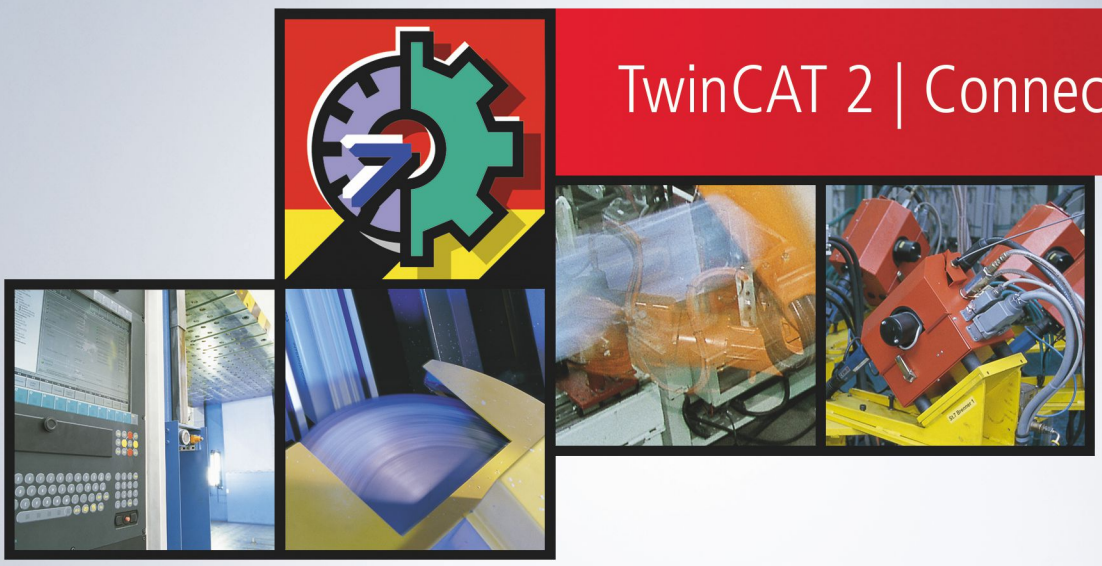
BECKHOFF New Automation Technology

Handbuch | DE

TX1000

TwinCAT 2 | ADS-DLL



07.09.2023 | Version: 1.2

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

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

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

Disclaimer

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

Marken

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

Patente

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

EtherCAT®

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

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

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.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

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 Einleitung

Die **TcAdsDll** stellt Funktionen zur Kommunikation mit anderen ADS-Geräten zur Verfügung.

- Kommunikation zu lokalen **TwinCAT Systemen** oder Remote **TwinCAT Systemen** über den **TwinCAT Message** Router.
- Kommunikation zu Remote **TwinCAT Systemen** über **TCP/IP** für **Win32 Systeme**.

Die TCAdsDll stellt die TwinCAT ADS Client Funktionen zur Verfügung. Diese Funktionen werden für 2 unterschiedliche Wege angeboten:

- Über C API [▶ 11]
- Über COM Interfaces [▶ 34]

Es wird empfohlen, die Bibliothek mit der kostenlosen TwinCAT-CP-Version zu verwenden.

3 API

Die TcAdsDll stellt Funktionen zur Verfügung, um mit anderen ADS-Geräten über den TwinCAT-Router mit dem C API Interface zu kommunizieren. Weitere Informationen zu ADS finden Sie unter TwinCAT ADS.

Die meisten in dieser Anleitung gezeigten Beispiele sind in Visual C++ Service Pack 3 realisiert. Da keines der Beispiele sehr komplex ist, sind tiefgreifende Kenntnisse in C/C++ nicht erforderlich.

3.1 Funktionen

3.1.1 AdsGetDllVersion

Liefert die Versionnummer, Revisionnummer und die Buildnummer der ADS-DLL zurück.

```
LONG AdsGetDllVersion(
 void
);
```

Parameter

-

Rückgabewert

Der Rückgabewert vom Type long enthält in codierter Form die drei oben genannten Angaben über die ADS-DLL.

Beispiel

Siehe Beispiel 1: DLL-Version auslesen [▶ 46].

3.1.2 AdsPortOpen

Stellt eine Verbindung (Kommunikationsport) zum TwinCAT Messagerouter her.

```
LONG AdsPortOpen(
 void
);
```

Parameter

-

Rückgabewert

Es wird die Portnummer zurückgegeben, die der ADS-Router dem Programm zugewiesen hat.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

3.1.3 AdsPortClose

Die Verbindung (Kommunikationsport) zum TwinCAT Messagerouter wird beendet.

```
LONG AdsPortClose(
 void
);
```

Parameter

-

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

3.1.4 AdsGetLocalAddress

Liefert die eigene NetId und die eigene Portnummer zurück.

```
LONG AdsGetLocalAddress(
  PAmsAddr  pAddr
);
```

Parameter

pAddr

[out] Zeiger auf die Struktur vom Typ AmsAddr [▶ 30].

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.1.5 AdsSyncWriteReq

Schreibt Daten synchron in ein ADS-Gerät.

```
LONG AdsSyncWriteReq(
  PAmsAddr  pAddr,
  ULONG     nIndexGroup,
  ULONG     nIndexOffset,
  ULONG     nLength,
  PVOID     pData
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Länge der Daten in Byte, die in den ADS-Server geschrieben werden.

pData

[in] Zeiger auf Daten, die in den ADS-Server geschrieben werden.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.1.6 AdsSyncReadReq

Liest Daten synchron aus einem ADS-Server.

```
LONG AdsSyncReadReq(
  PAmsAddr  pAddr,
  ULONG     nIndexGroup,
  ULONG     nIndexOffset,
  ULONG     nLength,
  PVOID     pData
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Länge der Daten in Byte.

pData

[out] Zeiger auf Datenbuffer welche die Daten aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 3: Synchron aus der SPS lesen [▶ 47].

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.1.7 AdsSyncReadReqEx

Liest Daten synchron aus einem ADS-Server.

```
LONG AdsSyncReadReqEx(
  PAmsAddr  pAddr,
  ULONG     nIndexGroup,
  ULONG     nIndexOffset,
  ULONG     nLength,
  PVOID     pData,
  ULONG*    pcbReturn);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Länge der Daten in Byte.

pData

[out] Zeiger auf Datenbuffer welcher die Daten aufnimmt.

pcbReturn

[out] Zeiger auf eine Variable. Diese Variable liefert bei Erfolg die Anzahl der tatsächlich gelesenen Datenbytes zurück.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.1.8 AdsSyncReadWriteReq

Schreibt Daten synchron in ein ADS-Server und bekommt von dem ADS-Gerät Daten wieder zurück.

```
LONG AdsSyncReadWriteReq(
  PAmsAddr  pAddr,
  ULONG     nIndexGroup,
  ULONG     nIndexOffset,
  ULONG     nReadLength,
  PVOID     pReadData,
  ULONG     nWriteLength,
  PVOID     pWriteData
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nReadLength

[in] Länge der Daten in Byte, die das ADS-Gerät zurückliefert.

pReadData

[out] Buffer mit Daten, die das ADS-Gerät zurückliefert.

nWriteLength

[in] Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden

pWriteData

[out] Buffer mit Daten, die in das ADS-Gerät geschrieben werden.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 7: Zugriff auf ein Array in der SPS [▶ 50]

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.1.9 AdsSyncReadWriteReqEx

Schreibt Daten synchron in ein ADS-Server und bekommt von dem ADS-Gerät Daten wieder zurück.

```
LONG AdsSyncReadWriteReqEx(
  PAmsAddr  pAddr,
  ULONG     nIndexGroup,
  ULONG     nIndexOffset,
  ULONG     nReadLength,
  PVOID     pReadData,
  ULONG     nWriteLength,
  PVOID     pWriteData,
  ULONG*    pcbReturn
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nReadLength

[in] Länge der Daten in Byte, die das ADS-Gerät zurückliefert.

pReadData

[out] Buffer mit Daten, die das ADS-Gerät zurückliefert.

nWriteLength

[in] Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden

pWriteData

[out] Buffer mit Daten, die in das ADS-Gerät geschrieben werden.

pcbReturn

[out] Zeiger auf eine Variable. Diese Variable liefert bei Erfolg die Anzahl der tatsächlich gelesenen Datenbytes zurück.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.1.10 AdsSyncReadDeviceInfoReq

Liest die Bezeichnung und die Versionsnummer von einem ADS-Server.

```
LONG AdsSyncReadDeviceInfoReq(
  PAmsAddr      pAddr,
  PCHAR         pDevName,
  PAdsVersionp  Version
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

pDevName

[out] Zeiger auf eine Zeichenkette, welche der Name von dem ADS-Gerät aufnimmt.

pVersion

[out] Adresse einer Variablen vom Typ AdsVersion [▶ 31], welche die Versionsnummer, Revisionsnummer und die Buildnummer aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 5: ADS-Status auslesen [▶ 48].

Sehen Sie dazu auch

- AmsAddr [▶ 30]
- AdsVersion [▶ 31]

3.1.11 AdsSyncWriteControlReq

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

```
LONG AdsSyncWriteControlReq(
  PAmsAddr pAddr,
  USHORT   nAdsState,
  USHORT   nDeviceState,
  ULONG    nLength,
  PVOID    pData
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nAdsState

[in] neuer ADS-Status.

nDeviceState

[in] neuer Geräte-Status.

nLength

[in] Länge der Daten in Byte.

pData

[in] Zeiger auf Daten die zusätzlich zum Ads-Gerät geschickt werden.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

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 unterschieden zwischen den Status des Gerätes selbst (DeviceState) und dem Status der ADS-Schnittstelle von dem ADS-Gerät (AdsState). Die möglichen Zustände, die die ADS-Schnittstelle annehmen kann, ist durch die ADS-Spezifikation festgelegt

Beispiel

Siehe Beispiel 6: SPS starten/stoppen [▶ 49].

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.1.12 AdsSyncReadStateReq

Liest den ADS-Status und den Geräte-Status von einem ADS-Server.

```
LONG AdsSyncReadStateReq(
  PAmsAddr pAddr,
  USHORT   *pAdsState,
  PUSHORT  pDeviceState
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

pAdsState

[out] Adresse einer Variablen, welche den ADS-Status aufnimmt (Siehe Datentyp ADSSTATE [▶ 33]).

pDeviceState

[out] Adresse einer Variablen, welche den Geräte-Status aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Anmerkungen

Jedes ADS-Gerät kann seinen aktuellen Zustand anderen ADS-Geräten mitteilen. Dabei wird unterschieden zwischen den Status des Gerätes selbst (DeviceState) und dem Status der ADS-Schnittstelle von dem ADS-Gerät (AdsState). Die möglichen Zustände, die die ADS-Schnittstelle annehmen kann, ist durch die ADS-Spezifikation festgelegt.
In Beispiel 11 [▶ 55] wird gezeigt, wie mit Hilfe einer Callback-Funktion die Änderung erkannt werden kann.

Beispiel

Siehe Beispiel 4: ADS-Status auslesen [▶ 48].

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.1.13 AdsSyncAddDeviceNotificationReq

Innerhalb eines ADS-Servers (z.B. SPS) wird eine Notification (Bekanntmachung) definiert. Beim Eintreten bestimmter Ereignisse, wird eine Funktion (Callbackfunktion) im ADS-Client (C-Programm) aufgerufen.

```
LONG AdsSyncAddDeviceNotificationReq(
  PAmsAddr                pAddr,
  ULONG                   nIndexGroup,
  ULONG                   nIndexOffset,
  PAdsNotificationAttrib  pNoteAttrib,
  PAdsNotificationFuncEx  pNoteFunc,
  ULONG                   hUser,
  PULONG                  pNotification
);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] IndexGroup.

nIndexOffset

[in] IndexOffset.

pNoteAttrib

[in] Zeiger auf die Struktur [▶ 31], welche weitere Informationen enthält.

pNoteFunc

[in] Name der Callback-Funktion [▶ 21].

hUser

[in] 32-Bit Wert, welcher an die Callback-Funktion übergeben wird.

pNotification

[out] Adresse der Variablen, die das Handle der Notification aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Limitierung :

Pro ADS-Port steht eine begrenzte Anzahl von 550 Notifications zur Verfügung.

Anmerkungen

Wird der TwinCAT-Router gestoppt und wieder gestartet, so sind die Notifications ungültig. Mit der Funktion AdsAmsRegisterRouterNotification() [▶ 20] können Sie dieses Ereignis abfangen.

Beispiel

Siehe Beispiel 8: Ereignisgesteuertes Lesen [▶ 51]

Sehen Sie dazu auch

- AmsAddr [▶ 30]
- AdsNotificationAttrib [▶ 31]
- PAdsNotificationFuncEx [▶ 21]

3.1.14 AdsSyncDelDeviceNotificationReq

Eine zuvor definierte Notification (Bekanntmachung) wird in einem ADS-Server gelöscht.

```
LONG AdsSyncDelDeviceNotificationReq(
PAmsAddrpAddr,
  ULONGhNotification);
```

Parameter

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

hNotification

[out] Adresse der Variablen, die das Handle der Notification enthält.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 8: Ereignisgesteuertes Lesen [▶ 51]

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.1.15 AdsSyncSetTimeout

Verändert die Timeoutzeit für die ADS-Funktionen. Der Standardwert ist 5000ms.

```
LONG AdsSyncSetTimeout(
  LONG   nMs
);
```

Parameter

nMs

[in] Timeoutzeit in ms.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

-

3.1.16 AdsAmsRegisterRouterNotification

Mit Hilfe der Funktion AdsAmsRegisterNotificationReq() kann eine Statusänderung des TwinCAT-Routers erkannt werden. Bei jeder Statusänderung wird die angegebende Callback-Funktion aufgerufen. Durch die Funktion AdsAmsUnRegisterNotification() [▶ 20] wird die Statusüberwachung des Routers wieder beendet.

```
LONG AdsAmsRegisterRouterNotification(
PAmsRouterNotificationFuncEx   pNoteFunc,
);
```

Parameter

pNoteFunc

[in] Name der Callback-Funktion [▶ 21]

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Bemerkungen:

- Implementiert ab TcAdsDLL-File-Version: 2.8.0.21 (ausgeliefert mit TwinCAT 2.9 Build > 941).
- Eine Verbindung zum TwinCAT-Router kann nur dann aufgebaut werden, wenn auf dem lokalem PC auch TwinCAT installiert wurde. Auf einem System ohne TwinCAT liefert die Funktion einen Fehler zurück.

Beispiel

Sehen Sie dazu auch

- PAmsRouterNotificationFuncEx [▶ 21]

3.1.17 AdsAmsUnRegisterRouterNotification

Durch die Funktion AdsAmsUnRegisterNotification() wird die Statusüberwachung des TwinCAT-Routers beendet. Siehe auch AdsAmsRegisterNotificationReq() [▶ 20].

```
LONG AdsAmsUnRegisterRouterNotification(
  void
);
```

Parameter

-

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Bemerkungen:

- Implementiert ab TcAdsDLL-File-Version: 2.8.0.21 (ausgeliefert mit TwinCAT 2.9 Build > 941).

- Eine Verbindung zum TwinCAT-Router kann nur dann aufgebaut werden, wenn auf dem lokalem PC auch TwinCAT installiert wurde. Auf einem System ohne TwinCAT liefert die Funktion einen Fehler zurück.

Beispiel

-

3.1.18 PAmsRouterNotificationFuncEx

Typdefinition der Callback-Funktion, die von der Funktion AdsAmsRegisterRouterNotification [▶ 20] benötigt wird.

```
typedef void  ( __stdcall *PAmsRouterNotificationFuncEx)( long nEvent );
```

3.1.19 PAdsNotificationFuncEx

Typdefinition der Callback-Funktion, die von der Funktion AdsSyncAddDeviceNotificationReq [▶ 18] benötigt wird.

```
typedef void (__stdcall *PAdsNotificationFuncEx)
(AmsAddr* pAddr, AdsNotificationHeader* pNotification, unsigned long hUser );
```

3.2 Erweiterte Funktionen

Mit den bisherigen Funktionen konnte nur ein Ads-Port pro Prozess kreiert werden. Dies ist vor allem für Mulithreaded Anwendungen nicht ausreichend, da die einzelnen Ads Kommandos sich gegenseitig blockieren. Mit den neuen Funktionen ist es nun möglich mehr als einen Port zu verwenden. Damit könnte z.B. pro Thread ein Ads-Port verwendet werden. Mit Hilfe der Funktion AdsPortOpenEx können neue Ports geöffnet werden. Die zurückgelieferte Port-Nummer wird dann als Parameter an die einzelnen Sync-Funktionen übergeben.

3.2.1 AdsPortOpenEx

Stellt eine Verbindung (Kommunikationsport) zum TwinCAT Messagerouter her. Im Gegensatz zu AdsPortOpen wird jedes Mal ein neuer Ads-Port geöffnet. Um mit diesem Port zu kommunizieren, müssen die erweiterten Ads-Funktionen verwendet werden. Diesen Funktionen wird die von AdsPortOpenEx zurückgegebene Port-Nummer als Parameter übergeben. Wenn kein TwinCAT-MessageRouter vorhanden ist wird die Funktion AdsPortOpenEx fehlschlagen.

```
LONG AdsPortOpenEx(
 void
);
```

Parameter

-

Rückgabewert

Portnummer des geöffneten Ads-Portes. Eine Rückgabewert von 0 bedeutet, dass der Aufruf fehlgeschlagen ist.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

3.2.2 AdsPortCloseEx

Die Verbindung (Kommunikationsport) zum TwinCAT Messagerouter wird beendet. Der Port der geschlossen werden soll, muss vorher durch einen Aufruf von AdsPortOpenEx geöffnet worden sein.

```
LONG AdsPortCloseEx(
  long nPort
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der vorher mit AdsPortOpenEx [▶ 21] geöffnet wurde.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

3.2.3 AdsGetLocalAddressEx

Liefert die eigene NetId und die eigene Portnummer zurück.

```
LONG AdsGetLocalAddressEx(
  long     port,
  PAmsAddr pAddr
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[out] Zeiger auf die Struktur vom Typ AmsAddr [▶ 30].

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.2.4 AdsSyncWriteReqEx

Schreibt Daten synchron in ein ADS-Gerät.

```
LONG AdsSyncWriteReqEx(
  LONG   port,PAmsAddrpAddr,
  ULONG  nIndexGroup,
  ULONG  nIndexOffset,
  ULONG  nLength,
  PVOID  pData
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Länge der Daten in Byte, die in den ADS-Server geschrieben werden.

pData

[in] Zeiger auf Daten, die in den ADS-Server geschrieben werden.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 2: Synchron in die SPS schreiben [▶ 46].

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.2.5 AdsSyncReadReqEx2

Liest Daten synchron aus einem ADS-Server.

```
LONG AdsSyncReadReqEx2(
  LONG    port,PAmsAddrpAddr,
  ULONG   nIndexGroup,
  ULONG   nIndexOffset,
  ULONG   nLength,
  PVOID   pData,
  ULONG*  pcbReturn
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Länge der Daten in Byte.

pData

[out] Zeiger auf Datenbuffer welcher die Daten aufnimmt.

pcbReturn

[out] Zeiger auf eine Variable. Diese Variable liefert bei Erfolg die Anzahl der tatsächlich gelesenen Datenbytes zurück.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.2.6 AdsSyncReadWriteReqEx2

Schreibt Daten synchron in ein ADS-Server und bekommt von dem ADS-Gerät Daten wieder zurück.

```
LONG AdsSyncReadWriteReqEx2(
  LONG      port,
  PAmsAddr  pAddr,
  ULONG     nIndexGroup,
  ULONG     nIndexOffset,
  ULONG     nReadLength,
  PVOID     pReadData,
  ULONG     nWriteLength,
  PVOID     pWriteData,
  ULONG*    pcbReturn
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nReadLength

[in] Länge der Daten in Byte, die das ADS-Gerät zurückliefert.

pReadData

[out] Buffer mit Daten, die das ADS-Gerät zurückliefert.

nWriteLength

[in] Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden

pWriteData

[out] Buffer mit Daten, die in das ADS-Gerät geschrieben werden.

pcbReturn

[out] Zeiger auf eine Variable. Diese Variable liefert bei Erfolg die Anzahl der tatsächlich gelesenen Datenbytes zurück.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.2.7 AdsSyncReadDeviceInfoReqEx

Liest die Bezeichnung und die Versionsnummer von einem ADS-Server.

```
LONG AdsSyncReadDeviceInfoReqEx(
  LONG          port,
  PAmsAddr      pAddr,
  PCHAR         pDevName,
  PAdsVersion   pVersion
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

pDevName

[out] Zeiger auf eine Zeichenkette, welche der Name von dem ADS-Gerät aufnimmt.

pVersion

[out] Adresse einer Variablen vom Typ AdsVersion [▶ 31], welche die Versionsnummer, Revisionsnummer und die Buildnummer aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 5: ADS-Informationen auslesen [▶ 48].

Sehen Sie dazu auch

- AmsAddr [▶ 30]
- AdsVersion [▶ 31]

3.2.8 AdsSyncWriteControlReqEx

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

```
LONG AdsSyncWriteControlReqEx(
  LONG      port,
  PAmsAddr  pAddr,
```

```
  USHORT    nAdsState,
  USHORT    nDeviceState,
  ULONG     nLength,
  PVOID     pData
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nAdsState

[in] neuer ADS-Status.

nDeviceState

[in] neuer Geräte-Status.

nLength

[in] Länge der Daten in Byte.

pData

[in] Zeiger auf Daten die zusätzlich zum Ads-Gerät geschickt werden.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

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 unterschieden zwischen den Status des Gerätes selbst (DeviceState) und dem Status der ADS-Schnittstelle von dem ADS-Gerät (AdsState). Die möglichen Zustände, die die ADS-Schnittstelle annehmen kann, ist durch die ADS-Spezifikation festgelegt

Beispiel

Siehe Beispiel 6: SPS starten/stoppen [▶ 49].

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.2.9 AdsSyncReadStateReqEx

Liest den ADS-Status und den Geräte-Status von einem ADS-Server.

```
LONG AdsSyncReadStateReqEx(
  LONG      port,
  PAmsAddr  pAddr,
  USHORT    *pAdsState,
  PUSHORT   pDeviceState
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

pAdsState

[out] Adresse einer Variablen, welche den ADS-Status aufnimmt (Siehe Datentyp ADSSTATE [▶ 33]).

pDeviceState

[out] Adresse einer Variablen, welche den Geräte-Status aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Anmerkungen

Jedes ADS-Gerät kann seinen aktuellen Zustand anderen ADS-Geräten mitteilen. Dabei wird unterschieden zwischen den Status des Gerätes selbst (DeviceState) und dem Status der ADS-Schnittstelle von dem ADS-Gerät (AdsState). Die möglichen Zustände, die die ADS-Schnittstelle annehmen kann, ist durch die ADS-Spezifikation festgelegt.
In Beispiel 11 [▶ 55] wird gezeigt, wie mit Hilfe einer Callback-Funktion die Änderung erkannt werden kann.

Beispiel

Siehe Beispiel 4: ADS-Status auslesen [▶ 48].

Sehen Sie dazu auch

- AmsAddr [▶ 30]

3.2.10 AdsSyncAddDeviceNotificationReqEx

Innerhalb eines ADS-Servers (z.B. SPS) wird eine Notification (Bekanntmachung) definiert. Beim Eintreten bestimmter Ereignisse, wird eine Funktion (Callbackfunktion) im ADS-Client (C-Programm) aufgerufen.

```
LONG AdsSyncAddDeviceNotificationReqEx(
  LONG                    port,
  PAmsAddr                pAddr,
  ULONG                   nIndexGroup,
  ULONG                   nIndexOffset,
  PAdsNotificationAttrib  pNoteAttrib,
  PAdsNotificationFuncEx  pNoteFunc,
  ULONG                   hUser,
  PULONG                  pNotification
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

nIndexGroup

[in] IndexGroup.

nIndexOffset

[in] IndexOffset.

pNoteAttrib

[in] Zeiger auf die Struktur [▶ 31], welche weitere Informationen enthält.

pNoteFunc

[in] Name der Callback-Funktion [▶ 21].

hUser

[in] 32-Bit Wert, welcher an die Callback-Funktion übergeben wird.

pNotification

[out] Adresse der Variablen, die das Handle der Notification aufnimmt.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Limitierung :

Pro ADS-Port steht eine begrenzte Anzahl von 550 Notifikations zur Verfügung.

Anmerkungen

Wird der TwinCAT-Router gestoppt und wieder gestartet, so sind die Notifications ungültig. Mit der Funktion AdsAmsRegisterRouterNotification() [▶ 20] können Sie dieses Ereignis abfangen.

Beispiel

Siehe Beispiel 8: Ereignisgesteuertes Lesen [▶ 51]

Sehen Sie dazu auch

- AmsAddr [▶ 30]
- AdsNotificationAttrib [▶ 31]
- PAdsNotificationFuncEx [▶ 21]

3.2.11 AdsSyncDelDeviceNotificationReqEx

Eine zuvor definierte Notification (Bekanntmachung) wird in einem ADS-Server gelöscht.

```
LONG AdsSyncDelDeviceNotificationReqEx(
  LONG       port,
  PAmsAddr   pAddr,
  ULONG      hNotification
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pAddr

[in] Struktur [▶ 30] mit NetId und Portnummer vom ADS-Server.

hNotification

[out] Adresse der Variablen, die das Handle der Notification enthält.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

Siehe Beispiel 8: Ereignisgesteuertes Lesen [▶ 51]

Sehen Sie dazu auch

AmsAddr [▶ 30]

3.2.12 AdsSyncSetTimeoutEx

Verändert die Timeoutzeit für die ADS-Funktionen. Der Standardwert ist 5000ms.

```
LONG AdsSyncSetTimeoutEx(
  LONG   port,
  LONG   nMs
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

nMs

[in] Timeoutzeit in ms.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

-

3.2.13 AdsSyncGetTimeoutEx

Liefert die eingestellte Timeoutzeit für die ADS-Funktionen. Der Standardwert ist 5000ms.

```
LONG AdsSyncGetTimeoutEx(
  LONG       port,
  LONG*      pnMs
);
```

Parameter

port

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pnMs

[out] Puffer für die Timeoutzeit in ms.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

-

3.2.14 AdsAmsPortEnabledEx

Liefert den Status der ADS-Client-Verbindung.

```
LONG AdsAmsPortEnabledEx(
  LONG       nPort,
  BOOL*      pbEnabled
);
```

Parameter

nPort

[in] Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx [▶ 21] oder AdsPortOpen [▶ 11] geöffnet worden ist.

pbEnabled

[out] Puffer für den Statuswert.

Rückgabewert

Gibt den Fehlerstatus der Funktion zurück.

Beispiel

-

3.3 Strukturen

3.3.1 AmsAddr

In dieser Struktur kann die vollständige Adresse eines ADS-Gerätes abgelegt werden.

```
typedef struct {
  AmsNetId  netId;
  USHORT    port;
} AmsAddr, *PAmsAddr;
```

Elemente

NetId NetId [▶ 30].

port Portnummer.

3.3.2 AmsNetId

Die NetId eines ADS-Gerätes kann in dieser Struktur dargestellt werden.

```
typedef struct {
  UCHAR  b[6];
} AmsNetId, *PAmsNetId;
```

Elemente

b[6]

NetId, bestehend aus 6 Ziffern.

Bemerkung

Die Struktur besteht aus einem Array mit 6 Elementen vom Typ UCHAR. Jedes Element des Arrays darf die Werte von 1 bis 255 annehmen. Die NetId wird mit dem TwinCAT System Service eingestellt.

3.3.3 AdsVersion

Die Struktur beinhaltet die Versionsnummer, Revisionsnummer und Buildnummer.

```
typedef struct {
  UCHAR    version;
  UCHAR    revision;
  USHORT   build;
} AdsVersion, *PAdsVersion;
```

Elemente

version

Versionsnummer.

revision

Revisionsnummer.

build

Buildnummer.

3.3.4 AdsNotificationAttrib

Diese Struktur beinhaltet bei der Definition einer Notification sämtliche Attribute.

```
typedef struct {
  ULONG        cbLength;
  ADSTRANSMODE nTransMode;
  ULONG        nMaxDelay;
  ULONG        nCycleTime;
} AdsNotificationAttrib, *PAdsNotificationAttrib;
```

Elemente

cbLength

Länge der Daten, die an die Callback-Funktion übergeben werden sollen.

nTransMode: ADSTRANSMODE [▶ 33]

ADSTRANS_SERVERCYCLE: Die Callback-Funktion der Notification wird zyklisch aufgerufen.

ADSTRANS_SERVERONCHA: Die Callback-Funktion der Notification wird nur bei Änderung des Wertes aufgerufen.

nMaxDelay

Spätestens nach dieser Zeit wird die Callback-Funktion der Notification aufgerufen. Die Einheit ist in 100ns.

nCycleTime

In diesem Zeitinvall prüft der ADS-Server, ob sich die Variable verändert hat. Die Einheit ist in 100ns

Anmerkungen

Die Übertragung zwischen der Echtzeit und der ADS-DLL wird durch einen FIFO gepuffert. Jeder Wert, der per Callback-Funktion übertragen werden soll, wird zuvor von TwinCAT in den FIFO geschrieben. Ist der Puffer voll, oder ist die Zeit nMaxDelay abgelaufen, so wird für jeden Eintrag die Callback-Funktion aufgerufen. Dabei wird durch den Parameter nTransMode das Verhalten in folgender Weise beeinflusst:

ADSTRANS_SERVERCYCLE
Es werden in dem Zeitintervall nCycleTime die Werte zyklisch in den FIFO geschrieben. Der kleinste mögliche Wert für nCycleTime, ist die Zykluszeit des Ads-Servers; bei der SPS ist dieses die Zykluszeit der Task. Die Cycletime kann in 1ms-Schritten ausgewertet werden. Geben Sie als Cycletime 0ms an, so wird in jedem Zyklus der Task der Wert in den FIFO geschrieben.

ADSTRANS_SERVERONCHA
Es wird nur dann ein Wert in den FIFO geschrieben, wenn sich dieser geändert hat. Die Abtastung in der Echtzeit erfolgt in der Zeit, die in nCycleTime angegeben wird. Die Cycletime kann in 1ms-Schritten ausgewertet werden. Geben Sie als Cycletime 0ms an, so wird bei jeder Änderung die Variablen in den FIFIO geschrieben.

HINWEIS

Zu viele Leseoperationen

Durch zu viele Leseoperationen kann das System so stark ausgelastet werden, daß die Bedieneroberfläche stark verlangsamt wird.

- Setzen Sie die Cycletime auf möglichst angemessene Werte und beenden Sie jedesmal Verbindungen, wenn diese nicht mehr benötigt werden.

Sehen Sie dazu auch

ADSTRANSMODE [▶ 33]

3.3.5 AdsNotificationHeader

Diese Struktur wird an die Callback-Funktion mit übergeben.

```
typedef struct {
  ULONG    hNotification;
  __int64  nTimeStamp;
  ULONG    cbSampleSize;
  UCHAR    data[ANYSIZE_ARRAY];
} AdsNotificationHeader, *PAdsNotificationHeader;
```

Elemente

hNotification

Handel der Notification. Wird beim Definieren der Notification festgelegt;

nTimeStamp

Zeitstempel in FILETIME-Format.

cbSampleSize

Anzahl der Bytes die übertragen wurden.

data[ANY_SIZE_ARRAY]

Array mit den übertragenden Daten.

Bemerkung

Der Zeitstempel wird im FILETIME-Format übertragen. FILETIME ist eine 64-Bit Variable, welche in 100ns Schritten die Uhrzeit und das Datum beginnend vom 01.01.1601 darstellt. Die lokale Zeitverschiebung wird nicht berücksichtigt; es wird die Coordinated Universal Time (UTC) benutzt. Wollen Sie auf einzelne Elemente (Tag, Monat, Jahr, Stunde, Minute, Sekunde, usw.) zugreifen, müssen Sie den Zeitstempel vom FILETIME-Format in das SYSTEMTIME-Format umwandeln und anschließend die Zeit, unter Berücksichtigung des örtlichen Standorts (local time), umrechnen.

Beispiel

Siehe Beispiel 8: Ereignisgesteuertes Lesen [▶ 51]

3.4 Enums

3.4.1 ADSSTATE

```
typedef enum nAdsState {
  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,
  ADSSTATE_ERROR        = 11,
  ADSSTATE_SHUTDOWN     = 12,
  ADSSTATE_SUSPEND      = 13,
  ADSSTATE_RESUME       = 14,
  ADSSTATE_CONFIG       = 15,   // system is in config mode
  ADSSTATE_RECONFIG     = 16,   // system should restart in config mode
  ADSSTATE_MAXSTATES
} ADSSTATE;
```

3.4.2 ADSTRANSMODE

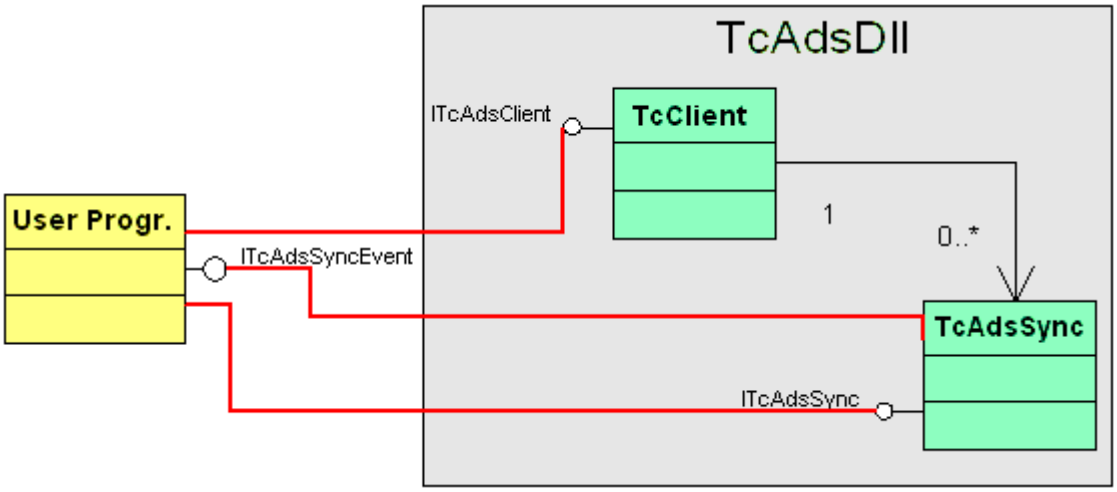
```
typedef enum nAdsTransMode {
  ADSTRANS_NOTRANS      = 0,
  ADSTRANS_CLIENTCYCLE  = 1,
  ADSTRANS_CLIENT1REQ   = 2,
  ADSTRANS_SERVERCYCLE  = 3,
  ADSTRANS_SERVERONCHA  = 4
} ADSTRANSMODE;
```

4 COM-API

Die TcAdsDll stellt Funktionen zur Kommunikation mit anderen ADS Geräten über die COM Schnittstelle des TwinCAT-Routers zu Verfügung. Weitere Informationen zu ADS finden Sie unter TwinCAT ADS.

Die COM Klasse TcClient [▶ 34] unterstützt die Anwenderprogramme, eine Verbindung von einem ADS Gerät zum lokalen PC oder zu einem Remote PC aufzubauen. Die TcAdsDll unterstützt ein multi-threaded Einwählmodell. Sie kann von multi-threaded und single-threaded COM clients verwendet werden. Wenn die TcAdsDll von single-threaded COM clients benutzt wird, werden die Methodenaufrufe durch einen Ordner synchronisiert. Der Ordner ist in der TcAdsDll kompiliert, es ist keine weitere proxy-stub-dll notwendig.

Der TcClient gibt für jede einzelne Verbindung zu einem ADS Gerät ein Objekt vom Typ TcAdsSync [▶ 34] zurück. Diese Klasse unterstützt die synchrone ADS Kommunikation zum ADS Gerät. Die Klasse TcAdsSync stellt die Funktionen über die Default Schnittstelle ITcAdsSync [▶ 36] zur Verfügung. Um die ADS Notification vom TcAdsSync Objekt zu erreichen, muss das Anwenderprogramm die Event Schnittstelle ITcAdsSyncEvent [▶ 41] implementieren und verbinden.



4.1 Classes

Die TcAdsDll stellt eine Schnittstelle nach außen durch COM (Component Object Model) zur Verfügung.

CoClasses	Beschreibung
TcClient [▶ 34]	Die Haupt-Klasse der TcAdsDll. Unterstützt eine class factory um eine ADS client Verbindung herzustellen.

4.1.1 TcClient

Das Objekt TcClient stellt eine class factory zur Erstellung einer ADS client Verbindung zur Verfügung.

Schnittstelle	Beschreibung
ITcClient [▶ 35]	Schnittstelle unterstützt eine class factory

4.1.2 TcAdsSync

Das Objekt TcAdsSync unterstützt die ADS Kommunikation zu einem ADS Gerät. TcAdsSync hat keine class factory und kann nur durch einen Aufruf erstellt werden.

An ITcClient::Connect [▶ 35]

Schnittstelle	Beschreibung
ITcAdsSync [▶ 36]	Schnittstelle, die die ADS Kommunikations-Funktionen bereitstellt.

4.2 Interfaces

4.2.1 ITcClient

Die ITcClient Schnittstelle stellt eine class factory zum Erstellen eines Objektes zur Kommunikation mit einem ADS Gerät zur Verfügung. Die Schnittstelle leitet sich von IUnknown ab.

IUnknown Methoden	Beschreibung
QueryInterface	Gibt einen Pointer zur Schnittstelle zurück, die angefragt wurde.
AddRef	Inkrementiert den Referenzzähler
Release	Dekrementiert den Referenzzähler

ITcClient Methoden	Beschreibung
Connect [▶ 35]	Erstellt ein Objekt vom Typ ITcAdsSync, das die synchrone ADS Kommunikation zu einem einzelnen ADS Gerät unterstützt.

4.2.1.1 Connect

Erstellt eine neues ADS client Kommunikations-Objekt für ein einzelnes ADS Gerät durch eine vorgegebene AdsNet Id und Port Nummer.

```
HRESULT Connect(
  AmsNetId*    pAmsNetId,
  long         nPort,
  ITcAdsSync** pipTcAdsSync
);
```

Parameter

pAmsNetId	[in] Variable präsentiert die Ams Net Id durch den Structurtyp AmsNetId. Wenn die Net Id auf 0.0.0.0.0.0 gesetzt ist, wird die Verbindung zum lokalen TwinCAT System hergestellt. Wenn auf dem Client PC kein TWinCAt System installiert ist, benutzt die Verbindung TCP/IP. Die Einschränkung bei der TCP/IP Verbindung ist, dass der Client nur Verbindungen zum Haupt Remote Gerät mit der AMS Net Id = TCP/IP address + 1.1. herstellen kann.
nPort	[in] ADS Port Nummer des ADS Geräts, mit dem kommuniziert werden soll.
ITcAdsSync	[out, retval] Gibt einen Pointer zu einem ITcAdsSync [▶ 36] Pointer zurück, der das Objekt enthält, das für die ADS Kommunikation verwendet wird.

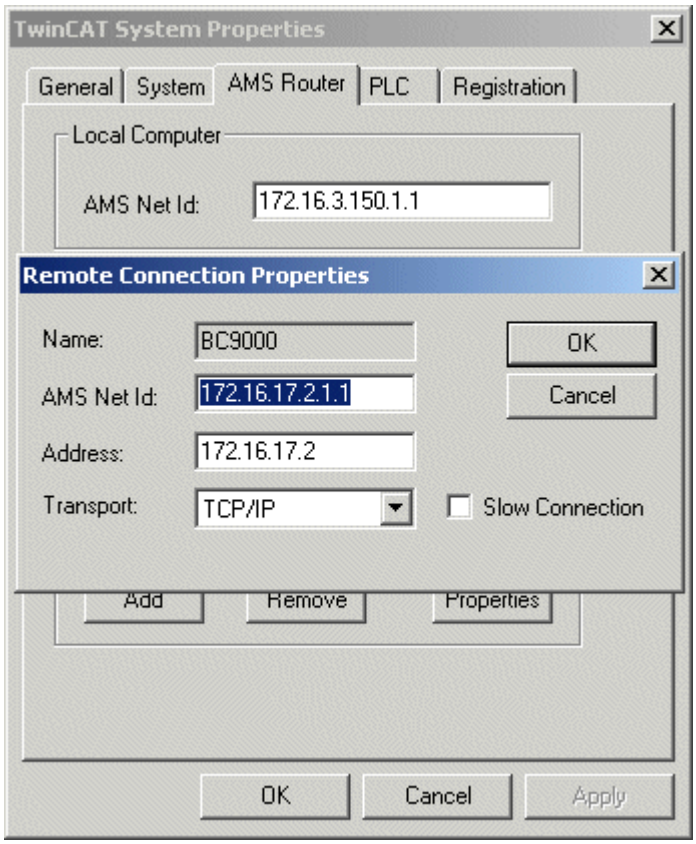
Rückgabe Werte

S_OK	Die Verbindungsfunktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

Um eine Verbindung zu einem Remote TwinCAT System herzustellen, muss das Remote Gerät auf die Liste der Remote Computer im TwinCAT System hinzugefügt werden. Wenn auf dem Client PC und dem Remote PC ein TwinCAT System installiert ist, muss der Client PC in die Liste der Remote Computer im Remote PC

eingetragen werden und umgekehrt. Wenn auf dem Client PC kein TwinCAT System installiert ist, muss der Client PC nur in die Liste der Remote Computer im Remote PC eingetragen werden. In diesem Fall besteht die AMS Net Id lediglich aus der TCP/IP Adresse + 1.1. .



4.2.2 ITcAdsSync

Die ITcClient Schnittstelle stellt einem Client die Funktionalität zur Kommunikation mit einem ADS Gerät zur Verfügung. Die Schnittstelle leitet sich von IUnknown ab.

IUnknown Methoden	Beschreibung
QueryInterface	Gibt einen Pointer zur angefragten Schnittstelle zurück
AddRef	Inkrementiert den Referenzzähler
Release	Dekrementiert den Referenzzähler

ITcAdsSync Methoden und Eigenschaften	Beschreibung
Write [▶ 37]	Schreibt einen Wert an eine Variable in einem ADS Gerät als Bytestream.
Read [▶ 37]	Liest den Wert einer Variablen aus einem ADS Gerät als Bytestream.
ReadWrite [▶ 38]	Schreibt einen Wert an ein ADS Gerät und liest das Ergebnis in einem Schritt.
WriteControl [▶ 38]	Setzt den Status des ADS Systems und Geräts.
AddDeviceNotification [▶ 39]	Verbindet eine Variable mit dem Client. Der Client wird durch ein Event angezeigt.
DelDeviceNotification [▶ 39]	Entfernt die Verbindung einer Variablen.
ReadDeviceInfo [▶ 40]	Liest Geräte Informationen aus dem ADS Gerät.
ReadState [▶ 40]	Liest den Status des ADS Geräts.
Timeout [▶ 40]	Setzt die Zeit, nach der der Client eine Timeout Warnung von allen übrigen ADS Kommandos erhält.

