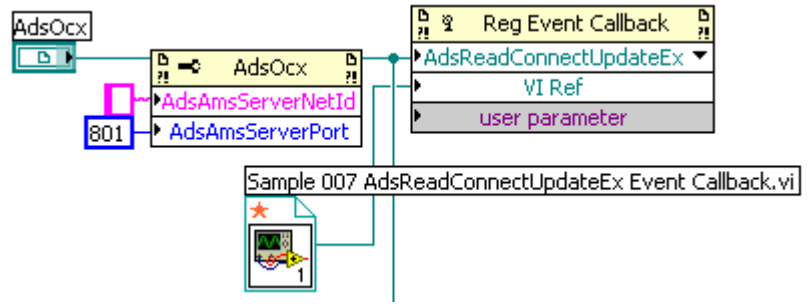


Abb. 5: TcAdsO44

Die Methode **AdsReadVarConnectEx** stellt eine feste Verbindung zwischen LabVIEW™ und einer SPS-Variablen. Das zurückgelieferte Handle identifiziert die Verbindung. Wenn die Verbindung nicht mehr benötigt wird, wird sie mit **AdsDisconnectEx** gelöst.

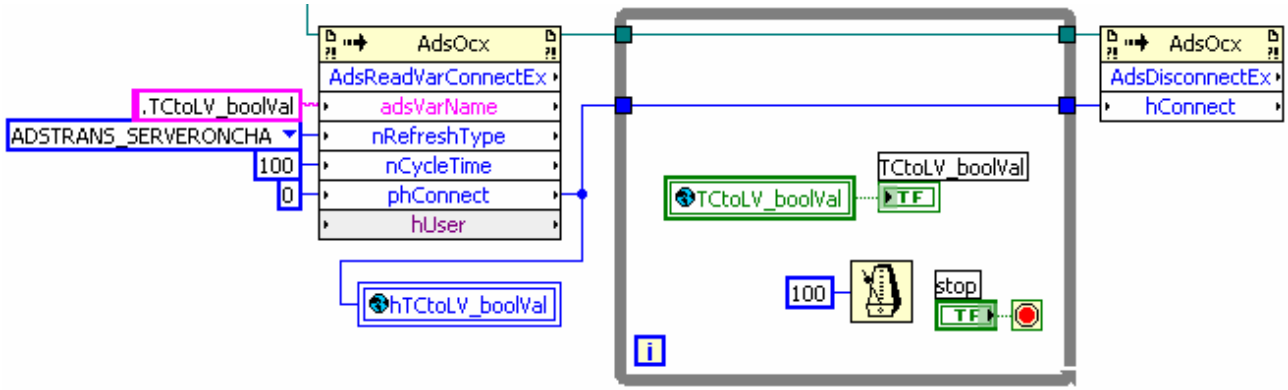


Abb. 6: TcAdsO45

Die Daten werden beim Aufruf des Callback-Vi als Variant übertragen. Anhand des Handles können die Variablen in den richtigen Typ umgewandelt und der richtigen globalen LabVIEW™- Variablen zugewiesen werden.

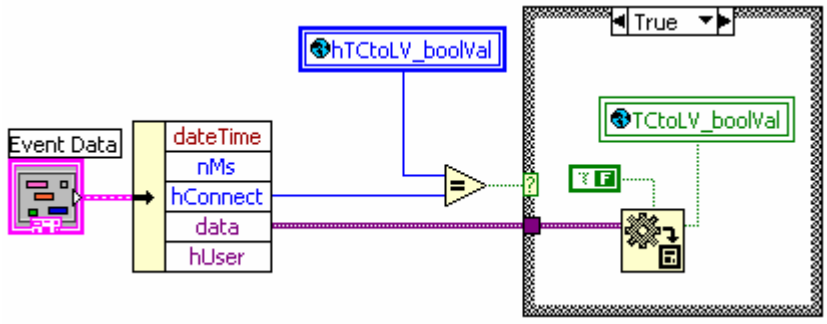


Abb. 7: TcAdsO46

5.3.9 Ereignisgesteuertes Lesen, Strukturvariablen

Methode: **AdsReadVarConvertConnect**

Beispieldateien :