4.2.2.1 Write

Schreibt einen Wert an eine Variable in einem ADS Gerät als Bytestream.

```
HRESULT Write(indexGroup, indexOffset, cbLen, pData);
```

Parameter

indexGroup	[in] Variable vom Typ long die die Index Group der Variable, auf die geschrieben werden soll, enthält.
indexOffset	[in] Variable vom Typ long die den Index Offset der Variable, auf die geschrieben werden soll, enthält.
cbLen	[in] Anzahl der Bytes die in die Variable geschrieben werden sollen.
pData	[in, size_is(cbLen)] Pointer auf das erste Element eines Bytearrays mit der Länge **cbLen** der Daten, die auf eine Variable in einem ADS Gerät geschrieben werden sollen.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

4.2.2.2 Read

Liest den Wert einer Variable aus einem ADS Gerät als Bytestream.

```
HRESULT Read(
  long indexGroup,
  long indexOffset,
  long cbLen,
  long* pcbRead,
  [out, size_is(cbLen),
  length_is(*pcbRead)] byte* pData
);
```

Parameter

indexGroup	[in] Variable vom Typ long die die Index Group der Variable, die gelesen werden soll, enthält.
indexOffset	[in] Variable vom Typ long die den Index Offset der Variable, die gelesen werden soll, enthält.
cbLen	[in] Anzahl der Bytes die aus der Variablen gelesen werden sollen.
pcbRead	[out] Pointer auf eine Variable, die die Anzahl der (wirklich) gelesenen Bytes zurückgibt.
pData	[out, size_is(cbLen), length_is(*pcbRead)] Pointer auf das erste Element eines Bytearrays mit der Länge **cbLen** der Daten, die aus einer Variable in einem ADS Gerät gelesen werden sollen.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

4.2.2.3 ReadWrite

Schreibt einen Wert an ein ADS Gerät und liest das Ergebnis in einem Schritt.

```
HRESULT ReadWrite(
  long  indexGroup,
  long  indexOffset,
  long  cbRdLen,
  long* pcbRead,
  byte* pRdData,
  [in]  long cbWrLen,
  [in,  size_is(cbWrLen)] byte* pWrData
);
```

Parameter

indexGroup	[in] Variable vom Typ long die die Index Group der Variable, die gelesen werden soll, enthält.
indexOffset	[in] Variable vom Typ long die den Index Offset der Variable, die gelesen werden soll, enthält.
cbRdLen	[in] Anzahl der Bytes die aus der Variable gelesen werden sollen.
pcbRead	[out] Pointer auf eine Variable, die die Anzahl der (wirklich) gelesenen Bytes zurückgibt.
pRdData	[out, size_is(cbRdLen), length_is(*pcbRead)] Pointer auf das erste Element eines Bytearrays mit der Länge **cbRdLen** der Daten, die aus einer Variable in einem ADS Gerät gelesen werden sollen.
cbWrLen	[in] Anzahl der Bytes die in die Variable geschrieben werden sollen.
pWrData	[in, size_is(pWrData)] Pointer auf das erste Element eines Bytearrays mit der Länge **cbRdLen** der Daten, die auf eine Variable in einem ADS Gerät geschrieben werden sollen.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

4.2.2.4 WriteControl

Setzt den Status des ADS Systems und Geräts.

```
HRESULT WriteControl(
  ADSSTATE adsState,
  ADSSTATE deviceState,
  long     cbLen,
  byte*    pData
);
```

Parameters

adsState	[in] Der Status als ADSSTATE [▶ 43], der in das ADS system geschrieben werden soll.
deviceState	[in] Der Status als ADSSTATE [▶ 43], der in das ADS Gerät geschrtieben werden soll.
cbLen	[in] Anzahl der Bytes die in die Variable geschrieben werden sollen.
pData	[in, size_is(cbLen)] Pointer auf das erste Element eines Bytearrays mit der Länge **cbLen** der zusätzlichen Daten die in das ADS Gerät geschrieben werden sollen.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES	Ein Fehler tritt auf.

Sehen Sie dazu auch

ADSERRORCODES [▶ 42]

4.2.2.5 AddDeviceNotification

Verbindet eine Variable mit dem Client. Der Client wird durch ein Event angezeigt.

```
HRESULT AddDeviceNotification(
  long         indexGroup,
  long         indexOffset,
  long         cbLenData,
  ADSTRANSMODE transMode,
  long         nMaxDelay,
  long         nCycleTime,
  long*        phNotification
);
```

Parameter

indexGroup	[in] Variable vom Typ long die die Index Group der Variable, auf die geschrieben werden soll, enthält.
indexOffset	[in] Variable vom Typ long die den Index Offset der Variable, auf die geschrieben werden soll, enthält.
cbLenData	[in] Anzahl der Bytes die aus der verbundenen Variablen gelesen werden sollen.
transMode	[out] Modus, wie die Variable mit dem Typ ADSTRANSMODE [▶ 44] verbunden ist.
nMaxDelay	[in] Die Zeit mit einer Auflösung von 100 ns nach der ein callback der implementierten _ITcAdsSyncEvent [▶ 41] Schnittstelle erwartet wird.
nCycleTime	[in] Die Zeit mit einer Auflösung von 100 ns , wie die Variable gesammelt werden soll.
phNotification	[out, retval] Pointer auf das Handle, das allein die Verbindung zur Variablen kennzeichnet.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

Eine nCycleTime= 10000 und nMaxDelay=100000 würde alle 10 ms 10 Werte mit der Auflösung von 1 ms erhalten.

4.2.2.6 DelDeviceNotification

Entfernt die Verbindung einer Variablen, die vorher mit einer AddDeviceNotification [▶ 39] hergestellt wurde.

```
HRESULT DelDeviceNotification(
  long  hNotication
);
```

Parameter

phNotification	[in] Handle der vorher aufgebauten Verbindung.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

Eine nCycleTime= 10000 und nMaxDelay=100000 würde alle 10 ms 10 Werte mit der Auflösung von 1 ms erhalten.

4.2.2.7 ReadDeviceInfo

Liest Geräte Informationen aus dem ADS Gerät.

```
HRESULT ReadDeviceInfo(
  BSTR*       pName,
  AdsVersion* pVersion
);
```

Parameter

pName	[out] Variable die den BSTR String, der das ADS Gerät beschreibt, enthält.
pVersion	[out] Pointer auf eine Variable des Types AdsVersion [▶ 42] , welche die Versionsnummer enthält.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

4.2.2.8 ReadState

Liest den Status des ADS Geräts und des ADS Systems.

```
HRESULT ReadState(
  ADSSTATE* pAdsState,
  ADSSTATE* pDeviceState
);
```

Parameter

pAdsState	[out] Pointer auf eine Variable des Typs ADSSTATE [▶ 43] die den Status des ADS Systems enthält.
pDeviceState	[out] Pointer auf eine Variable des Typs ADSSTATE [▶ 43] die den Status des ADS Geräts enthält.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Anmerkungen

4.2.2.9 get_Timeout

Diese Eigenschaft wird benutzt, um den Timeout Wert in Millisekunden für alle ADS Funktionen der ITcAdsSync [▶ 36] Schnittstelle zuzuweisen oder abzurufen.

Abrufen des Timeout Wertes.

```
HRESULT get_Timeout(long *pTime);
```

Parameters

pTime	[out, retval] Pointer auf eine Variable die den aktuellen Timeout Wert enthält.

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

Neuen Timeout Wert zuweisen

HRESULT put_Timeout(long nTime);**Parameter**

pTime[in] Variable die den neuen Timeout Wert enthält.**Rückgabe Werte**

S_OKDie Funktion wurde erfolgreich aufgerufen.ADSERRORCODESEin Fehler tritt auf.**Anmerkungen**

4.2.3 _ITcAdsSyncEvent

Die **_ITcAdsSyncEvent** Schnittstelle ist die Event Schnittstelle, die ein Client implementieren muss, wenn er eine ADS Notification für verbundene Variablen bekommen will. Die Schnittstelle leitet sich von IUnknown ab.

IUnknown Methoden	Beschreibung
QueryInterface	Gibt einen Pointer zur angefragten Schnittstelle zurück
AddRef	Inkrementiert den Referenzzähler
Release	Dekrementiert den Referenzzähler

_ITcAdsSyncEvent Methoden	Beschreibung
DeviceNotification [▶ 41]	Diese Methode wird vom Server für alle verbundenen Variablen dieses ADS Geräts aufgerufen

4.2.3.1 DeviceNotification

_ITcAdsSyncEvent [▶ 41]

Diese Methode wird vom Server für alle verbundenen Variablen dieses ADS Geräts aufgerufen. Das Event tritt für solche Variablen auf, die vorher durch eine AddDeviceNotification [▶ 39] verbunden wurden.

```
HRESULT DeviceNotification(
  [in]TimeStamp*            pTime,
  [in] long                 hNotification,
  [in] long                 cbLen,
  [in, size_is(cbLen)]byte* pData
);
```

Parameter

pTime	[in] Pointer auf eine Variable des Typs TimeStamp [▶ 42] die die genaue Zeit enthält, zu der die verbundene Variable gesammelt wurde.
hNotification	[in] Handle, das eine einzelne verbundene Variable identifiziert. Das Handle wurde zurückgegeben, wenndie Variable durch einen Aufruf der Methode AddDeviceNotification [▶ 39] verbunden wurde.
cbLen	[in] Anzahl der Bytes die durch diese Methode empfangen wurden.
pData	[in, size_is(cbLen)]. Pointer auf das erste Element eines Bytearrays mit der Länge **cbLen** das die Daten der verbundenen Variablen enthält

Rückgabe Werte

S_OK	Die Funktion wurde erfolgreich aufgerufen.
ADSERRORCODES [▶ 42]	Ein Fehler tritt auf.

4.2.4 Strukturen

4.2.4.1 AdsVersion

Die Struktur AdsVersion stellt die Versionsnummer dar, aufgesplittet in Version, Revision und Build Nummer.

```
struct AdsVersion
{
    BYTE version;
    BYTE revision;
    short build;
};
```

4.2.4.2 TimeStamp

Die Struktur TimeStamp stellt eine Windows **FILETIME** Datenstruktur dar. Es handelt sich um einen 64bit Wert, der die Anzahl der 100-Nanosekunden Intervalle seit dem 1. Januar 1601 angibt.

Durch dieses Mittel legt Win32 das Datum und die Zeit fest.

```
struct TimeStamp
{
    long nLow;
    long nHigh;
};
```

Anmerkungen

4.2.5 Enums

4.2.5.1 ADSERRORCODES

Die Aufzählung des Typs **ADSERRORCODE** beschreibt ADS Fehlercodes mit den folgenden Werten:

Konst.	Hex Wert	Beschreibung
ADS_E_ERROR	0x98117000	Offset des ADS Fehlers übergeben von COM HRESULT
ADS_E_SRVNOTSUPP	0x98117001	Angefragter Service wird vom ADS Gerät nicht unterstützt.
_E_INVALIDGRP	0x98117002	Ungültige index group
ADS_E_INVALIDOFFSET	0x98117003	Ungültiger index offset
ADS_E_INVALIDACCESS	0x98117004	Lesen und Schreiben nicht erlaubt
ADSERR_DEVICE_INVALIDSIZE	0x98117005	Parametergröße ist nicht korrekt
ADS_E_INVALIDDATA	0x98117006	Ungültige(r) Datenwert(e)
ADS_E_NOTREADY	0x98117007	Gerät ist nicht bereit
ADS_E_BUSY	0x98117008	Gerät beschäftigt
ADS_E_INVALIDCONTEXT	0x98117009	Ungültiger Kontext
ADS_E_NOMEMORY	0x9811700A	Speicherplatz unzureichend (out of memory)
ADS_E_INVALIDPARM	0x9811700B	Ungültige Parameterwerte
ADS_E_NOTFOUND	0x9811700C	Nicht gefunden (Dateien, ...)
ADS_E_SYNTAX	0x9811700D	Syntaxfehler in Befehl oder Datei
ADS_E_INCOMPATIBLE	0x9811700E	Objekte nicht kompatibel
ADS_E_EXISTS	0x9811700F	Objekt existiert bereits

Konst.	Hex Wert	Beschreibung
ADS_E_SYMBOLNOTFOUND	0x98117010	Symbol nicht gefunden
ADS_E_SYMBOLVERSIONINVALID	0x98117011	Symbolversion ungültig
ADS_E_INVALIDSTATE	0x98117012	Server befindet sich im ungültigen Status
ADS_E_TRANSMODENOTSUPP	0x98117013	AdsTransMode nicht unterstützt
ADS_E_NOTIFYHNDINVALID	0x98117014	Notification handle ist ungültig
ADS_E_CLIENTUNKNOWN	0x98117015	Notification client ist nicht registriert
ADS_E_NOMOREHDLS	0x98117016	Keine weiteren notification handles
ADS_E_INVALIDWATCHSIZE	0x98117017	Zu große Größe für Watch
ADS_E_NOTINIT	0x98117018	Gerät ist nicht initialisiert
ADS_E_TIMEOUT	0x98117019	Gerät hat ein timeout
ADS_E_NOINTERFACE	0x9811701A	Anfrage an Schnittstelle gescheitert
ADS_E_INVALIDINTERFACE	0x9811701B	Falsche Schnittstelle erforderlich
ADS_E_INVALIDCLSID	0x9811701C	class ID ungültig
ADS_E_INVALIDOBJID	0x9811701D	object ID ungültig
ADS_E_CLIENT_ERROR	0x98117040	Fehler class: client Fehler
ADS_E_CLIENT_INVALIDPARM	0x98117041	Ungültiger Parameter bei Serviceaufruf
ADS_E_CLIENT_LISTEMPTY	0x98117042	Polling Liste ist leer
ADS_E_CLIENT_VARUSED	0x98117043	var connection bereits in Benutzung
ADS_E_CLIENT_DUPLINVOKEID	0x98117044	invoke id In Benutzung
ADS_E_CLIENT_SYNCTIMEOUT	0x98117045	timeout verstrichen
ADS_E_CLIENT_W32ERROR	0x98117046	Fehler im Win32 Subsystem
ADS_E_CLIENT_TIMEOUTINVALID	0x98117047	
ADS_E_CLIENT_PORTNOTOPEN	0x98117048	
ADS_E_CLIENT_NOAMSADDR	0x98117049	
ADS_E_CLIENT_SYNCINTERNAL	0x98117050	Interner Fehler in ADS Sync
ADS_E_CLIENT_ADDHASH	0x98117051	hash table overflow
ADS_E_CLIENT_REMOVEHASH	0x98117052	key nicht in hash table gefunden
ADS_E_CLIENT_NOMORESYM	0x98117053	Keine weiteren Symbole im Speicher

4.2.5.2 ADSSTATE

Die Aufzählung des Typs **ADSSTATE** beschreibt den ADS Status mit den folgenden Werten:

Konst	Int Wert	Beschreibung
ADSSTATE_INVALID	0	Ungültiger Status
ADSSTATE_IDLE	1	Idles Status
ADSSTATE_RESET	2	Reset Status
ADSSTATE_INIT	3	initialisiert
ADSSTATE_START	4	gestartet
ADSSTATE_RUN	5	läuft
ADSSTATE_STOP	6	gestoppt
ADSSTATE_SAVECFG	7	Gesicherte Konfiguration
ADSSTATE_LOADCFG	8	Geladene Konfiguration
ADSSTATE_POWERFAILURE	9	Spannungsausfall
ADSSTATE_POWERGOOD	10	Spannung vorhanden
ADSSTATE_ERROR	11	Fehlerstatus
ADSSTATE_SHUTDOWN	12	Herunterfahren (Shut down)
ADSSTATE_SUSPEND	13	verschoben
ADSSTATE_RESUME	14	wiederaufgenommen

Konst	Int Wert	Beschreibung
ADSSTATE_CONFIG	15	system is in config mode
ADSSTATE_RECONFIG	16	system should restart in config mode

4.2.5.3 ADSTRANSMODE

Die Aufzählung des Typs **ADSTRANSMODE** beschreibt den Betrieb einer Geräte notification mit den folgenden Werten:

Konst	Int Wert	Beschreibung
ADSTRANS_NOTRANS	0	
ADSTRANS_CLIENTCYCLE	1	
ADSTRANS_CLIENTONCHA	2	
ADSTRANS_SERVERCYCLE	3	
ADSTRANS_SERVERONCHA	4	
ADSTRANS_CLIENT1REQ	5	

5 Beispiele

5.1 Beispiele: Visual C++

Beispiel	Beschreibung
Beispiel 1	DLL-Version auslesen [▶ 46]
Beispiel 2	Synchron in die SPS schreiben [▶ 46]
Beispiel 3	Synchron aus der SPS lesen [▶ 47]
Beispiel 4	ADS-Status auslesen [▶ 48]
Beispiel 5	ADS-Information auslesen [▶ 48]
Beispiel 6	SPS starten/stoppen [▶ 49]
Beispiel 7	Zugriff auf ein Array in der SPS [▶ 50]
Beispiel 8	Ereignisgesteuertes Lesen [▶ 51]
Beispiel 9	Zugriff per Variablenname [▶ 52]
Beispiel 10	SPS-Variablendeklaration auslesen [▶ 53]
Beispiel 11	Statusänderung vom TwinCAT-Router und der SPS erkennen [▶ 55]
Beispiel 12	Änderungen an der Symboltabelle ereignisgesteuert erkennen [▶ 56]
Beispiel 13	Reserviert
Beispiel 14	Upload der Symbolinformation für eine einzelne Variable [▶ 57]
Beispiel 15	Auslesen der SPS via Multi-Threading [▶ 59]
Beispiel 16	SPS-Variablendeklaration auslesen [▶ 62]
Beispiel 17	ADS-Summenkommando: Lesen und Schreiben [▶ 63]
Beispiel 18	ADS-Summenkommando: Holen und freigeben [▶ 64]
Beispiel 19	Reserviert
Beispiel 20	Übertragen von Strukturen an die SPS [▶ 68]
Beispiel 21	Lesen und Schreiben von DATE/TIME-Variablen [▶ 69]

Dokumente hierzu

- sample20-c-ads-dll-writestructure.zip (Resources/zip/12470841739.zip)
- sample21-c-ads-dll-accessdatetimevariables.zip (Resources/zip/12470843147.zip)

5.1.1 Einbinden in Visual C++

TwinCAT ADS API

Die ADS-Bibliotheken werden mit jeder TwinCAT-Installation ausgeliefert und befinden sich im Verzeichnis "..**\TwinCAT\AdsApi**". Wenn sie lokal keine Laufzeit oder IO benötigen, können sie die kostenlose Variante TwinCAT CP verwenden, um einen ADS Client zu entwickeln.

Header Dateien einbinden

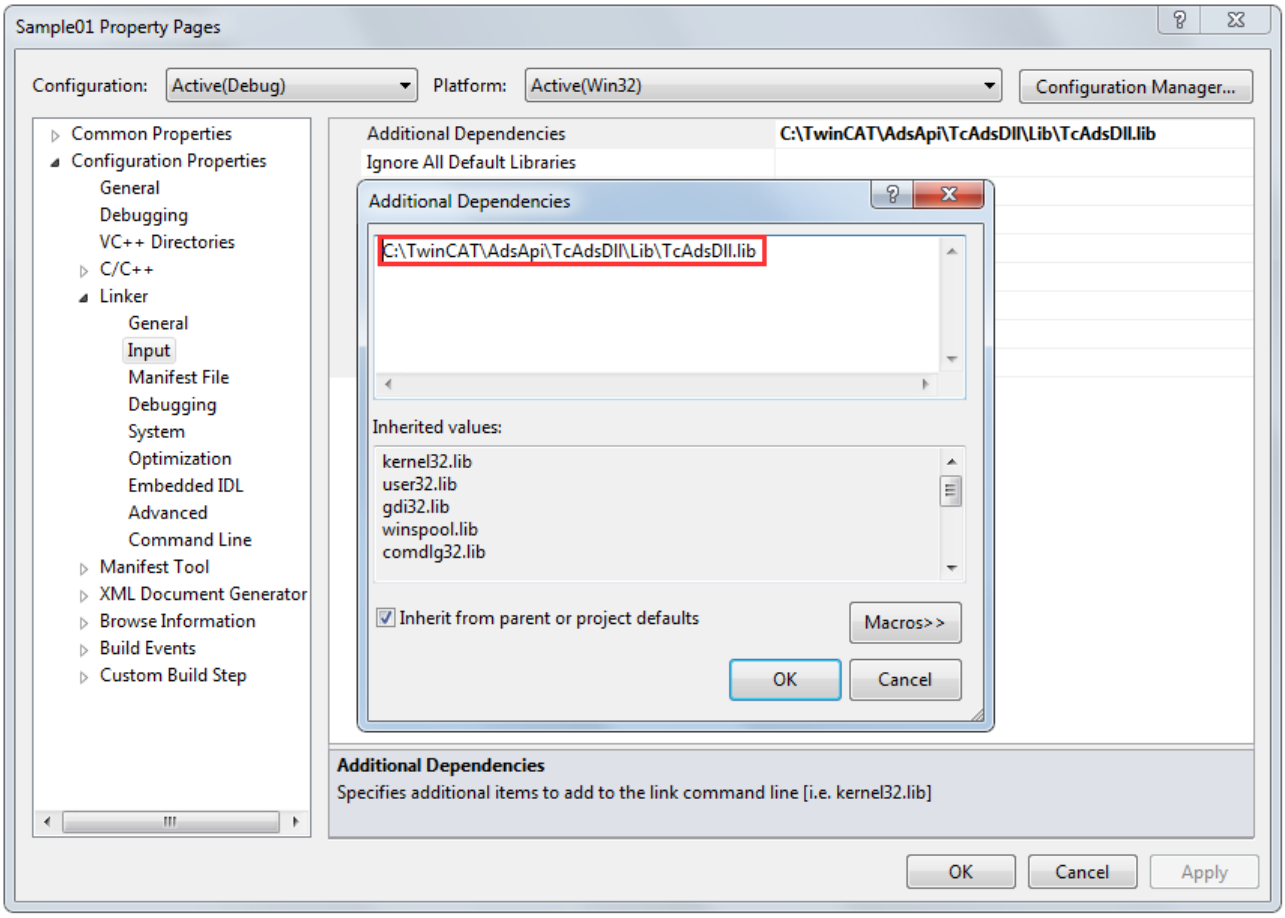
Um die Funktionalität der TcAdsDll.dll in einem Projekt nutzen zu können, müssen die Headerdateien TcAdsApi.h und TcAdsDef.h in das Projekt eingefügt werden.

```
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"
```

Bibliothek zum Projekt hinzufügen

Um die Funktionalität der TcAdsDll nutzen zu können, muss die TcAdsDll.lib eingebunden werden.

In Visual Studio muss der Menüpunkt "Project Properties" angewählt werden. Im Dialog "Project Properties" ist der Bereich "Configuration Properties\Linker\Input" anzuwählen. Um die Bibliothek einzubinden, muss der Pfad der TcAdsDll.lib in der Textbox "Additional Dependencies" eingetragen werden.



5.1.2 DLL-Version auslesen

Dieses Programm ermittelt die Version der DLL-Datei.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470817803.zip

```
#include <iostream.h>
#include <conio.h>
#include <windows.h>

// Insert ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  long    nTemp;
  AdsVersion* pDLLVersion;

  nTemp = AdsGetDllVersion();
  pDLLVersion = (AdsVersion *)&nTemp;
  cout << "Version: " << (int)pDLLVersion->version << '\n';
  cout << "Revision: " << (int)pDLLVersion->revision << '\n';
  cout << "Build: " << pDLLVersion->build << '\n';
  cout.flush();
  getch();
}
```

5.1.3 Synchron in die SPS schreiben

Bei diesem Beispielprogramm wird der eingegebene Wert in das Merkerdoppelwort 0 geschrieben.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470819211.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  long     nErr, nPort;
  AmsAddr  Addr;
  PAmsAddr pAddr = &Addr;
  DWORD    dwData;

  // Kommunikationsport auf dem ADS Router öffnen
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // TwinCAT2 RTS1 Port = 801
  pAddr->port = 801;

  // Wert vom Benutzer einlesen, welcher zur SPS geschrieben werden sollen
  cout << "Value: ";
  cin >> dwData;

  // Wert in MD0 schreiben
  nErr = AdsSyncWriteReq( pAddr, 0x4020, 0x0, 0x4, &dwData );
  if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

  _getch();
  // Kommunikationsport schließen
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

5.1.4 Synchron aus der SPS lesen

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

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470820619.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  long      nErr, nPort;
  AmsAddr   Addr;
  PAmsAddr  pAddr = &Addr;
  DWORD     dwData;

  // Kommunikationsport auf dem ADS Router öffnen
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // TwinCAT2 RTS1 Port = 801
  pAddr->port = 801;

  // Werte aus MD0 auslesen und anzeigen
  do
  {
    nErr = AdsSyncReadReq(pAddr, 0x4020, 0x0, 0x4, &dwData);
    if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
    else cout << dwData << '\n';
    cout.flush();
  }
  while (getch() == '\r');    // mit Carriage return nächsten Wert auslesen, sonst Ende
```

```
  // Kommunikationsport schließen
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

5.1.5 ADS-Status auslesen

Dieses Programm ließ den Status der SPS aus. Die Variable vom Typ ADSSTATE enthält Informationen, ob sich z. B. die SPS im RUN oder STOP Zustand befindet.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470822027.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  ADSSTATE   nAdsState;
  USHORT    nDeviceState;
  long  nErr, nPort;
  AmsAddr       Addr;
  PAmsAddr   pAddr = &Addr;

  // Kommunikationsport auf dem ADS Router öffnen
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) nErr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // TwinCAT2 RTS1 Port = 801
  pAddr->port = 801;

  do
  {
    nErr = AdsSyncReadStateReq(pAddr, &nAdsState, &nDeviceState);
    if (nErr)
      cerr << "Error: AdsSyncReadStateReq: " << nErr << '\n';
    else
    {
      cout << "AdsState: " << nAdsState << '\n';
      cout << "DeviceState: " << nDeviceState << '\n';
    }
    cout.flush();
  }
  while ( getch() == '\r'); // Carriage return weiter, sonst Ende

  // Kommunikationsport schließen
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

5.1.6 ADS-Informationen auslesen

Jedes ADS-Gerät enthält eine Versionsnummer und eine Bezeichnung. Das Beispielprogramm liest diese Informationen der SPS aus und zeigt sie am Bildschirm an.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470823435.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  LONG       nErr, nPort;
  AdsVersion     Version;
  AdsVersion     *pVersion = &Version;
```

```
  char       pDevName[50];
  AmsAddr    Addr;
  PAmsAddr       pAddr = &Addr;

  // Kommunikationsport auf dem ADS Router öffnen
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // TwinCAT2 RTS1 Port = 801
  pAddr->port = 801;

  nErr = AdsSyncReadDeviceInfoReq(pAddr, pDevName, pVersion);
  if (nErr)
    cerr << "Error: AdsSyncReadDeviceInfoReq: " << nErr << '\n';
  else
  {
    cout << "Name: " << pDevName << '\n';
    cout << "Version: " << (int)pVersion->version << '\n';
    cout << "Revision: " << (int)pVersion->revision << '\n';
    cout << "Build: " << pVersion->build << '\n';
  }
  cout.flush();
  _getch();

  // Kommunikationsport schließen
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

5.1.7 SPS starten/stoppen

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

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470824843.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  USHORT      nAdsState;
  USHORT      nDeviceState = 0;
  long    nErr, nPort;
  int     ch;
  void    *pData = NULL;
  AmsAddr     Addr;
  PAmsAddr    pAddr = &Addr;

  // Kommunikationsport auf dem ADS Router öffnen
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // TwinCAT2 RTS1
  pAddr->port = 801;

  cout << "(R) -> PLC Run\n";
  cout << "(S) -> PLC Stop\n";
  cout.flush();
  ch = getch();
  ch = toupper(ch);
  while ( (ch == 'R') || (ch == 'S') )
  {
    switch (ch)
    {
      case 'R':
    nAdsState = ADSSTATE_RUN;
    break;
      case 'S':
    nAdsState = ADSSTATE_STOP;
    break;
    }
```

```
    nErr = AdsSyncWriteControlReq (pAddr, nAdsState, nDeviceState, 0, pData);
    if (nErr) cerr << "Error: AdsSyncWriteControlReq: " << nErr << '\n';
    ch = _getch();
    ch = toupper(ch);
  }

  // Kommunikationsport schließen
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

5.1.8 Zugriff auf ein Array in der SPS

Ein Array, welches sich in der SPS befindet, soll ausgelesen werden. Die Variable wird hierbei per Handle angesprochen. Die Funktion AdsSyncReadReq() wird die Länge des gesamten Arrays übergeben. Als Variable wird die Adresse des ersten Array-Elements verwendet.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470826251.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  long      nErr, nPort;
  AmsAddr       Addr;
  PAmsAddr      pAddr = &Addr;
  unsigned long lHdlVar;
  int       nIndex;
  short     Data[10];
  char      szVar []={"MAIN.PLCVar"};

  // Open communication port on the ADS router
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
  pAddr->port = AMSPORT_R0_PLC_RTS1;

  // Fetch handle for the PLC variable
   nErr = AdsSyncReadWriteReq(  pAddr,
                ADSIGRP_SYM_HNDBYNAME,
                0x0,
                sizeof(lHdlVar),
                lHdlVar,
                sizeof(szVar),
                szVar);

  if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';
  // Read values of the PLC variables (by handle)
  nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(Data), &Data[0]);
  if (nErr)
    cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
  else
  {
    for (nIndex = 0; nIndex < 10; nIndex++)
      cout << "Data[" << nIndex << "]: " << Data[nIndex] << '\n';
  }
  cout.flush();
  _getch();
  // Close communication port
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

SPS Programm

```
PROGRAM MAIN
VAR
    PLCVar          : ARRAY[0..9] OF WORD;
    TP_1            : ARRAY[0..9] OF TP;
    TOGGEL          : BOOL;
    nIndex          : INT;
```

```
END_VAR

TOGGEL := NOT TOGGEL;

FOR nIndex := 0 TO 9 DO
    TP_1[nIndex]( IN := TOGGEL, PT := t#1s);
    IF (TP_1[nIndex].Q = 0) THEN
        PLCVar[nIndex] := PLCVar[nIndex] + 1 + nIndex;
    END_IF
END_FOR
```

5.1.9 Ereignisgesteuertes Lesen

Sollen in einer Bedieneroberfläche kontinuierlich Werte aus der SPS oder NC angezeigt werden, so ist das Benutzen von AdsSyncReadReq() [▶ 13] sehr aufwendig, da diese Funktion zyklisch aufgerufen werden muss. Durch das Definieren sogenannter Notifications (Meldungen) kann ein TwinCAT Server dazu veranlasst werden, Werte über ADS an ein anderes ADS-Gerät zu übertragen. Hierbei wird unterschieden, ob der TwinCAT Server die Werte zyklisch oder nur bei Veränderung übertragen soll.

Mit der Funktion AdsSyncAddDeviceNotificationReq() [▶ 18] wird eine Notification erzeugt. Die Callback-Funktion wird anschließend selbständig von TwinCAT aufgerufen. Mit AdsSyncDelDeviceNotificationReq() [▶ 19] wird die Notification wieder gelöscht. Da die Anzahl der Notifications begrenzt ist, sollten Sie in Ihrem Programm dafür sorgen, das nicht mehr benötigte Notifications gelöscht werden. Weitere Informationen finden Sie bei der Beschreibung zur Struktur AdsNotificationAttrib [▶ 31].

Das folgende Programm startet eine Notification auf dem Handle einer Variablen. Bei jeder Änderung der SPS-Variablen, wird die Callback-Funktion aufgerufen. Als Parameter enthält die Callback-Funktion unter anderem eine Variable vom Typ AdsNotificationHeader [▶ 32]. In dieser Struktur sind alle notwendigen Informationen (Wert, Zeitstempel, ...) enthalten.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470827659.zip

HINWEIS

Zeitintensive Aktionen oder ADS-Befehle

Im Callback dürfen keine zeitintensiven Aktionen oder ADS-Befehle ausgeführt werden.

```
#include <iostream>
#include <conio.h>
#include <windows.h>
#include <winbase.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

using namespace std;

void _stdcall Callback(AmsAddr*, AdsNotificationHeader*, unsigned long);

void main()
{
  long           nErr, nPort;
  AmsAddr        Addr;
  PAmsAddr           pAddr = &Addr;
  ULONG          hNotification, hUser;
  AdsNotificationAttrib  adsNotificationAttrib;
  char       szVar []={"MAIN.PlcVar"};

  // Open communication port on the ADS router
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
  pAddr->port = 801;

  // Set the attributes of the notification
  adsNotificationAttrib.cbLength = 4;
  adsNotificationAttrib.nTransMode = ADSTRANS_SERVERONCHA;
  adsNotificationAttrib.nMaxDelay = 0;
  adsNotificationAttrib.nCycleTime = 10000000; // Unit = 100ns, 1sec = 10000000

  // Get variable handle
```

```
  nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(hUser), &hUser, sizeof(szVar)
, szVar);
  if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';

  // Initiate the transmission of the PLC-variable
  nErr = AdsSyncAddDeviceNotificationReq(       pAddr,
                   ADSIGRP_SYM_VALBYHND,
                   hUser,
                   &adsNotificationAttrib,
                   Callback,
                   hUser,
                   &hNotification);

  if (nErr) cerr << "Error: AdsSyncAddDeviceNotificationReq: " << nErr << '\n';
  cout.flush();

  // Wait for user intraction (keystroke)
  _getch();

  // Finish the transmission of the PLC-variable
  nErr = AdsSyncDelDeviceNotificationReq(pAddr, hNotification);
  if (nErr) cerr << "Error: AdsSyncDelDeviceNotificationReq: " << nErr << '\n';

  // Release  the variable handle
  nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(hUser), &hUser);
  if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

  // Close the communication port
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}

// Callback-function
void __stdcall Callback(AmsAddr* pAddr, AdsNotificationHeader* pNotification, ULONG hUser)
{
  int             nIndex;
  static ULONG        nCount = 0;
  SYSTEMTIME          SystemTime, LocalTime;
  FILETIME        FileTime;
  LARGE_INTEGER       LargeInteger;
  TIME_ZONE_INFORMATION   TimeZoneInformation;

  cout << ++nCount << ". Notification:\n";

  // print (to screen)) the value of the variable
  cout << "Value: " << *(ULONG *)pNotification->data << '\n';
  cout << "Notification: " << pNotification->hNotification << '\n';

  // Convert the timestamp into SYSTEMTIME
  LargeInteger.QuadPart = pNotification->nTimeStamp;
  FileTime.dwLowDateTime = (DWORD)LargeInteger.LowPart;
  FileTime.dwHighDateTime = (DWORD)LargeInteger.HighPart;
  FileTimeToSystemTime(&FileTime, &SystemTime);

  // Convert the time value Zeit to local time
  GetTimeZoneInformation(&TimeZoneInformation);
  SystemTimeToTzSpecificLocalTime(&TimeZoneInformation, &SystemTime, &LocalTime);

  // Print out the timestamp
  cout << LocalTime.wHour << ":" << LocalTime.wMinute << ":" << LocalTime.wSecond << '.' << LocalTim
e.wMilliseconds;

  // Print out the ADS-address of the sender
  cout << "\nServerNetId: ";
  for (nIndex = 0; nIndex < 6; nIndex++)
    cout << (int)pAddr->netId.b[nIndex] << ".";
  cout << " Port: " << pAddr->port << "\n\n";
  cout.flush();
}
```

5.1.10 Zugriff per Variablenname

Alle Daten, die ADS-Geräte nach Außen zur Verfügung stellen, sind in IndexGroups und IndexOffset organisiert. Eine IndexGroup kann man sich als Tabelle vorstellen, wobei jeder Eintrag über den IndexOffset angesprochen wird. Die TwinCAT-SPS verfügt z. B. über IndexGroups in denen die Variablen abgelegt sind, die zum Ein-/Ausgangs- oder Merkerbereich gehören. Außerdem stehen in der TwinCAT-SPS IndexGroups zur Verfügung, mit denen Systemfunktionen angesprochen werden können.

Für das Beispielprogramm sind die IndexGroups ADSIGRP_SYM_HNDBYNAME und ADSIGRP_SYM_VALBYHND von Bedeutung. Über die IndexGroup ADSIGRP_SYM_HNDBYNAME wird von einer SPS-Variablen, die namentlich genannt wird, ein Handle angefordert. Mit Hilfe dieses Handles und der IndexGroup ADSIGRP_SYM_VALBYHND kann auf die Variable zugegriffen werden. Als IndexOffset wird der Handle der Variablen übergeben.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470829067.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>

// ADS headers
#include "c:\twincat\adsapi\tcadsdll\include\tcadsdef.h"
#include "c:\twincat\adsapi\tcadsdll\include\tcadsapi.h"

using namespace std;

void main()
{
  long      nErr, nPort;
  AmsAddr   Addr;
  PAmsAddr  pAddr = &Addr;
  ULONG     lHdlVar, nData;
  char      szVar []={"MAIN.PLCVar"};

  // Open communication port on the ADS router
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
  pAddr->port = AMSPORT_R0_PLC_RTS1;

  // Get the handle of the PLC-variable
  nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlVar), &lHdlVar, sizeof(sz
Var), szVar);
  if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';
  do
  {
    // Read the value of a PLC-variable, via handle
    nErr = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nData );
    if (nErr)
      cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
    else
      cout << "Value: " << nData << '\n';
    cout.flush();
    if (nData > 10)
    {
      // Write the value of the PLC variable to 0
      nData = 0;
      nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nData);
      if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';
    }
  }
  while ( _getch() == '\r');  // Get next value with ENTER-button

  // Release the handle of the PLC-variable
  nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlVar), &lHdlVar);
  if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

  // Close the communication port
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

5.1.11 SPS-Variablendeklaration (statisch) auslesen

Dieses Beispiel beschreibt das Auslesen der statischen SPS-Symbole.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470830475.zip

Bei dem Zugriff auf die Variablendeklaration werden folgende Informationen übertragen:

- Variablenname
- Datentyp
- Länge

- Adresse (IndexGroup / IndexOffset)
- Kommentar

Sämtliche oben aufgelisteten Information werden in einen Datenstream übertragen. Bevor dieser ausgelesen werden kann, wird mit dem ersten AdsSyncReadReq() [▶ 13] die Länge ermittelt. Bei dem zweiten AdsSyncReadReq() werden die eigentlichen Daten übertragen. Die Variable *pchSymbols* ist ein Pointer, der auf diesen Bereich zeigt. In der FOR-Schleife wird für jede einzelne SPS-Variable der entsprechende Datenbereich in die Struktur *pAdsSymbolEntry* kopiert. In dieser Struktur sind die einzelnen Informationen der SPS-Variablen abgelegt. Die Makros PADSSYMBOLNAME, PADSSYMBOLTYPE und PADSSYMBOLCOMMENT vereinfachen das Auswerten dieser Daten.

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>
#include <assert.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void main()
{
  long          nErr, nPort;
  char          *pchSymbols = NULL;
  UINT          uiIndex;
  AmsAddr           Addr;
  PAmsAddr          pAddr = &Addr;
  AdsSymbolUploadInfo   tAdsSymbolUploadInfo;
  PAdsSymbolEntry       pAdsSymbolEntry;

  // Open communication port on the ADS router
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
  pAddr->port = 801;

  // Read the length of the variable declaration
  nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_UPLOADINFO, 0x0, sizeof(tAdsSymbolUploadInfo), &tAdsSymbo
lUploadInfo);
  if (!nErr)
  {
      pchSymbols = new char[tAdsSymbolUploadInfo.nSymSize];
      assert(pchSymbols);

      // Read information of the PLC-variable
      nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_UPLOAD, 0, tAdsSymbolUploadInfo.nSymSize, pchSymbols)
;
      if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '\n';

      // Print out the information of the PLC-variable
      pAdsSymbolEntry = (PAdsSymbolEntry)pchSymbols;
      for (uiIndex = 0; uiIndex < tAdsSymbolUploadInfo.nSymbols; uiIndex++)
      {
        cout << PADSSYMBOLNAME(pAdsSymbolEntry) << "\t\t"
             << pAdsSymbolEntry->iGroup << '\t'
             << pAdsSymbolEntry->iOffs << '\t'
             << pAdsSymbolEntry->size << '\t'
             << PADSSYMBOLTYPE(pAdsSymbolEntry) << '\t'
             << PADSSYMBOLCOMMENT(pAdsSymbolEntry) << '\n';
        pAdsSymbolEntry = PADSNEXTSYMBOLENTRY(pAdsSymbolEntry); cout.flush();
      }
  }
  else
  {
      cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
  }
  getch();

  // Close the communication port
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';

  // Release the allocated memory
  if (pchSymbols) delete(pchSymbols);
}
```

5.1.12 Statusänderung vom TwinCAT-Router und der SPS erkennen

Zur Laufzeit einer Applikation ist es oft von Bedeutung, den Status von TwinCAT und/oder dessen Komponenten abzufragen; z. B. ob die SPS im RUN-Status ist. Damit dieses nicht ständig abgefragt werden muß, kann mit Hilfe von Callback-Funktionen eine Statusänderung auf sehr effektive Weise erkannt werden. Mit dem folgenden Beispielprogramm wird der Status der SPS (Laufzeitsystem 1) und der des TwinCAT-Routers überwacht.

Durch den Aufruf der Funktion AdsAmsRegisterRouterNotification() [▶ 20] wird bei jeder Statusänderung des TwinCAT-Routers die angegebende Callback-Funktion aufgerufen. Anhand des Parameters, der übergeben wird, kann der aktuelle Status abgefragt werden.

Mit Hilfe der Funktion AdsSyncAddDeviceNotificationReq() [▶ 18] wird der Status der SPS überwacht. Die Daten, die an die Callback-Funktion übergeben werden, stellen der aktuellen Status der SPS da.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470831883.zip

```
#include <iostream.h>
#include <conio.h>
#include <windows.h>
#include <winbase.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

void __stdcall Callback(AmsAddr*, AdsNotificationHeader*, ULONG);
void __stdcall RouterCall(LONG);

void main()
{
  LONG            nErr, nPort;
  ULONG           hNotification, hUser = 0;
  AmsAddr         Addr;
  PAmsAddr        pAddr = &Addr;
  AdsNotificationAttrib   adsNotificationAttrib;

  // Open communication port on the ADS router
  nPort = AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // Select Port: TwinCAT2 PLC1
  pAddr->port = 801;

  nErr = AdsAmsRegisterRouterNotification(&RouterCall);
  if (nErr) cerr << "Error: AdsAmsRegisterRouterNotification: " << nErr << '\n';

  // Invoke notification
  adsNotificationAttrib.cbLength       = sizeof(short);
  adsNotificationAttrib.nTransMode     = ADSTRANS_SERVERONCHA;
  adsNotificationAttrib.nMaxDelay      = 0; // report each change directly
  adsNotificationAttrib.dwChangeFilter = 0;
  nErr = AdsSyncAddDeviceNotificationReq(pAddr, ADSIGRP_DEVICE_DATA, ADSIOFFS_DEVDATA_ADSSTATE, &ads
NotificationAttrib, Callback, hUser, &hNotification);
  if (nErr) cerr << "Error: AdsSyncAddDeviceNotificationReq: " << nErr << "\n";
  getch();

  // The following calls return errors if TwinCAT is halted
  nErr = AdsSyncDelDeviceNotificationReq(pAddr, hNotification);
  if (nErr) cerr << "Error: AdsSyncDelDeviceNotificationReq: " << nErr << '\n';

  nErr = AdsAmsUnRegisterRouterNotification();
  if (nErr) cerr << "Error: AdsAmsUnRegisterRouterNotification: " << nErr << '\n';

  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';

  return;
}

// ADS state callback function
void __stdcall Callback(AmsAddr* pAddr, AdsNotificationHeader* pNotification, ULONG hUser)
{
  INT       nIndex;
  nIndex = *(short *)pNotification->data;
  switch (nIndex)
  {
```

```
  case ADSSTATE_RUN:
    cout << "PLC run\n";
    break;
  case ADSSTATE_STOP:
    cout << "PLC stop\n";
    break;
  default :
    cout << "PLC ADS-State" << nIndex << "\n";
    break;
  }
  cout.flush ();
}

// TwinCAT router callback function
void __stdcall RouterCall (long nReason)
{
  switch (nReason)
  {
  case AMSEVENT_ROUTERSTOP:
    cout << "TwinCAT-Router stop\n";
    break;
  case AMSEVENT_ROUTERSTART:
    cout << "TwinCAT-Router start\n";
    break;
  case AMSEVENT_ROUTERREMOVED:
    cout << "TwinCAT-Router removed\n";
    break;
  default:
    cout << "TwinCAT-Router AMS-Event " << nReason << "\n";
    break;
  }
  cout.flush ();
}
```

5.1.13 Änderungen an der Symboltabelle ereignisgesteuert erkennen

ADS-Geräte, die Symbolnamen unterstützen (SPS, NC, ...), legen die Symbolnamen in eine interne Tabelle ab. Dabei wird jedem Symbol ein Handle zugeordnet. Das Symbolhandle wird benötigt, um auf die Variablen zugreifen zu können (siehe auch Beispiel 9 [▶ 52]). Ändert sich die Symboltabelle, z. B. weil ein neues SPS-Programm in die Steuerung geschrieben wird, so müssen auch die Handles neu ermittelt werden. Wie Änderungen an der Symboltabelle erkannt werden können, zeigt das folgende Beispiel.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470833291.zip

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>
#include <winbase.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"


void __stdcall SymbolChanged(AmsAddr*, AdsNotificationHeader*, unsigned long);

void main()
{
  long            nErr;
  AmsAddr         Addr;
  PAmsAddr            pAddr = &Addr;
  ULONG           hNotification;
  AdsNotificationAttrib  adsNotificationAttrib;

  // Open communication port on the ADS router
  AdsPortOpen();
  nErr = AdsGetLocalAddress(pAddr);
  if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

  // Select Port: TwinCAT2 PLC1
  pAddr->port = 801;

  // Specify attributes of the notification
  adsNotificationAttrib.cbLength = 1;
  adsNotificationAttrib.nTransMode = ADSTRANS_SERVERONCHA;
```

```
  adsNotificationAttrib.nMaxDelay = 5000000; // 500ms
  adsNotificationAttrib.nCycleTime = 5000000; // 500ms

  // Start notification for changes to the symbol table
  nErr = AdsSyncAddDeviceNotificationReq(pAddr, ADSIGRP_SYM_VERSION, 0, &adsNotificationAttrib, Symb
olChanged, NULL, &hNotification);
  if (nErr) cerr << "Error: AdsSyncAddDeviceNotificationReq: " << nErr << '\n';

  // Wait for a key-press from the user
  _getch();

  // Stop notification
  nErr = AdsSyncDelDeviceNotificationReq(pAddr, hNotification);
  if (nErr) cerr << "Error: AdsSyncDelDeviceNotificationReq: " << nErr << '\n';

  // Close communication port
  nErr = AdsPortClose();
  if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}

// Callback function
void __stdcall SymbolChanged(AmsAddr* pAddr, AdsNotificationHeader* pNotification, ULONG hUser)
{
  cout << "Symboltabelle hat sich geändert!\n";
  cout.flush();
}
```

5.1.14 Auslesen einer Variablendeklaration

Bei dem Zugriff auf die Variablendeklaration werden folgende Informationen übertragen:

- Variablenname
- Datentyp
- Länge
- Adresse (IndexGroup / IndexOffset)
- Kommentar

Mit dem AdsSyncReadWriteReq() Aufruf wird die Information der Variablen ausgelesen. Der Name der Variable wird der Funktion im Parameter *pWriteData* übergeben. Nach dem Aufruf steht die angeforderte Information in der Variable *pAdsSymbolEntry*. In dieser Struktur sind die einzelnen Informationen der SPS-Variablen abgelegt. Die Makros PADSSYMBOLNAME, PADSSYMBOLTYPE und PADSSYMBOLCOMMENT vereinfachen das Auswerten dieser Daten. Als nächstes wird der Datentyp der Variable anhand von *pAdsSymbolEntry->dataType* ausgewertet.
Falls der Datentyp dem Typ UDINT oder ARRAY OF UDINT wird zusätzlich der Wert dieser Variable ausgelsen.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470834699.zip

```
#include <windows.h>
#include <conio.h>
#include <assert.h>
#include <string.h>
#include <iostream.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

typedef enum AdsDataTypeId
{
    ADST_VOID = VT_EMPTY,
    ADST_INT8 = VT_I1,
    ADST_UINT8 = VT_UI1,
    ADST_INT16 = VT_I2,
    ADST_UINT16 = VT_UI2,
    ADST_INT32 = VT_I4,
    ADST_UINT32 = VT_UI4,
    ADST_INT64 = VT_I8,
    ADST_UINT64 = VT_UI8,
    ADST_REAL32 = VT_R4,
    ADST_REAL64 = VT_R8,
    ADST_STRING = VT_LPSTR,
    ADST_WSTRING = VT_LPWSTR,
```

```
    ADST_REAL80 = VT_LPWSTR+1,
    ADST_BIT = VT_LPWSTR+2,
    ADST_BIGTYPE = VT_BLOB,
    ADST_MAXTYPES = VT_STORAGE,
} ADS_DATATYPE;

typedef struct _ValueString
{
    DWORD dwValue;
    char* szLabel;
} ValueString;

ValueString AdsDatatypeString[] =
{
    { VT_EMPTY, "ADST_VOID", },
    { VT_I1, "ADST_INT8", },
    { VT_UI1, "ADST_UINT8", },
    { VT_I2, "ADST_INT16", },
    { VT_UI2, "ADST_UINT16", },
    { VT_I4, "ADST_INT32", },
    { VT_UI4, "ADST_UINT32", },
    { VT_I8, "ADST_INT64", },
    { VT_UI8, "ADST_UINT64", },
    { VT_R4, "ADST_REAL32", },
    { VT_R8, "ADST_REAL64", },
    { VT_LPSTR, "ADST_STRING", },
    { VT_LPWSTR, "ADST_WSTRING", },
    { VT_LPWSTR+2, "ADST_BIT", },
    { VT_BLOB, "ADST_BIGTYPE", },
    { VT_STORAGE, "ADST_MAXTYPES", },
};

void main()
{
    long nErr, nPort;
    AmsAddr Addr;
    PAmsAddr pAddr = &Addr;
    char szVariable[255];
    BYTE buffer[0xFFFF];
    PAdsSymbolEntry pAdsSymbolEntry;

    // Open communication port on the ADS router
    nPort = AdsPortOpen();
    nErr = AdsGetLocalAddress(pAddr);
    if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

    // Select Port: TwinCAT2 PLC1
    pAddr->port = 801;

    for(;;)
    {
    cout << "Enter variable Name: ";
    cin >> szVariable;

    nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_INFOBYNAMEEX, 0, sizeof(buffer), buffer, strlen(sz
Variable)+1, szVariable);
    if (nErr)
    {
        cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
    }
    else
    {
        pAdsSymbolEntry = (PAdsSymbolEntry)buffer;
        cout << "Name: " << PADSSYMBOLNAME(pAdsSymbolEntry) << "\n"
        <<"Index Group: "<< pAdsSymbolEntry->iGroup << '\n'
        <<"Index Offset: "<< pAdsSymbolEntry->iOffs << '\n'
        <<"Size: "<< pAdsSymbolEntry->size << '\n'
        <<"Type: "<< (char*)PADSSYMBOLTYPE(pAdsSymbolEntry) << '\n'
        <<"Comment: "<< (char*)PADSSYMBOLCOMMENT(pAdsSymbolEntry) << '\n';

        switch(pAdsSymbolEntry->dataType)
        {
        case ADST_UINT32:
        {
            int nElements = pAdsSymbolEntry->size/sizeof(unsigned long);
            unsigned long *pVal = new unsigned long[nElements];
            cout << "Datatype: ADST_UINT32" <<'\n';
            AdsSyncReadReq(pAddr, pAdsSymbolEntry->iGroup, pAdsSymbolEntry->iOffs, pAdsSymbolEntry-
>size, pVal);
            if( nErr )
```

```
            {
            cerr << "Error: AdsSyncReadReq: Unable to read Value" << nErr << '\n';
            }
            else
            {
            cout << "Value: ";
            for( int i=0; i<nElements; i++ )
            {
                cout << pVal[i] << '\t';
            }
            cout << '\n';
            }
        }
        break;
        default:
        {
            int nType = sizeof(AdsDatatypeString)/sizeof(ValueString);
            for(int i=0; i< nType; i++)
            {
            if( AdsDatatypeString[i].dwValue == pAdsSymbolEntry->dataType )
            {
                cout << "Datatype:" << AdsDatatypeString[i].szLabel <<'\n';
                break;
            }
            }
            if( i == nType )
            cout << "Datatype:" << "Unknown datatype:" << pAdsSymbolEntry->dataType <<'\n';
        }
        break;
        }
    }
    cout << "Exit(y/n)" << '\n';
    cout.flush();

    if(_getch() == 'y')
        break;
    }
    // Close communication port
    nErr = AdsPortClose();
    if (nErr) cerr << "Fehler: AdsPortClose: " << nErr << '\n';
}
```

```
VAR_GLOBAL
    PLCVarBYTE AT %MB0: BYTE; (* PLCVarBYTE *)
    PLCVarWORD AT %MB4: WORD; (* PLCVarWORD *)
    PLCVarDWORD AT %MB8: DWORD; (* PLCVarDWORD *)
    PLCVarBOOL AT %MB12: BOOL; (* PLCVarBOOL *)
    PLCVarINT AT %MB16: INT; (* PLCVarINT *)
    PLCVarSINT AT %MB20: SINT; (* PLCVarSINT *)
    PLCVarUSINT AT %MB24: USINT; (* PLCVarUSINT *)
    PLCVarUINT AT %MB28: UINT; (* PLCVarUINT *)
    PLCVarDINT AT %MB32: DINT; (* PLCVarDINT *)
    PLCVarUDINT AT %MB34: UDINT := 1234; (* PLCVarUDINT *)
    PLCVarREAL AT %MB38: REAL; (* PLCVarREAL *)
    PLCVarLREAL AT %MB42: LREAL; (* PLCVarLREAL *)

    PLCVarUDINTArray :ARRAY[0..5] OF UDINT; (* PLCVarINTArray*)
END_VAR
```

5.1.15 Multi-Threading

Das Beispiel erläutert das Erzeugen zweier Threads, die in unterschiedlichen Zeitintervalen auf eine SPS zugreifen.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470836107.zip

Multithreading Anwendung mit ADS Funktionalitäten

Verwenden Sie die erweiterten ADS Methoden [▶ 21], wenn Sie eine Multithreading Anwendung mit ADS Funktionalitäten entwickeln wollen.

```
#include <iostream>
#include <windows.h>
#include <conio.h>
```

```
// ADS headers
#include "c:\twincat\adsapi\tcadsdll\include\tcadsdef.h"
#include "c:\twincat\adsapi\tcadsdll\include\tcadsapi.h"

using namespace std;

DWORD WINAPI ThreadFunction1(LPVOID parameter)
{
    long        nErr, nPort;
    AmsAddr     Addr;
    PAmsAddr    pAddr = &Addr;
    ULONG       lHdlVar, nData;
    ULONG       cbReturn = 0;
    bool        bRun = true;
    char        szVar []={"MAIN.PLCVarRTS1"};

    // Open communication port on local ADS router
    // Use the ADS-Ex-Methods if you implement multi threading

    nPort = AdsPortOpenEx();
    nErr = AdsGetLocalAddressEx(nPort, pAddr);
    if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

    // TwinCAT2 RTS1 Port
    pAddr->port = 801;

    // Get the handle of the PLC-variable
    nErr = AdsSyncReadWriteReqEx2(nPort, pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlVar), &lHdlVa
r, sizeof(szVar), szVar, &cbReturn);
    if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';

    while(bRun)
    {

        DWORD wait = 0;

        // Wait 100ms for stopEvent
        wait = WaitForSingleObject(parameter, 100);

        // If timeout elapse read plc data
        if(WAIT_TIMEOUT == wait)
        {
            // Read the value of a PLC-variable, via handle
            nErr = AdsSyncReadReqEx2(nPort, pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nD
ata, &cbReturn);
            if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
            else cout << "Actual thread --> Thread1 Value: " << nData << '\n';
            cout.flush();
        }

        else
        {
            bRun = false;
        }
    }

    // Release the handle of the PLC-variable
    nErr = AdsSyncWriteReqEx(nPort, pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlVar), &lHdlVar);
    if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

    // Close the communication port
    nErr = AdsPortCloseEx(nPort);
    if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';

    return 1;
}

DWORD WINAPI ThreadFunction2(LPVOID parameter)
{
    long        nErr, nPort;
    AmsAddr     Addr;
    PAmsAddr    pAddr = &Addr;
    ULONG       lHdlVar, nData;
    ULONG       cbReturn = 0;
    bool        bRun = true;
    char        szVar []={"MAIN.PLCVarRTS2"};

    // Open communication port on the local ADS router
    // Use the ADS-Ex-Methods if you implement multi threading and open an additional port
```

```
    nPort = AdsPortOpenEx();
    nErr = AdsGetLocalAddressEx(nPort, pAddr);
    if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

    // TwinCAT2 RTS1 Port
    pAddr->port = 801;

    // Get the handle of the PLC-variable
    nErr = AdsSyncReadWriteReqEx2(nPort, pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlVar), &lHdlVa
r, sizeof(szVar), szVar, &cbReturn);
    if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';

    while(bRun)
    {
        DWORD wait = 0;

        // Wait 500ms for stopEvent
        wait = WaitForSingleObject(parameter, 500);

        // If timeout elapse read plc data
        if(WAIT_TIMEOUT == wait)
        {
            // Read the value of a PLC-variable, via handle
            nErr = AdsSyncReadReqEx2(nPort, pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nD
ata, &cbReturn);
            if (nErr)
                cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
            else
                cout << "Actual thread --> Thread2 Value: " << nData << '\n';
            cout.flush();
        }

        else
        {
            bRun = false;
        }
    }

    // Release the handle of the PLC-variable
    nErr = AdsSyncWriteReqEx(nPort, pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlVar), &lHdlVar);
    if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

    // Close the communication port
    nErr = AdsPortCloseEx(nPort);
    if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';

    return 1;
}

void main()
{
    // StopEvents are used to end the threads
    HANDLE hStopThread1 = CreateEvent(0, FALSE, FALSE, "StopEvent1");
    HANDLE hStopThread2 = CreateEvent(0, FALSE, FALSE, "StopEvent2");

    DWORD threadId1 = 1;
    DWORD threadId2 = 2;

    // Thread creation
    HANDLE hThread1 = CreateThread(0,               // no security
                0,              // default stack size
                ThreadFunction1,
                (void*)hStopThread1,    // parameter
                0,              // creation flags
                &threadId1          // threadId
                );

    HANDLE hThread2 = CreateThread(0,               // no security
                0,              // default stack size
                ThreadFunction2,
                (void*)hStopThread2,    // parameter
                0,              // creation flags
                &threadId2          // threadId
                );

    // Loop only for demonstration
    for(int i = 0; i < 10; ++i)
    {
        cout << "Actual thread --> Main Thread" << '\n';
        Sleep(1000);
```

```
    }

    // Stop threads
    SetEvent(hStopThread1);
    SetEvent(hStopThread2);
    cout << "Press key to exit: ";
    _getch();
}
```

5.1.16 Auslesen der Variablendeklaration (dynamisch)

In diesem Beispiel wird beschrieben, wie die SPS-Symbolinformationen auf eine effizientere dynamische Weise hochgeladen werden können.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470837515.zip

Die Informationen des SPS-Symbols bestehen aus den folgenden Teilen:

- variablename
- datatype
- length
- address (IndexGroup / IndexOffset)
- comment

Wir empfehlen dringend, NICHT mit dieser IndexGroup/IndexOffset für die ADS-Kommunikation zu arbeiten, sondern stattdessen Handles von Symbolen für die ADS-Kommunikation zu verwenden. Nach dem Hochladen der Informationen "name", "datatype" und "length" ist es sinnvoll, ein Handle für dieses Symbol anzufordern.

Einlesen der wichtigsten Informationen über ADS in die interne Klasse **"CAdsParseSymbols"** :

```
// Read major symbol information via ADS from device

AdsSymbolUploadInfo2 info;
nResult = AdsSyncReadReq(&m_amsAddr, ADSIGRP_SYM_UPLOADINFO2, 0, sizeof(info), &info);
if ( nResult == ADSERR_NOERR )
{
    // size of symbol information
    PBYTE pSym = new BYTE[info.nSymSize];
    if ( pSym )
    {
        // upload symbols (instances)
        nResult = AdsSyncReadReq(&m_amsAddr, ADSIGRP_SYM_UPLOAD, 0, info.nSymSize, pSym);
        if ( nResult == ADSERR_NOERR )
        {
            // get size of datatype description
            PBYTE pDT = new BYTE[info.nDatatypeSize];
            if ( pDT )
            {
                // upload datatype descriptions
                nResult = AdsSyncReadReq(&m_amsAddr, ADSIGRP_SYM_DT_UPLOAD, 0, info.nDatatypeSize, p
DT);
                if ( nResult == ADSERR_NOERR )
                {
                    // create class-object for each datatype-description
                    m_pDynSymbols = new CAdsParseSymbols(pSym, info.nSymSize, pDT, info.nDatatypeSiz
e);
                    if ( m_pDynSymbols == NULL )
                        nResult = ADSERR_DEVICE_NOMEMORY;
                    }
                    delete pDT;
                }
            }
            delete pSym;
        }
    }
}
```

Get Parent:
Die Routine Get Parent springt NICHT zu dem direkten Parent eines Childs. Stattdessen springt dieser Befehl zum nächsten Eintrag des direkten Parent.

Get Sibling:
Bei Auswahl dieser Option wird das nächste Symbol innerhalb der aktuellen Hierarchiestufe angezeigt. Symbole, die untergeordnete Informationen enthalten, werden angezeigt, aber die untergeordneten Elemente werden nicht angezeigt.

Get Child:
Wenn das aktuelle Symbol Informationen über ein untergeordnetes Symbol enthält (das aktuelle Symbol ist also eine Instanz einer Datentypbeschreibung), wird mit diesem Befehl die nächste Hierarchieebene aufgerufen und die Informationen über das erste untergeordnete Objekt zurückgegeben.

Get Next:
Wenn Sie auf diese Schaltfläche klicken, wird das nächste Symbol aus der internen Klasse "CAdsParseSymbols" extrahiert und angezeigt.

Wenn Sie immer nur diese Option wählen, können Sie vom ersten ADS-Symbol durch den Symbolbaum der Hierarchie bis zum Ende der Liste navigieren.

5.1.17 ADS-sum command: Read or Write a list of variables with one single ADS-command

https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470838923.zip

This sample describes how to read multiple single variables with one single ADS API call.

Note that ADS is just a transport layer, but there could be important side effects : So read the requirements and keep the limitations in mind.

Background :

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

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

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

REQUIREMENTS AND IMPORTANT LIMITATIONS :

Note that ADS is just a transport layer, but there could be important side effects. So read the requirements and take care on limitations :

- **Version of target ADS Device:**
  ADS itself is just the transport layer, but the requested ADS device has to support the ADS-Command.

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

- **Number of Sub-ADS calls : Highly recommended to max. 500 !**
  If the PLC is processing one ADS request, it will solely work on this single ADS request BEFORE starting the next PLC cycle.
  As a result one single ADS request with 200.000 sub-Ads-requests would cause that PLC would collect and copy 200.000 variables into one single ADS response, before starting next PLC.
  So this large number of ads-sub-commands will jitter the PLC execution !
  We highly recommend to not request more than 500 Ads-Sub commands

```
// This code snippet uses ADSIGRP_SUMUP_READ with IndexGroup 0xF080 and IndexOffset as number
of ADS-sub-commands
// Demonstrates how to read a list of variables, see full demo-code
// Use ADS-
ReadWrite request : "Write" the requested data down to ADS device and "Read" the received answ
er
 nErr = AdsSyncReadWriteReq(    pAddr,
                0xF080,         // Sum-Command, response will contain ADS-
error code for each ADS-Sub-command
                reqNum,     // number of ADS-Sub-Commands
                4*reqNum+reqSize, // number requested bytes in the sample two variables each 4
 bytes. NOTE : we request additional "error"-flag(long) for each ADS-sub commands
                (void*)(mAdsSumBufferRes), // provide buffer for response
                12*reqNum,      // send 12 bytes for each variable (each ads-
Sub command consist of 3 * ULONG : IG, IO, Len)
                &parReq);
```

```
// This code snippet uses ADSIGRP_SUMUP_WRITE with IndexGroup 0xF081 and IndexOffset as number of AD
S-sub-commands
// Demonstrates how to write a list of variables, see full demo-code
// Use ADS-
ReadWrite request : "Write" the send a list of data to list of variables down to ADS device and "Rea
d" the received return codes
 nErr = AdsSyncReadWriteReq(    pAddr,
                0xF081,     // ADS list-write command
                reqNum,     // number of ADS-Sub commands
                4*reqNum,           // we expect an ADS-error-return-code (long) for each ADS-
Sub command
                (void*)
(mAdsSumBufferRes), // provide space for the response containing the return codes
                16*reqNum,      // cbyteLen : in THIS sample we send two variables each 4 byte
                        // --> send 16 bytes (IG1, IO1, Len1, IG2, IO2, Len2, Data1, Data2)
                &parReq);   // buffer with data
```

5.1.18 ADS-Summenkommando: Holen und Freigeben von mehreren Handles

Dieses Beispiel zeigt, wie man unter Zuhilfenahme des ADS-Summenkommandos, viele Handles holen und wieder freigeben kann. Aufgebaut als AdsSyncReadWriteRequest, dient es als Behälter, in dem die Unterkommandos transportiert werden.

Anforderung: TwinCAT 2.11 >= Build 1550

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470840331.zip

1. Handles holen

Als erstes, werden die benötigten Header eingebunden.

```
#include <iostream.h>
#include <windows.h>
#include <conio.h>
```

```
// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"
```

Es folgt das Definieren einer Struktur, Deklarieren von Variablen und Reservieren von Speicher.

```
// Structure declaration for valuestypedef struct dataReq
    {
        unsigned long       indexGroup;     // index group in ADS server interfaceunsigned long
  indexOffset;    // index offset in ADS server interfaceunsigned long       rlength;        // coun
t of bytes to readunsigned long       wlength;        // count of bytes to write
    }TDataReq, *PTDataReq;


    // Variables declaration
    AmsAddr     Addr;
    LONG        nErr, nPort;
    PAmsAddr        pAddr = &Addr;

    char        szVar1[] = {".bVar01"};
    char        szVar2[] = {".bVar02"};

    // Allocate memory
    ULONG   cbReq   = ( sizeof(TDataReq)*2 ) + sizeof(szVar1) + sizeof(szVar2);
    BYTE*   pBuffReq    = new BYTE[cbReq];

    BYTE*   pBuffRes    = new BYTE[24];

    // Put structure over memory
    PTDataReq           pDataReq = (PTDataReq)pBuffReq;
    ULONG*      pDataRes = (ULONG*)pBuffRes;
```

Die zu übertragenden Werte werden hinter die letzte Struktur geschrieben.

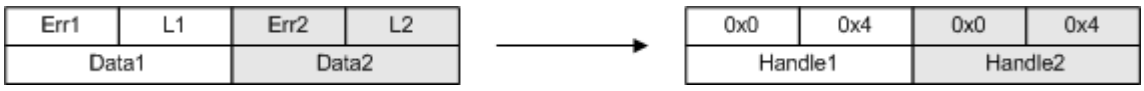

```
// pDataReq-> structure 1
    pDataReq->indexGroup    = ADSIGRP_SYM_HNDBYNAME;
    pDataReq->indexOffset   = 0x0;
    pDataReq->rlength           = sizeof(ULONG);
    pDataReq->wlength           = sizeof(szVar1);

    // Skip to next structure
    pDataReq = pDataReq+1;

    // pDataReq-> structure 2
    pDataReq->indexGroup    = ADSIGRP_SYM_HNDBYNAME;
    pDataReq->indexOffset   = 0x0;
    pDataReq->rlength           = sizeof(ULONG);
    pDataReq->wlength           = sizeof(szVar2);

    // Skip to write data 1char*   szVarName = ( (char*)pDataReq ) + sizeof(TDataReq);
    strncpy( szVarName, szVar1, sizeof(szVar1) );

    // Skip to write data 2
    szVarName = szVarName + sizeof(szVar1);
    strncpy( szVarName, szVar2, sizeof(szVar2) );
```

Für die Kommunikation wird ein Port geöffnet und die lokale Adresse übergeben. Kommt es zur Übertragung wird vorher der Port vom Laufzeitsystem 1 der Adresse zugewiesen.

Die Parameter für das Summenkommando bestehen aus *IndexGroup (0xf082)* - Aufruf des Summenkommandos, *IndexOffset (0x2)* - Anzahl der Unterkommandos, *ReadLength (0x18)* - Größenangabe der zu lesenden Daten, *ReadData (pBuffRes)* - Speicher, der gelesene Daten entgegen nimmt, *WriteLength (cbReq)* - Größenangabe der zu sendenen Daten und *WriteLength (pBuffReq)* - Speicher, der zu sendende Daten enthält.

```
    // Kommunikationsport auf dem ADS Router öffnen
    nPort = AdsPortOpen();
    nErr  = AdsGetLocalAddress(pAddr);

    cout << "open port:  ";
        if (nErr == 0)
        {
            cout << "OK" << '\n';

            // Get handles
            pAddr->port = AMSPORT_R0_PLC_RTS1;
            nErr = AdsSyncReadWriteReq(
                pAddr,
                0xf082,     // ADS list-read-write command
                0x2,        // number of ADS-sub commands
                0x18,       // we expect an ADS-error-return-code for each ADS-sub command
                pBuffRes,           // provide space for the response containing the return codes
                cbReq,      // cbReq : send 48 bytes (IG1, IO1, RLen1, WLen1,// IG2, IO2, RLen2, WLe
n2, Data1, Data2)
                pBuffReq );     // buffer with data
        }
        else {cout << "ERROR [" << nErr << "]" << '\n';};

    cout << "connect:    ";
        if (nErr == 0)
        {
            cout << "OK" << '\n';

            // Skip to handle 1 and examine the value
            ULONG nVarHandle = *( (ULONG*)pBuffRes );
            if (nVarHandle != 0)
            {
                cout << " > handle1: ";
                cout << "ERROR [" << nVarHandle << "]" << '\n';
            }

            // Skip to handle 2 and examine the value
            nVarHandle = *( (ULONG*)pBuffRes + 2 );
            if (nVarHandle != 0)
            {
                cout << " > handle2: ";
                cout << "ERROR [" << nVarHandle << "]" << '\n';
            }
        }
        else {cout << "ERROR [" << nErr << "]" << '\n';};
```

Nach dem Senden des Request, erwarten wir einen ADS Fehlercode und Länge für jeden Handle den wir versuchen zu bekommen.



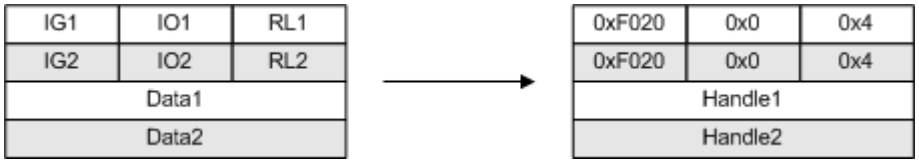
2. Handles freigeben

Es wird ein weiteres Mal eine Struktur definiert, Variablen deklariert und Speicher reserviert.

```
// Structure declaration for valuestypedef struct dataRel
    {
        unsigned long       indexGroup;     // index group in ADS server interfaceunsigned long
  indexOffset;    // index offset in ADS server interfaceunsigned long       length;     // count of
 bytes to write
    }TDataRel, *PTDataRel;


    // Variables declaration
    ULONG*  nVar1   = (ULONG*)pBuffRes+4;
    ULONG*  nVar2   = (ULONG*)pBuffRes+5;

    // Allocate memory
    ULONG   cbRel      = sizeof(TDataRel)*2 + sizeof(ULONG)*2;
    BYTE*   pBuffRel    = new BYTE[cbRel];

    ULONG   cbRelRes    = sizeof(ULONG)*2;
    BYTE*   pBuffRelRes = new BYTE[cbRelRes];
```

```
    // Put structure over memory
    PTDataRel  pDataRel    = (PTDataRel)pBuffRel;
    ULONG*    pDataRelRes = (ULONG*)pBuffRelRes;
```

Die zu empfangenen Werte werden wieder hinter die letzte Struktur geschrieben.




```
// pDataRel-> structure 1
    pDataRel->indexGroup    = ADSIGRP_SYM_RELEASEHND;
    pDataRel->indexOffset   = 0x0;
    pDataRel->length        = sizeof(ULONG);

    // Skip to next structure
    pDataRel++;

    // pDataReq-> structure 2
    pDataRel->indexGroup    = ADSIGRP_SYM_RELEASEHND;
    pDataRel->indexOffset   = 0x0;
    pDataRel->length        = sizeof(ULONG);

    // Skip to next structure
    pDataRel++;

    // Write handles into structure
    memcpy(    pDataRel,   nVar1, sizeof(ULONG) );
    memcpy( (ULONG*)pDataRel+1, nVar2, sizeof(ULONG) );
```

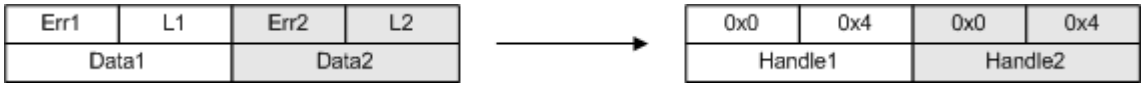
Für die Freigabe der Handles wird die bestehende Verbindung benutzt.

Die Parameter für das Summenkommando bestehen aus *IndexGroup (0xf081)* - Aufruf des Summenkommandos, *IndexOffset (0x2)* - Anzahl der Unterkommandos, *ReadLength (cbRelRes)* - Größenangabe der zu lesenden Daten, *ReadData (pBuffRelRes)* - Speicher, der gelesene Daten entgegen nimmt, *WriteLength (cbRel)* - Größenangabe der zu sendenen Daten und *WriteLength (pBuffRel)* - Speicher, der zu sendene Daten enthält.Zum Schluss müssen die Handles freigegeben und der Port geschlossen werden.

```
// Release handles
    nErr = AdsSyncReadWriteReq(
        pAddr,
        0xf081,     // ADS list-write command
        0x2,        // number of ADS-sub commands
        cbRelRes,           // we expect an ADS-error-return-code for each ADS-sub command
        pBuffRelRes,    // provide space for the response containing the return codes
        cbRel,      // cbReq : send 40 bytes (IG1, IO1, Len1, IG2, IO2, Len2, Data1, Data2)
        pBuffRel );     // buffer with data

    cout << "disconnect: ";
        if (nErr == 0)
        {
            cout << "OK" << '\n';

            // Skip to handle 1 and examine the value
            ULONG nVarHandle = *( (ULONG*)pBuffRes );
            if (nVarHandle != 0)
            {
                cout << " > handle1: ";
                cout << "ERROR [" << nVarHandle << "]" << '\n';
            }

            // Skip to handle 2 and examine the value
            nVarHandle = *( (ULONG*)pBuffRes + 2 );
            if (nVarHandle != 0)
            {
                cout << " > handle2: ";
                cout << "ERROR [" << nVarHandle << "]" << '\n';
            }
        }
        else {cout << "ERROR [" << nErr << "]" << '\n';};
```

```
    // Close the communication port
    nErr = AdsPortClose();
    cout << "close port: ";
        if (nErr == 0) {cout << "OK" << '\n' << "--------------------" << '\n';}
        else {cout << "ERROR [" << nErr << "]" << '\n' << "--------------------" << '\n';}

    cout.flush();

    // Wait for key press
    getch();
}
```

Als Antwort erhalten wir einen ADS Fehlercode für jeden Handle den wir versuchen freizugeben.



5.1.19 Übertragen von Strukturen an die SPS

Dieses Beispiel zeigt die Übertragung einer Struktur an die SPS per ADS. Die Struktur besteht aus Elementen von verschiedenen Datentypen.

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470841739.zip

```
#include <stdio.h>
#include <tchar.h>
#include "windows.h"

#include "c:\twincat\adsapi\tcadsdll\include\tcadsdef.h"
#include "c:\twincat\adsapi\tcadsdll\include\tcadsapi.h"

// Create new struct
typedef struct PlcStruct {
    INT16   shortVal;
    INT32   intVal;
    byte    byteVal;
    DOUBLE  doubleVal;
    FLOAT   floatVal;
} SPlcVar, *pSPlcVar;

int _tmain(int argc, _TCHAR* argv[])
{
    long     nErr, nPort;
    AmsAddr  Addr;
    PAmsAddr pAddr = &Addr;
    ULONG     lHdlVar;

    // New struct. Assign test values
    PlcStruct PlcVar;

    PlcVar.shortVal = 1;
    PlcVar.intVal = 2;
    PlcVar.byteVal = 3;
    PlcVar.doubleVal = 4.04;
    PlcVar.floatVal = (FLOAT)5.05;

    // Declare PLC variable which should notify changes
    char      szVar []={"MAIN.PLCVar"};


    // Extract values from struct and write to byte array
    // Circumvent memory holes caused by padding
    BYTE *pData = new BYTE[19];
    int nIOffs = 0;

    memcpy_s(&pData[nIOffs], 19, &PlcVar.shortVal, 2);
    nIOffs += 2;
    memcpy_s(&pData[nIOffs], 17, &PlcVar.intVal, 4);
    nIOffs += 4;
    memcpy_s(&pData[nIOffs], 13, &PlcVar.byteVal, 1);
    nIOffs++;
    memcpy_s(&pData[nIOffs], 12, &PlcVar.doubleVal, 8);
    nIOffs += 8;
    memcpy_s(&pData[nIOffs], 4, &PlcVar.floatVal, 4);
```

```
    // Open communication port on the ADS router
    nPort = AdsPortOpen();
    nErr = AdsGetLocalAddress(pAddr);
    if (nErr) printf("Error: Ads: Open port: %d\n", nErr);
    pAddr->port = 801;

    // Get variable handle
    nErr = AdsSyncReadWriteReq(pAddr,
                ADSIGRP_SYM_HNDBYNAME,
                0x0,
                sizeof(lHdlVar),
                &lHdlVar,
                sizeof(szVar),
                szVar);

    // Write the struct to the Plc
    AdsSyncWriteReq(    pAddr,
            ADSIGRP_SYM_VALBYHND,   // IndexGroup
            lHdlVar,            // IndexOffset
            0x13,           // Size of struct
            (void*) pData);

    if (nErr) printf("Error: Ads: Write struct: %d\n", nErr);
    else  printf("Structure successful written\n");

    // Close communication
    delete [] pData;

    //Release handle of plc variable
    nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlVar), &lHdlVar);
    if (nErr) printf("Error: AdsSyncWriteReq: %d \n", nErr);

    nErr = AdsPortClose();
    if (nErr) printf("Error: Ads: Close port: %d\n", nErr);

    getchar();
}
```

5.1.20 Lesen und Schreiben von DATE/TIME-Variablen

Die SPS enthält die TIME-Variable MAIN.Time1 und die DT-Variable MAIN.Date1. Dieses Beispielprojekt zeigt, wie diese Werte gelesen, angezeigt und geschrieben werden können:

Download: https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470843147.zip

```
#include <time.h>

// ADS headers
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"

#define TIME_LENGHT 56
#define DATE_LENGHT 62
#define MON_START 1
#define YEAR_START 1900

int _tmain(int argc, _TCHAR* argv[])
{
    long lErr, lPort;
    long lTime, lMs, lSek, lMin, lHour, lDay;
    AmsAddr  Addr;
    PAmsAddr pAddr = &Addr;
    DWORD dwTime, dwDate;
    ULONG     lHdlTime, lHdlDate;

    // Declare PLC variable
    char      szPlcTime []={"MAIN.Time1"};
    char      szPlcDate []={"MAIN.Date1"};

    // Open the communication
    lPort = AdsPortOpen();
    lErr = AdsGetLocalAddress(pAddr);
    if(lErr) printf_s((char*)"Error: Getting local adress: 0x%i  \n", lErr);

    // TwinCAT2 RTS1 Port = 801, TwinCAT3 RTS1 Port = 851
    pAddr->port = 801;

    // Get variable handle
```

```
    lErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlTime), &lHdlTime, sizeo
f(szPlcTime), szPlcTime);
    lErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlDate), &lHdlDate, sizeo
f(szPlcDate), szPlcDate);

    // Read from MAIN.Time1
    lErr = AdsSyncReadReq(pAddr,
                          ADSIGRP_SYM_VALBYHND,   // IndexGroup
                          lHdlTime,               // IndexOffset
                          0x4,            // Size of DWORD
                          &dwTime);
    if(lErr) printf_s((char*)"Error: Read time variable: 0x%i  \n", lErr);

    //Convert DWORD to Time
    lTime = (long)dwTime;
    lMs         = (lTime % 1000);
    lSek    = (lTime / 1000) % 60;
    lMin    = (lTime / 60000) % 60;
    lHour   = (lTime / 3600000) % 24;
    lDay    = (lTime / 86400000) % 365;

    wchar_t szTime[TIME_LENGHT];
    wsprintf(szTime, L"Time from PLC: %dd %dh %dm %ds %dms \n",
                        lDay,
                        lHour,
                        lMin,
                        lSek,
                        lMs);
    wprintf_s(szTime);

    //Write to MAIN.Time1
    //Manipulate DWORD for demonstration
    dwTime += 3600000;  //Add 3600000ms (One hour)

    //AdsWrite
    lErr = AdsSyncWriteReq(pAddr,
                          ADSIGRP_SYM_VALBYHND,   //IndexGroup
                          lHdlTime,               //IndexOffset
                          0x4,
                          &dwTime);
    if(lErr) printf_s((char*)"Error: Write time variable: 0x%i \n", lErr);

    //Read from MAIN.Date1
    //AdsRead
    lErr = AdsSyncReadReq(pAddr,
                          ADSIGRP_SYM_VALBYHND,   //IndexGroup
                          lHdlDate,               //IndexOffset
                          0x4,
                          &dwDate);
    if(lErr) printf_s((char*)"Error: Read date variable: 0x%i  \n", lErr);

    //Convert long to date
    time_t tDate(dwDate);
    tm tmDate;

    gmtime_s(&tmDate, &tDate);

    wchar_t szDate[DATE_LENGHT];
    wsprintf(szDate, L"Date from PLC: %d/%d/%d %d:%d:%d \n",
                tmDate.tm_mday,
                tmDate.tm_mon + MON_START,
                tmDate.tm_year + YEAR_START,
                tmDate.tm_hour,
                tmDate.tm_min,
                tmDate.tm_sec);
    wprintf_s(szDate);

    //Write to MAIN.Date1
    //Manipulate DWORD for demonstration
    dwDate += 3600; //Add 3600s (One hour)

    //AdsWrite
    lErr = AdsSyncWriteReq(pAddr,
                          ADSIGRP_SYM_VALBYHND,   //IndexGroup
                          lHdlDate,               //IndexOffset
                          0x4,
                          &dwDate);
    if(lErr) printf_s((char*)L"Error: Write time variable: 0x%i \n", lErr);

    //Releases handles of plc variable
```

```
    lErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlTime), &lHdlTime);
    if (lErr) printf("Error: AdsSyncWriteReq: %d \n", lErr);
    lErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlDate), &lHdlDate);
    if (lErr) printf("Error: AdsSyncWriteReq: %d \n", lErr);

    //Close the communication
    lErr = AdsPortClose();
    if(lErr) printf_s((char*)L"Error: Closing connection: 0x%i \n", lErr);

    printf_s("\nPress enter to exit..");
    getchar();
}
```

5.2 Beispiele: Visual C++ für Windows CE

Beschreibung	Quelltexte
Beispiel 1: Auslesen einer SPS-Variablen	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470844555.zip
Beispiel 2: Wert in eine SPS-Variable schreiben	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470845963.zip
Beispiel 3: Zugriff per Variablenname	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470847371.zip
Beispiel 4: Ereignisgesteuertes Lesen	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12470848779.zip

5.2.1 Verbindung mit Visual Studio

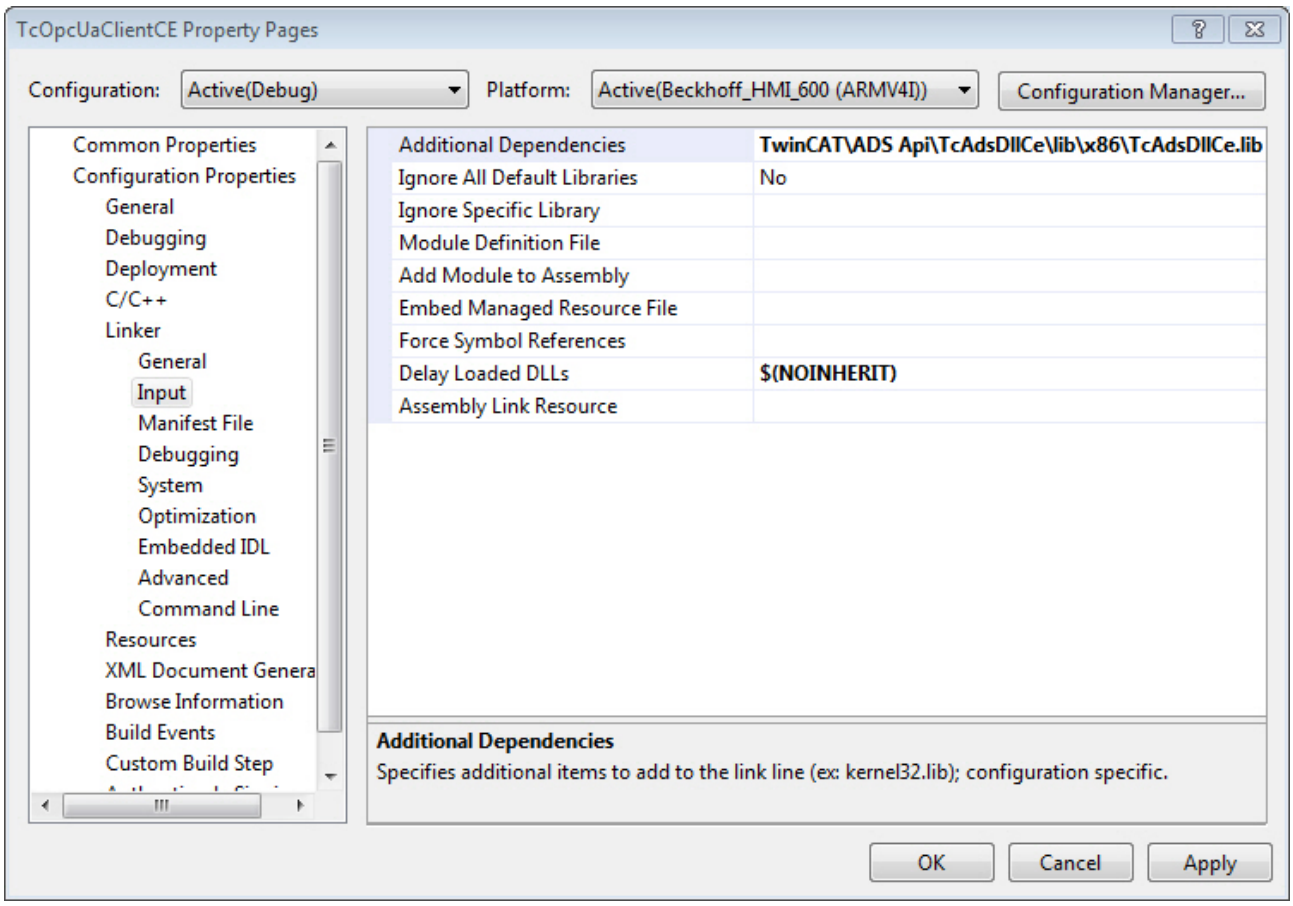
Um die **TcAdsDllCe** in ihrem **Visual Studio** Projekt zu verwenden, müssen Sie folgende Schritte durchführen:

- Project Settings Dialog öffnen
- Link Tab auswählen und im Editierfeld *Object/library modules* die für ihren Prozessortyp spezifische Bibliothek auswählen:

- StrongArm: TwinCATDirectory\AdsApi\TcAdsDll\CE\lib\arm\TcAdsDllCe.lib

- MIPS: StrongArm: TwinCATDirectory\AdsApi\TcAdsDll\CE\lib\mips\TcAdsDllCe.lib

- X86: TwinCATDirectory\AdsApi\TcAdsDll\CE\lib\x86\TcAdsDllCe.lib



Die Methoden der TcAdsDllCE.dll sind kompatibel zur TcAdsDll.dll.