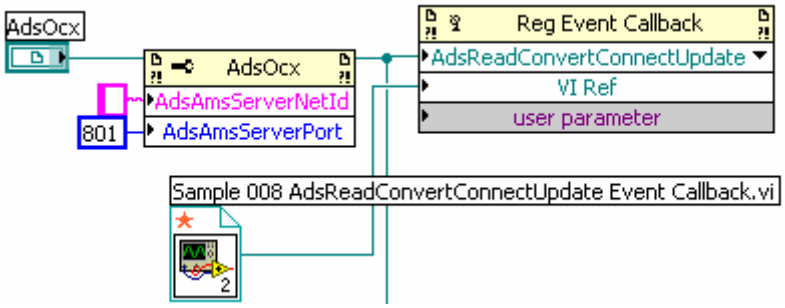
https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967700747.zip,

TwinCAT Deklaration:

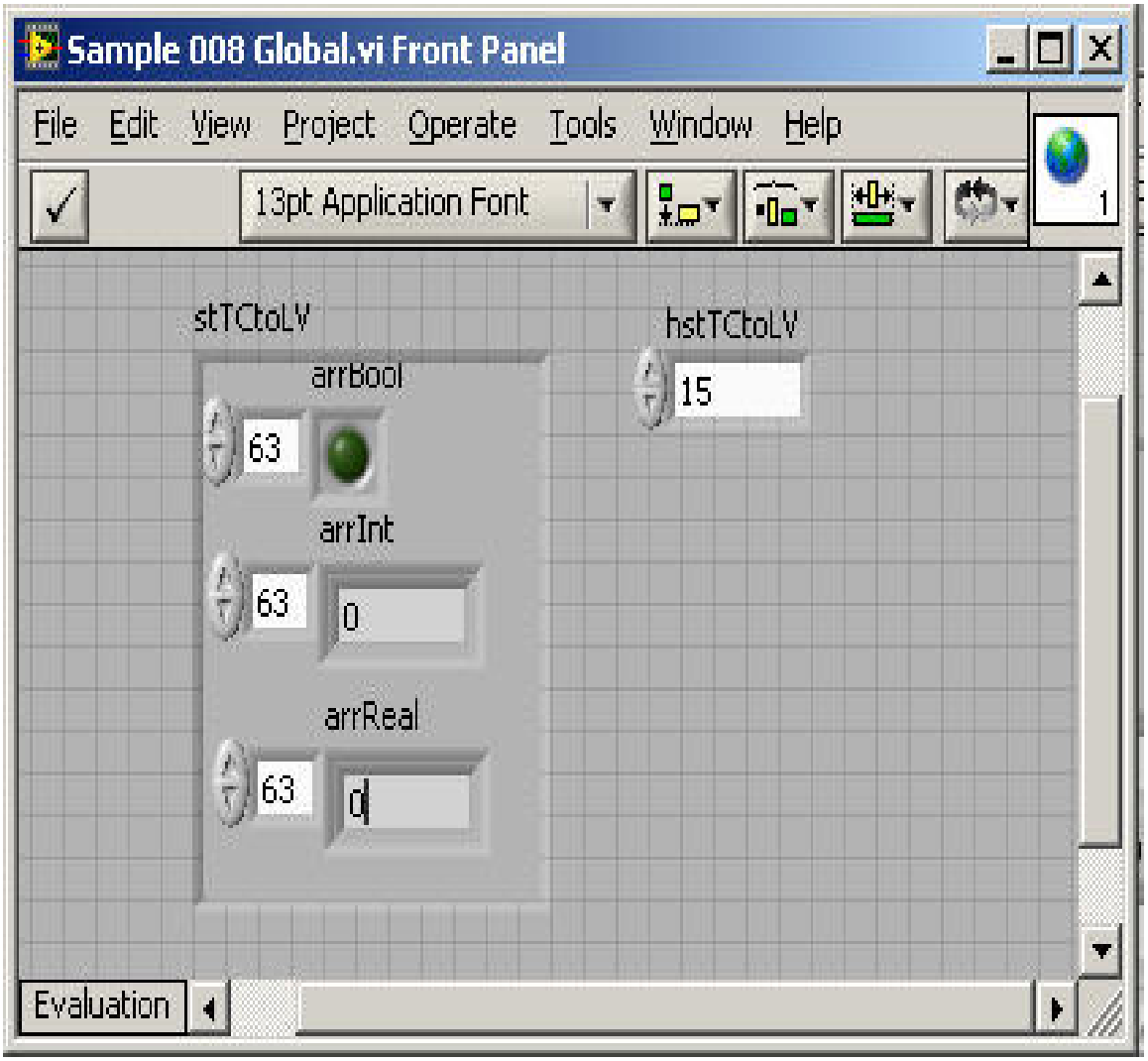
```
TYPE ST_DataExchange:
   STRUCT arrBool:
       ARRAY[0..63] OF BOOL;
       arrInt  : ARRAY[0..63] OF INT;
       arrReal : ARRAY[0..63] OF REAL;
   END_STRUCT
END_TYPE
```

```
   stTCtoLV AT%MB1000: ST_DataExchange;
```