Zusätzlich benötigen sie das passende SDK für Windows CE. Sie finden es auf unseren FTP Server.

Visual Studio kann sich direkt mit einem Windows CE Gerät verbinden, um z.B die Applikation zu übertragen und online zu debuggen.

Lese Sie zum Konfigurieren der Verbindung bitte folgenden MSDN-Artikel.

5.3 Beispiele: Borland C++ Builder

Tab. 1: ADS-DLL Beispiele for Windows NT/2000/XP/CE

Beschreibung	Quelltexte
Einbinden in Borland C++ Builder [▶ 72]	
Beispiel 1: DLL-Version auslesen [▶ 74]	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473611531.exe
Beispiel 2: Ereignisgesteueertes Lesen [▶ 75]	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473612939.exe

5.3.1 Einbinden in Borland C++ Builder 5.0

Notwendige Dateien:

- TcAdsDll.dll - Dynamische Funktions-Bibliothek
- TcAdsApi.h - Deklarationen der Ads-Funktionen (C++ Headerdatei)
- TcAdsDef.h - Deklarationen der Strukturen und Konstanten (C++ Headerdatei)

Die TcAdsDll.dll befindet sich im ‚System32'-Verzeichnis von Windows. Die TcAdsApi.h und TcAdsDef.h befinden sich im *..\ADS Api\TcAdsDll\Include*-Verzeichnis von TwinCAT.

Notwendige Tools:

- Borland C++ Builder 5.0
- IMPLIB.EXE - generiert eine neue Import-Library TcAdsDLL.Lib aus der TcAdsDll.dll

Bemerkung

Die mit der TwinCAT -Installation mitgelieferte Library-Datei: TcAdsDll.lib kann aus Kompatibilitäts-Gründen nicht in Borland C++ Builder Projekten eingebunden und kompiliert werden. Beim Versuch, diese Lib zu linken, bekommt man folgende Fehlermeldung:

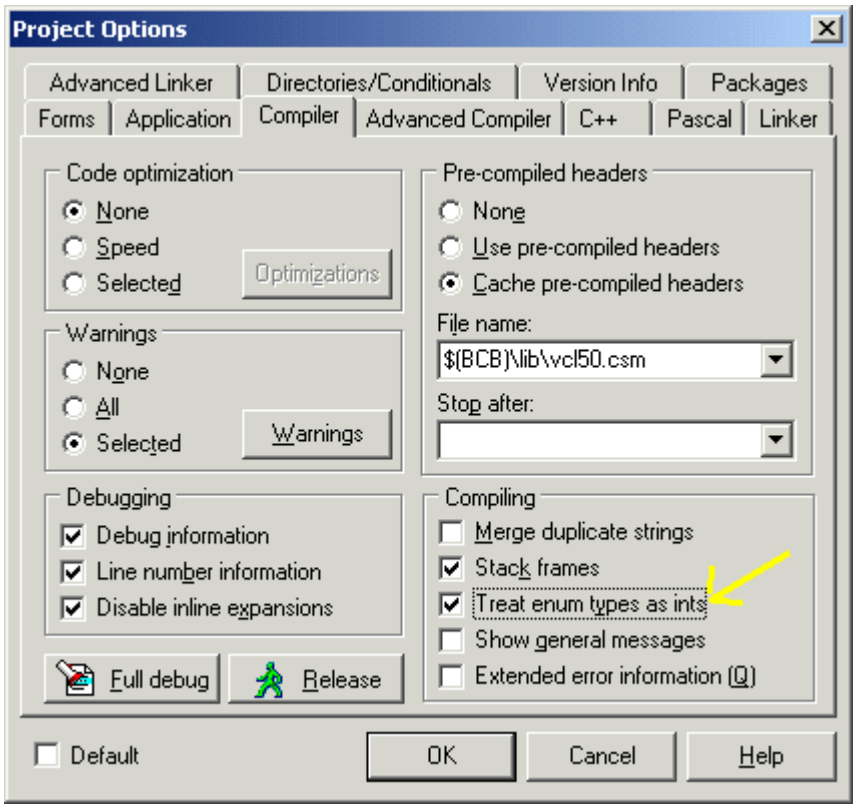
```
[Linker Error]
'E:\BORLAND\CBUILDER5\PROJECTS\SAMPLE1\TCADSDLL.LIB'
contains invalid OMF record, type 0x21 (possibly COFF)
```

Mit dem Borland-Utility: IMPLIB.EXE kann eine neue Library aus der TcAdsDll.dll generiert werden. Diese kann dann fehlerfrei kompiliert und gelinkt werden. IMPLIB.EXE befindet sich standardmäßig in dem Borland ...\Bin - Verzeichnis. Benutzen Sie dafür folgendes Kommando auf dem DOS-Prompt:

```
IMPLIB -a TcAdsDll
TcAdsDll.dll
```

Mit dem Schalter **-a** werden Microsoft-Kompatible Aliasnamen für die Dll-Funktionen generiert.

Für die Aufzählungstypen (Enums) in der TcAdsDEF.h müssen 4 Bytes (32 Bits) Speicher alloziert werden. Stellen Sie sicher daß unter den Projekt->Optionen auf dem Kompiler-Reiter folgende Option angewählt ist: "Treat enum types as ints":



Um auf die DLL-Funktionen zugreifen zu können müssen die beiden Header-Dateien in das Borland C++ Builder Projekt eingebunden werden. Z. B.:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"

#include "c:\TwinCAT\ADS Api\TcAdsDll\Include\TcAdsDEF.h"
```

```
#include "c:\TwinCAT\ADS Api\TcAdsDll\Include\TcAdsAPI.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

Die TcAdsDll.Lib kann über das Menue Project->Add to project... dem Projekt hinzugefügt werden.

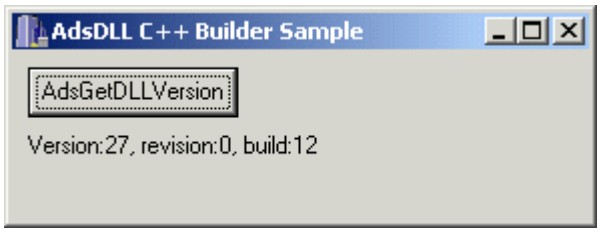
5.3.2 ADS-DLL-Version auslesen

Systemvoraussetzungen:

- Borland Builder 5.0;
- TwinCAT Version 2.9 oder höher;

Dieses Programm ermittelt die Version der DLL-Datei.

Beispielprogramm 'BCB: DLL-Version auslesen'(https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473611531.exe) entpacken.



Die C++ Applikation:

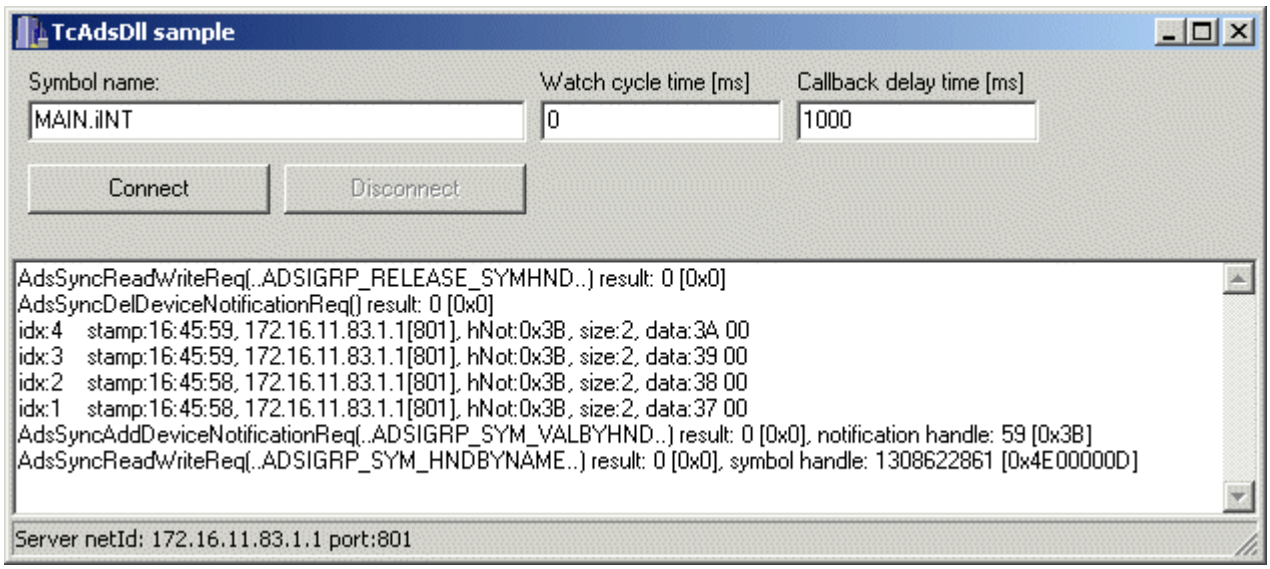
```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "AdsDllSampleUnit.h"
#include "c:\TwinCAT\ADS Api\TcAdsDll\Include\TcAdsDEF.h"
#include "c:\TwinCAT\ADS Api\TcAdsDll\Include\TcAdsAPI.h"


//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;


//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::GetVersionButtonClick(TObject *Sender)
{
    long    nTemp;
    AdsVersion* pDLLVersion;

    nTemp = AdsGetDllVersion();
    pDLLVersion = (AdsVersion *)&nTemp;

    Label1->Caption=Format("Version:%d, revision:%d, build:%d",
    ARRAYOFCONST(((int)pDLLVersion->version, (int)pDLLVersion->revision, (int)pDLLVersion->build)));
}
//---------------------------------------------------------------------------
```

5.3.3 Ereignisgesteuertes Lesen (by symbol name)

Systemvoraussetzungen:

- Borland Builder 5.0;
- TwinCAT Version 2.10 Build 1328 oder höher;

Beispielprogramm 'Ereignisgesteuertes Lesen (by symbol name) (https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473612939.exe) entpacken.



5.4 Beispiele: Delphi

5.4.1 API-Interface

5.4.1.1 Einbinden in Delphi (API-Schnittstelle)

Benötigte Dateien:

Zum Einbinden der DLL in Delphi-Programme benötigen Sie folgende Dateien:

- TcAdsDll.dll - dynamische Funktionsbibliothek

Die TcAdsDll.dll befindet sich im ‚System32'-Verzeichnis von Windows NT/2000.

Optional für statisches Linken:
HINWEIS: Die Deklarationsdateien TcAdsAPI.pas und TcAdsDEF.pas sind im Archiv https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip enthalten.

- Portierte TcAdsAPI.h C++ Headerdatei mit den Deklarationen der Ads-Funktionen: TcAdsAPI.pas
- Portierte TcAdsDEF.h C++ Headerdatei mit den Deklarationen der Strukturen und Konstanten: TcAdsDEF.pas

Um die Bespiele selbst übersetzen zu können benötigen Sie:

- Delphi 5.0 oder höher;
- ADS Summenkommando-Beispiele: Delphi 6.0 **+ Update Pack 2** oder höher;
- ADS Multithreading-Beispiel: Delphi 7.0 **+ Update 7.1** oder höher;

Theoretisch können Sie die Beispiele auch mit einer neueren Delphi-Version bearbeiten. Oft sind nach der Konvertierung des Projekts zusätzliche Anpassungen im Sourcecode an die neue Delphi Version vorzunehmen. Diese Anpassungen können z. B. durch eine Änderung im Compiler bei der Behandlung der VARIANT-Datentypen oder Stringdatentypen notwendig sein.

DLLs in Delphi aufrufen

Bevor Sie Routinen aufrufen können, die in der TcAdsDLL definiert sind, müssen Sie diese Routinen importieren. Der Import kann auf zwei Arten durchgeführt werden: Sie deklarieren eine Prozedur oder Funktion als external, oder Sie rufen die Windows-API direkt auf. Bei beiden Methoden werden die Routinen erst zur Laufzeit zur Anwendung gelinkt. Das bedeutet, dass die DLL zur Compilierzeit nicht benötigt wird. Es bedeutet aber auch, dass der Compiler nicht überprüfen kann, ob eine Routine korrekt importiert wird.

Statisches laden der DLL-Funktionen

Die einfachste Art, eine Prozedur oder Funktion zu importieren, besteht darin, sie mit der Direktive *external* zu deklarieren:

```
procedure MyDllFunction; external
'MYLIB.DLL';
```

Durch diese Deklaration wird beim Programmstart die Datei *MYLIB.DLL*geladen. Während der Ausführung des Programms bezieht sich der Bezeichner *MyDllFunction* immer auf denselben Eintrittspunkt in derselben DLL.

Sie können die Deklaration einer importierten Routine direkt in das Programm oder die Unit einfügen, in dem bzw. der sie aufgerufen wird. Um Ihre Programme leichter pflegen zu können, sollten Sie aber alle external-Deklarationen in einer separaten „Import-Unit“ zusammenfassen. Diese Unit enthält dann auch die Konstanten und Typen, die für die Schnittstelle zur DLL erforderlich sind (die Unit Windows von Delphi ist dafür ein gutes Beispiel). Andere Module, die auf die Import-Unit zugreifen, können alle darin deklarierten Routinen aufrufen.

Ausführliche Informationen über external-Deklarationen und dynamisches Laden über Aufrufe der Windows-API finden Sie in der Delphi Online-Hilfe.

Folgende Regeln sind bei einer Portierung der TcAdsDLL in Pascal-Applikationen einzuhalten:

- Alle C++ Makros müssen in entsprechenden Pascal-Funktionen umgewandelt werden;
- Alle C++ Datentypen müssen in Pascal-Typen umgewandelt werden;
- Die Datentypen der C++ Funktionsparameter müssen in entsprechenden Pascal-Datentypen umgewandelt werden;

Die wichtigsten Deklarationen der TcAdsDLL.DLL wurden in zwei Units zusammengefasst: TcAdsDEF.pas und TcAdsAPI.pas. Wenn Sie die Funktionen statisch laden wollen, müssen diese Units über eine Uses-Klausel in die Delphi-Applikation eingebunden werden.

Infos über Delphis memory manager in Multithreading-Applikationen

Der "memory manager" von Delphi benutzt "critical sections" um Threadsicherheit in Multithreading-Applikationen gewährleisten zu können. Die "critical sections" werden aber nur dann verwendet, wenn eine nicht so bekannte Systemvariable **IsMultiThread** auf True gesetzt wurde. Die **IsMultiThread** Variable ist in dem unit ‘System.pas’ definiert, welches implizit von allen Delphi units verwendet wird.

Immer dann, wenn die Applikation oder eine eingebundene Bibliothek mehrere Threads verwendet, müsste die Variable IsMultithread auf True gesetzt werden. Andernfalls ist die Applikation oder die Bibliothek nicht "thread safe".
IsMultiThread wird aber nur dann implizit gesetzt, wenn der Entwickler in der Applikation oder der Bibliothek das Delphi TThread Objekt benutzt.

Der memory manager von Delphi geht z. B. davon aus, dass es sich um eine Singlethread-Applikation handelt, wenn:

- Eine DLL eingebunden wird die mehrere Threads verwendet;
- Kein TThread Objekt vom Entwickler benutzt wird;

Die Zugriffe auf gemeinsame Resourcen werden in diesem Fall nicht verriegelt.

Die TcAdsDLL.dll verwendet, aber eigene Threads. **Aus diesem Grund muss sichergestellt werden, dass die IsMultiThread Variable auf True gesetzt ist sobald die TcAdsDLL in eine Delphi-Applikation eingebunden wird.**

In den aktuellen Versionen der Units: TcAdsDEF.pas und TcAdsAPI.pas wird die IsMultiThread Variable in der initialization section auf True gesetzt. Der Entwickler ist aber selbst dafür verantwortlich sicherzustellen, dass die Variable in einer Multithreading-Applikation zum richtigen Zeitpunkt auf jeden Fall gesetzt ist.

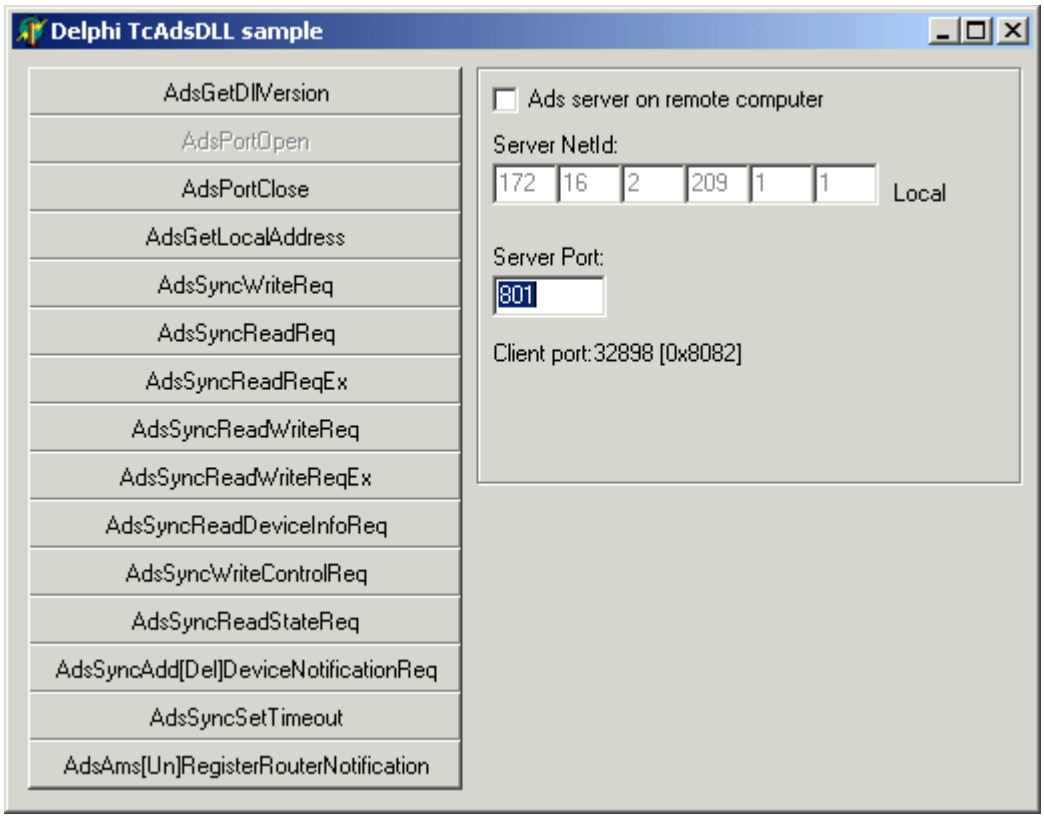
5.4.1.2 All-In-One Beispiel

In dieser Beispielapplikation werden alle Funktionen der DLL präsentiert und können durch einen Mausklick auf den entsprechenden Button ausprobiert werden. Wurde der Port erfolgreich geöffnet, dann kann auch ein Remote-PC für die Kommunikation ausgewählt werden.

Voraussetzungen:

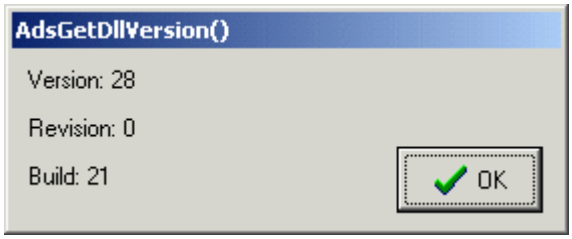
- Delphi 5.0 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selber übersetzen möchten;

Bei einem Mausklick auf eine der Tasten in der Grafik werden Sie zur Seite mit dem Sourcecode weiter geleitet.



Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473387915.exe
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466742027.exe

5.4.1.2.1 AdsGetDllVersion



Delphi 5 Programm

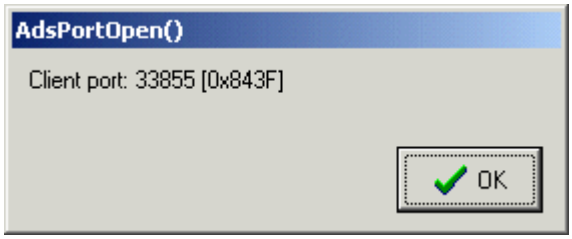
```
unit frmAdsGetDllVersionUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons;

type
  TfrmAdsGetDllVersion = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

implementation

{$R *.DFM}

procedure TfrmAdsGetDllVersion.FormCreate(Sender: TObject);
var tmp : Longword;
    pVersion :PAdsVersion;
begin
    tmp := AdsGetDllVersion();
    pVersion := PAdsVersion(@tmp);
    Label1.Caption := Format( 'Version: %d', [pVersion.version]);
    Label2.Caption := Format( 'Revision: %d', [pVersion.revision]);
    Label3.Caption := Format( 'Build: %d', [pVersion.build]);
end;

end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466742027.exe
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466742027.exe

Dokumente hierzu

- delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.2 AdsPortOpen



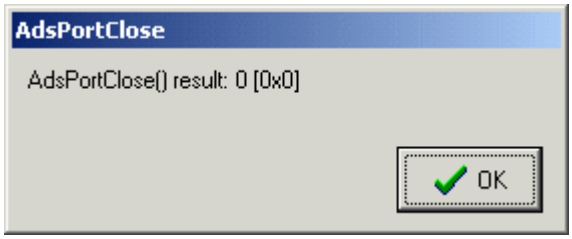
Delphi 5 Programm

```
unit frmAdsPortOpenUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons;

type
  TfrmAdsPortOpen = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    port : Longint;
  end;

implementation

{$R *.DFM}
//////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsPortOpen.FormCreate(Sender: TObject);
begin
     port := AdsPortOpen();
     Label1.Caption := Format( 'Client port: %d [0x%x]', [port,port] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.3 AdsPortClose



Delphi 5 Programm

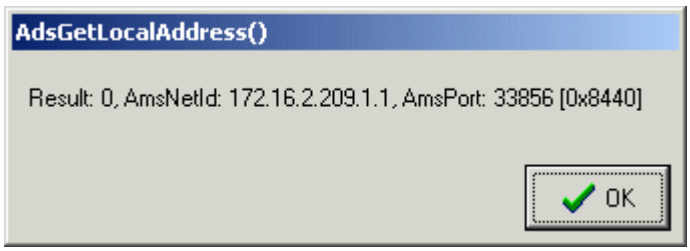
```
unit frmAdsPortCloseUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons;

type
  TfrmAdsPortclose = class(TForm)
```

```
    Label1: TLabel;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

implementation

{$R *.DFM}
////////////////////////////////////////////////////////////
procedure TfrmAdsPortclose.FormCreate(Sender: TObject);
var result : longint;
begin
     result :=AdsPortClose();
     Label1.Caption := Format( 'AdsPortClose() result: %d [0x%x]', [result, result]);
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.4 AdsGetLocalAddress



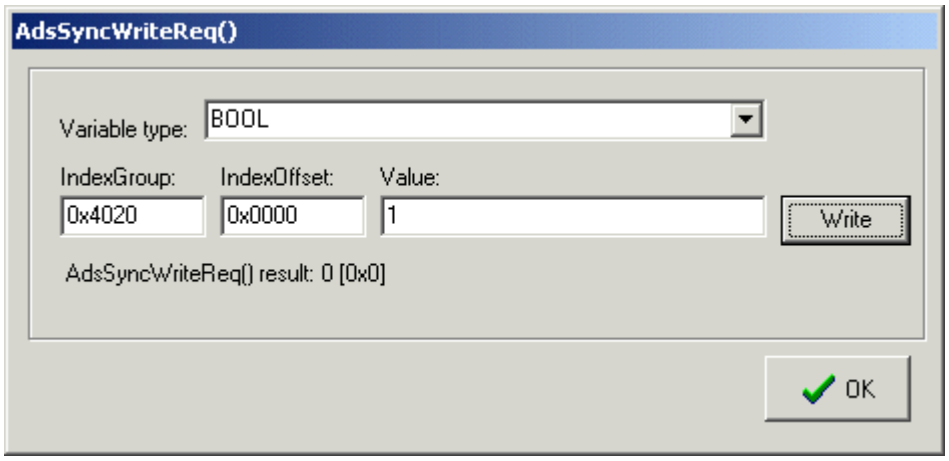
Delphi 5 Programm

```
unit frmAdsGetLocalAddressUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons ;

type
  TfrmAdsGetLocalAddress = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    addr : TAmsAddr;
  end;

implementation

{$R *.DFM}
////////////////////////////////////////////////////////////////////////
procedure TfrmAdsGetLocalAddress.FormCreate(Sender: TObject);
var result : Longint;
begin
     result  := AdsGetLocalAddress( @addr );

     Label1.Caption := Format( 'Result: %d, AmsNetId: %d.%d.%d.%d.%d.%d, AmsPort: %d [0x%x]',[
            result,addr.netid.b[0],addr.netid.b[1],addr.netid.b[2],
            addr.netid.b[3],addr.netid.b[4],addr.netid.b[5],
            addr.port,  addr.port] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.5 AdsSyncWriteReq



Delphi 5 Programm

```
unit frmAdsSyncWriteReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls ;

type
  TfrmAdsSyncWriteReq = class(TForm)
    editIO: TEdit;
    editIG: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    editValue: TEdit;
    Label1: TLabel;
    Label5: TLabel;
    ComboBox1: TComboBox;
    Label4: TLabel;
    BitBtn1: TBitBtn;
    Button1: TButton;
    Bevel1: TBevel;
    procedure ComboBox1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private-Deklarationen }
    serverAddr      : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );
  end;

type TPlcDataTypes =
(
    plcBOOL,
    plcBYTE,  //and USINT
    plcWORD,  //and UINT
    plcDWORD, //and UDINT
    plcSINT,
    plcINT,
    plcDINT,
    plcREAL,
    plcLREAL,
    plcSTRING
);


implementation

{$R *.DFM}
```

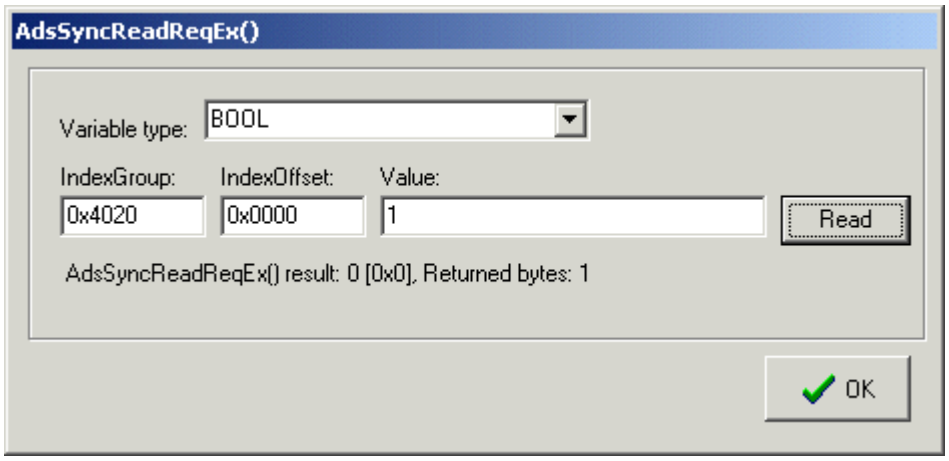
```
////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncWriteReq.InitData( destAddr : TAmsAddr );
begin
     serverAddr := destAddr;
     ComboBox1.itemindex := 0;
     ComboBox1Change(self);
end;
////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncWriteReq.Button1Click(Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
    indexGroup      : Longword;
    indexOffset     : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT        : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;
begin
     result := 0;
     dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

     indexGroup := Longword(StrToInt(editIG.Text));
     indexOffset := Longword(StrToInt(editIO.Text));

     case dataType of
       plcBOOL:
         begin
          nBYTE := Byte(StrToInt(editValue.Text) And $01);
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(nBYTE), @nBYTE  );
         end;
       plcBYTE:
         begin
          nBYTE := Byte(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(nBYTE), @nBYTE  );
         end;
       plcWORD:
         begin
          nWORD := Word(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(nWORD), @nWORD  );
         end;
       plcDWORD:
         begin
          nDWORD := Longword(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(nDWORD), @nDWORD  )
;
         end;
       plcSINT:
         begin
          iSINT := Shortint(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(iSINT), @iSINT  );
         end;
       plcINT:
         begin
          iINT := Smallint(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(iINT), @iINT  );
         end;
       plcDINT:
         begin
          iDINT := Longint(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(iDINT), @iDINT  );
         end;
       plcREAL:
         begin
          fREAL := StrToFloat(editValue.Text);
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(fREAL), @fREAL  );
         end;
       plcLREAL:
         begin
          fLREAL := StrToFloat(editValue.Text);
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(fLREAL), @fLREAL  )
;
         end;
       plcSTRING:
         begin
```

```
        sSTRING := AnsiString(editValue.Text);
        if sSTRING = '' then
          sSTRING := ''#00;
        result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, Length(sSTRING)
+1, @sSTRING[1] );
         end;
     end;

     Label5.Caption := Format( 'AdsSyncWriteReq() result: %d [0x%x]', [result, result] );
end;

//////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncWriteReq.ComboBox1Change(Sender: TObject);
begin
     case  ComboBox1.ItemIndex of
       0:
          editValue.Text := '1';
       1..6:
          editValue.Text := '7';
       7..8:
          editValue.Text := '12.34';
       9:
          editValue.Text := 'my string';
     end;
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.6 AdsSyncReadReqEx



Delphi 5 Programm

```
unit frmAdsSyncReadReqExUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls ;

type
  TfrmAdsSyncReadReqEx = class(TForm)
    editIO: TEdit;
    editIG: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    editValue: TEdit;
    Label1: TLabel;
    Label5: TLabel;
    ComboBox1: TComboBox;
    Label4: TLabel;
    BitBtn1: TBitBtn;
    Button1: TButton;
    Bevel1: TBevel;
    procedure Button1Click(Sender: TObject);
```

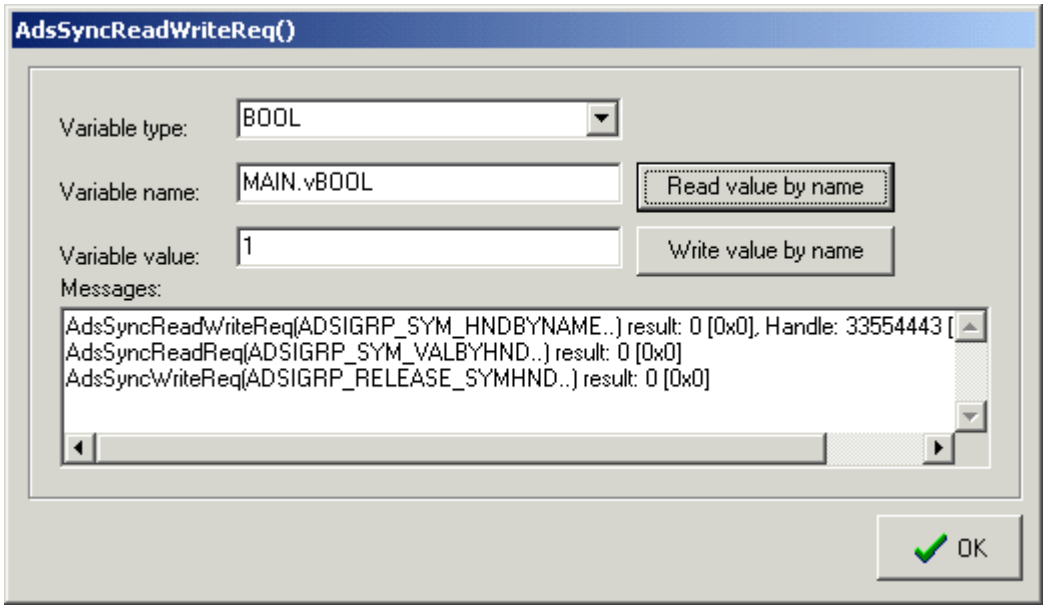
```
  private
    serverAddr      : TAmsAddr;
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );
  end;

type TPlcDataTypes =
(
    plcBOOL,
    plcBYTE,  //and USINT
    plcWORD,  //and UINT
    plcDWORD, //and UDINT
    plcSINT,
    plcINT,
    plcDINT,
    plcREAL,
    plcLREAL,
    plcSTRING
);

implementation

{$R *.DFM}
////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadReqEx.InitData( destAddr : TAmsAddr);
begin
     serverAddr := destAddr;
     ComboBox1.itemindex := 0;
end;
////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadReqEx.Button1Click( Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
    indexGroup      : Longword;
    indexOffset     : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT            : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;

    cbReturn        : Longword;
begin
     result := 0;
     dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

     indexGroup := Longword(StrToInt(editIG.Text));
     indexOffset := Longword(StrToInt(editIO.Text));

     case dataType of
       plcBOOL:
         begin
          result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(nBYTE), @nBYTE, @c
bReturn );
          editValue.Text := IntToStr(nByte And $01);
         end;
       plcBYTE:
         begin
          result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(nBYTE), @nBYTE, @c
bReturn  );
          editValue.Text := IntToStr(nByte);
         end;
       plcWORD:
         begin
          result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(nWORD), @nWORD, @c
bReturn  );
          editValue.Text := IntToStr(nWORD);
         end;
       plcDWORD:
         begin
          result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(nDWORD), @nDWORD,
@cbReturn  );
          editValue.Text := IntToStr(nDWORD);
```

```
        end;
      plcSINT:
        begin
         result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(iSINT), @iSINT, @c
bReturn  );
         editValue.Text := IntToStr(iSINT);
        end;
      plcINT:
        begin
         result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(iINT), @iINT, @cbR
eturn  );
         editValue.Text := IntToStr(iINT);
        end;
      plcDINT:
        begin
         result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(iDINT), @iDINT, @c
bReturn  );
         editValue.Text := IntToStr(iDINT);
        end;
      plcREAL:
        begin
         result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(fREAL), @fREAL, @c
bReturn  );
         editValue.Text := FloatToStr(fREAL);
        end;
      plcLREAL:
        begin
         result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, sizeof(fLREAL), @fLREAL,
@cbReturn  );
         editValue.Text := FloatToStr(fLREAL);
        end;
      plcSTRING:
        begin
         SetLength( sSTRING, 256 );//make sure the delphi string size is >= SIZEOF(plc string)
         result := AdsSyncReadReqEx(@serverAddr, indexGroup, indexOffset, Length(sSTRING), @sSTRING
[1], @cbReturn  );
         editValue.Text := String(sSTRING);
        end;
    end;

    Label5.Caption := Format( 'AdsSyncReadReqEx() result: %d [0x%x], Returned bytes: %d', [result,
result, cbReturn] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.7 AdsSyncReadWriteReq



Delphi 5 Programm

```
unit frmAdsSyncReadWriteReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls ;

type
  TfrmAdsSyncReadWriteReq = class(TForm)
    editValue: TEdit;
    Label1: TLabel;
    ComboBox1: TComboBox;
    Label4: TLabel;
    BitBtn1: TBitBtn;
    Button1: TButton;
    Bevel1: TBevel;
    editName: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private-Deklarationen }
    serverAddr      : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );
  end;

type TPlcDataTypes =
(
    plcBOOL,
    plcBYTE,  //and USINT
    plcWORD,  //and UINT
    plcDWORD, //and UDINT
    plcSINT,
    plcINT,
    plcDINT,
    plcREAL,
    plcLREAL,
    plcSTRING
);


implementation

{$R *.DFM}
////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadWriteReq.InitData( destAddr : TAmsAddr);
begin
     serverAddr := destAddr;
     ComboBox1.itemindex := 0;
end;
////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadWriteReq.Button1Click( Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
    hVar            : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT            : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;
    sName           : AnsiString;
begin
     Memo1.Lines.clear;
     dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

     sName := AnsiString(editName.Text);
     result := AdsSyncReadWriteReq(@serverAddr, ADSIGRP_SYM_HNDBYNAME, $0000, sizeof(hVar), @hVar, L
```

```
ength(sName)+1, @sName[1] );
     Memo1.Lines.add( Format( 'AdsSyncReadWriteReq(ADSIGRP_SYM_HNDBYNAME..) result: %d [0x%x], Handl
e: %d [0x%x]', [result, result, hVar, hVar]));

     if result = ADSERR_NOERR then
     begin

       case dataType of
       plcBOOL:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE );
          editValue.Text := IntToStr(nByte And $01);
         end;
       plcBYTE:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE );
          editValue.Text := IntToStr(nByte);
         end;
       plcWORD:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nWORD), @nWORD );
          editValue.Text := IntToStr(nWORD);
         end;
       plcDWORD:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nDWORD), @nDWORD
);
          editValue.Text := IntToStr(nDWORD);
         end;
       plcSINT:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iSINT), @iSINT  )
;
          editValue.Text := IntToStr(iSINT);
         end;
       plcINT:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iINT), @iINT  );
          editValue.Text := IntToStr(iINT);
         end;
       plcDINT:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iDINT), @iDINT );
          editValue.Text := IntToStr(iDINT);
         end;
       plcREAL:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fREAL), @fREAL );
          editValue.Text := FloatToStr(fREAL);
         end;
       plcLREAL:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fLREAL), @fLREAL
);
          editValue.Text := FloatToStr(fLREAL);
         end;
       plcSTRING:
         begin
          SetLength(sSTRING, 256);//make sure the delphi string size is >= SIZEOF(plc string)
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, Length(sSTRING), @sSTRIN
G[1] );
          editValue.Text := String(sSTRING);
         end;
       end;

       Memo1.Lines.add( Format( 'AdsSyncReadReq(ADSIGRP_SYM_VALBYHND..) result: %d [0x%x]', [result,
 result]));

       result := AdsSyncWriteReq( @serverAddr, ADSIGRP_RELEASE_SYMHND, $0000, sizeof( hVar), @hVar )
;

       Memo1.Lines.add( Format( 'AdsSyncWriteReq(ADSIGRP_RELEASE_SYMHND..) result: %d [0x%x]', [resu
lt, result]));

     end;
end;
////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadWriteReq.Button2Click(Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
```

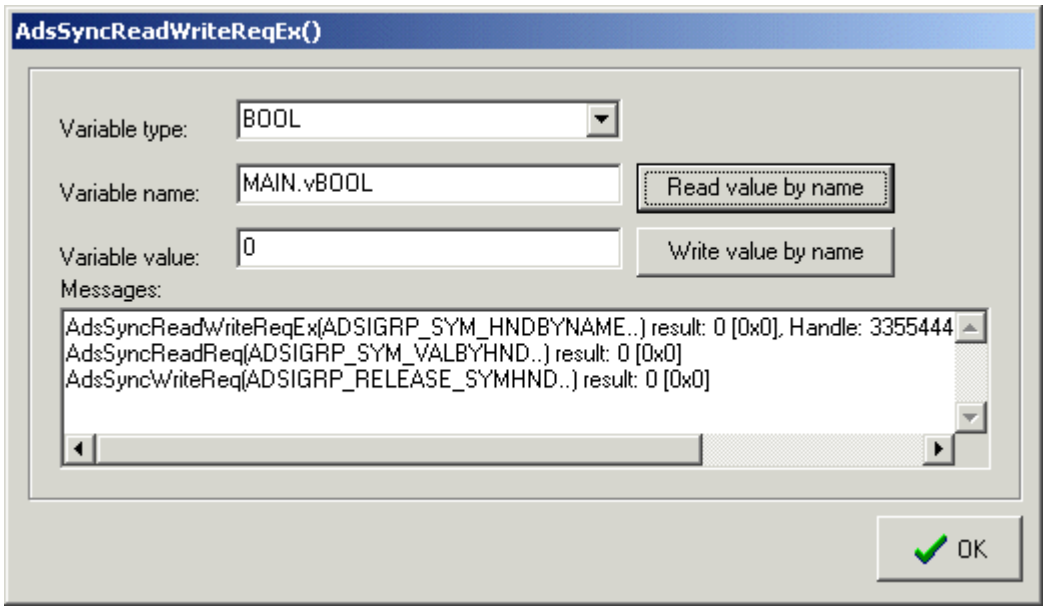
```
    hVar        : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT        : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;
    sName           : AnsiString;
begin
     Memo1.Lines.clear;
     dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

     sName := AnsiString(editName.Text);
     result := AdsSyncReadWriteReq(@serverAddr, ADSIGRP_SYM_HNDBYNAME, $0000, sizeof(hVar), @hVar, L
ength(sName)+1, @sName[1] );
     Memo1.Lines.add( Format( 'AdsSyncReadWriteReq(ADSIGRP_SYM_HNDBYNAME..) result: %d [0x%x], Handl
e: %d [0x%x]', [result, result, hVar, hVar]));

     if result = ADSERR_NOERR then
     begin

       case dataType of
       plcBOOL:
         begin
          nByte := Byte(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE )
;
         end;
       plcBYTE:
         begin
          nByte := Byte(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE )
;
         end;
       plcWORD:
         begin
          nWORD := Word(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nWORD), @nWORD )
;
         end;
       plcDWORD:
         begin
          nDWORD := Longword(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nDWORD), @nDWORD
 );
         end;
       plcSINT:
         begin
          iSINT := Shortint(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iSINT), @iSINT
);
         end;
       plcINT:
         begin
          iINT := Smallint(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iINT), @iINT  );
         end;
       plcDINT:
         begin
          iDINT := Longint(StrToInt(editValue.Text));
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iDINT), @iDINT )
;
         end;
       plcREAL:
         begin
          fREAL := StrToFloat(editValue.Text);
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fREAL), @fREAL )
;
         end;
       plcLREAL:
         begin
          fLREAL := StrToFloat(editValue.Text);
          result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fLREAL), @fLREAL
 );
         end;
       plcSTRING:
```

```
        begin
         sSTRING := AnsiString(editValue.Text);
         if sSTRING = '' then
           sSTRING := ''#00;
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, Length(sSTRING)
+1, @sSTRING[1] );
        end;
      end;

      Memo1.Lines.add( Format( 'AdsSyncWriteReq(ADSIGRP_SYM_VALBYHND..) result: %d [0x%x]', [result
, result]));

      result := AdsSyncWriteReq( @serverAddr, ADSIGRP_RELEASE_SYMHND, $0000, sizeof( hVar), @hVar )
;

      Memo1.Lines.add( Format( 'AdsSyncWriteReq(ADSIGRP_RELEASE_SYMHND..) result: %d [0x%x]', [resu
lt, result]));

    end;
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.8 AdsSyncReadWriteReqEx



Delphi 5 Programm

```
unit frmAdsSyncReadWriteReqExUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls ;

type
  TfrmAdsSyncReadWriteReqEx = class(TForm)
    editValue: TEdit;
    Label1: TLabel;
    ComboBox1: TComboBox;
    Label4: TLabel;
    BitBtn1: TBitBtn;
    Button1: TButton;
    Bevel1: TBevel;
    editName: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
```

```
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private-Deklarationen }
    serverAddr      : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );
  end;

type TPlcDataTypes =
(
    plcBOOL,
    plcBYTE,  //and USINT
    plcWORD,  //and UINT
    plcDWORD, //and UDINT
    plcSINT,
    plcINT,
    plcDINT,
    plcREAL,
    plcLREAL,
    plcSTRING
);


implementation

{$R *.DFM}
//////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadWriteReqEx.InitData( destAddr : TAmsAddr);
begin
     serverAddr := destAddr;
     ComboBox1.itemindex := 0;
end;
//////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadWriteReqEx.Button1Click( Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
    hVar        : Longword;
    cbReturn        : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT        : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;
    sName           : AnsiString;
begin
     Memo1.Lines.clear;
     dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

     sName := AnsiString(editName.Text);
     result := AdsSyncReadWriteReqEx(@serverAddr, ADSIGRP_SYM_HNDBYNAME, $0000, sizeof(hVar), @hVar,
 Length(sName)+1, @sName[1], @cbReturn );
     Memo1.Lines.add( Format( 'AdsSyncReadWriteReqEx(ADSIGRP_SYM_HNDBYNAME..) result: %d [0x%x], Han
dle: %d [0x%x], Returned bytes: %d', [result, result, hVar, hVar, cbReturn]));

     if result = ADSERR_NOERR then
     begin

       case dataType of
       plcBOOL:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE );
          editValue.Text := IntToStr(nByte And $01);
         end;
       plcBYTE:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE );
          editValue.Text := IntToStr(nByte);
         end;
       plcWORD:
         begin
          result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nWORD), @nWORD );
          editValue.Text := IntToStr(nWORD);
```

```
        end;
      plcDWORD:
        begin
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nDWORD), @nDWORD
);
         editValue.Text := IntToStr(nDWORD);
        end;
      plcSINT:
        begin
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iSINT), @iSINT  )
;
         editValue.Text := IntToStr(iSINT);
        end;
      plcINT:
        begin
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iINT), @iINT  );
         editValue.Text := IntToStr(iINT);
        end;
      plcDINT:
        begin
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iDINT), @iDINT );
         editValue.Text := IntToStr(iDINT);
        end;
      plcREAL:
        begin
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fREAL), @fREAL );
         editValue.Text := FloatToStr(fREAL);
        end;
      plcLREAL:
        begin
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fLREAL), @fLREAL
);
         editValue.Text := FloatToStr(fLREAL);
        end;
      plcSTRING:
        begin
         SetLength(sSTRING,256);//make sure the delphi string size is >= SIZEOF(plc string)
         result := AdsSyncReadReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, Length(sSTRING), @sSTRIN
G[1] );
         editValue.Text := String(sSTRING);
        end;
      end;

      Memo1.Lines.add( Format( 'AdsSyncReadReq(ADSIGRP_SYM_VALBYHND..) result: %d [0x%x]', [result,
 result]));

      result := AdsSyncWriteReq( @serverAddr, ADSIGRP_RELEASE_SYMHND, $0000, sizeof( hVar), @hVar )
;

      Memo1.Lines.add( Format( 'AdsSyncWriteReq(ADSIGRP_RELEASE_SYMHND..) result: %d [0x%x]', [resu
lt, result]));

    end;
end;
//////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadWriteReqEx.Button2Click(Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
    hVar        : Longword;
    cbReturn        : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT        : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;
    sName           : AnsiString;
begin
    Memo1.Lines.clear;
    dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

    sName := AnsiString(editName.Text);
    result := AdsSyncReadWriteReqEx(@serverAddr, ADSIGRP_SYM_HNDBYNAME, $0000, sizeof(hVar), @hVar,
 Length(sName)+1, @sName[1], @cbReturn );
    Memo1.Lines.add( Format( 'AdsSyncReadWriteReqEx(ADSIGRP_SYM_HNDBYNAME..) result: %d [0x%x], Han
dle: %d [0x%x], Returned bytes: %d', [result, result, hVar, hVar, cbReturn]));
```

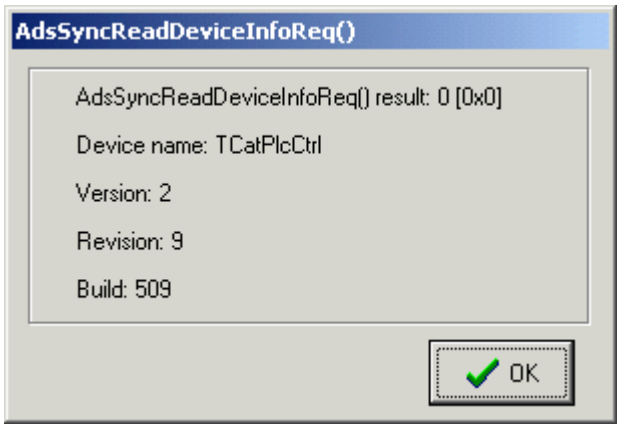
```
    if result = ADSERR_NOERR then
    begin

      case dataType of
      plcBOOL:
        begin
         nByte := Byte(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE )
;
        end;
      plcBYTE:
        begin
         nByte := Byte(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nBYTE), @nBYTE )
;
        end;
      plcWORD:
        begin
         nWORD := Word(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nWORD), @nWORD )
;
        end;
      plcDWORD:
        begin
         nDWORD := Longword(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(nDWORD), @nDWORD
 );
        end;
      plcSINT:
        begin
         iSINT := Shortint(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iSINT), @iSINT
);
        end;
      plcINT:
        begin
         iINT := Smallint(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iINT), @iINT  );
        end;
      plcDINT:
        begin
         iDINT := Longint(StrToInt(editValue.Text));
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(iDINT), @iDINT )
;
        end;
      plcREAL:
        begin
         fREAL := StrToFloat(editValue.Text);
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fREAL), @fREAL )
;
        end;
      plcLREAL:
        begin
         fLREAL := StrToFloat(editValue.Text);
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(fLREAL), @fLREAL
 );
        end;
      plcSTRING:
        begin
         sSTRING := AnsiString(editValue.Text);
         if sSTRING = '' then
           sSTRING := ''#00;
         result := AdsSyncWriteReq(@serverAddr, ADSIGRP_SYM_VALBYHND, hVar, Length(sSTRING)
+1, @sSTRING[1] );
        end;
      end;

      Memo1.Lines.add( Format( 'AdsSyncWriteReq(ADSIGRP_SYM_VALBYHND..) result: %d [0x%x]', [result
, result]));

      result := AdsSyncWriteReq( @serverAddr, ADSIGRP_RELEASE_SYMHND, $0000, sizeof( hVar), @hVar )
;

      Memo1.Lines.add( Format( 'AdsSyncWriteReq(ADSIGRP_RELEASE_SYMHND..) result: %d [0x%x]', [resu
lt, result]));

    end;
```

```
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.9 AdsSyncReadDeviceInfoReq



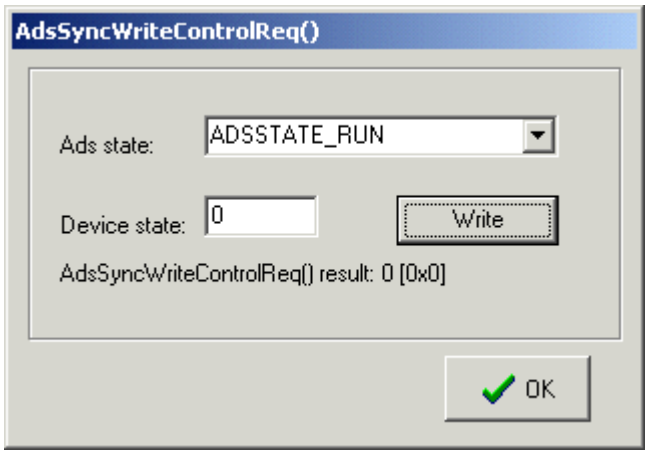
Delphi 5 Programm

```
unit frmAdsSyncReadDeviceInfoReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls;

type
  TfrmAdsSyncReadDeviceInfoReq = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Bevel1: TBevel;
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );

  end;

implementation

{$R *.DFM}
procedure TfrmAdsSyncReadDeviceInfoReq.InitData( destAddr : TAmsAddr);
var result : Longint;
    sDevName : AnsiString;
    adsDevVersion : TAdsVersion;
begin
     SetLength(sDevName, ADS_FIXEDNAMESIZE + 1);
     result := AdsSyncReadDeviceInfoReq(@destAddr, @sDevName[1], @adsDevVersion );
     Label1.Caption := Format( 'AdsSyncReadDeviceInfoReq() result: %d [0x%x]', [result,result] );
     Label2.Caption := Format( 'Device name: %s', [String(sDevName)] );
     Label3.Caption := Format( 'Version: %d', [adsDevVersion.Version] );
     Label4.Caption := Format( 'Revision: %d', [adsDevVersion.Revision] );
     Label5.Caption := Format( 'Build: %d', [adsDevVersion.Build] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.10 AdsSyncWriteControlReq



Delphi 5 Programm

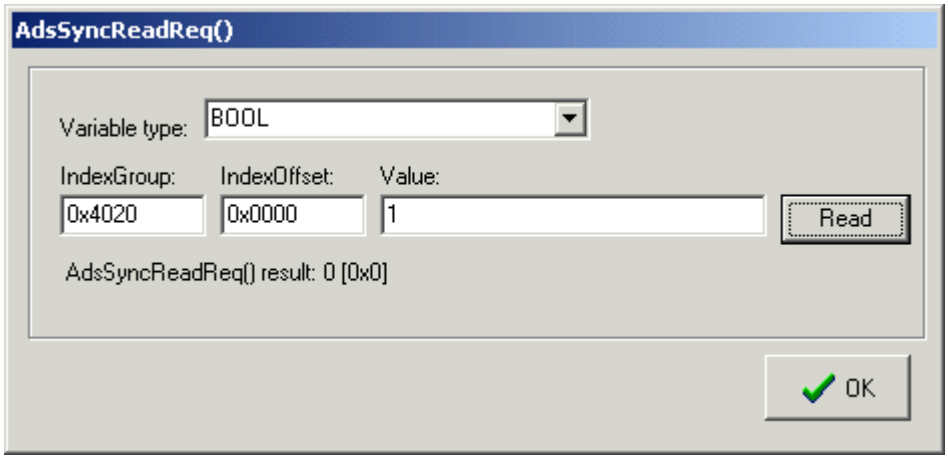
```
unit frmAdsSyncWriteControlReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls;

type
  TfrmAdsSyncWriteControlReq = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    ComboAdsState: TComboBox;
    editDevState: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Label3: TLabel;
    Bevel1: TBevel;
    procedure Button1Click(Sender: TObject);
  private
    { Private-Deklarationen }
    serverAddr : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );

  end;

implementation

{$R *.DFM}
//////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncWriteControlReq.InitData( destAddr : TAmsAddr);
begin
     serverAddr := destAddr;
     ComboAdsstate.Items.Add('ADSSTATE_INVALID');
     ComboAdsstate.Items.Add('ADSSTATE_IDLE');
     ComboAdsstate.Items.Add('ADSSTATE_RESET');
     ComboAdsstate.Items.Add('ADSSTATE_INIT');
     ComboAdsstate.Items.Add('ADSSTATE_START');
     ComboAdsstate.Items.Add('ADSSTATE_RUN');
     ComboAdsstate.Items.Add('ADSSTATE_STOP');
     ComboAdsstate.Items.Add('ADSSTATE_SAVECFG');
     ComboAdsstate.Items.Add('ADSSTATE_LOADCFG');
     ComboAdsstate.Items.Add('ADSSTATE_POWERFAILURE');
     ComboAdsstate.Items.Add('ADSSTATE_POWERGOOD');
     ComboAdsstate.Items.Add('ADSSTATE_ERROR');
     ComboAdsstate.Items.Add('ADSSTATE_SHUTDOWN');
     ComboAdsstate.Items.Add('ADSSTATE_SUSPEND');
     ComboAdsstate.Items.Add('ADSSTATE_RESUME');
     ComboAdsstate.Items.Add('ADSSTATE_CONFIG');// system is in config mode
     ComboAdsstate.Items.Add('ADSSTATE_RECONFIG');// system should restart in config mode
```

```
     ComboAdsState.ItemIndex := 5;
end;
//////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncWriteControlReq.Button1Click(Sender: TObject);
var result : Longint;
begin
     result := AdsSyncWriteControlReq( @serverAddr, Word( ComboAdsState.itemIndex ), Word(StrToInt(e
ditDevState.Text)), 0, Nil);
     Label3.Caption := Format('AdsSyncWriteControlReq() result: %d [0x%x]',[result,result] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.11 AdsSyncReadReq



Delphi 5 Programm

```
unit frmAdsSyncReadReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls ;

type
  TfrmAdsSyncReadReq = class(TForm)
    editIO: TEdit;
    editIG: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    editValue: TEdit;
    Label1: TLabel;
    Label5: TLabel;
    ComboBox1: TComboBox;
    Label4: TLabel;
    BitBtn1: TBitBtn;
    Button1: TButton;
    Bevel1: TBevel;
    procedure Button1Click(Sender: TObject);
  private
    { Private-Deklarationen }
    serverAddr      : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr);
  end;

type TPlcDataTypes =
(
    plcBOOL,
    plcBYTE,  //and USINT
    plcWORD,  //and UINT
    plcDWORD, //and UDINT
```

```
    plcSINT,
    plcINT,
    plcDINT,
    plcREAL,
    plcLREAL,
    plcSTRING
);


implementation

{$R *.DFM}
////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadReq.InitData( destAddr : TAmsAddr);
begin
     serverAddr := destAddr;
     ComboBox1.itemindex := 0;
end;
////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadReq.Button1Click( Sender: TObject);
var result          : longword;
    dataType        : TPlcDataTypes;
    indexGroup      : Longword;
    indexOffset     : Longword;

    nBYTE           : Byte; //and USINT
    nWORD           : Word; //and UINT
    nDWORD          : Longword; //and UDINT
    iSINT           : Shortint;
    iINT            : Smallint;
    iDINT           : Longint;
    fREAL           : Single;
    fLREAL          : Double;
    sSTRING         : AnsiString;
begin
     result := 0;
     dataType :=  TPlcDataTypes(ComboBox1.ItemIndex);

     indexGroup := Longword(StrToInt(editIG.Text));
     indexOffset := Longword(StrToInt(editIO.Text));

     case dataType of
       plcBOOL:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(nBYTE), @nBYTE  );
          editValue.Text := IntToStr(nByte And $01);
         end;
       plcBYTE:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(nBYTE), @nBYTE  );
          editValue.Text := IntToStr(nByte);
         end;
       plcWORD:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(nWORD), @nWORD  );
          editValue.Text := IntToStr(nWORD);
         end;
       plcDWORD:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(nDWORD), @nDWORD  );
          editValue.Text := IntToStr(nDWORD);
         end;
       plcSINT:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(iSINT), @iSINT  );
          editValue.Text := IntToStr(iSINT);
         end;
       plcINT:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(iINT), @iINT  );
          editValue.Text := IntToStr(iINT);
         end;
       plcDINT:
         begin
          result := AdsSyncWriteReq(@serverAddr, indexGroup, indexOffset, sizeof(iDINT), @iDINT  );
          editValue.Text := IntToStr(iDINT);
         end;
       plcREAL:
         begin
          result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(fREAL), @fREAL  );
```

```
          editValue.Text := FloatToStr(fREAL);
        end;
      plcLREAL:
        begin
         result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, sizeof(fLREAL), @fLREAL  );
         editValue.Text := FloatToStr(fLREAL);
        end;
      plcSTRING:
        begin
         SetLength(sSTRING, 256);//make sure the delphi string size is >= SIZEOF(plc string)
         result := AdsSyncReadReq(@serverAddr, indexGroup, indexOffset, Length(sSTRING), @sSTRING[1
]  );
         editValue.Text := String(sSTRING);
        end;
    end;

    Label5.Caption := Format( 'AdsSyncReadReq() result: %d [0x%x]', [result, result] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.12 AdsSyncReadStateReq

Delphi 5 Programm

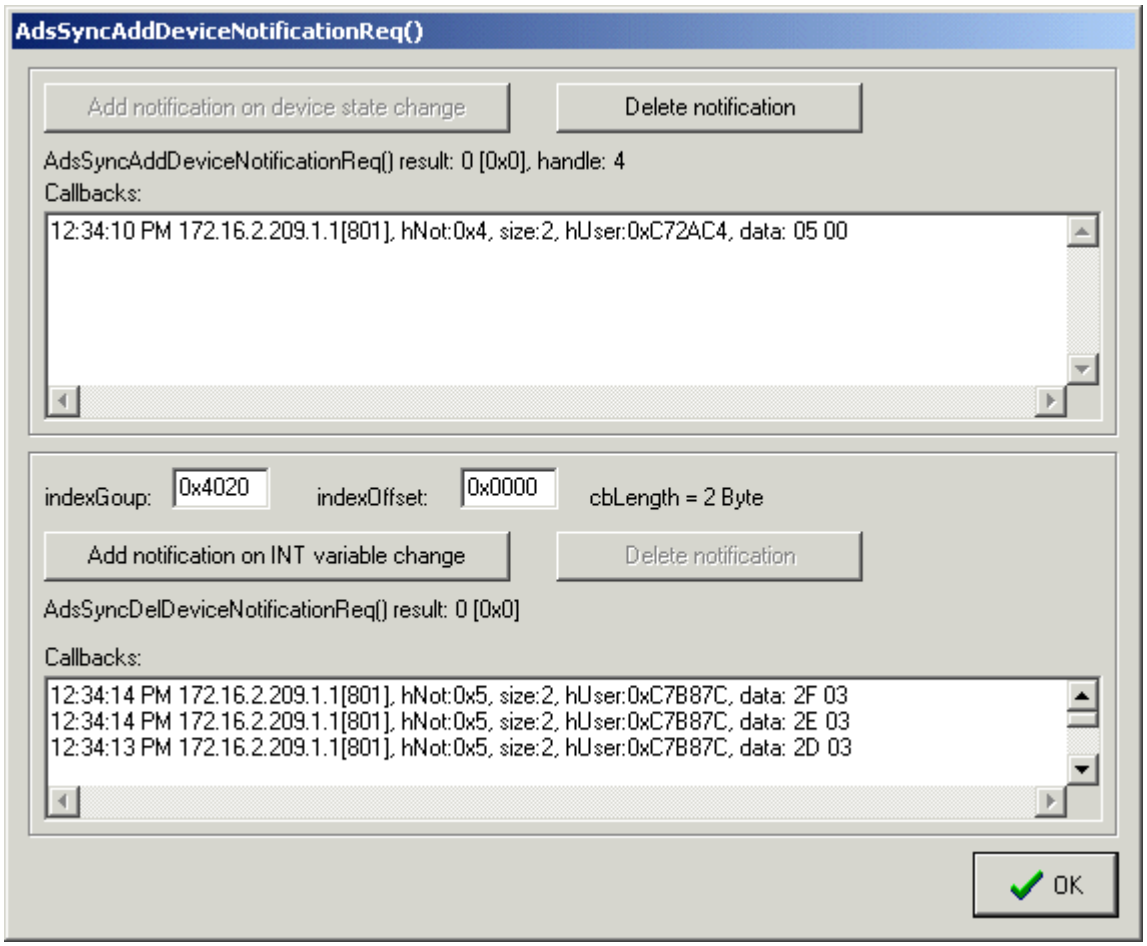
```
unit frmAdsSyncReadStateReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls;

type
  TfrmAdsSyncReadStateReq = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    ComboAdsState: TComboBox;
    editDevState: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Label3: TLabel;
    Bevel1: TBevel;
    procedure Button1Click(Sender: TObject);
  private
    { Private-Deklarationen }
    serverAddr : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );

  end;
```

```
implementation

{$R *.DFM}
////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadStateReq.InitData( destAddr : TAmsAddr);
begin
     serverAddr := destAddr;
     ComboAdsstate.Items.Add('ADSSTATE_INVALID');
     ComboAdsstate.Items.Add('ADSSTATE_IDLE');
     ComboAdsstate.Items.Add('ADSSTATE_RESET');
     ComboAdsstate.Items.Add('ADSSTATE_INIT');
     ComboAdsstate.Items.Add('ADSSTATE_START');
     ComboAdsstate.Items.Add('ADSSTATE_RUN');
     ComboAdsstate.Items.Add('ADSSTATE_STOP');
     ComboAdsstate.Items.Add('ADSSTATE_SAVECFG');
     ComboAdsstate.Items.Add('ADSSTATE_LOADCFG');
     ComboAdsstate.Items.Add('ADSSTATE_POWERFAILURE');
     ComboAdsstate.Items.Add('ADSSTATE_POWERGOOD');
     ComboAdsstate.Items.Add('ADSSTATE_ERROR');
     ComboAdsstate.Items.Add('ADSSTATE_SHUTDOWN');
     ComboAdsstate.Items.Add('ADSSTATE_SUSPEND');
     ComboAdsstate.Items.Add('ADSSTATE_RESUME');
     ComboAdsstate.Items.Add('ADSSTATE_CONFIG');// system is in config mode
     ComboAdsstate.Items.Add('ADSSTATE_RECONFIG');// system should restart in config mode

     Button1Click( self );
end;
////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncReadStateReq.Button1Click(Sender: TObject);
var result : Longint;
    adsState, devState : Word;
begin
     adsState := 0;
     devState := 0;

     result := AdsSyncReadStateReq( @serverAddr, @adsState, @devState);
     ComboAdsState.ItemIndex := adsState;
     editDevState.Text :=  IntToStr(devState);
     Label3.Caption := Format('AdsSyncReadStateReq() result: %d [0x%x]',[result,result] );
end;

end.
```

Dokumente hierzu

- delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.13 AdsSyncAddDeviceNotificationReq/ AdsSyncDelDeviceNotificationReq



Delphi 5 Programm

```
unit frmAdsSyncAddDeviceNotificationReqUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls, Math;

//define your own message ids
const WM_ADSDEVICENOTIFICATION = WM_APP + 400;

type
  TfrmAdsSyncAddDeviceNotificationReq = class(TForm)
    BitBtn1: TBitBtn;
    Bevel1: TBevel;
    Button1: TButton;
    Label1: TLabel;
    Memo1: TMemo;
    Button2: TButton;
    Label2: TLabel;
    Bevel2: TBevel;
    Button3: TButton;
    Button4: TButton;
    Memo2: TMemo;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Label5: TLabel;
    Edit2: TEdit;
    Label6: TLabel;
    Label7: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
```

```
    procedure Button4Click(Sender: TObject);
  protected
    procedure WMAdsDeviceNotification( var Message: TMessage ); message WM_ADSDEVICENOTIFICATION;
  private
    { Private-Deklarationen }
    serverAddr : TAmsAddr;
    hDevNotification : Longword;
    hSymValNotification : Longword;
  public
    { Public-Deklarationen }
    procedure InitData( addr : TAmsAddr);
    destructor Destroy; override;
  end;

{$ALIGN OFF}
type TThreadListItem = record
     netId       : TAmsNetId;
     port        : Word;
       hNotify     : Longword;
       stamp       : TDateTime;
       cbSize      : Longword;
     destObj     : ^TStrings;
     data        : Byte;    //Array[1..ANYSIZE_ARRAY] of Byte;
end;
PThreadListItem = ^TThreadListItem;
{$ALIGN ON}

var
    DevThreadList : TThreadList;
    wndHandle :HWND;

implementation
{$R *.DFM}

////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncAddDeviceNotificationReq.InitData( addr : TAmsAddr);
begin
     DevThreadList := TThreadList.Create();
     serverAddr := addr;
     wndHandle := Handle;
end;
////////////////////////////////////////////////////////////////////////////
destructor TfrmAdsSyncAddDeviceNotificationReq.Destroy();
var X : integer;
begin
     //delete notifications on exit
     if hDevNotification <> 0 then
      Button2Click( self );
     if hSymValNotification <> 0 then
      Button4Click( self );

     With DevThreadList.LockList do
     try
    for X := 0 to Count-1 do
        FreeMem( Items[X], sizeof(TThreadListItem) + PThreadListItem(Items[X])^.cbSize - 1 );
     finally
    DevThreadList.UnlockList;
    DevThreadList.Destroy;
     end;

     inherited Destroy;
end;
////////////////////////////////////////////////////////////////////////////
Procedure Callback(     pAddr:PAmsAddr; pNotification:PAdsNotificationHeader;   hUser:Longword ); st
dcall;
var pItem : PThreadListItem;
    FileTime        : _FILETIME;
    SystemTime, LocalTime    : _SYSTEMTIME;
    TimeZoneInformation     : _TIME_ZONE_INFORMATION;
begin
    GetMem( pItem, sizeof(TThreadListItem) + pNotification^.cbSampleSize - 1 );
    pItem^.netId := pAddr^.netId;
    pItem^.port := pAddr^.port;
    pItem^.hNotify := pNotification^.hNotification;
    pItem^.cbSize := pNotification^.cbSampleSize;

    FileTime:=pNotification^.nTimeStamp;
    FileTimeToSystemTime(FileTime, SystemTime);
    GetTimeZoneInformation(TimeZoneInformation);
    SystemTimeToTzSpecificLocalTime(@TimeZoneInformation, SystemTime, LocalTime);
```

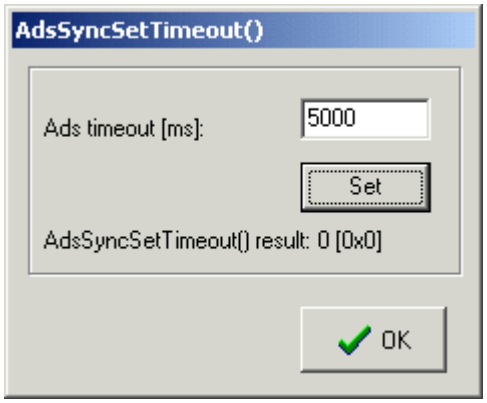
```
    pItem^.stamp := SystemTimeToDateTime(LocalTime);

    pItem^.destObj := Ptr(hUser);
    //copy the notification data
    Move( pNotification^.data, pItem^.data, pNotification^.cbSampleSize);
    with DevThreadList.LockList do
    try
       Add( pItem );
    finally
       DevThreadList.UnlockList;
       PostMessage( wndHandle, WM_ADSDEVICENOTIFICATION, 0, 0);
    end;
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncAddDeviceNotificationReq.WMAdsDeviceNotification( var Message: TMessage );
var pItem           : PThreadListItem;
    X,i             : integer;
    dataAsStr           : String;
begin
    with DevThreadList.LockList do
    try
       for X := 0 to Count-1 do
       begin
      pItem := Items[X];
      { convert data to hex string }
      dataAsStr := '';
      for i:= 0 to math.Min(pItem^.cbSize-1, 20) do
          dataAsStr := dataAsStr + IntToHex( PByte(PAnsiChar(@pItem^.data)+i)^, 2 ) + ' ' ;

      pItem^.destObj^.Insert(0,Format( '%s %d.%d.%d.%d.%d.%d[%d], hNot:0x%x, size:
%d, hUser:0x%x, data: %s',
                    [TimeToStr(pItem^.stamp), pItem^.netId.b[0], pItem^.netId.b[1], pItem^.netId.b[
2],
                    pItem^.netId.b[3], pItem^.netId.b[4], pItem^.netId.b[5], pItem^.port,
                    pItem^.hNotify, pItem^.cbSize, Longword(pItem^.destObj),
                    dataAsStr ]));
      FreeMem( pItem, sizeof(TThreadListItem) + pItem^.cbSize - 1 ); //free item memory
       end;
       Clear();
    finally
       DevThreadList.UnlockList;
    end;
    inherited;
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncAddDeviceNotificationReq.Button1Click(Sender: TObject);
var result : Longint;
    adsNotificationAttrib : TAdsNotificationAttrib;
begin
     adsNotificationAttrib.cbLength := 2;
     adsNotificationAttrib.nTransMode := ADSTRANS_SERVERONCHA;
     adsNotificationAttrib.nMaxDelay := 0;
     adsNotificationAttrib.nCycleTime := 0;

     result := AdsSyncAddDeviceNotificationReq( @serverAddr, ADSIGRP_DEVICE_DATA,
                         ADSIOFFS_DEVDATA_ADSSTATE,
                         @adsNotificationAttrib,
                         @Callback,
                         Longword(@Memo1.lines), @hDevNotification  );

     Label1.Caption := Format( 'AdsSyncAddDeviceNotificationReq() result: %d [0x%x], handle: 0x%x',
[result, result, hDevNotification] );
     Memo1.Lines.Clear;
     Button1.Enabled := not(result = ADSERR_NOERR);
     Button2.Enabled := (result = ADSERR_NOERR);
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncAddDeviceNotificationReq.Button2Click(  Sender: TObject);
var result : longint;
begin
     result := AdsSyncDelDeviceNotificationReq( @serverAddr, hDevNotification );
     hDevNotification := 0;
     Label1.Caption := Format( 'AdsSyncDelDeviceNotificationReq() result: %d [0x%x]', [result,result
] );
     Button1.Enabled := true;
     Button2.Enabled := false;
end;

////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncAddDeviceNotificationReq.Button3Click(  Sender: TObject);
```

```
var result : Longword;
    adsNotificationAttrib : TAdsNotificationAttrib;
begin
     adsNotificationAttrib.cbLength := 2; //INT
     adsNotificationAttrib.nTransMode := ADSTRANS_SERVERONCHA;
     adsNotificationAttrib.nMaxDelay := 10000000;//1 second
     adsNotificationAttrib.nCycleTime := 0;

     result := AdsSyncAddDeviceNotificationReq( @serverAddr, Longword(StrToInt(Edit1.Text)),
                         Longword(StrToInt(Edit2.Text)),
                         @adsNotificationAttrib,
                         @Callback,
                         Longword(@Memo2.lines), @hSymValNotification  );

     Label3.Caption := Format( 'AdsSyncAddDeviceNotificationReq() result: %d [0x%x], handle: 0x%x',
[result, result, hSymValNotification] );
     Memo2.Lines.Clear;
     Button3.Enabled := not(result = ADSERR_NOERR);
     Button4.Enabled := (result = ADSERR_NOERR);
end;
//////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncAddDeviceNotificationReq.Button4Click(  Sender: TObject);
var result : Longword;
begin
     result := AdsSyncDelDeviceNotificationReq( @serverAddr, hSymValNotification );
     hSymValNotification := 0;
     Label3.Caption := Format( 'AdsSyncDelDeviceNotificationReq() result: %d [0x%x]', [result,result
] );
     Button3.Enabled := true;
     Button4.Enabled := false;
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.14 AdsSyncSetTimeout



Delphi 5 Programm

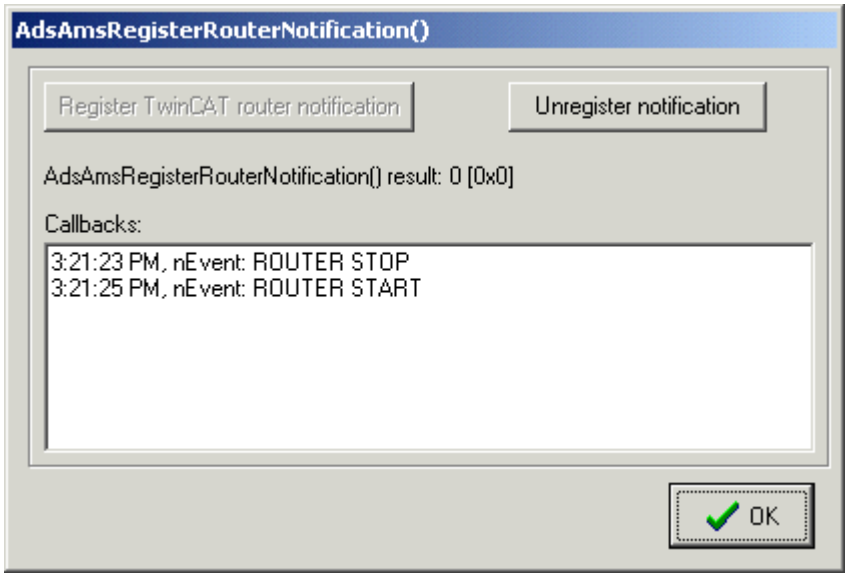
```
unit frmAdsSyncSetTimeoutUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls;

type
  TfrmAdsSyncSetTimeout = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    Bevel1: TBevel;
    editTimeout: TEdit;
    Label2: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private-Deklarationen }
```

```
    serverAddr : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr );
  end;


implementation

{$R *.DFM}
////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncSetTimeout.InitData( destAddr : TAmsAddr );
begin
     serverAddr := destAddr;
end;
////////////////////////////////////////////////////////////////
procedure TfrmAdsSyncSetTimeout.Button1Click(Sender: TObject);
var result : longint;
begin
     result := AdsSyncSetTimeout(StrToInt(editTimeout.Text));
     Label1.Caption := Format( 'AdsSyncSetTimeout() result: %d [0x%x]', [result,result] );
end;

end.
```

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.2.15 AdsAms[Un]RegisterRouterNotification



Delphi 5 Programm

```
unit frmAdsAmsRegisterRouterNotificationUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, TcAdsDef, TcAdsApi, Buttons, ExtCtrls;

//define your own message id
const WM_ADSROUTERNOTIFICATION = WM_APP + 405;


type
  TfrmAdsAmsRegisterRouterNotification = class(TForm)
    BitBtn1: TBitBtn;
    Bevel1: TBevel;
    Button1: TButton;
    Label1: TLabel;
    Memo1: TMemo;
    Button2: TButton;
    Label2: TLabel;
```

```
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  protected
    procedure WMAdsRouterNotification( var Message: TMessage ); message WM_ADSROUTERNOTIFICATION;
  private
    { Private-Deklarationen }
    serverAddr : TAmsAddr;
  public
    { Public-Deklarationen }
    procedure InitData( destAddr : TAmsAddr);
  end;

var wndHandle : HWND;

implementation
{$R *.DFM}
////////////////////////////////////////////////////////////////////////////
Procedure RouterCallback( nEvent:Longint ); stdcall;
begin
     PostMessage(wndHandle, WM_ADSROUTERNOTIFICATION, nEvent, 0);
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsAmsRegisterRouterNotification.WMAdsRouterNotification( var Message: TMessage );
var sState : String;
begin
     case Message.WParam of
      AMSEVENT_ROUTERSTOP: sState := 'ROUTER STOP';
      AMSEVENT_ROUTERSTART: sState:= 'ROUTER START';
      AMSEVENT_ROUTERREMOVED: sState:= 'ROUTER REMOVED';
     else
     sState := Format( 'Other: %d', [Message.WParam]);
     end;

     Memo1.lines.add(Format('%s, nEvent: %s',[TimeToStr(time), sState]));
     inherited;
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsAmsRegisterRouterNotification.InitData( destAddr : TAmsAddr );
begin
     serverAddr := destAddr;
     wndHandle := Handle;//Save the window handle
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsAmsRegisterRouterNotification.Button1Click(Sender: TObject);
var result : longint;
begin
     result := AdsAmsRegisterRouterNotification( @RouterCallback );
     Label1.Caption := Format( 'AdsAmsRegisterRouterNotification() result: %d [0x%x]', [result,resul
t] );
     Button1.Enabled := not(result = ADSERR_NOERR);
     Button2.Enabled := (result = ADSERR_NOERR);
end;
////////////////////////////////////////////////////////////////////////////
procedure TfrmAdsAmsRegisterRouterNotification.Button2Click(  Sender: TObject);
var result : longint;
begin
     Memo1.Lines.Clear();
     result := AdsAmsUnRegisterRouterNotification();
     Label1.Caption := Format( 'AdsAmsUnRegisterRouterNotification() result: %d [0x%x]', [result,res
ult] );
     Button1.Enabled := true;
     Button2.Enabled := false;
end;

end.
```

Dokumente hierzu

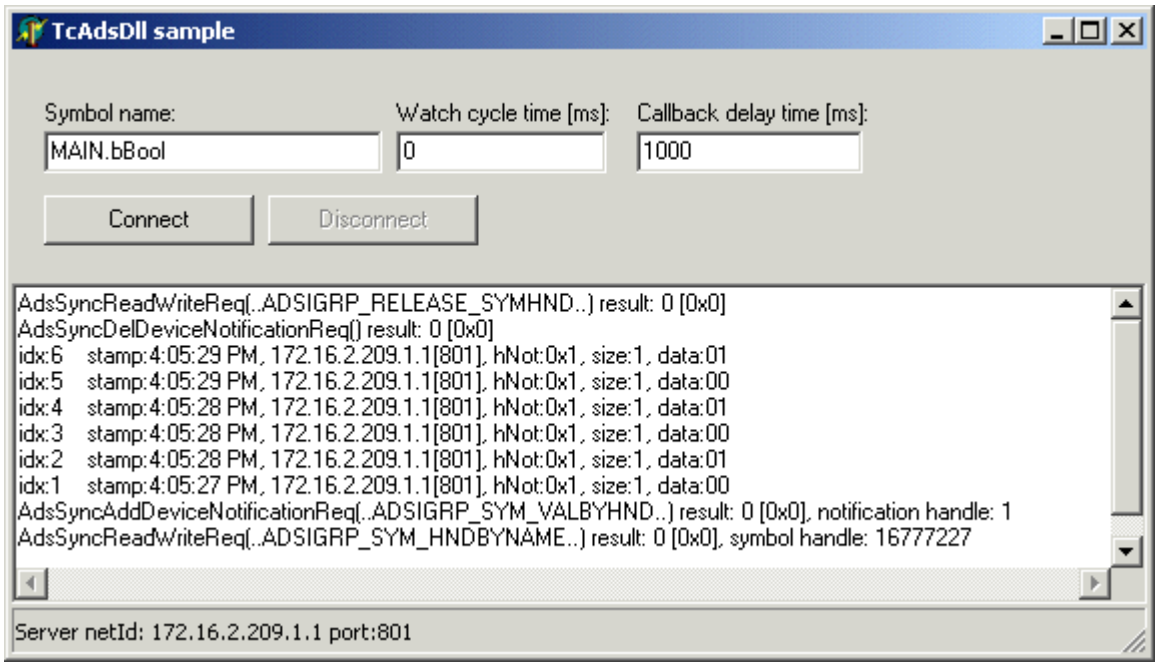
delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.3 Ereignisgesteuertes Lesen (by symbol name)

Voraussetzungen:

- Delphi 5.0 oder höher
- TcAdsDLL.DLL

- TcAdsDEF.pas und TcAdsAPI.pas, falls Sie den Quelltext selbst übersetzen möchten



Deklarationsdateien

Die Deklarationsdateien TcAdsAPI.pas und TcAdsDEF.pas sind im Archiv delphi_adsdll_api_units.zip enthalten.