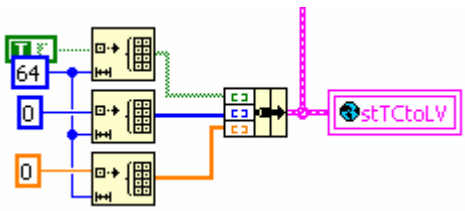
1. Callback-Vi für die Ereignismethode **AdsReadConvertConnectUpdate** registrieren



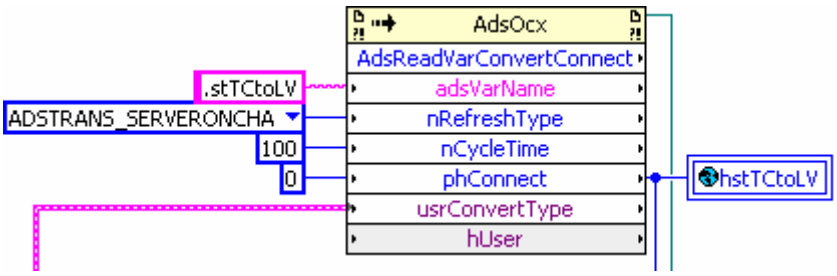
2. Globale Variablen:
   - Cluster-Variable als Abbildung der TwinCAT-Struktur erstellen.
   - globale Handle-Variable für die Unterscheidung der Events



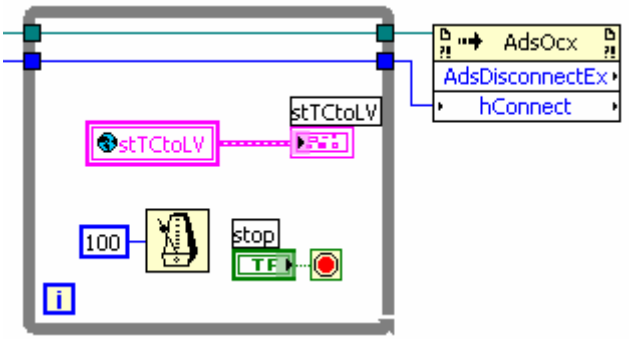
3. Initialisieren der Datenstruktur als Abbildung der TwinCAT-Struktur