Delphi 5 Programm

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, TcAdsDef, TcAdsAPI,
  StdCtrls, Math, ComCtrls;
const WM_ADSDEVICENOTIFICATION = WM_APP + 400;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Button2: TButton;
    Memo1: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    editDelayTime: TEdit;
    Label3: TLabel;
    editCycleTime: TEdit;
    StatusBar1: TStatusBar;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure WMAdsDeviceNotification( var Message: TMessage ); message WM_ADSDEVICENOTIFICATION;
    procedure FormDestroy(Sender: TObject);
  private
    server : TAmsAddr;
    hVar : Longword;
    hNotification : Longword;
    nCnt      : Longword;
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;
{$ALIGN OFF}
type
    TNotifyData = record
        netid        : TAmsNetId;
        port         : Word;
        hNotification    : Longword;
        dateTime         : TDateTime;
        cbSampleSize     : Longword;
        hUser        : Longword;
        data         : Byte;
```

```
    end;
PNotifyData = ^TNotifyData;
{$ALIGN ON}
var
  Form1: TForm1;
  DevThreadList : TThreadList;
implementation
{$R *.DFM}
```

Implementierung der Notification Callback-Funktion

```
//////////////////////////////////////////////////////////////////////////////////////////
Procedure NotificationCallback( pAddr:PAmsAddr; pNotification:PAdsNotificationHeader;   hUser:Longwo
rd ); stdcall;
var pItem           : PNotifyData;
    FileTime        : _FILETIME;
    SystemTime, LocalTime   : _SYSTEMTIME;
    TimeZoneInformation     : _TIME_ZONE_INFORMATION;
begin
     System.GetMem( pItem, sizeof(TNotifyData) - 1 + pNotification^.cbSampleSize );
     pItem^.netId := pAddr^.netId;
     pItem^.port := pAddr^.port;
     pItem^.hNotification := pNotification^.hNotification;
     pItem^.cbSampleSize := pNotification^.cbSampleSize;
     System.Move( pNotification^.data, pItem^.data, pNotification^.cbSampleSize);
     FileTime:=pNotification^.nTimeStamp;
     GetTimeZoneInformation(TimeZoneInformation);
     FileTimeToSystemTime(FileTime, SystemTime);
     SystemTimeToTzSpecificLocalTime(@TimeZoneInformation, SystemTime, LocalTime);
     pItem^.dateTime:=SystemTimeToDateTime(LocalTime);
     with DevThreadList.LockList do
     try
      Add( pItem );
     finally
      DevThreadList.UnlockList;
      PostMessage( HWND(hUser), WM_ADSDEVICENOTIFICATION, 0, 0);
     end;
end;
```

Synchronisierung von Main-Thread und Callback-Thread mit Hilfe der Windows-Messages

```
//////////////////////////////////////////////////////////////////////////////////////////
procedure TForm1.WMAdsDeviceNotification( var Message: TMessage );
var pItem           : PNotifyData;
    X               : integer;
    sHex            : String;
    i               : Longword;
begin
    with DevThreadList.LockList do
    try
       Memo1.Lines.BeginUpdate();
       for X := 0 to Count-1 do
       begin
       nCnt := nCnt + 1;
       pItem := Items[X];
       {convert data to hex string}
       sHex := '';
       if pItem^.cbSampleSize > 0 then
       begin
        for i:= 0 to math.Min(pItem^.cbSampleSize-1, 20) do
            sHex := sHex + IntToHex( PBYTE(PAnsiChar(@pItem^.data) + i)^, 2 ) + ' ' ;
       end;
       {}
       Memo1.Lines.Insert(0,Format( 'idx:%-4d stamp:%s, %d.%d.%d.%d.%d.%d[%d], hNot:0x%x, size:
%d, data:%s',
                [nCnt,TimeToStr(pItem^.dateTime), pItem^.netId.b[0], pItem^.netId.b[1], pItem^.netId.
b[2],
                pItem^.netId.b[3], pItem^.netId.b[4], pItem^.netId.b[5], pItem^.port,
                pItem^.hNotification, pItem^.cbSampleSize, sHex]));
       FreeMem( pItem, sizeof(TNotifyData) -1 + pItem^.cbSampleSize );
       end;
       Clear();
   finally
      if memo1.Lines.Count > 1000 then
    memo1.Lines.Clear();
      Memo1.Lines.EndUpdate();
      DevThreadList.UnlockList;
```

```
   end;
   inherited;
end;
```

Verbindung zur SPS herstellen

```
////////////////////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
begin
     AdsPortOpen();
     AdsGetLocalAddress(@server);
     server.port := AMSPORT_R0_PLC_RTS1;
     DevThreadList := TThreadList.Create();
     StatusBar1.SimpleText := Format('Server netId: %d.%d.%d.%d.%d.%d port:
%d', [server.netId.b[0], server.netId.b[1],server.netId.b[2],
               server.netId.b[3], server.netId.b[4],server.netId.b[5], server.port]);
end;
```

Resourcen freigeben, Verbindung trennen

```
////////////////////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormDestroy(Sender: TObject);
var X : integer;
begin
    if hNotification <> 0 then
    Button2Click(Sender);
     with DevThreadList.LockList do
     try
      for X := 0 to Count -1 do
          FreeMem( Items[X], sizeof(TNotifyData) + PNotifyData(Items[X])^.cbSampleSize - 1 );
      Clear;
     finally
      DevThreadList.UnlockList;
      DevThreadList.Destroy;
     end;
     AdsPortClose();
end;
```

Notification anmelden

```
////////////////////////////////////////////////////////////////////////////////////////////
procedure TForm1.Button1Click(Sender: TObject);
var result : Longint;
    adsNotificationAttrib : TAdsNotificationAttrib;
    szVarName : AnsiString;
begin
     Memo1.Lines.Clear;
     nCnt := 0;
     { get symbol handle by symbol name }
     szVarName := AnsiString(Edit1.Text);
     result := AdsSyncReadWriteReq( @server, ADSIGRP_SYM_HNDBYNAME, 0,
                    sizeof(hVar), @hVar, Length(szVarName)+1, @szVarName[1] );
     Memo1.Lines.Insert(0, Format('AdsSyncReadWriteReq(..ADSIGRP_SYM_HNDBYNAME..) result: %d [0x%x],
 symbol handle: 0x%x', [result, result, hVar]));
     { add notification (connect to variable)}
     adsNotificationAttrib.cbLength := 20;{Should be right}
     adsNotificationAttrib.nTransMode := ADSTRANS_SERVERONCHA;
     adsNotificationAttrib.nMaxDelay := StrToInt(editDelayTime.Text) * 10000; //ms
     adsNotificationAttrib.nCycleTime := StrToInt(editCycleTime.Text) * 10000;//ms
     result := AdsSyncAddDeviceNotificationReq( @server, ADSIGRP_SYM_VALBYHND,
                         hVar,
                         @adsNotificationAttrib,
                         @NotificationCallback,
                         Handle, @hNotification );
     Memo1.Lines.Insert(0, Format( 'AdsSyncAddDeviceNotificationReq(..ADSIGRP_SYM_VALBYHND..) result
: %d [0x%x], notification handle: 0x%x', [result, result, hNotification] ));
     Button1.Enabled := not( result = ADSERR_NOERR);
     Button2.Enabled := ( result = ADSERR_NOERR);
end;
```

Notification abmelden

```
////////////////////////////////////////////////////////////////////////////////////////
procedure TForm1.Button2Click(Sender: TObject);
var result : Longint;
begin
     { delete notification }
     result := AdsSyncDelDeviceNotificationReq( @server, hNotification );
     hNotification := 0;
     Memo1.Lines.Insert(0, Format( 'AdsSyncDelDeviceNotificationReq() result: %d [0x%x]', [result,re
sult] ));
     {release symbol handle}
     result := AdsSyncWriteReq( @server, ADSIGRP_RELEASE_SYMHND, 0,sizeof(hVar), @hVar);
     Memo1.Lines.Insert(0, Format('AdsSyncReadWriteReq(..ADSIGRP_RELEASE_SYMHND..) result: %d [0x%x]
', [result, result]));
     Button1.Enabled := true;
     Button2.Enabled := false;
end;
initialization

end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473389579.exe
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466743435.exe

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

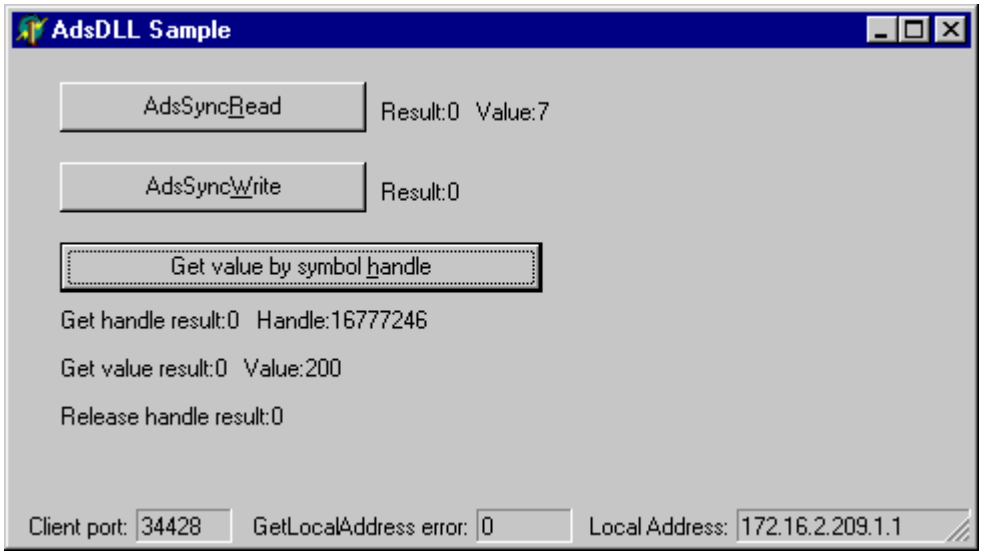
5.4.1.4 Synchroner Zugriff auf die SPS-Variablen

Voraussetzungen:

- Delphi 5.0 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selbst übersetzen möchten;

Aufgabenstellung

Aus der Windows-Applikation soll synchron schreibend und lesend auf eine SPS-Variable zugegriffen werden. Die Variable ist vom Typ Integer (16 Bit) und liegt auf dem Offset 520 in dem Merkerbereich des ersten Laufzeitsystems der SPS.



Das SPS-Programm

Die SPS-Variable wird beim Start der SPS mit dem Wert 7 initialisiert.

```
PROGRAM MAIN
VAR
    VARINT16 AT%MB520 :INT:=7;
END_VAR
```

Delphi 5 Programm

Die Deklarationen der benutzten TcAdsDLL-Funktionen befinden sich in den Pascal-Units *TcAdsDEF.pas und TcAdsAPI.pas*. Diese wurden über eine Uses-Klausel in das Projekt eingebunden.

```
unit DelphiAdsDLLSample1Unit;

interface
uses
TcAdsDEF, TcAdsAPI, Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,

StdCtrls, ExtCtrls, ComCtrls;
```

Die Beispielapplikation

In der Event-Funktion *FormCreate* wird die DLL-Funktion AdsPortOpen [▶ 11]aufgerufen. Beim Erfolg liefert diese Funktion die geöffnete Portnummer, sonst eine Null. Eventuelle Fehler werden über eine Message-Box an den Benutzer ausgegeben. Beim Beenden der Anwendung muß der Port wieder geschlossen werden. In der Event-Funktion *FormDestroy* wird dabei die DLL-Funktion AdsPortClose [▶ 11]aufgerufen.

Wurde der Ads-Port erfolgreich geöffnet, dann wird eine weitere DLL-Funktion aufgerufen: AdsGetLocalAddress [▶ 12]. Diese liefert die AMS-Addresse [▶ 30] des lokalen TwinCAT-PC's zurück. Die AMS-Adresse wird für Schreib/Lese-Zugriffe auf die SPS-Variablen benötigt.

```
var
  Form1     : TForm1;
  ClientPort    : longint;
  LocalAddr     : TAmsAddr;
  IndexGroup    : Integer;
  IndexOffset   : Integer;
  varHandle     : Integer;
  Result    : longint;
  varValue      : Smallint;
  const varName: AnsiString = 'MAIN.VARINT16'; //Symbol name
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientPort:= AdsPortOpen();
  if ClientPort > 0 then {OK}
    begin
      Result:=AdsGetLocalAddress( @LocalAddr );
      if Result = 0 Then  {OK}
    begin
      AdsState.Panels.Items[5].Text:= Format('%d.%d.%d.%d.%d.%d',[
          LocalAddr.netId.b[0], LocalAddr.netId.b[1], LocalAddr.netId.b[2],
          LocalAddr.netId.b[3], LocalAddr.netId.b[4], LocalAddr.netId.b[5]]);
      AdsState.Panels.Items[1].Text:=IntToStr( ClientPort );
      LocalAddr.port := 801; //Set the Ads port number for read/write/readwrite
      IndexGroup := $00004020; {memory range}
      IndexOffset := 520; {byte offset of the PLC variable}
    end
      else  {error}
    begin
      AdsState.Panels.Items[3].Text:=IntToStr( Result );
      MessageBox(NULL,'AdsGetLocalAddress()', 'Error!', MB_OK)
    end;
     end
  else  {error}
     MessageBox( NULL,'AdsPortOpen()', 'Error!', MB_OK );
end;
procedure TForm1.FormDestroy(Sender: TObject);
```

```
begin
     AdsPortClose();
end;
```

Bei der Betätigung des Buttons *AdsSyncRead,* wird der Wert der SPS-Variablen synchron über deren lokierte Adresse ausgelesen und auf der Form ausgegeben. Dabei wird in der Event-Funktion *OnSyncReadButtonClick* die DLL-Funktion AdsSyncReadReq [▶ 13] aufgerufen. Die Parameter *IndexGroup, IndexOffset* sind als globale Variablen definiert. Deren Werte werden beim Start der Applikation entsprechend gesetzt.

```
procedure TForm1.OnSyncReadButtonClick(Sender:
TObject);

begin

     Result := AdsSyncReadReq( @LocalAddr,
IndexGroup, IndexOffset, sizeof(varValue), @varValue );

     ReadLabel.Caption := Format( 'Result:
%d, Value: %d', [Result, varValue]);

end;
```

Ein Schreib-Zugriff auf die Variable wurde auf ähnliche Weise implementiert. In der Event-Funktion *OnSyncWriteButtonClick* wird die DLL-Funktion AdsSyncWriteReq [▶ 12] aufgerufen und der Wert der SPS-Variablen =200 gesetzt.

```
procedure
TForm1.OnSyncWriteButtonClick(Sender: TObject);

begin

    varValue := 200;

    Result := AdsSyncWriteReq( @LocalAddr,
IndexGroup, IndexOffset, sizeof(varValue), @varValue );

    WriteLabel.Caption := Format( 'Result:
%d', [Result]);

end;
```

Ein gleichzeitiger Lese- und Schreib-Zugriff auf SPS-Variablen kann über die DLL-Funktion AdsSyncReadWriteReq [▶ 14] realisiert werden. In dem Beispiel wird der Wert der Variablen über ein Handle gelesen. Dabei wird über die *AdsSyncReadWriteReq*-Funktion das Handle der Variablen zuerst angefordert. Mit diesem Handle kann dann der Wert der Variablen über die Funktion *AdsSyncReadReq* gelesen werden. Das Handle muß dann über die Funktion *AdsSyncWriteReq* frei gegeben werden.

```
procedure
TForm1.OnGetValueByHandleClick(Sender: TObject);

begin

     {get handle}

     Result := AdsSyncReadWriteReq( @LocalAddr,
ADSIGRP_SYM_HNDBYNAME, 0, sizeof(varHandle), @varHandle,
Length(varName) + 1, @varName[1] );

     GetHandleLabel.Caption := Format('Get
handle result: %d, Handle: 0x%x', [Result, varHandle] );

     {get value}

     Result := AdsSyncReadReq( @LocalAddr,
ADSIGRP_SYM_VALBYHND, varHandle, sizeof(varValue), @varValue
);

     GetValueLabel.Caption := Format( 'Get
value result: %d, Value: %d', [Result, varValue] );

     {release handle}

     Result := AdsSyncWriteReq( @LocalAddr,
ADSIGRP_RELEASE_SYMHND, 0, sizeof(varHandle), @varHandle );
```

```
    ReleaseHandleLabel.Caption := Format(
'Release handle result: %d', [Result] );

end;
```

Sollen die Werte mehrerer SPS-Variablen öfter asynchron gelesen werden, dann können die benötigten Handles in der Event-Funktion *FormCreate* beim Start der Applikation angefordert werden. Erst beim Beenden der Applikation in der Event-Funktion *FormDestroy* können sie wieder frei gegeben werden.

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473376267.exe
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466744843.exe

5.4.1.5 Statusänderungen vom TwinCAT-Router und der SPS erkennen

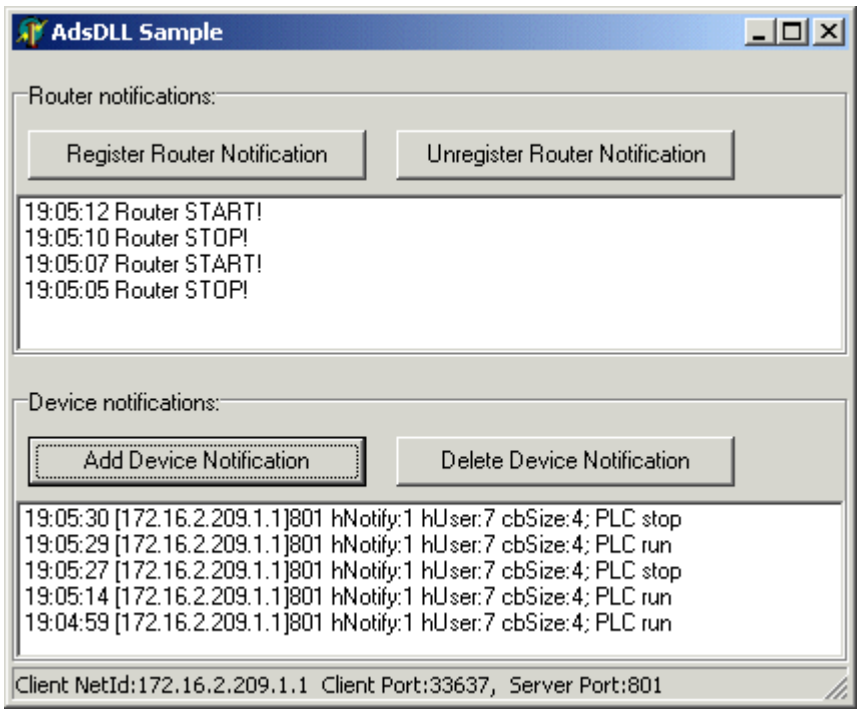
Voraussetzungen:

- Delphi 5.0 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selber übersetzen möchten;

Beschreibung

Zur Laufzeit einer Applikation ist es oft von Bedeutung, den Status von TwinCAT und/oder dessen Komponenten (Devices) abzufragen (ob die SPS sich z. B. im RUN-Status befindet). Um die benötigten Statusinformationen nicht immer pollend abzufragen gibt es die Möglichkeit, eine Mitteilung (Notification) bei den TwinCAT Komponenten anzumelden. Diese können dann mit Hilfe von Callback-Funktionen eine Statusänderung an die angemeldeten Client-Applikationen melden. In dem folgenden Beispielprogramm wird der Status der SPS (Laufzeitsystem 1) und der des TwinCAT-Routers auf dem lokalem Rechner überwacht. Die Applikation kann nur den Status des TwinCAT-Routers auf dem lokalem Rechner überwachen. D.h. der Status eines Routers auf einem Remote-PC kann auf diese Weise nicht überwacht werden. Der Status der SPS kann dagegen sowohl auf dem lokalen Rechner als auch auf einem Remote-PC mit den vorgestellten Funktionen überwacht werden.

Beim Start der Applikation wird eine Verbindung zum TwinCAT Router auf dem lokalen PC aufgebaut. Durch das Betätigen der entsprechenden Tasten wird eine Notification entweder an den TwinCAT-Router oder an das erste Laufzeitsystem der SPS angemeldet. Die ankommenden Mitteilungen (Callbacks) werden zwei Listen hinzugefügt. Über weitere Tasten können die Notifications bei den jeweiligen TwinCAT Geräten (Devices) gelöscht werden. Beim Beenden der Applikation wird die Verbindung zum TwinCAT Router geschlossen. Durch Starten/Stoppen des TwinCAT Systems über die Taskleiste werden z. B. die Router-Notifications und durch Starten/Stoppen der SPS die Device-Notifications gesendet. Die angemeldeten Device-Notifications werden von den TwinCAT Komponenten selbst verwaltet. Aus diesem Grund müssen die Device-Notifications beim Stoppen des TwinCAT Systems neu angemeldet werden, da die Komponente (hier das Laufzeitsystem der SPS) beim TwinCAT System-Stop aus dem Speicher entfernt wird.



Delphi 5 Programm

In der Event-Funktion *FormCreate* wird die DLL-Funktion AdsPortOpen [▶ 11]aufgerufen. Beim Erfolg liefert diese Funktion eine Portnummer, sonst eine Null. Eventuelle Fehler werden über eine Message-Box an den Benutzer ausgegeben. Wurde der Ads-Port erfolgreich geöffnet, dann wird eine weitere DLL-Funktion aufgerufen: AdsGetLocalAddress [▶ 12]. Diese liefert die AMS-Addresse [▶ 30] des lokalen TwinCAT-PC's (auf dem unsere Applikation läuft) zurück. Um die Notifications beim ersten Laufzeitsystem der SPS anzumelden wird noch eine weitere AMS-Addresse generiert. Diese besitzt die gleiche NetId wie unsere Applikation, weil wir die SPS auf dem gleichen TwinCAT-PC ansprechen wollen. Durch Zuweisung der Portnummer 801 wird das erste LZS ausgewählt. Beim Beenden der Anwendung muß der Port wieder geschlossen werden. In der Event-Funktion *FormDestroy* wird dabei die DLL-Funktion AdsPortClose [▶ 11] aufgerufen.

```
unit AdsDLLEventSampleForm;
interface
uses
  TcAdsDEF, TcAdsAPI, Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Math
,
  StdCtrls, ComCtrls;
const WM_ADSROUTERNOTIFICATION = WM_APP + 400;
const WM_ADSDEVICENOTIFICATION = WM_APP + 401;
type
  TForm1 = class(TForm)
    StatusBar1: TStatusBar;
    GroupBox1: TGroupBox;
    RegRouterNotify: TButton;
    UnregRouterNotify: TButton;
    ListBox1: TListBox;
    GroupBox2: TGroupBox;
    AddDevNotify: TButton;
    DelDevNotify: TButton;
    ListBox2: TListBox;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure RegRouterNotifyClick(Sender: TObject);
    procedure UnregRouterNotifyClick(Sender: TObject);
    procedure AddDevNotifyClick(Sender: TObject);
    procedure DelDevNotifyClick(Sender: TObject);
    procedure WMAdsRouterNotification( var Message: TMessage ); message WM_ADSROUTERNOTIFICATION;
    procedure WMAdsDeviceNotification( var Message: TMessage ); message WM_ADSDEVICENOTIFICATION;
  private
    { Private declarations }
    LocalAddr     :TAmsAddr;
    ServerAddr    :TAmsAddr;
    hNotification :DWORD;
  public
```

```
    { Public declarations }
  end;
{$ALIGN OFF}
type TThreadListItem = record
     netId       : TAmsNetId;
     port        : Word;
     hNotify     : Longword;
     stamp       : _FILETIME;  {int64}
     cbSize      : Longword;
     hUser       : Longword;
     data        : Byte;    //Array[1..ANYSIZE_ARRAY] of Byte;
     end;
PThreadListItem = ^TThreadListItem;
{$ALIGN ON}
var
  Form1: TForm1;
  wndHandle : HWND;
  DevThreadList : TThreadList;
implementation
{$R *.DFM}
////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
var AdsResult:Longint;
    ClientPort:Word;
begin
  GroupBox1.Caption := 'Router notifications:';
  GroupBox2.Caption := 'Device notifications:';
  StatusBar1.SimplePanel := true;
  wndHandle := Handle;
  DevThreadList := TThreadList.Create();
  ClientPort:= AdsPortOpen();
  if ClientPort = 0 then {error!}
    ShowMessage( 'AdsPortOpen() error!' )
  else  {OK}
  begin
    AdsResult:=AdsGetLocalAddress( @LocalAddr );
    if AdsResult = 0 then {OK}
    begin
      ServerAddr.netId:=LocalAddr.netId;{connect to the PLC on the local PC}
      ServerAddr.port:=801; {first RTS}
      StatusBar1.SimpleText:=Format('Client NetId:%d.%d.%d.%d.%d.%d  Client Port:%d,  Server Port:
%d',[
          LocalAddr.netId.b[0], LocalAddr.netId.b[1], LocalAddr.netId.b[2],
          LocalAddr.netId.b[3], LocalAddr.netId.b[4], LocalAddr.netId.b[5],
          ClientPort, ServerAddr.port]);
    end
    else
      ShowMessageFmt('AdsGetLocalAddress() error:%d', [AdsResult]);
  end;
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormDestroy(Sender: TObject);
var AdsResult:longint;
    X : integer;
begin
  UnregRouterNotifyClick( Sender );
  DelDevNotifyClick( Sender );
  With DevThreadList.LockList do
  try
     for X := 0 to Count-1 do
     FreeMem( Items[X], sizeof(TThreadListItem) + PThreadListItem(Items[X])^.cbSize - 1 );
     Clear();
  finally
     DevThreadList.UnlockList;
     DevThreadList.Destroy;
  end;
  AdsResult := AdsPortClose();
  if AdsResult > 0 then
    ShowMessageFmt('AdsPortClose() error:%d', [AdsResult]);
end;
```

Anmelden/Abmelden der Router-Notifications

Eine Notification beim Router wird durch den Aufruf der Funktion AdsAmsRegisterRouterNotification() [▶ 20] angemeldet. Die Router-Notification kann immer nur bei dem Router auf dem lokalen TwinCAT-PC angemeldet werden. Als einziger Funktionsparameter wird ein Funktionszeiger auf unsere Callback-Funktion (in unserem Fall ist es eigentlich eine Prozedur) übergeben. Über den Funktionsaufruf AdsAmsUnRegisterRouterNotification() [▶ 20] wird die Notification vom TwinCAT-Router gelöscht.

```
procedure TForm1.RegRouterNotifyClick(Sender: TObject);
var AdsResult:longint;
begin
  AdsResult:= AdsAmsRegisterRouterNotification(@AdsRouterCallback);
  if AdsResult > 0 then
    ListBox1.Items.Insert(0, Format('AdsAmsRegisterRouterNotification() error:%d', [AdsResult]));
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.UnregRouterNotifyClick(Sender: TObject);
var AdsResult:longint;
begin
  AdsResult:=AdsAmsUnRegisterRouterNotification();
  if AdsResult > 0 then
    ListBox1.Items.Insert(0, Format('AdsAmsUnRegisterRouterNotification() error:%d', [AdsResult]));
end;
```

Anmelden/Abmelden der Device-Notifications

Die Device-Notifications werden nicht an einer zentralen Stelle, sondern von der TwinCAT Komponente (Device) selbst verwaltet. Die Device-Notifications können auch bei einem Remote TwinCAT-PC angemeldet werden. Das Zielgerät wird dabei über die sogenannte AdsAms-Adresse ausgewählt. Die Notification wird durch den aufruf der DLL-Funktion AdsSyncAddDeviceNotificationReq() [▶ 18] angemeldet. Vorher müssen die Attribute der Mitteilung festgelegt werden. Diese werden als Parameter neben dem Funktionszeiger der Callback-Routine an die DLL-Funktion übergeben. Beim Erfolg liefert die Funktion kein Fehler und ein Notification-Handle. Dieses Handle wird dann benötigt um die Notification mit der DLL-Funktion AdsSyncDelDeviceNotificationReq() [▶ 19] zu löschen. Eine Applikation kann gleichzeitig mehrere Notifications anmelden. Jede Notification kann über ein Notification-Handle oder über ein User-Handle identifiziert werden. Diese Parameter werden immer in der Callback-Funktion an die Applikation zurückgesendet.

```
procedure TForm1.AddDevNotifyClick(Sender: TObject);
var AdsResult:Longint;
    huser :Longword;
    adsNotificationAttrib :TadsNotificationAttrib;
begin
  adsNotificationAttrib.cbLength       := sizeof(DWORD);
  adsNotificationAttrib.nTransMode     := ADSTRANS_SERVERONCHA;
  adsNotificationAttrib.nMaxDelay      := 0; // jede Aenderung sofort melden
  adsNotificationAttrib.nCycleTime     := 0; //
  hUser := 7;
  AdsResult:=AdsSyncAddDeviceNotificationReq( @ServerAddr,
            ADSIGRP_DEVICE_DATA,
            ADSIOFFS_DEVDATA_ADSSTATE,
            @adsNotificationAttrib,
            @AdsDeviceCallback, hUser, @hNotification  );
  if AdsResult > 0 then
    ListBox2.Items.Insert(0, Format('AdsSyncAddDeviceNotificationReq() error:%d', [AdsResult]));
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.DelDevNotifyClick(Sender: TObject);
var AdsResult:Longint;
begin
  AdsResult := AdsSyncDelDeviceNotificationReq( @ServerAddr, hNotification );
  if AdsResult > 0 then
    ListBox2.Items.Insert(0, Format('AdsSyncDelDeviceNotificationReq() error:%d', [AdsResult]));
end;
```

Die Callback-Funktionen

Die Callback-Funktionen für den Router-Callback und den DeviceNotification-Callback wurden als Proceduren definiert. Es können aber genauso Funktionen sein. Da die C++ Funktionen keine Rückgabeparameter zurückliefern und in Pascal eine Funktion immer einen Parameter zurückliefern muß, wurden für die Callback-Funktionen Proceduren gewählt. Sie können in der Callback-Funktion nicht erneut eine weitere DLL-Funktion aufrufen. Diese könnte dann einen weiteren Callback auslösen usw. Um die

Callback-Funktionen zu entkoppeln und mit dem Thread der Anwedung zu synchronisieren wurden PostMessage-API-Funktionen benutzt. Die Router-Callback-Daten werden an den Applikations-Thread direkt über die Message-Parameter übergeben. Die Daten der Device-Notification werden an den Applikations-Thread über eine Thread-Sichere-Liste übergeben.

```
Procedure AdsDeviceCallback(     pAddr:PAmsAddr; pNotification:PAdsNotificationHeader;   hUser:Longw
ord ); stdcall;
var pItem : PThreadListItem;
begin
    pItem := Nil;
    try
       GetMem( pItem, sizeof(TThreadListItem) + pNotification^.cbSampleSize - 1 );
       pItem^.netId := pAddr^.netId;
       pItem^.port := pAddr^.port;
       pItem^.hNotify := pNotification^.hNotification;
       pItem^.stamp := pNotification^.nTimeStamp;
       pItem^.cbSize := pNotification^.cbSampleSize;
       pItem^.hUser := hUser;
       //copy the notification data
       Move( pNotification^.data, pItem^.data, pNotification^.cbSampleSize);
    finally
    with DevThreadList.LockList do
    try
       Add( pItem );
    finally
       DevThreadList.UnlockList;
       PostMessage(wndHandle, WM_ADSDEVICENOTIFICATION, 0, 0);
    end;
    end;
end;
////////////////////////////////////////////////////////////////////////////
procedure AdsRouterCallback( nEvent:Longint ); stdcall;
begin
  PostMessage(wndHandle, WM_ADSROUTERNOTIFICATION, nEvent, 0);
end;
```

```
procedure TForm1.WMAdsRouterNotification( var Message: TMessage );
var tmpString:String;
begin
  case Message.WParam of
    AMSEVENT_ROUTERSTOP:
      tmpString:='Router STOP!';
    AMSEVENT_ROUTERSTART:
      tmpString:='Router START!';
    AMSEVENT_ROUTERREMOVED:
      tmpString:='Router REMOVED!';
  else
      tmpString:=Format('Unknown state:%d', [Message.WParam]);
  end;
  ListBox1.Items.Insert(0, Format('%s %s', [TimeToStr(time), tmpString]));
  inherited;
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.WMAdsDeviceNotification( var Message: TMessage );
var pItem           : PThreadListItem;
    X               : integer;
    FileTime        : _FILETIME;
    SystemTime, LocalTime   : _SYSTEMTIME;
    TimeZoneInformation     : _TIME_ZONE_INFORMATION;
    DateTime        : TDateTime;
    adsState        : Smallint;
    sState          : String;
    cbData          : Longword;
begin
    with DevThreadList.LockList do
    try
       for X := 0 to Count-1 do
       begin
       pItem := Items[X];
       {convert file time to local system time}
       FileTime:=pItem^.stamp;
       FileTimeToSystemTime(FileTime, SystemTime);
       GetTimeZoneInformation(TimeZoneInformation);
       SystemTimeToTzSpecificLocalTime(@TimeZoneInformation, SystemTime, LocalTime);
       DateTime:=SystemTimeToDateTime(LocalTime);
       cbData :=Min(pItem^.cbSize, sizeof(adsState));
       System.Move( pItem^.data, adsState, cbData );
```

```
      case adsState of
       ADSSTATE_STOP:  sState := 'STOP';
       ADSSTATE_RUN:  sState := 'RUN';
      else
       sState := Format('Other: %d', [adsState]);
      end;
      ListBox2.Items.Add(Format( '%s %d.%d.%d.%d.%d.%d[%d], hNot:0x%x, size:%d, hUser:0x%x, State:
%s',
                    [TimeToStr(DateTime), pItem^.netId.b[0], pItem^.netId.b[1], pItem^.netId.b[2],
                    pItem^.netId.b[3], pItem^.netId.b[4], pItem^.netId.b[5], pItem^.port,
                    pItem^.hNotify, pItem^.cbSize, pItem^.hUser,
                    sState ]));
      FreeMem( pItem, sizeof(TThreadListItem) + pItem^.cbSize - 1 ); //free item memory
      end;
      Clear();
   finally
      DevThreadList.UnlockList;
   end;
   inherited;
end;
```

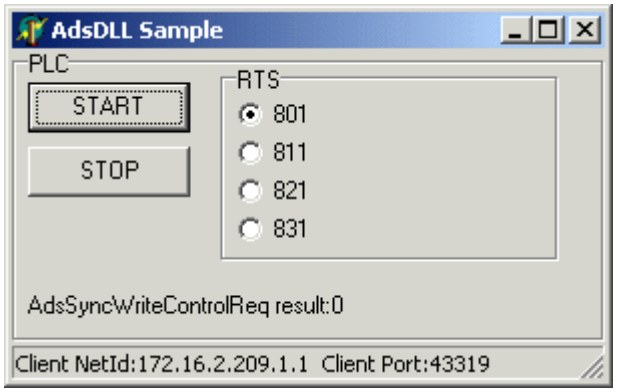
Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473377931.exe
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466746251.exe

5.4.1.6 SPS starten/stoppen

Mit dem folgenden Programm wird das Laufzeitsystem 1bis 4 der SPS gestartet, bzw. gestoppt.

Voraussetzungen:

- Delphi 5.0 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selbst übersetzen möchten;



Delphi 5 Programm:

```
unit AdsDLLWriteControlForm;
interface
uses
  TcAdsDEF, TcAdsAPI, Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    StatusBar1: TStatusBar;
    GroupBox1: TGroupBox;
    Start: TButton;
    Stop: TButton;
    RadioGroup1: TRadioGroup;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure StartClick(Sender: TObject);
    procedure StopClick(Sender: TObject);
```

```
    procedure RadioGroup1Click(Sender: TObject);
  private
    { Private declarations }
    LocalAddr     :TAmsAddr;
    ServerAddr    :TAmsAddr;
    AdsResult     :Longint;
    ClientPort    :Word;
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
begin
  GroupBox1.Caption := 'PLC';
  GroupBox1.Align := alClient;
  RadioGroup1.Caption := 'RTS';
  RadioGroup1.Items.Add('801');
  RadioGroup1.Items.Add('811');
  RadioGroup1.Items.Add('821');
  RadioGroup1.Items.Add('831');
  RadioGroup1.ItemIndex:=0;
  StatusBar1.SimplePanel := true;
  Label1.Caption:='Ads result:';
  ClientPort:= AdsPortOpen();
  if ClientPort = 0 then {error!}
    ShowMessage('AdsPortOpen() error!')
  else  {OK}
  begin
    AdsResult:=AdsGetLocalAddress( @LocalAddr );
    if AdsResult = 0 then {OK}
    begin
      ServerAddr.netId:=LocalAddr.netId;{connect to the PLC on the local PC}
      ServerAddr.port:=801; {select first RTS}
      StatusBar1.SimpleText:=Format('Client NetId:%d.%d.%d.%d.%d.%d  Client Port:%d',[
          LocalAddr.netId.b[0], LocalAddr.netId.b[1], LocalAddr.netId.b[2],
          LocalAddr.netId.b[3], LocalAddr.netId.b[4], LocalAddr.netId.b[5],
          ClientPort]);
    end
    else
      ShowMessageFmt('AdsGetLocalAddress() error:%d', [AdsResult]);
  end;
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormDestroy(Sender: TObject);
begin
  AdsResult := AdsPortClose();
  if AdsResult > 0 then
    ShowMessageFmt('AdsPortClose() error:%d', [AdsResult]);
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.StartClick(Sender: TObject);
begin
  AdsResult:=AdsSyncWriteControlReq( @ServerAddr, ADSSTATE_RUN, 0, 0, Nil );
  Label1.Caption := Format('AdsSyncWriteControlReq result:%d', [AdsResult]);
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.StopClick(Sender: TObject);
begin
  AdsResult:=AdsSyncWriteControlReq( @ServerAddr, ADSSTATE_STOP, 0, 0, Nil );
  Label1.Caption := Format('AdsSyncWriteControlReq result:%d', [AdsResult]);
end;
////////////////////////////////////////////////////////////////////////////
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  ServerAddr.port := StrToInt(RadioGroup1.Items[RadioGroup1.ItemIndex]);
end;
end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473379595.exe
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466747659.exe

5.4.1.7 Beispiel mit erweiterten Funktionen (für Multithreaded-Applikationen)

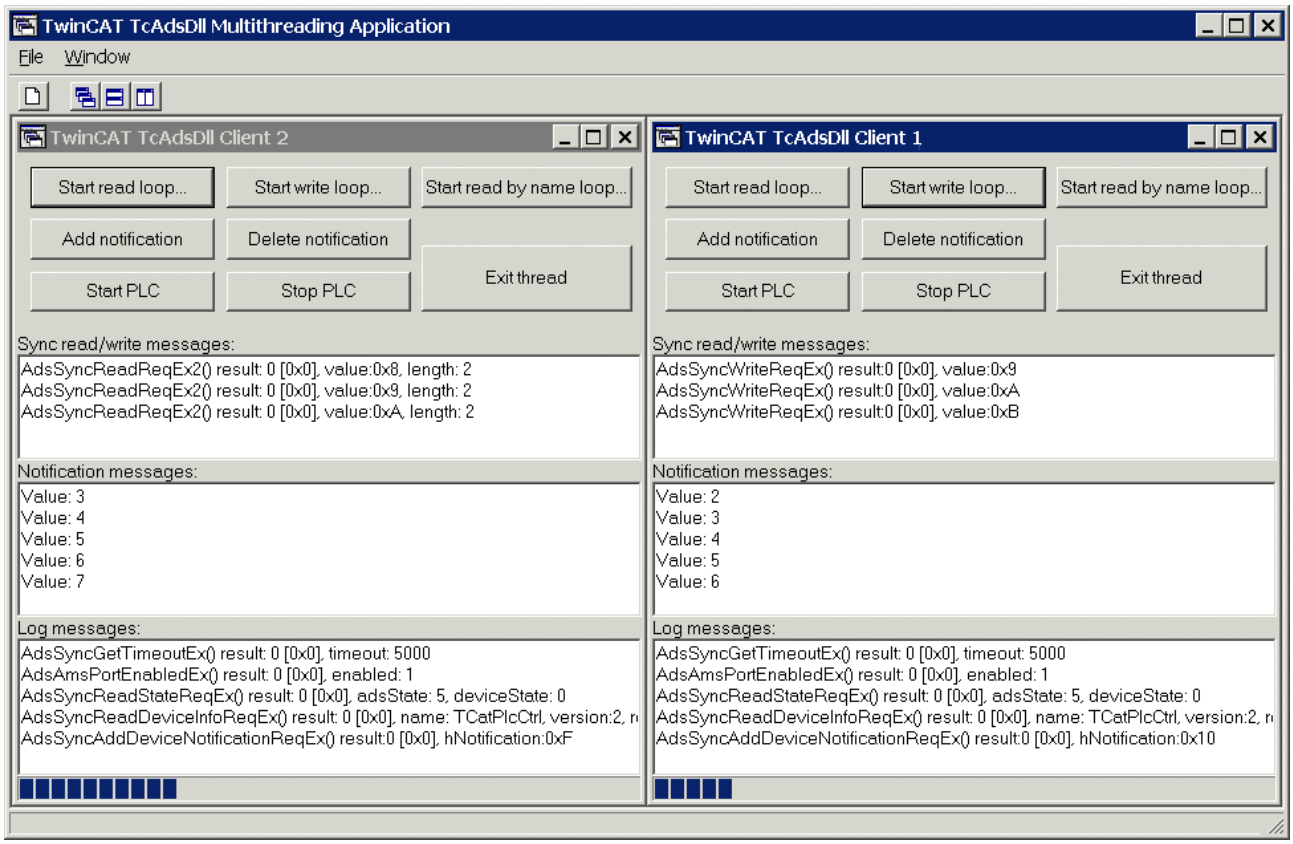
Bei der Applikation handelt es sich um eine einfache Delphi-MDI-Applikation. Jedes Child-Fenster (GUI) besitzt einen eigenen Ads-Thread (ADS-Port für die Datenkommunikation) in dem die erweiterten Ads-Funktionen aufgerufen werden. Die Applikation demonstriert wie ADS-Funktionsaufrufe (längere Aktionen) in einen separaten Thread ausgelagert werden können ohne den Main-Thread zu blokieren. Die Applikation kann als Startpunkt für eigene Implementierungen benutzt werden.

Die Ads-Thread-Funktionen wurden in einer separaten Unit: **AdsThread.pas** implementiert. Die Child-Fensterfunktionen wurden ebenfalls in einer separaten Unit: **CHILDWIN.PAS / CHILDWIN.DFM** implementiert.

Die Kommunikation zwischen der GUI (Child-Fenster) und dem Ads-Thread wurde mit Hilfe der WM-Messages realisiert (PostThreadMessage-/SendMessage-Funktionen). Dadurch wurde der Main-Thread von den Ads-Threads entkoppelt. Es können aber auch andere Synchronisierungsmechanismen benutzt werden.

Voraussetzungen:

- Delphi 7.0 + Update 7.1 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selber übersetzen möchten;



Beschreibung

Starten sie das dazugehörige SPS-Projekt **Sample.pro** und die **MDIAPP.exe**. Mit File->New kann eine neue Ads-Client-Verbindung und das dazugehörige GUI-Fenster erzeugt werden. Erzeugen sie mehrere Child-Fenster.