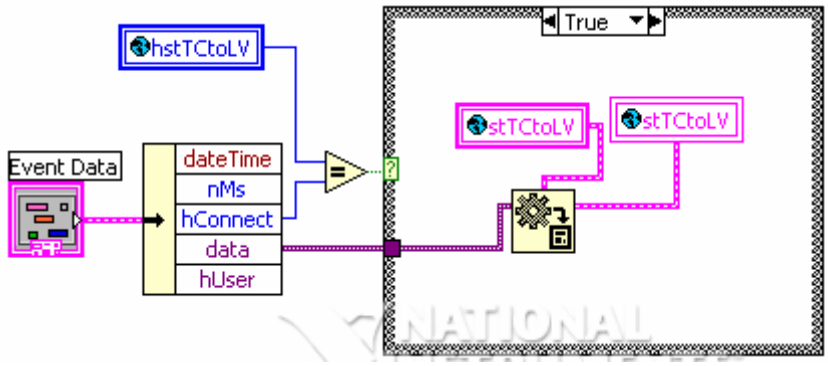
4. Herstellen der Datenverbindung und speichern des Handles der Verbindung



5. Zyklischer Zugriff auf die globalen Daten und Löschen der Verbindung



6. Eventbehandlung im Callback-Vi
   Anhand des übergebenen Handles hConnect kann das Callback-Vi entscheiden, für welche Variable das Event aufgerufen wurde und den in Data übergebenen Wert der richtigen Variablen zuweisen.

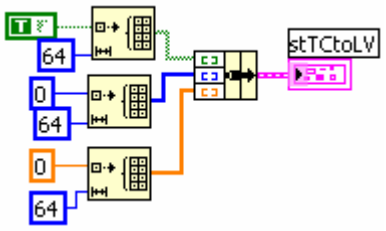


5.3.10 Ereignisgesteuertes Lesen mit Daten-Referenzübergabe an Callback-vi

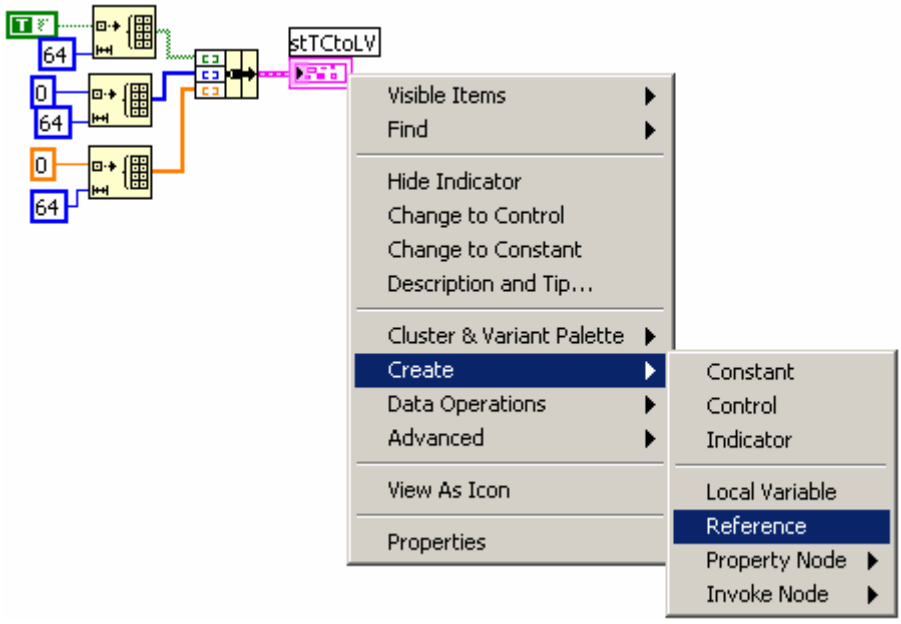
Wenn nur eine Variable per Connect gelesen wird, kann die Referenz auf die Variable an das Callback-Vi übergeben werden. Die Verwendung von globalen Variablen erübrigt sich dadurch. Das Callback-vi schreibt per Referenz direkt auf die Variable des aufrufenden Vi.