```
PROGRAM MAIN
VAR
    vUINT AT%MB0 :UINT;
END_VAR
```

```
vUINT := vUINT +1;
```

Bei einem Mausklick auf die entsprechenden Buttons werden folgende Aktionen gestartet:

- Start read loop...: Liest synchron (by address) in einer For-Schleife die vUINT-Variable (Dauer ca. 10 Sekunden).
- Start write loop...: Schreibt synchron (by address) in einer For-Schleife die vUINT-Variable (Dauer ca. 10 Sekunden).
- Start read by name loop...: Liest synchron (by variable name) in einer For-Schleife die vUINT-Variable (Dauer ca. 10 Sekunden).
- Add notification: Meldet eine Notification an. Immer wenn sich der Wert der vUINT-Variablen ändert wird er an die Client-Applikation übertragen un im (Notification mesages) Memo-Control angezeigt.
- Delete notification: Deaktiviert die Notification.
- Start PLC: Startet die SPS.
- Stop PLC: Stoppt die SPS.
- Exit thread: Beendet den Ads-Thread (verlässt den TThread-Execute-Aufruf).

Zum Test wählen Sie z. B. in einem Fenster den *Start read loop...-Button* und in einem anderem Fenster *Start write loop...-Button*.

Delphi 7 Programm

Main.pas:

```
unit MAIN;
interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Buttons, Messages, ExtCtrls, ComCtrls, StdActns,
  ActnList, ToolWin, ImgList, CHILDWIN, TcAdsDef, TcAdsAPI, AdsThread;
const
  DEFAULT_SERVERADDR : TAmsAddr =    ( netID : ( b : (0,0,0,0,0,0 ) );
//  DEFAULT_SERVERADDR : TAmsAddr =    ( netID : ( b : (172,16,7,53,1,1 ) );
                       port : 801);
  DEFAULT_ADSTIMEOUT : Longint = 5000;  {default ads timeout = 5 seconds}
  DEFAULT_IG  : Longint = $00004020;    {memory range}
  DEFAULT_IO  : Longint = $00000000;    {byte offset of the PLC variable}
type
  TMainForm = class(TForm)
    MainMenu1: TMainMenu;
    ActionList1: TActionList;
    StatusBar: TStatusBar;
    ImageList1: TImageList;
    File1: TMenuItem;
    FileNewItem: TMenuItem;
    FileCloseItem: TMenuItem;
    FileExitItem: TMenuItem;
    FileExit1: TAction;
    FileClose1: TWindowClose;
    Window1: TMenuItem;
    WindowCascadeItem: TMenuItem;
    WindowTileItem: TMenuItem;
    WindowTileItem2: TMenuItem;
    WindowMinimizeItem: TMenuItem;
    WindowArrangeItem: TMenuItem;
    WindowCascade1: TWindowCascade;
    WindowTileHorizontal1: TWindowTileHorizontal;
    WindowTileVertical1: TWindowTileVertical;
    WindowMinimizeAll1: TWindowMinimizeAll;
    WindowArrangeAll1: TWindowArrange;
    ToolBar2: TToolBar;
    ToolButton9: TToolButton;
    ToolButton8: TToolButton;
    ToolButton10: TToolButton;
    ToolButton11: TToolButton;
    procedure FileNew1Execute(Sender: TObject);
    procedure FileExit1Execute(Sender: TObject);
  private
    { Private declarations }
    procedure CreateMDIChild(const Name: string);
  public
    { Public declarations }
  end;
var
  MainForm: TMainForm;
implementation
```

```
{$R *.dfm}
////////////////////////////////////////////////////////////////////////////
procedure TMainForm.CreateMDIChild(const Name: string);
var
  Child: TMDIChild;
  params : TAdsParams;
begin
  params.hOwner := 0;
  params.server := DEFAULT_SERVERADDR;
  params.timeout := DEFAULT_ADSTIMEOUT;
  params.varIG   := DEFAULT_IG;
  params.varIO := DEFAULT_IO;
  params.length := 2; // INT
  params.name := 'MAIN.VUINT';
  { create a new MDI child window }
  Child := TMDIChild.Create(Application, params );
  Child.Caption := Name;
end;
////////////////////////////////////////////////////////////////////////////
procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
  CreateMDIChild('TwinCAT TcAdsDll Client ' + IntToStr(MDIChildCount + 1));
end;
////////////////////////////////////////////////////////////////////////////
procedure TMainForm.FileExit1Execute(Sender: TObject);
begin
  Close;
end;
end.
```

ChildWin.pas

```
unit CHILDWIN;
interface
uses Windows, Classes, Messages, Graphics, Forms, Controls, StdCtrls, TcAdsDef, TcAdsApi, AdsThread,
  ComCtrls;
type
  TMDIChild = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Memo2: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    Memo3: TMemo;
    Label3: TLabel;
    Button7: TButton;
    Button8: TButton;
    ProgressBar1: TProgressBar;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
    adsThread : TAdsThread;
    procedure OnNotification(var Message: TMessage); message WM_NOTIFICATION;
    procedure OnReadWrite(var Message: TMessage); message WM_READWRITE;
    procedure OnLog(var Message: TMessage); message WM_LOG;
    procedure OnProgress(var Message: TMessage); message WM_PROGRESS;
  public
    { Public declarations }
    constructor Create( AOwner: TComponent; params : TAdsParams );
  end;
implementation
{$R *.dfm}
////////////////////////////////////////////////////////////////
constructor TMDIChild.Create(AOwner: TComponent; params : TAdsParams );
begin
  inherited Create(AOwner);   // create child window
```

```
  params.hOwner := self.Handle; // set owner window handle
  adsThread := TAdsThread.Create(params ); // create ads thread  (open ads connection)
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.FormDestroy(Sender: TObject);
begin
  adsThread.Free();// release thread resources (close ads connection)
  inherited;
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.OnLog(var Message: TMessage);
begin
 Memo1.Lines.Add(String(Message.WParam));
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.OnReadWrite(var Message: TMessage);
begin
  Memo3.Lines.Add(String(Message.WParam));
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.OnNotification(var Message: TMessage);
begin
  Memo2.Lines.Add(String(Message.WParam));
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.OnProgress(var Message: TMessage);
begin
  if Message.LParam = 0 then
    Progressbar1.Position := Message.WParam
  else
  begin
    Progressbar1.Min := Message.WParam;
    Progressbar1.Max := Message.LParam;
  end;
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button1Click(Sender: TObject);
begin
  Memo3.Lines.Clear();
  PostThreadMessage( adsThread.ThreadID, WM_READLOOP, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button2Click(Sender: TObject);
begin
  Memo3.Lines.Clear();
  PostThreadMessage( adsThread.ThreadID, WM_WRITELOOP, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button3Click(Sender: TObject);
begin
  Memo2.Lines.Clear();
  PostThreadMessage( adsThread.ThreadID, WM_ADDNOTIFICATION, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button4Click(Sender: TObject);
begin
  PostThreadMessage( adsThread.ThreadID, WM_DELNOTIFICATION, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button5Click(Sender: TObject);
begin
  PostThreadMessage( adsThread.ThreadID, WM_STARTPLC, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button6Click(Sender: TObject);
begin
  PostThreadMessage( adsThread.ThreadID, WM_STOPPLC, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button8Click(Sender: TObject);
begin
  PostThreadMessage( adsThread.ThreadID, WM_READBYNAME, 0, 0 );
end;
////////////////////////////////////////////////////////////////////
procedure TMDIChild.Button7Click(Sender: TObject);
```

```
begin
 adsThread.Terminate();
 Button1.Enabled := false;
 Button2.Enabled := false;
 Button3.Enabled := false;
 Button4.Enabled := false;
 Button5.Enabled := false;
 Button6.Enabled := false;
 Button7.Enabled := false;
 Button8.Enabled := false;
end;
end.
```

AdsThread.pas:

```
unit AdsThread;
interface
uses
  Messages, Windows, Classes, SysUtils, TcAdsDef, TcAdsApi;
const
  // messages send from form to ads thread
  WM_READLOOP       = WM_APP + $0001;
  WM_WRITELOOP      = WM_APP + $0002;
  WM_ADDNOTIFICATION = WM_APP + $0003;
  WM_DELNOTIFICATION = WM_APP + $0004;
  WM_STARTPLC       = WM_APP + $0005;
  WM_STOPPLC    = WM_APP + $0006;
  WM_READBYNAME     = WM_APP + $0007;
  // messages send from ads thread to form
  WM_LOG        = WM_APP + $1001;
  WM_NOTIFICATION   = WM_APP + $1002;
  WM_READWRITE      = WM_APP + $1003;
  WM_PROGRESS       = WM_APP + $1004;
  MAX_TEST_LOOPS  : Longint = 100;
type
  TAdsParams = record
      hOwner : HWND;
      server  : TAmsAddr;  // TwincAT ADS server network address
      timeout : Longint;  // ads communication timeout
      varIG   : Longint;  // index offset of the plc variable
      varIO   : Longint;  // index group of the plc variable
      length  : Longint; // byte size of the plc variable
      name    : String; // plc variable symbol name
  end;
type
  TAdsThread = class(TThread)
  private
    { Private declarations }
    port    : Longint;
    params   : TAdsParams;
    hNotification : Longint;  // notification handle
    procedure LogMessage( msg : String );
    procedure LogReadWrite( msg : String );
    procedure StartProgress( min : Longint; max: Longint );
    procedure DoProgress( position : Longint );
    function ReadByName( port : Longint; pAddr:PAmsAddr; sName : String; pReadData:Pointer; cbReadLe
ngth : Longword; pcbReturn :PLONGWORD ): Longint;
    procedure OnReadLoop(var Message: TMessage); message WM_READLOOP;
    procedure OnWriteLoop(var Message: TMessage); message WM_WRITELOOP;
    procedure OnAddNotification(var Message: TMessage); message WM_ADDNOTIFICATION;
    procedure OnDelNotification(var Message: TMessage); message WM_DELNOTIFICATION;
    procedure OnStartPlc(var Message: TMessage); message WM_STARTPLC;
    procedure OnStopPlc(var Message: TMessage); message WM_STOPPLC;
    procedure OnReadByName(var Message: TMessage); message WM_READBYNAME;
  protected
    procedure Execute; override;
  public
    constructor Create( params : TAdsParams );
  end;
implementation
```

```
//////////////////////////////////////////////////////////////////////////
// Notification callback
Procedure Callback( pAddr:PAmsAddr; pNotification:PAdsNotificationHeader; hUser:Longword ); stdcall;
var sValue : String;
    nValue : Word;
begin
  // only to test
```

```
  if pNotification^.cbSampleSize = sizeof(nValue) then
  begin
    nValue := (PWORD(@pNotification^.data))^;
    sValue := Format('Value: %d', [ nValue ] );
    SendMessage( HWND(hUser), WM_NOTIFICATION, Integer(sValue), 0 );
  end;
end;
```

```
////////////////////////////////////////////////////////////////////////////
constructor TAdsThread.Create( params : TAdsParams );
begin
  hNotification := 0;
  port := 0;
  self.params := params; // save parameter for later use
  inherited Create(false);
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.Execute();
var result      : Longint;
    client      : TAmsAddr;
    oldTimeout  : Longint;
    enabled     : LongBool;
    adsState, deviceState : Word;
    adsVersion: TAdsVersion;
    szDevName : AnsiString;
    pDllVersion : PAdsVersion;
    Msg: TMsg;
    DMsg: TMessage;
begin
  LogMessage( 'Ads thread started!' );
  result := AdsGetDllVersion();
  if result <> 0 then
  begin
    pDllVersion := PAdsVersion(@result);
    LogMessage( Format('---- TcAdsDll.dll version:%d, revision:%d, build:%d ----',
        [pDllVersion^.version, pDllVersion^.revision, pDllVersion^.build]));
  end;
  port := 0;
  port := AdsPortOpenEx();
  if port <> 0 then
  begin
    result := AdsGetLocalAddressEx( port, @client );
    LogMessage(Format('AdsGetLocalAddressEx() result: %d [0x%x]', [result, result]));
    if result = 0 then
    begin

      if (params.server.netId.b[0] = 0)
      And (params.server.netId.b[1] = 0)
      And (params.server.netId.b[2] = 0)
      And (params.server.netId.b[3] = 0)
      And (params.server.netId.b[4] = 0)
      And (params.server.netId.b[5] = 0)then
      begin
    params.server.netId := client.netId;
      end;
      // show client address
      LogMessage(Format('Client port:%d [0x%x], netID:%d.%d.%d.%d.%d.%d',
              [client.port, client.port,
              client.netid.b[0],client.netid.b[1],client.netid.b[2],
              client.netid.b[3],client.netid.b[4],client.netid.b[5]]));
      // show server address
      LogMessage(Format('Server port:%d [0x%x], netID:%d.%d.%d.%d.%d.%d',
              [params.server.port, params.server.port,
              params.server.netid.b[0],params.server.netid.b[1],params.server.netid.b[2],
              params.server.netid.b[3],params.server.netid.b[4],params.server.netid.b[5]]));
      // read current ads timeout
      oldTimeout := 0;
      result := AdsSyncGetTimeoutEx( port, @oldTimeout );
      LogMessage(Format('AdsSyncGetTimeoutEx() result: %d [0x%x], timeout: %d', [result, result, old
Timeout]));
      if oldTimeout <> params.timeout then
      begin
    // set new ads timeout
    result := AdsSyncSetTimeoutEx( port, params.timeout );
    LogMessage(Format('AdsSyncSetTimeoutEx() result: %d [0x%x], timeout: %d', [result, result, param
```

```
s.timeout]));
      end;
      result := AdsAmsPortEnabledEx( port, @enabled );
      LogMessage(Format('AdsAmsPortEnabledEx() result: %d [0x%x], enabled: %d', [result, result, Ord
(enabled)]));
      if enabled then
      begin
    result := AdsSyncReadStateReqEx( port, @params.server, @adsState, @deviceState );
    LogMessage(Format('AdsSyncReadStateReqEx() result: %d [0x%x], adsState: %d, deviceState: %d',
          [result, result, adsState, deviceState]));
    SetLength( szDevName, ADS_FIXEDNAMESIZE + 1 );
    result := AdsSyncReadDeviceInfoReqEx( port, @params.server, @szDevName[1], @adsVersion );
    LogMessage(Format('AdsSyncReadDeviceInfoReqEx() result: %d [0x%x], name: %s, version:
%d, revision:%d, build:%d',
          [result, result, szDevName, adsVersion.version, adsVersion.revision, adsVersion.build]));
      end;
      ////////////////////////////////////////////////
      PeekMessage(Msg,0,0,0,PM_NOREMOVE); // Create Message Queue
      repeat
    if PeekMessage(Msg,0,0,0,PM_REMOVE) then
    begin
      DMsg.Msg:=Msg.message;
      DMsg.wParam:=Msg.wParam;
      DMsg.lParam:=Msg.lParam;
      DMsg.Result:=0;
      Dispatch(DMsg);
    end;
    Sleep(10);
      until Terminated;
      if hNotification <> 0 then
      begin
     result := AdsSyncDelDeviceNotificationReqEx( port, @params.server, hNotification );
     hNotification := 0;
      end;
      ////////////////////////////////////////////////
      result := AdsPortcloseEx(port); // close ads port
      LogMessage(Format('AdsPortcloseEx() result: %d [0x%x]', [result, result]));
    end
  end
  else
    LogMessage('AdsPortOpenEx() failed!');
  LogMessage( 'Ads thread stopped!' );
end;
```

```
//////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnReadLoop(var Message: TMessage);
var i, error : Longint;
    varValue    : Word;
    cbReturned    : Longint;
begin
  StartProgress( 0, MAX_TEST_LOOPS);
  for i:= 1 to MAX_TEST_LOOPS  do
  begin
    cbReturned := 0;
    varValue := 0;
    error := AdsSyncReadReqEx2( port, @params.server, params.varIG, params.varIO, sizeof(varValue),
@varValue, @cbReturned );
    LogReadWrite( Format('AdsSyncReadReqEx2() result: %d [0x%x], value:0x%x, length: %d', [error, er
ror, varValue, cbReturned]));
    if Terminated then break;
    DoProgress(i);
    Sleep(100);
  end;
end;
```

```
//////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnWriteLoop(var Message: TMessage);
var i, error : Longint;
    varValue    : Word;
begin
  StartProgress( 0, MAX_TEST_LOOPS);
  varValue := 1;
  for i:= 1 to MAX_TEST_LOOPS  do
  begin
    varValue := i; // write some test value
    error := AdsSyncWriteReqEx( port, @params.server, params.varIG, params.varIO, sizeof(varValue),
```

```
@varValue );
    LogReadWrite( Format('AdsSyncWriteReqEx() result:
%d [0x%x], value:0x%x', [error, error, varValue]));
    if Terminated then break;
    DoProgress(i);
    Sleep(100);
  end;
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnReadByName(var Message: TMessage);
var i, cbReturned, error : Longint;
    varValue : Word;
begin
  StartProgress( 0, MAX_TEST_LOOPS);
  for i:= 1 to MAX_TEST_LOOPS  do
  begin
    cbReturned := 0;
    varValue := 0;
    error := ReadByName( port, @params.server, params.name, @varValue, sizeof(varValue), @cbReturned
);
    LogReadWrite( Format('Ads read value by name result:
%d [0x%x], value: 0x%x, length: %d', [error, error, varValue, cbReturned]));
    if Terminated then break;
    DoProgress(i);
    Sleep(100);
  end;
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnAddNotification(var Message: TMessage);
var error : Longword;
    adsNotificationAttrib : TAdsNotificationAttrib;
begin
     adsNotificationAttrib.cbLength := params.length;
     adsNotificationAttrib.nTransMode := ADSTRANS_SERVERONCHA;
     adsNotificationAttrib.nMaxDelay := 10000000;//1 second
     adsNotificationAttrib.nCycleTime := 0;
     error := AdsSyncAddDeviceNotificationReqEx( port, @params.server,
                         params.varIG,
                         params.varIO,
                         @adsNotificationAttrib,
                         @Callback,
                         params.hOwner, // send window handle as user data
                         @hNotification  );
    LogMessage( Format('AdsSyncAddDeviceNotificationReqEx() result:
%d [0x%x], hNotification:0x%x', [error, error, hNotification]));
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnDelNotification(var Message: TMessage);
var error : Longint;
begin
  if hNotification <> 0 then
  begin
    error := AdsSyncDelDeviceNotificationReqEx( port, @params.server, hNotification );
    hNotification := 0;
    LogMessage( Format('AdsSyncDelDeviceNotificationReqEx() result:
%d [0x%x], hNotification:0x%x', [error, error, hNotification]));
  end;
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnStartPlc(var Message: TMessage);
var error : Longint;
begin
  error:=AdsSyncWriteControlReqEx( port, @params.server, ADSSTATE_RUN, 0, 0, Nil );
  LogMessage( Format('AdsSyncWriteControlReqEx() result: %d [0x%x]', [error, error]));
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TAdsThread.OnStopPlc(var Message: TMessage);
var error : Longint;
begin
  error:=AdsSyncWriteControlReqEx( port, @params.server, ADSSTATE_STOP, 0, 0, Nil );
  LogMessage( Format('AdsSyncWriteControlReqEx() result: %d [0x%x]', [error, error]));
end;
```

```
////////////////////////////////////////////////////////////////////////////
function TAdsThread.ReadByName( port : Longint; pAddr:PAmsAddr; sName : String; pReadData:Pointer; c
bReadLength : Longword; pcbReturn :PLONGWORD ) : Longint;
var hSymbol : Longword;
    error : Longint;
    varName : AnsiString;
begin
  {get handle}
  hSymbol := 0;
  varName := AnsiString(sName);
  error := AdsSyncReadWriteReqEx2( port, pAddr, ADSIGRP_SYM_HNDBYNAME, 0, sizeof(hSymbol), @hSymbol,
 Length(varName)+1, @varName[1], pcbReturn );
  if error = 0 then
  begin
    {get value}
    error := AdsSyncReadReqEx2( port, pAddr, ADSIGRP_SYM_VALBYHND, hSymbol, cbReadLength, pReadData,
 pcbReturn );
    {release handle}
    error := AdsSyncWriteReqEx( port, pAddr, ADSIGRP_RELEASE_SYMHND, 0, sizeof(hSymbol), @hSymbol );
  end;
  result := error;
end;
end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473381259.exe
Delphi 7 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466749067.exe

Dokumente hierzu

delphi_adsdll_api_units.zip (Resources/zip/12466785675.zip)

5.4.1.8 ADS Summenkommando: Lesen oder schreiben von mehreren Variablen mit einem ADS-Befehl.

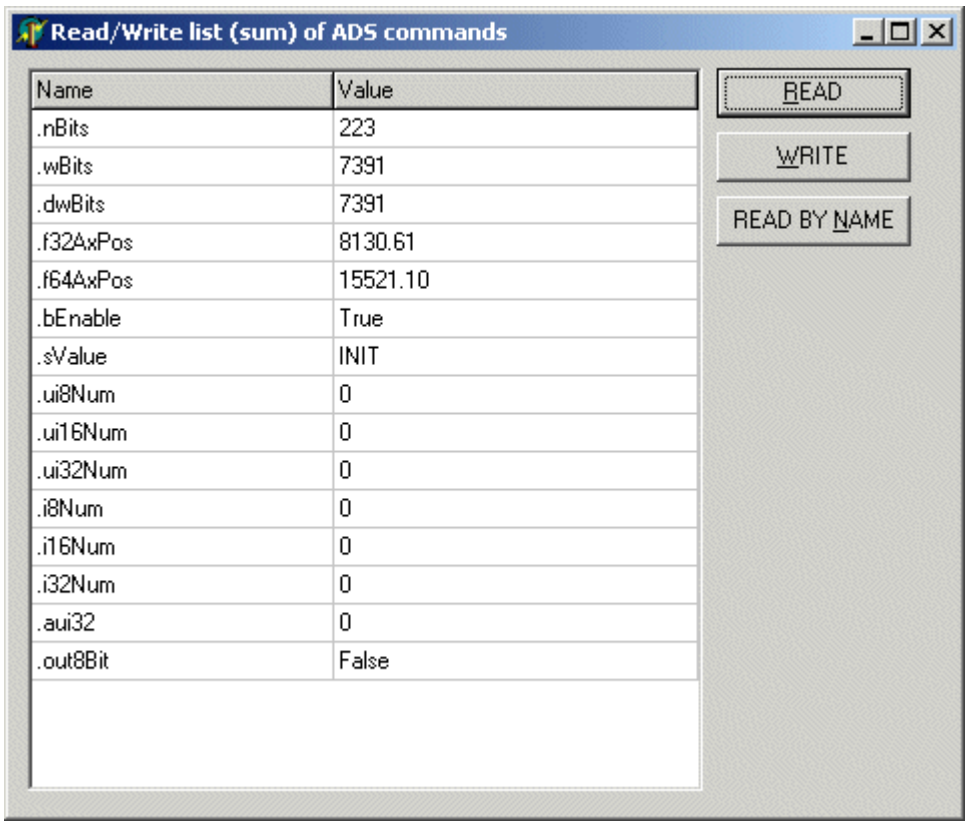
Voraussetzungen:

- **TwinCAT v2.10 Build >= 1324**;
- Delphi 6.0 + Update Pack 2 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selber übersetzen möchten;

Aufgabenstellung

Es sollen 15-SPS-Variablen, unterschiedlichen Typs mit einem ADS-Summenkommando gelesen oder geschrieben werden.

In Delphi ist es möglich offene Arrayparameter an Funktionen zu übergeben. D.h. die Anzahl der Arrayelemente in so einem Funktionsparameter ist nicht festgelegt. In der TcAdsApi.pas-Datei wurden drei neue Funktionen implementiert: **AdsSyncSumReadReq**, **AdsSyncSumWriteReq** und **AdsSyncSumReadWriteReq**. Diese Funktionen benutzen die Funktionalität der offenen Arrayparameter und vereinfachen die Benutzung des Summenkomandos unter Delphi. Das Beispielprojekt nutzt diese neuen Wrapper-Funktionen.



Das SPS-Programm

Die SPS-Variablen wurden als globale Variablen deklariert. Bei einer gestarteten SPS werden Wertänderungen der Variablen im einfachen MAIN-Programm simuliert.

```
VAR_GLOBAL
    nBits : BYTE;
    wBits : WORD;
    dwBits : DWORD;
    f32AxPos : REAL;
    f64AxPos : LREAL;
    bEnable : BOOL;
    sValue : STRING := 'INIT';
    ui8Num : USINT;
    ui16Num : UINT;
    ui32Num : UDINT;
    i8Num : SINT;
    i16Num : INT;
    i32Num : DINT;
    aui32 : ARRAY[0..9] OF DWORD;
    out8Bit AT%QX0.1 : BOOL;
END_VAR
```

Delphi 6 Programm

Die Deklarationen der benutzten TcAdsDLL-Funktionen befinden sich in den Pascal-Units *TcAdsDEF.pas und TcAdsAPI.pas*. Diese wurden über eine Uses-Klausel in das Projekt eingebunden.

```
unit Unit1;
interface
uses
  TcAdsDEF, TcAdsAPI, Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, ValEdit, ComCtrls, StdCtrls, Math;
```

Beim Programmstart wird der Kommunikationsport geöffnet und beim Beenden geschlossen. Die benötigten Symbolinformationen der SPS-Variablen wie Index-Group, index-Offset usw. werden beim Programmstart ebenfalls mit Hilfe eines Summenkommandos einmalig gelesen.

```
//////////////////////////////////////////////////////////////////////////
type
    TAdsSymbolInfo = packed record
    entry   : TAdsSymbolEntry; // ADS symbol info
    name    : AnsiString; // ADS symbol name
```

```
    arraySize : Longword;
    case typeID : ADS_DATATYPE of       //variable part value
    ADST_VOID:();               //nothing
    ADST_INT16:(vINT : Smallint);       //2 byte signed number
    ADST_INT32:(vDINT: Longint);    //4 byte signed number
    ADST_REAL32:(vREAL : Single);       //4 byte real
    ADST_REAL64:(vLREAL : Double);      //8 byte real
    ADST_INT8:(vSINT : Shortint);       //1 byte signed number
    ADST_UINT8:(vBYTE : Byte);      //1 byte unsigned number
    ADST_UINT16:(vWORD : Word);     //2 byte unsigned number
    ADST_UINT32:(vDWORD : DWORD);       //4 byte unsigned number
    ADST_STRING:(vSTRING : Array[0..80] OF AnsiChar); //null terminated string
    ADST_BIT:(vBOOL : Boolean);     //boolean
    ADST_BIGTYPE:(data : Array[0..255] OF Byte); //data byte array
    end;

//////////////////////////////////////////////////////////////////////
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    ValueListEditor1: TValueListEditor;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    infos : Array of TAdsSymbolInfo;
    client : Longint;
    amsAddr : TAmsAddr;
    { Public declarations }
    procedure ReadListOfSymbolInfo();
    procedure ReadListOfVars();
    procedure WriteListOfVars();
    procedure ReadListByName();
    procedure DataToView( var info : TAdsSymbolInfo);
    procedure ViewToData( var info : TAdsSymbolInfo);
  end;



var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}
//////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
var adsErr : Longint;
begin
    Button1.Enabled := True;
    Button2.Enabled := False;
    Button3.Enabled := False;
    client := AdsPortOpen(); // open ads port
    if client <> 0 then
    begin
    adsErr := AdsGetLocalAddress( @amsAddr );
    if adsErr = 0 then
    begin
        amsAddr.port := AMSPORT_R0_PLC_RTS1; // set port to 801  (first PLC run time system)
        ReadListOfSymbolInfo();// read symbol info
    end
    else
        ShowMessageFmt('AdsGetLocalAddress() error: %d [0x%X]', [adsErr, adsErr]);
    end
    else
    ShowMessage('AdsPortOpen() failed!');
end;
```

```
//////////////////////////////////////////////////////////////////////
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    if client <> 0 then
    AdsPortClose();// close ads port
end;
```

Symbolinformationen lesen

```
//////////////////////////////////////////////////////////////////////////
procedure TForm1.ReadListOfSymbolInfo();
var igList, ioList, writeLenList, readLenList, retLenList : Array of Longword;
    writePtrList, readPtrList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
    nInfos : Longword;
begin
    nInfos := 15;
    SetLength(infos, nInfos);
    infos[0].name := '.nBits';
    infos[1].name := '.wBits';
    infos[2].name := '.dwBits';
    infos[3].name := '.f32AxPos';
    infos[4].name := '.f64AxPos';
    infos[5].name := '.bEnable';
    infos[6].name := '.sValue';
    infos[7].name := '.ui8Num';
    infos[8].name := '.ui16Num';
    infos[9].name := '.ui32Num';
    infos[10].name := '.i8Num';
    infos[11].name := '.i16Num';
    infos[12].name := '.i32Num';
    infos[13].name := '.aui32';
    infos[14].name := '.out8Bit';
    SetLength(igList, nInfos);
    SetLength(ioList, nInfos);
    SetLength(readLenList, nInfos);
    SetLength(readPtrList, nInfos);
    SetLength(writeLenList, nInfos);
    SetLength(writePtrList, nInfos);
    SetLength(resList, nInfos);
    SetLength(retLenList, nInfos);
    for i:=Low(infos) to High(infos) do
    begin
    igList[i]       := ADSIGRP_SYM_INFOBYNAMEEX;
    ioList[i]       := $0;
    readLenList[i]  := SizeOf(infos[i].entry);
    readPtrList[i]  := @infos[i].entry;
    writeLenList[i] := Length(infos[i].name) + 1;
    writePtrList[i] := @infos[i].name[1];
    resList[i]      := 0;
    retLenList[i]   := 0;
    end;
    adsErr := AdsSyncSumReadWriteReq(   @amsAddr, igList, ioList,
                    readLenList, readPtrList,
                    writeLenList, writePtrList,
                    resList,retLenList );
    if adsErr = 0 then
    begin
    for i:=Low(resList) to High(resList) do
    begin
        if resList[i] <> 0 then
        ShowMessageFmt('AdsSyncSumReadWriteReq(reading info "%s") error: %d [0x%X]', [infos[i].name,
 resList[i], resList[i]]);
    end
    else
    ShowMessageFmt('AdsSyncSumReadWriteReq() error: %d [0x%X]', [adsErr, adsErr]);
end;
```

Werte lesen

```
//////////////////////////////////////////////////////////////////////////
procedure TForm1.Button1Click(Sender: TObject);
begin
    ValueListEditor1.Strings.Clear();
    ReadListOfVars();
end;
```

```
//////////////////////////////////////////////////////////////////////////
procedure TForm1.ReadListOfVars();
var igList, ioList, lenList : Array of Longword;
    ptrList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
begin
```

```
    SetLength(igList, Length(infos));
    SetLength(ioList, Length(infos));
    SetLength(lenList, Length(infos));
    SetLength(ptrList, Length(infos));
    SetLength(resList, Length(infos));
    for i:=Low(infos) to High(infos) do
    begin
    igList[i] := infos[i].entry.iGroup;
    ioList[i] := infos[i].entry.iOffs;
    lenList[i] := infos[i].entry.size;
    ptrList[i] := @infos[i].data[0];
    resList[i] := 0;
    end;
    adsErr := AdsSyncSumReadReq(@amsAddr, igList, ioList, lenList, ptrList, resList );
    if adsErr = 0 then
    begin
    for i:= Low(resList) to High(resList) do
    begin
        if resList[i] <> 0 then
        ShowMessageFmt('AdsSyncSumReadReq(reading value "%s") error: %d [0x%X]', [infos[i].name, res
List[i], resList[i]]);
        DataToView( infos[i] )// Output read data
    end;
    end
    else
    ShowMessageFmt('AdsSyncSumReadReq() error: %d [0x%X]', [adsErr, adsErr]);
    Button2.Enabled := adsErr = 0;
    Button3.Enabled := adsErr = 0;
end;
```

Werte schreiben

```
////////////////////////////////////////////////////////////////////////
procedure TForm1.Button2Click(Sender: TObject);
begin
    WriteListOfVars();
end;
```

```
////////////////////////////////////////////////////////////////////////
procedure TForm1.WriteListOfVars();
var igList, ioList, lenList : Array of Longword;
    srcList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
begin
    SetLength(igList, Length(infos));
    SetLength(ioList, Length(infos));
    SetLength(lenList, Length(infos));
    SetLength(srcList, Length(infos));
    SetLength(resList, Length(infos));
    for i:= Low(infos) to High(infos) do
    begin
    try
        ViewToData( infos[i] ); // refresh value
    finally
        igList[i] := infos[i].entry.iGroup;
        ioList[i] := infos[i].entry.iOffs;
        lenList[i] := infos[i].entry.size;
        srcList[i] := @infos[i].data[0];
        resList[i] := 0;
    end;
    end;
    adsErr := AdsSyncSumWriteReq(  @amsAddr,
                    igList,
                    ioList,
                    lenList,
                    srcList,
                    resList );
    if adsErr = 0 then
    begin
    for i:= Low(resList) to High(resList) do
    begin
        if resList[i] <> 0 then
        ShowMessageFmt('AdsSyncSumWriteReq(writing value "%s") error: %d [0x%X]', [infos[i].name, re
sList[i], resList[i]]);
    end;
    end
    else
    ShowMessageFmt('AdsSyncSumWriteReq() error: %d [0x%X]', [adsErr, adsErr]);
end;
```

Werte lesen "by name"

```
////////////////////////////////////////////////////////////////////////
procedure TForm1.Button3Click(Sender: TObject);
begin
    ValueListEditor1.Strings.Clear();
    ReadListByName();
end;
```

```
////////////////////////////////////////////////////////////////////////
procedure TForm1.ReadListByName();
var igList, ioList, writeLenList, readLenList, retLenList : Array of Longword;
    writePtrList, readPtrList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
begin
    SetLength(igList, Length(infos));
    SetLength(ioList, Length(infos));
    SetLength(readLenList, Length(infos));
    SetLength(readPtrList, Length(infos));
    SetLength(writeLenList, Length(infos));
    SetLength(writePtrList, Length(infos));
    SetLength(resList, Length(infos));
    SetLength(retLenList, Length(infos));
    for i:=Low(infos) to High(infos) do
    begin
    igList[i]       := ADSIGRP_SYM_VALBYNAME;
    ioList[i]       := $0;
    readLenList[i]  := SizeOf(infos[i].data);
    readPtrList[i]  := @infos[i].data[0];
    writeLenList[i] := Length(infos[i].name) + 1;
    writePtrList[i] := @infos[i].name[1];
    resList[i]      := 0;
    retLenList[i]   := 0;
    end;
    adsErr := AdsSyncSumReadWriteReq(   @amsAddr, igList, ioList,
                    readLenList, readPtrList,
                    writeLenList, writePtrList,
                    resList,retLenList );
    if adsErr = 0 then
    begin
    for i:=Low(resList) to High(resList) do
    begin
        if resList[i] <> 0 then
        ShowMessageFmt('AdsSyncSumReadWriteReq(reading value "%s) error:
%d [0x%X]', [infos[i].name, resList[i], resList[i]]);
        DataToView( infos[i] )// Output read data
    end;
    end
    else
    ShowMessageFmt('AdsSyncSumReadWriteReq() error: %d [0x%X]', [adsErr, adsErr]);
end;
```

Einfache Proceduren um gelesenen Werte im Dialog anzeigen zu können oder die zu schreibenden Werte vom Dialog in Daten die übertragen werden müssen zu konvertieren.

```
////////////////////////////////////////////////////////////////////////
procedure TForm1.DataToView( var info : TAdsSymbolInfo );
var sValue : String;
    index : integer;
    list : TStringList;
    bPickList : Boolean;
begin
    list := TStringList.Create();
    bPickList := False;
    sValue := '';
    info.arraySize := 0;
    info.typeID := ADS_DATATYPE(info.entry.dataType);
    case info.typeID of
    ADST_INT8:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vSINT);
        sValue := Format('%d', [info.vSINT]);
        end;
    ADST_INT16:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vINT);
        sValue := Format('%d', [info.vINT]);
```

```
        end;
    ADST_INT32:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vDINT);
        sValue := Format('%d', [info.vDINT]);
        end;
    ADST_UINT8:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vBYTE);
        sValue := Format('%u', [info.vBYTE]);
        end;
    ADST_UINT16:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vWORD);
        sValue := Format('%u', [info.vWORD]);
        end;
    ADST_UINT32:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vDWORD);
        sValue := Format('%u', [info.vDWORD]);
        end;
    ADST_BIT:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vBOOL);
        sValue := Format('%s', [BoolToStr(info.vBOOL, true)]);
        list.Add('True');
        list.Add('False');
        bPickList := True;
        end;
    ADST_STRING:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vSTRING);
        sValue := Format('%s', [Trim(String(info.vSTRING))]); //
Trim = Cut characters behind of string end delimiter
        end;
    ADST_REAL32:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vREAL);
        sValue := Format('%f', [info.vREAL]);
        end;
    ADST_REAL64:
        begin
        info.arraySize := info.entry.size Div sizeof(info.vLREAL);
        sValue := Format('%f', [info.vLREAL]);
        end;
       ADST_BIGTYPE:
        sValue := 'BIGTYPE';
    else
    sValue := 'Unknown type?';
    end;
    index := ValueListEditor1.Strings.Add(Format('%s=%s', [info.name, sValue]));
    if bPickList then
    begin
    ValueListEditor1.itemProps[index].EditStyle := esPickList;
    ValueListEditor1.itemProps[index].PickList.AddStrings(list);
    end;
    list.Destroy();
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TForm1.ViewToData( var info : TAdsSymbolInfo);
var sValue : String;
begin
    try
    sValue := ValueListEditor1.Values[String(info.name)];
    case info.typeID of
        ADST_INT8:
        begin
            info.vSINT := Shortint(StrToInt(sValue));
        end;
        ADST_INT16:
        begin
            info.vINT := Smallint(StrToInt(sValue));
        end;
        ADST_INT32:
        begin
            info.vDINT := Longint(StrToInt64(sValue));
        end;
        ADST_UINT8:
        begin
```

```
            info.vBYTE := Byte(StrToInt(sValue));
        end;
        ADST_UINT16:
        begin
            info.vWORD := Word(StrToInt(sValue));
        end;
        ADST_UINT32:
        begin
            info.vDWORD := Longword(StrToInt64(sValue));
        end;
        ADST_BIT:
        begin
            info.vBOOL := StrToBool(sValue);
        end;
        ADST_STRING:
        begin
            StrPCopy(@info.vSTRING[0], AnsiString(sValue));
        end;
        ADST_REAL32:
        begin
            info.vREAL:= StrToFloat(sValue);
        end;
        ADST_REAL64:
        begin
            info.vLREAL:= StrToFloat(sValue);
        end;
       ADST_BIGTYPE:
        ;
    else
    ;
    end;
    except
     on E: EConvertError do
        ShowMessage(E.ClassName + E.Message);
    end;
end;

end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473382923.exe
Delphi 6 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466750475.exe

5.4.1.9 ADS Summenkommando: Das Holen und Freigeben von mehreren Variable-Handles mit einem ADS-Kommando.

Voraussetzungen:

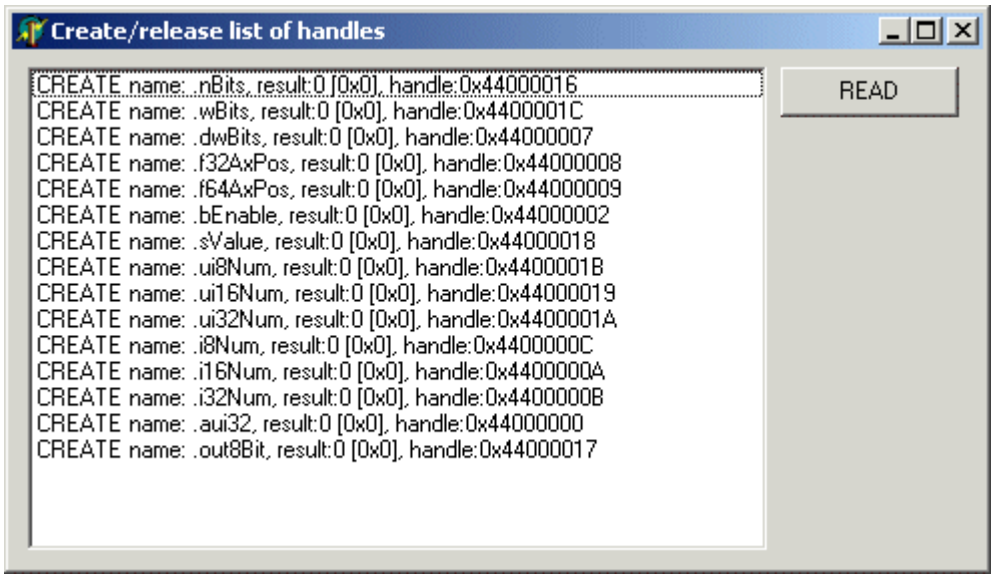
- **TwinCAT v2.11 Build >= 1550**;
- Delphi 6.0 + Update Pack 2 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selber übersetzen möchten;

Aufgabenstellung

Beim Start der einfachen Windows-Applikation sollen Handles für 15 SPS-Variablen mit einem Summenkommando geholt werden. Mit diesen Handles kann z. B. auf die Variablen lesend zugegriffen werden. Beim Beenden der Applikation sollen die Handles mit einem Summenkommando freigegeben werden.

In Delphi ist es möglich offene Arrayparameter an Funktionen zu übergeben. D.h. die Anzahl der Arrayelemente in so einem Funktionsparameter ist nicht festgelegt. In der TcAdsApi.pas-Datei wurden drei neue Funktionen implementiert: **AdsSyncSumReadReq**, **AdsSyncSumWriteReq** und

AdsSyncSumReadWriteReq. Diese Funktionen benutzen die Funktionalität der offenen Arrayparameter und vereinfachen die Benutzung des Summenkomandos unter Delphi. Das Beispielprojekt nutzt diese neuen Wrapper-Funktionen.



Das SPS-Programm

Die SPS-Variablen wurden als globale Variablen deklariert. Bei einer gestarteten SPS werden Wertänderungen der Variablen im einfachen MAIN-Programm simuliert.

```
VAR_GLOBAL
    nBits : BYTE;
    wBits : WORD;
    dwBits : DWORD;
    f32AxPos : REAL;
    f64AxPos : LREAL;
    bEnable : BOOL;
    sValue : STRING := 'INIT';
    ui8Num : USINT;
    ui16Num : UINT;
    ui32Num : UDINT;
    i8Num : SINT;
    i16Num : INT;
    i32Num : DINT;
    aui32 : ARRAY[0..9] OF DWORD;
    out8Bit AT%QX0.1 : BOOL;
END_VAR
```

Delphi 6 Programm

Die Deklarationen der benutzten TcAdsDLL-Funktionen befinden sich in den Pascal-Units *TcAdsDEF.pas und TcAdsAPI.pas*. Diese wurden über eine Uses-Klausel in das Projekt eingebunden.

```
unit Unit1;
interface
uses
  TcAdsDEF, TcAdsAPI, Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Math;
```

Beim Programmstart wird der Kommunikationsport geöffnet und beim Beenden geschlossen.