Beispieldateien: https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967702155.zip

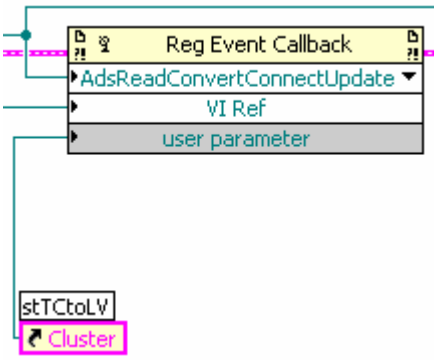
1. Ein LabVIEW™-Anzeigeelement vom richtigen Typ wird erstellt und initialisiert



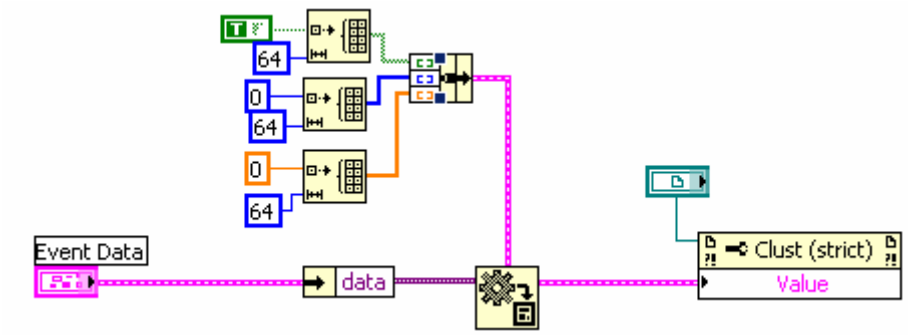
2. Erstellen der Referenz auf das Element



3. Übergabe der Referenz an das CallbackVi



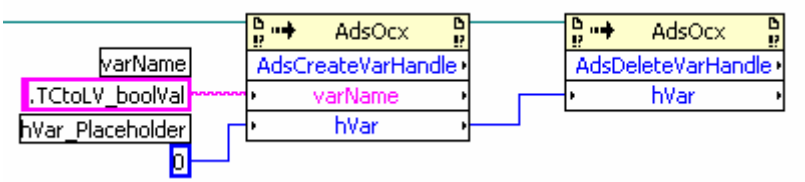
4. Zugriff auf die Referenzvariable im Callback-Vi
   Die Typenlose Variant-Variable muss in den richtigen Datentyp gewandelt werden, und dann an die Referenz-Variable übergeben.



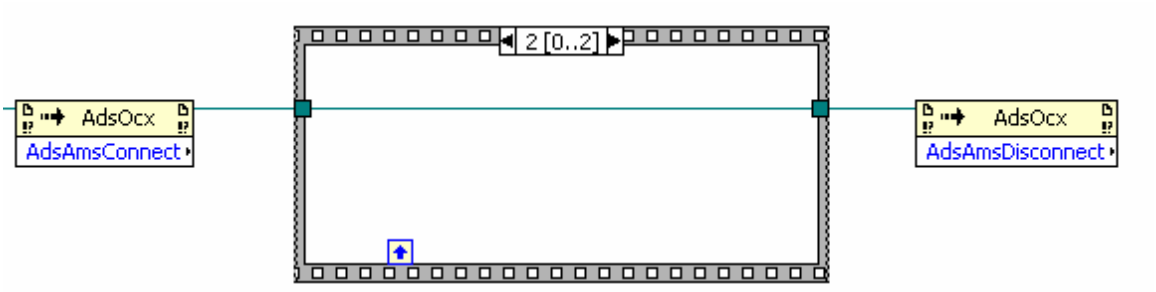
5.3.11 Allgemeine Methoden

1. Methoden **AdsCreateVarHandle** und **AdsDeleteVarHandle** werden verwendet, um auf PLC-Variablen per Name zuzugreifen
   PLC-Deklaration:

TCtoLV_boolVal AT%MX0.0: BOOL;
LabVIEW™:

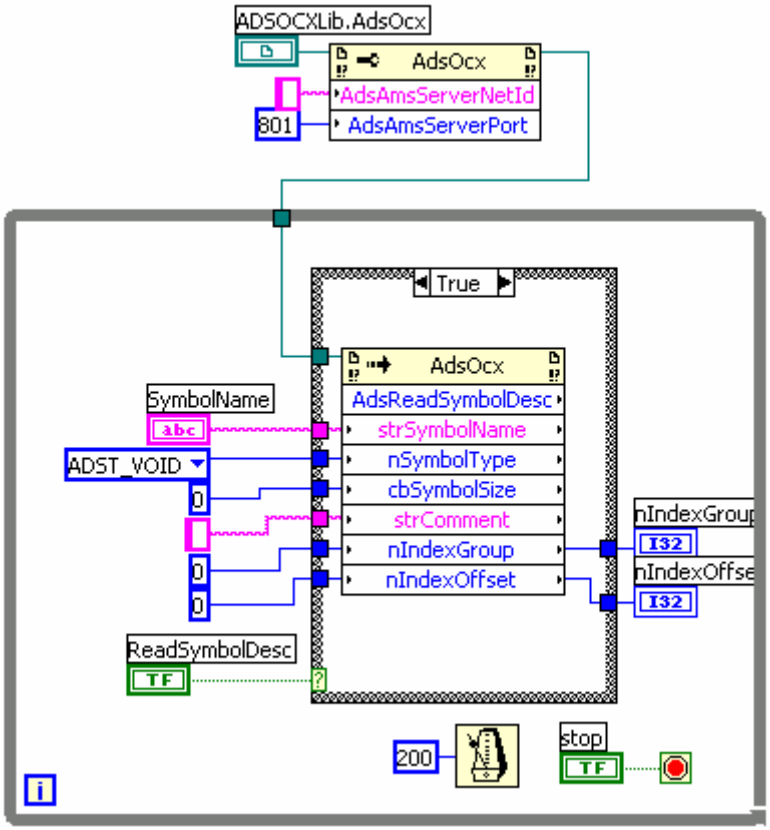


2. Methoden **AdsAmsConnect** und **AdsAmsDisconnect**
   Werden in der Start- bzw. Endphase aufgerufen, um das AdsOcx am Router an- bzw. vom Router abzumelden.



   Wenn das AdsOcx per AdsAmsDisconnect vom Router abgemeldet wurde, muss vor dem nächsten Aufruf einer AdsOcx-Methode AdsAmsConnect aufgerufen werden oder LabVIEW™ neu gestartet werden.

3. Die Methode **AdsReadSymbolDesc**
   Mit der Methode AdsReadSymbolDesc lassen sich zur Laufzeit Informationen zu einer benannten SPS-Variablen auslesen. So können z.B. die Adressdaten nIndexGroup und nIndexOffset gelesen werden, um dann per Adresse (evtl. auch mit der TcAdsDll) auf die Variable zuzugreifen. (s. https://infosys.beckhoff.com/content/1031/tcadsocx/Resources/11967703563.zip)


ADSOCXLib.AdsOcx
AdsOcx
AdsAmsServerNetId
801
AdsAmsServerPort
True
AdsOcx
AdsReadSymbolDesc
strSymbolName
nSymbolType
cbSymbolSize
strComment
nIndexGroup
nIndexOffset
SymbolName
abc
ADST_VOID
0
0
0
ReadSymbolDesc
TF
nIndexGroup
I32
nIndexOffse
I32
200
stop
TF

6 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 144]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 144]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 145]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 146]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.

Hex	Dec	HRESULT	Name	Beschreibung
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.

Hex	Dec	HRESULT	Name	Beschreibung
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.

Hex	Dec	HRESULT	Name	Beschreibung
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

Mehr Informationen:
www.beckhoff.de/automation

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