```
type TAdsHandleInfo = packed record // packed = 1 byte alignment
    name : AnsiString;// symbol name
    handle : Longword; // symbol handle
    data : Array Of Byte;// data bytes (resize the dynamic array to size of the plc variables)
end;
type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure CreateHandleList();
```

```
    procedure ReleaseHandleList();
    procedure ReadListByHandle();
    procedure Button1Click(Sender: TObject);
  private
    infos : Array of TAdsHandleInfo;
    client : Longint;
    amsAddr : TAmsAddr;
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

Handles holen

```
implementation
{$R *.dfm}
////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
var adsErr : Longint;
begin
    client := AdsPortOpen(); // open ads port
    if client <> 0 then
    begin
    adsErr := AdsGetLocalAddress( @amsAddr );
    if adsErr = 0 then
    begin
        amsAddr.port := AMSPORT_R0_PLC_RTS1; // set port to 801  (first PLC run time system)
        CreateHandleList();
    end
    else
        ShowMessageFmt('AdsGetLocalAddress() error: %d [0x%X]', [adsErr, adsErr]);
    end
    else
    ShowMessage('AdsPortOpen() failed!');
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TForm1.CreateHandleList();
var igList, ioList, writeLenList, readLenList, retLenList : Array of Longword;
    writePtrList, readPtrList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
    nInfos : Longword;
begin
    nInfos := 15;
    SetLength(infos, nInfos);
    infos[0].name := '.nBits';
    SetLength( infos[0].data, 1);//BYTE
    infos[1].name := '.wBits';
    SetLength( infos[1].data, 2);//WORD
    infos[2].name := '.dwBits';
    SetLength( infos[2].data, 4);//DWORD
    infos[3].name := '.f32AxPos';
    SetLength( infos[3].data, 4);//REAL
    infos[4].name := '.f64AxPos';
    SetLength( infos[4].data, 8);//LREAL
    infos[5].name := '.bEnable';
    SetLength( infos[5].data, 1);//BOOL
    infos[6].name := '.sValue';
    SetLength( infos[6].data, 81);//STRING(80) + 1 byte null delimiter
    infos[7].name := '.ui8Num';
    SetLength( infos[7].data, 1);//USINT
    infos[8].name := '.ui16Num';
    SetLength( infos[8].data, 2);//UINT
    infos[9].name := '.ui32Num';
    SetLength( infos[9].data, 4);//UDINT
    infos[10].name := '.i8Num';
    SetLength( infos[10].data, 1);//SINT
    infos[11].name := '.i16Num';
    SetLength( infos[11].data, 2);//INT
    infos[12].name := '.i32Num';
    SetLength( infos[12].data, 4);//DINT
    infos[13].name := '.aui32';
    SetLength( infos[13].data, 40);//ARRARY[..9] OF DWORD
    infos[14].name := '.out8Bit';
    SetLength( infos[14].data, 1);// 1 Byte
    SetLength(igList, nInfos);
    SetLength(ioList, nInfos);
```

```
    SetLength(readLenList, nInfos);
    SetLength(readPtrList, nInfos);
    SetLength(writeLenList, nInfos);
    SetLength(writePtrList, nInfos);
    SetLength(resList, nInfos);
    SetLength(retLenList, nInfos);
    for i:=Low(infos) to High(infos) do
    begin
    igList[i]       := ADSIGRP_SYM_HNDBYNAME;
    ioList[i]       := $0000000;
    readLenList[i]  := SizeOf(infos[i].handle);
    readPtrList[i]  := @infos[i].handle;
    writeLenList[i] := Length(infos[i].name) + 1;
    writePtrList[i] := @infos[i].name[1];
    resList[i]      := 0;
    retLenList[i]   := 0;
    end;
    adsErr := AdsSyncSumReadWriteReq(   @amsAddr, igList, ioList,
                    readLenList, readPtrList,
                    writeLenList, writePtrList,
                    resList,retLenList );
    // parse received data
    if adsErr = 0 then
    begin
    for i:=Low(resList) to High(resList) do
    begin
        if resList[i] <> 0 then
        ShowMessageFmt('AdsSyncSumReadWriteReq(ADSIGRP_SYM_HNDBYNAME) error:
%d [0x%X], handle:0x%X', [resList[i], resList[i], infos[i].handle]);
        ListBox1.Items.Add(Format('CREATE name: %s, result:
%d [0x%X], handle:0x%X', [infos[i].name, resList[i], resList[i], infos[i].handle] ));    end;
    end
    else
    ShowMessageFmt('AdsSyncSumReadWriteReq() error:%d [0x%X]', [adsErr, adsErr]);
end;
```

Handles freigeben

```
////////////////////////////////////////////////////////////////////////////
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    if client <> 0 then
    begin
    ReleaseHandleList();
    AdsPortClose();// close ads port
    end;
end;
```

```
////////////////////////////////////////////////////////////////////////////
procedure TForm1.ReleaseHandleList();
var igList, ioList, lenList : Array of Longword;
    srcList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
begin
    SetLength(igList, Length(infos));
    SetLength(ioList, Length(infos));
    SetLength(lenList, Length(infos));
    SetLength(srcList, Length(infos));
    SetLength(resList, Length(infos));
    for i:= Low(infos) to High(infos) do
    begin
    igList[i] := ADSIGRP_RELEASE_SYMHND;
    ioList[i] := $00000000;
    lenList[i] := SizeOf(infos[i].handle);
    srcList[i] := @infos[i].handle;
    resList[i] := 0;
    end;
    adsErr := AdsSyncSumWriteReq(  @amsAddr,
                    igList,
                    ioList,
                    lenList,
                    srcList,
                    resList );
    if adsErr = 0 then
    begin
    for i:= Low(resList) to High(resList) do
    begin
        if resList[i] = 0 then // release successfull
        infos[i].handle := 0// set to null
```

```
        else
        ShowMessageFmt('AdsSyncSumWriteReq(ADSIGRP_RELEASE_SYMHND) error: %d [0x%X], handle:0x%X', [
resList[i], resList[i], infos[i].handle]);
        ListBox1.Items.Add(Format('RELEASE name: %s, result:
%d [0x%X], handle:0x%X', [infos[i].name, resList[i], resList[i], infos[i].handle] ));
    end;
    end
    else
    ShowMessageFmt('AdsSyncSumWriteReq() error: %d [0x%X]', [adsErr, adsErr]);
end;
```

Werte lesen "by name"

```
//////////////////////////////////////////////////////////////////////////
procedure TForm1.Button1Click(Sender: TObject);
begin
    ListBox1.Items.Clear();
    ReadListByHandle();
end;
```

```
//////////////////////////////////////////////////////////////////////////
procedure TForm1.ReadListByHandle();
var igList, ioList, lenList : Array of Longword;
    destList : Array of Pointer;
    resList : Array of Longint;
    i, adsErr : Longint;
    b : Longint;
    sValue : String;
    cbValue : Longint;
begin
    SetLength(igList, Length(infos));
    SetLength(ioList, Length(infos));
    SetLength(lenList, Length(infos));
    SetLength(destList, Length(infos));
    SetLength(resList, Length(infos));
    for i:= Low(infos) to High(infos) do
    begin
    igList[i] := ADSIGRP_SYM_VALBYHND;
    ioList[i] := infos[i].handle;
    lenList[i] := Length(infos[i].data){ * SizeOf(infos[i].data[0])};
    destList[i] := @infos[i].data[0];
    resList[i] := 0;
    end;
    adsErr := AdsSyncSumReadReq(  @amsAddr,
                    igList,
                    ioList,
                    lenList,
                    destList,
                    resList );
    if adsErr = 0 then
    begin
    for i:= Low(resList) to High(resList) do
    begin
        if resList[i] = 0 then
        begin
        //TODO: save read data...
        //Convert data to hex string and show it on the form
        sValue := '';
        cbValue := Length(infos[i].data){ * SizeOf(infos[i].data[0])};
        if cbValue > 0 then
            for b:= 0 to (cbValue - 1) do
            sValue := sValue + IntToHex( infos[i].data[b], 2 ) + ' ' ;
        ListBox1.Items.Add(Format('READ name: %s, value:%s', [infos[i].name, sValue] ));
        end
        else
        ShowMessageFmt('AdsSyncSumReadReq(ADSIGRP_SYM_VALBYHND) error: %d [0x%X], handle:0x%X', [res
List[i], resList[i], infos[i].handle]);
    end;
    end
    else
    ShowMessageFmt('AdsSyncSumReadReq() error: %d [0x%X]', [adsErr, adsErr]);
end;
end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473384587.exe

Sprache / IDE	Beispielprogram auspacken
Delphi 6 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466751883.exe

5.4.1.10 Lesen und Schreiben von DT/DATE/TOD/TIME-Variablen

Voraussetzungen:

- **TwinCAT v2.11 Build >= 2034**;
- Delphi 7.0 + + Update 7.1 oder höher;
- TcAdsDLL.DLL;
- TcAdsDEF.pas und TcAdsAPI.pas, enthalten in der Datei https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466785675.zip, falls Sie den Quelltext selbst übersetzen möchten;

Aufgabenstellung

Das Datum und/oder Uhrzeit und eine Zeitdauer soll aus der Delphi-Applikation in die TwinCAT SPS geschrieben bzw. aus der TwinCAT SPS in die Delphi-Applikation eingelesen werden. Beim Schreiben der Werte in die SPS wird die in der visuellen Komponente **TDateTimePicker** eingestellte Uhrzeit und/oder Datum zuerst in das passende SPS-Format konvertiert und dann in die entsprechende SPS-Variable geschrieben. Die gelesenen Werte aus der TwinCAT SPS werden wiederum in das passenden Delphi-Format konvertiert und durch die **TDateTimePicker**-Komponente angezeigt. Die Zeitdauer soll in Millisekunden von einem TEdit control angezeigt bzw. editiert werden können.

In der Delphi-Applikation stehen unter anderem folgende Datentypen zur Verfügung:

- **TDateTime**: 64 Bit Fließkommazahl. Beinhaltet das Datum (ganzzahliger Teil) und die Uhrzeit (Bruchteil). Der ganzzahlige Anteil repräsentiert die Anzahl der vergangenen Tage seit dem 30.12.1899. Der Bruchteil gibt die Uhrzeit an (Bruchteil eines 24-Stunden Tages). Durch die Kodierung der Fließkommazahl ist die Auflösung bei sehr großen Werten kleiner (kleinere Anzahl der Nachkommastellen, die zur Verfügung stehen).
- **TDate**: 64 Bit Fließkommazahl, TDateTime-Aliastyp, beinhaltet nur das Datum (ganzzahliger Teil). Anzahl der vergangenen Tage seit dem 30.12.1899;
- **TTime**: 64 Bit Fließkommazahl, TDateTime-Aliastyp, beinhaltet nur die Uhrzeit (Bruchteil eines 24 Stunden Tages);
- **Longword**, 32 Bit Zählwert, in unserer Applikation soll er die Zeitdauer in Millisekunden Auflösung beinhalten (keine Uhrzeit);

Die Verwendung des TDateTime-Datentyps ermöglicht die Verwendung der zur Verfügung stehenden Delphi-Zeitkonvertierungsfunktionen. Insbesondere die zwei Konvertierungsfunktionen **DateTimeToUnix**() und **UnixToDateTime**(). Sie können aber stattdessen genauso einen 32 Bit Integer Wert oder einen anderen Datentyp verwenden, dort die Uhrzeit und Datum passend kodieren, konvertieren und dann in die SPS schreiben. Die unten vorgestellte Typkonvertierung/Mapping der Datentypen soll nicht als Vorschrift angesehen werden.

In der TwinCAT SPS stehen folgende Datentypen zur Verfügung:

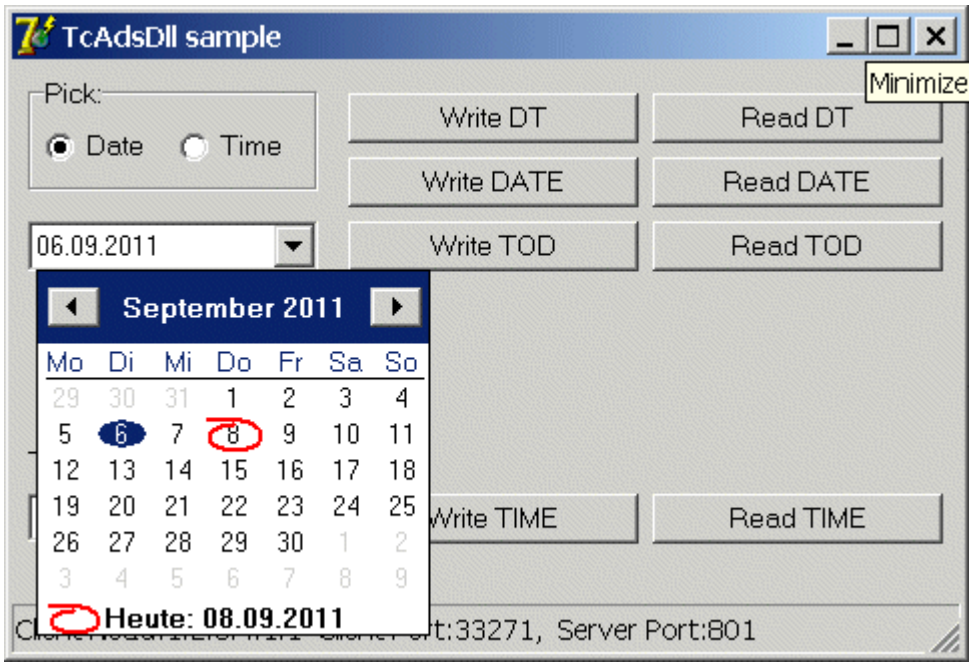
- DT bzw. DATE_AND_TIME: vorzeichenloser 32 Bit Zählwert, beinhaltet das Datum und Uhrzeit, Anzahl der vergangenen Sekunden seit dem 01.01.1970, **Auflösung in Sekunden**;
- DATE: vorzeichenloser 32 Bit Zählwert, beinhaltet nur das Datum, Anzahl der vergangenen Sekunden seit dem 01.01.1970. **Auflösung in Sekunden**;
- TOD bzw TIME_OF_DAY: vorzeichenloser 32 Bit Zählwert, beinhaltet nur die Uhrzeit. Anzahl der vergangenen Millisekunden seit dem Anfang des Tages (00:00Uhr), **Auflösung in Millisekunden**;
- TIME, vorzeichenloser 32 Bit Zählwert, beinhaltet die Zeitdauer (keine Uhrzeit), **Auflösung in Millisekunden**;

Beim Schreiben der Werte in die TwinCAT SPS wird folgende Typkonvertierung/Mapping durchgeführt:

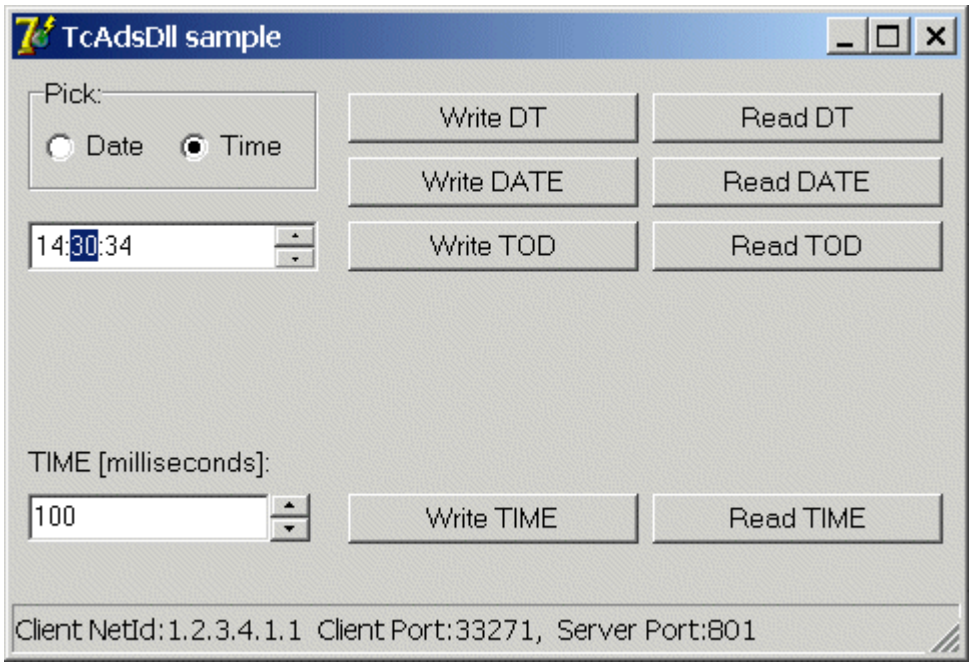
- TDateTimePicker.DateTime property (TDateTime) -> DT bzw. DATE_AND_TIME
- TDateTimePicker.Date property (TDate) -> DATE

- TDateTimePicker.Time property (TTime) -> TOD bzw. TIME_OF_DAY
- TEdit.Text property (Longword) -> TIME

Beim Lesen der Werte aus der TwinCAT SPS wird die Typkonvertierung in die umgekehrte Richtung durchgeführt.



Je nachdem, ob die Pick->Date- oder Pick->Time-Option angewählt wurde kann das Datum bzw. die Uhrzeit in der visuellen TDateTimePicker-Komponente eingestellt werden.



Die Zeitdauer in Millisekunden kann über das TEdit control im unteren Bereich eingestellt werden.

Beim Mausklick auf den entsprechenden Befehl-Button wird die Uhrzeit und/oder Datum oder die Zeitdauer in die TwinCAT-SPS geschrieben bzw. aus der SPS gelesen und angezeigt.

Das SPS-Programm

In MAIN wurde von jedem der SPS-Typen (Datum/Zeit) jeweils eine SPS-Variable deklariert. Die Delphi-Applikation soll die Werte schreiben oder lesen.

```
PROGRAM MAIN
VAR
    vDateAndTime    : DATE_AND_TIME;
    vDate       : DATE;
    vTimeOfDay      : TIME_OF_DAY;
    vTime       : TIME;
END_VAR
```

```
;
```

Delphi 7 Programm

Die Deklarationen der benutzten TcAdsDLL-Funktionen befinden sich in den Pascal-Units *TcAdsDEF.pas und TcAdsAPI.pas*. Diese wurden über eine Uses-Klausel in das Projekt eingebunden. Die Unit: DateUtils beinhaltet die benötigten Zeitkonvertierungsfunktionen und muss ebenfalls eingebunden werden.

```
unit Unit1;
interface
uses
  TcAdsDEF, TcAdsAPI, Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, Grids, Calendar, StdCtrls, DateUtils, ExtCtrls;
type
  TForm1 = class(TForm)
    Button2: TButton;
    Button3: TButton;
    StatusBar1: TStatusBar;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button9: TButton;
    Button8: TButton;
    Edit1: TEdit;
    UpDown1: TUpDown;
    RadioGroup1: TRadioGroup;
    DateTimePicker1: TDateTimePicker;
    Label2: TLabel;
    procedure Button2Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
  private
    { Private declarations }
    LocalAddr     : TAmsAddr;
    ServerAddr    : TAmsAddr;
    adsResult     : Longint;
    hDateAndTime  : Longword;
    hDate     : Longword;
    hTimeOfDay    : Longword;
    hTime     : Longword;
    function AdsCreateVarHandle( const name : AnsiString ) : Longword;
    function AdsReleaseVarHandle( hVar : Longword ) : boolean;
    function AdsWriteUI32( hVar : Longword; value : Longword ) : boolean;
    function AdsReadUI32( hVar : Longword; var value : Longword ) : boolean;
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

Aufbauen der Ads-Clientverbindung zur TwinCAT SPS

Die Delphi-Applikation baut beim Start eine Ads-Clientverbindung zur TwinCAT SPS (erstes Laufzeitsystem, Port: 801) auf und holt Handles für die 4 SPS-Variablen. Beim Beenden der Applikation werden die Handles freigegeben und zum Schluss die Ads-Clientverbindung geschlossen. Der Code zum Holen und Freigeben der Variablen-Handles wurde in zwei internen Hilfsfunktionen gekapselt: AdsCreateVarHandle() und AdsReleaseVarHandle().

```
////////////////////////////////////////////////////////////////
// Create ads client connection, create PLC variable handles
procedure TForm1.FormCreate(Sender: TObject);
var   ClientPort:Word;
begin
   StatusBar1.SimplePanel := true;
   ClientPort:= AdsPortOpen();
   if ClientPort = 0 then {error!}
      ShowMessage( 'AdsPortOpen() error!' )
   else  {OK}
   begin
      adsResult:=AdsGetLocalAddress( @LocalAddr );
      if adsResult = 0 then {OK}
      begin
     ServerAddr.netId:=LocalAddr.netId;{connect to the PLC on the local PC}
     ServerAddr.port:=801; {first RTS}
     StatusBar1.SimpleText:=Format('Client NetId:%d.%d.%d.%d.%d.%d  Client Port:%d,  Server Port:
%d',[
          LocalAddr.netId.b[0], LocalAddr.netId.b[1], LocalAddr.netId.b[2],
          LocalAddr.netId.b[3], LocalAddr.netId.b[4], LocalAddr.netId.b[5],
           ClientPort, ServerAddr.port]);
     hDateAndTime := AdsCreateVarHandle( 'MAIN.vDateAndTime' );
     hDate := AdsCreateVarHandle( 'MAIN.vDate' );
     hTimeOfDay := AdsCreateVarHandle( 'MAIN.vTimeOfDay' );
     hTime := AdsCreateVarHandle( 'MAIN.vTime' );
      end
      else
     ShowMessageFmt('AdsGetLocalAddress() error:%d', [adsResult]);
   end;
end;
```

Trennung der Ads-Clientverbindung zur TwinCAT SPS

```
////////////////////////////////////////////////////////////////
// Release PLC variable handles, close ads connection
procedure TForm1.FormDestroy(Sender: TObject);
begin
   AdsReleaseVarHandle( hDateAndTime );
   AdsReleaseVarHandle( hDate );
   AdsReleaseVarHandle( hTimeOfDay );
   AdsReleaseVarHandle( hTime );
   adsResult := AdsPortClose();
   if AdsResult > 0 then
      ShowMessageFmt('AdsPortClose() error:%d', [adsResult]);
end;
```

Handle der SPS-Variablen holen

```
////////////////////////////////////////////////////////////////
// Create PLC variable handle
function TForm1.AdsCreateVarHandle( const name : AnsiString ) : Longword;
var hVar : Longword;
begin
   adsResult := AdsSyncReadWriteReq( @serverAddr, ADSIGRP_SYM_HNDBYNAME, 0, sizeof(hVar), @hVar, Len
gth(name)+1, @name[1] );
   if adsResult <> 0 then
   begin
      ShowMessageFmt( 'AdsSyncReadWriteReq(ADSIGRP_SYM_HNDBYNAME) error: %d', [adsResult] );
      AdsCreateVarHandle := 0;
   end
   else
      AdsCreateVarHandle := hVar;
end;
```

Handle der SPS-Variablen freigeben

```
////////////////////////////////////////////////////////////////
// Release PLC variable handle
function TForm1.AdsReleaseVarHandle( hVar : Longword ) : boolean;
begin
   adsResult := AdsSyncWriteReq( @serverAddr, ADSIGRP_RELEASE_SYMHND, hVar, 0, Nil );
   if adsResult <> 0 then
      ShowMessageFmt( 'AdsSyncReadWriteReq(ADSIGRP_RELEASE_SYMHND) error: %d', [adsResult] );
   AdsReleaseVarHandle := adsResult = 0;
end;
```

Wert in die SPS schreiben (32 bit unsigned integer)

In der TwinCAT SPS werden die Zeitdatentypen intern in vorzeichenlose 32 Bit Zählwerten gehalten/ abgebildet. Nach der Konvertierung in das Zeitformat der SPS müssen die Werte als vorzeichenlose 32 Bit Zahl in die SPS geschrieben werden. Der Code zum Schreiben und Lesen der 32 Bit Werte wurde in zwei internen Hilfsfunktionen gekapselt: AdsWriteUI32() und AdsReadUI32().

```
////////////////////////////////////////////////////////////////////
// Write unsigned 32 bit integer value to the PLC
function TForm1.AdsWriteUI32( hVar : Longword; value : Longword ) : boolean;
begin
   adsResult := AdsSyncWriteReq( @ServerAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(value), @value );
   if adsResult <> 0 then
      ShowMessageFmt('AdsSyncWriteReq(ADSIGRP_SYM_VALBYHND) error:%d', [adsResult]);
   AdsWriteUI32 := adsResult = 0;
end;
```

Wert aus der SPS lesen (32 bit unsigned integer)

```
////////////////////////////////////////////////////////////////////
// Read unsigned 32 bit integer value from the PLC
function TForm1.AdsReadUI32( hVar : Longword; var value : Longword ) : boolean;
begin
   adsResult := AdsSyncReadReq( @ServerAddr, ADSIGRP_SYM_VALBYHND, hVar, sizeof(value), @value );
   if adsResult <> 0 then
      ShowMessageFmt('AdsSyncReadReq(ADSIGRP_SYM_VALBYHND) error:%d', [adsResult]);
   AdsReadUI32 := adsResult = 0;
end;
```

DATE_AND_TIME in die SPS schreiben

```
////////////////////////////////////////////////////////////////////
// Convert TDateTime -> DATE_AND_TIME
// Write DATE_AND_TIME value to the PLC
procedure TForm1.Button2Click(Sender: TObject);
var
   seconds : Longword;
   unix : Int64;
   dt : TDateTime;
begin
   // Get TDateTime value from TDateTimePicker component
   dt := DateTimePicker1.DateTime;
   // Truncate milliseconds (prevents from rounding up)
   dt := RecodeMillisecond( dt, 0 );
   // Convert TDateTime -> Unix time, 64 bit, seconds since 1.1.1970-00:00:00
   unix := DateTimeToUnix( dt );
   // Convert 64 bit -> 32 bit, seconds since 1.1.1970-00:00:00
   seconds := unix;
   // Wite DATE_AND_TIME value to the PLCAdsWriteUI32( hDateAndTime, seconds );
end;
```

DATE_AND_TIME aus der SPS lesen

```
////////////////////////////////////////////////////////////////////
// Read DATE_AND_TIME value from the PLC
// Convert DATE_AND_TIME -> TDateTime
procedure TForm1.Button3Click(Sender: TObject);
var
   seconds : Longword;
   unix : Int64;
   dt : TDateTime;
begin
   // Read DATE_AND_TIME value from PLC, 32 bit, seconds since 1.1.1970-00:00:00
   if AdsReadUI32( hDateAndTime, seconds ) then
   begin
      // Create Unix time, 64 bit, seconds since 1.1.1970-00:00:00
      unix := seconds;
      // Convert Unix time -> TDateTime
      dt := UnixToDateTime( unix );
      // Assign TDateTime value to the TDateTimePicker component
      DateTimePicker1.DateTime := dt;
   end;
end;
```

DATE in die SPS schreiben

```
////////////////////////////////////////////////////////////////////
// Convert TDate -> DATE
// Write DATE value to the PLC
procedure TForm1.Button4Click(Sender: TObject);
var
   seconds : Longword;
   unix  : Int64;
   d : TDate;
begin
   // Get TDate value from TDateTimePicker component
   d := DateOf( DateTimePicker1.DateTime );

   // Convert TDate -> Unix time, 64 bit, seconds since 1.1.1970-00:00:00
   unix := DateTimeToUnix( d );
   // Convert 64 bit -> 32 bit, seconds since 1.1.1970-00:00:00
   seconds := unix;
   // Wite DATE value to the PLCAdsWriteUI32( hDate, seconds );
end;
```

DATE aus der SPS lesen

```
////////////////////////////////////////////////////////////////////
// Read DATE value from the PLC
// Convert DATE -> TDate
procedure TForm1.Button5Click(Sender: TObject);
var
   seconds : Longword;
   unix  : Int64;
   d : TDate;
begin
   // Read DATE value from PLC, 32 bit, seconds since 1.1.1970-00:00:00
   if AdsReadUI32( hDate, seconds ) then
   begin
      // Create Unix time, 64 bit, seconds since 1.1.1970-00:00:00
      unix := seconds;
      // Convert Unix time -> TDate
      d := DateOf( UnixToDateTime( unix ) );
      // Assign TDate value to the TDateTimePicker component
      DateTimePicker1.Date := d;
   end;
end;
```

TIME_OF_DAY in die SPS schreiben

```
////////////////////////////////////////////////////////////////////
// Convert TTime -> TIME_OF_DAY
// Write TIME_OF_DAY value to the PLC
procedure TForm1.Button6Click(Sender: TObject);
var
   milliseconds : Longword;
   t : TTime;
begin
   // Get TTime value from TDateTimePicker component
   t := TimeOf(DateTimePicker1.DateTime);
   // Truncate milliseconds (prevents from rounding up)
   t := RecodeMillisecond( t, 0 );
   // Convert TTime -> Milliseconds of the day, 32 bit
   milliseconds := MillisecondOfTheDay( t );
   // Write TIME_OF_DAY value to the PLCAdsWriteUI32( hTimeOfDay, milliseconds );
end;
```

TIME_OF_DAY aus der SPS lesen

```
////////////////////////////////////////////////////////////////////
// Read TIME_OF_DAY value from the PLC
// Convert TIME_OF_DAY -> TTime
procedure TForm1.Button7Click(Sender: TObject);
var
   milliseconds : Longword;
   unix : Int64;
   t    : TTime;
begin
   // Read TIME_OF_DAY value from PLC, 32 bit, milliseconds of the day
   if AdsReadUI32( hTimeOfDay, milliseconds ) then
   begin
      // Create Unix time, 64 bit, seconds since 1.1.1970-00:00:00
      unix := milliseconds Div 1000;
```

```
      // Convert Unix time -> TTime
      t := TimeOf( UnixToDateTime( unix ) );
      // Assign TTime value to the TDateTimePicker component
      DateTimePicker1.Time := t;
   end;
end;
```

TIME (Zeitdauer) in die SPS schreiben

```
////////////////////////////////////////////////////////////////
// Write TIME value to the PLC
procedure TForm1.Button8Click(Sender: TObject);
var milliseconds : Longword;
begin
   milliseconds := UpDown1.Position;
   AdsWriteUI32( hTime, milliseconds );
end;
```

TIME (Zeitdauer) aus der SPS lesen

```
////////////////////////////////////////////////////////////////
// Read TIME value from the PLC
procedure TForm1.Button9Click(Sender: TObject);
var milliseconds : Longword;
begin
   if AdsReadUI32( hTime, milliseconds ) then
      UpDown1.Position := milliseconds;
end;
```

```
end.
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473386251.exe
Delphi 7 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466753291.exe

5.4.2 COM-Interface

5.4.2.1 Einbinden in Delphi XE2 (COM-Schnittstelle)

Die Typbibliothek der TcAdsDll.dll muss importiert werden (TcAdsDll_TLB.pas Unit). Abhängig von der Delphi-Version müssen teilweise manuelle Anpassungen gemacht werden um diese fehlerfrei übersetzen zu können.

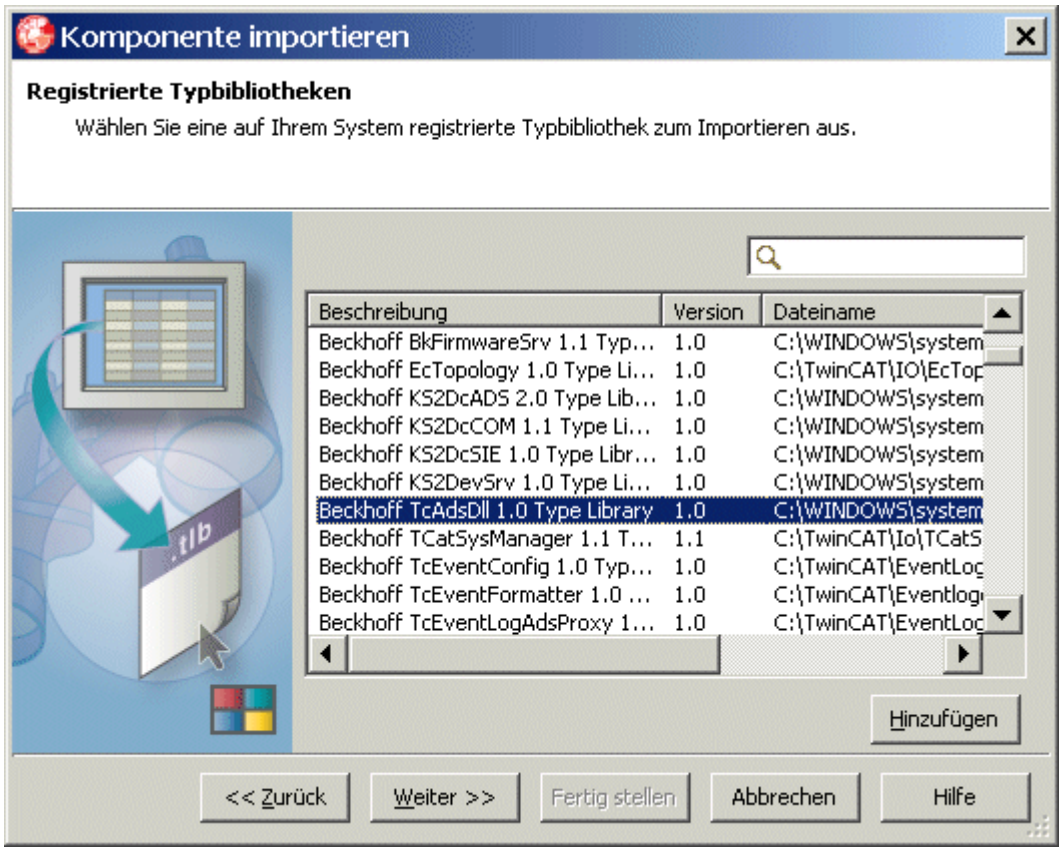
Systemvoraussetzungen

- Delphi XE2;
- TwinCAT 2.9 or higher;
- TcAdsDll.dll - dynamische Funktion-Bibliothek die sich im ‚System32'-Verzeichnis von Windows NT/2000/XP/.. befindet.

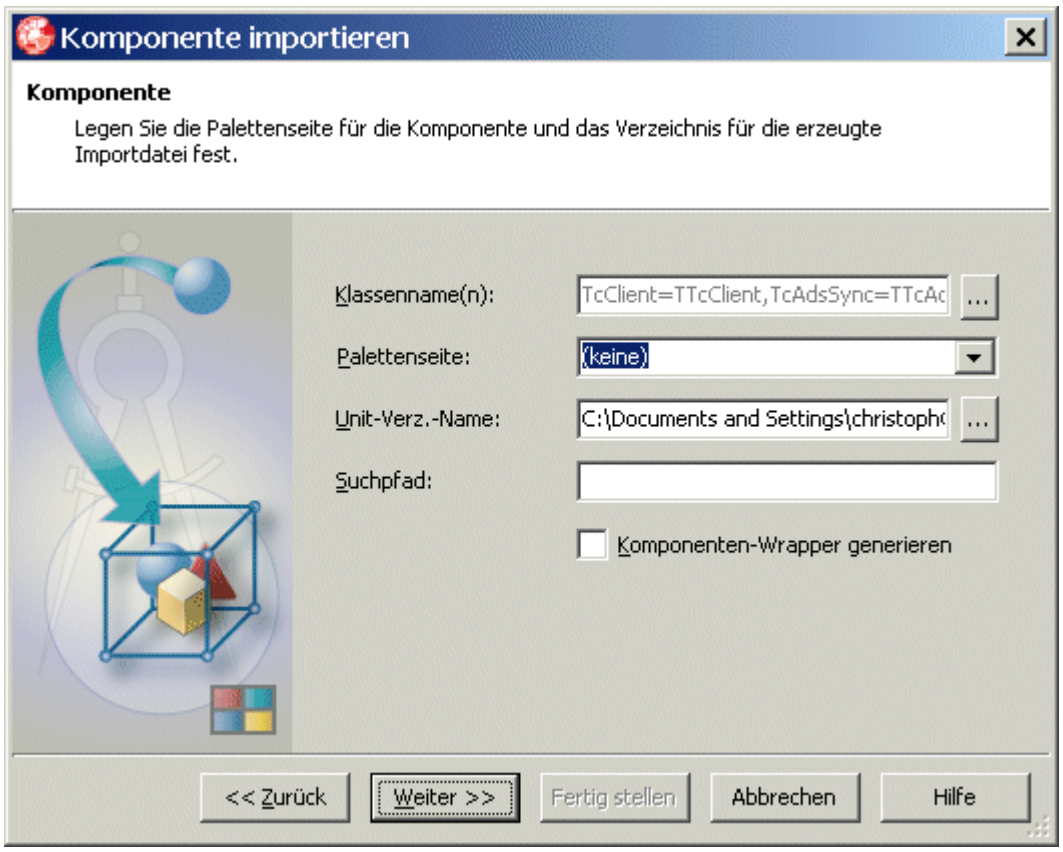
Importieren der Typbibliothek

1. Im Menue den Befehl: *Komponente -> Komponente importieren* aufrufen. Im Dialog dann die Option: *Typbibliothek importieren* anwählen und mit *Weiter* bestätigen.

2. Aus der Liste der registrierten Bibliotheken die *Beckhoff TcAdsDll x.x Type Library* auswählen und dann mit *Weiter* bestätigen.



3. Den nächsten Dialog ebenfalls mit *Weiter* bestätigen.



4. Im nächsten Dialog die Option: *Unit anlegen* auswählen und mit *Fertig stellen* bestätigen.

Die Typ-Bibliothek wird z. B. in dem Ordner **.../RAD Studio/9.0/Imports** generiert. Binden Sie bitte das Unit in ihre Applikation ein.

5.4.2.2 Einbinden in Delphi (COM-Schnittstelle)

Die Typbibliothek der TcAdsDll.dll muss importiert werden (TcAdsDll_TLB.pas Unit). Abhängig von der Delphi-Version müssen teilweise manuelle Anpassungen gemacht werden um diese fehlerfrei übersetzen zu können.

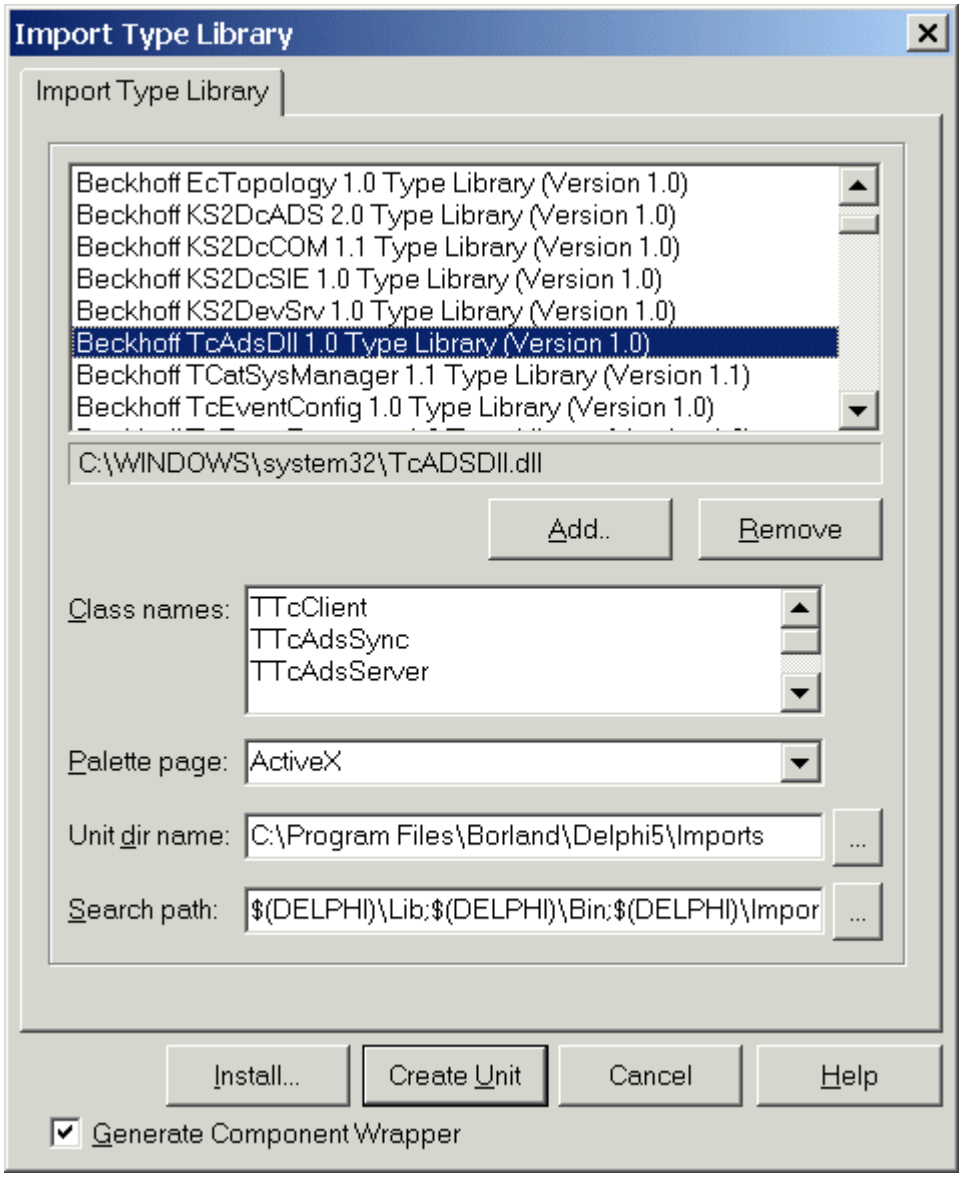
Systemvoraussetzungen

Zum Einbinden der DLL in Delphi-Programme benötigen Sie folgende Dateien:

- TcAdsDll.dll - dynamische Funktion-Bibliothek die sich im ‚System32'-Verzeichnis von Windows NT/ 2000 befindet.
- Delphi 5.0 or higher;
- TwinCAT 2.9 or higher;

Importieren der Typbibliothek

1. Über den Menübefehl: *Projekt -> Import Type Library* das Import Type Library-Dialogfenster aufrufen.



2. In dem Dialogfenster markieren Sie aus der Liste die *Beckhoff TcAdsDll x.x Type Library* und bestätigen mit einem Mausklick auf *Create Unit....*

Die Typ-Bibliothek wird z. B. in dem Ordner .../Delphi 5/Imports generiert. Binden Sie bitte das Unit in ihre Applikation ein.

5.4.2.3 Lesen und Schreiben von SPS Variablen

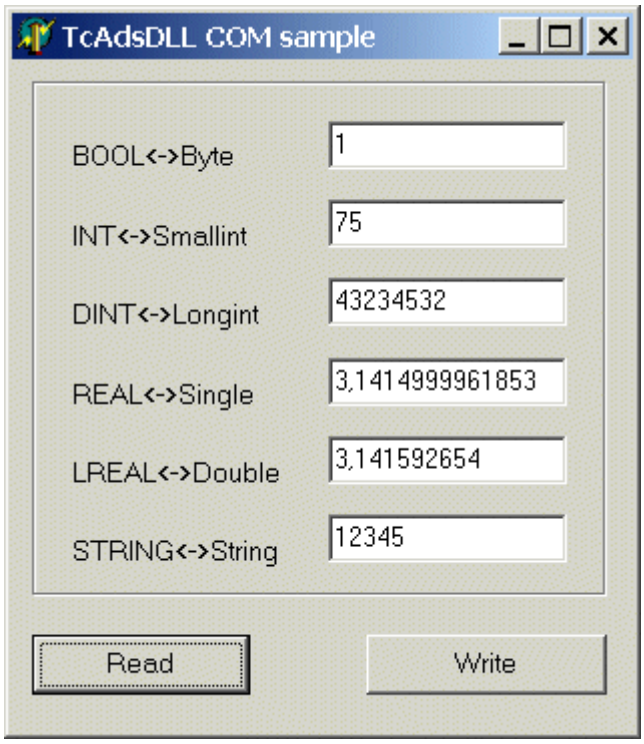
Voraussetzungen:

- Delphi 5.0 oder höher;
- TwinCAT 2.9 oder höher;
- Die Typbibliothek der TcAdsDll.dll muss importiert werden (TcAdsDll_TLB.pas Unit). Abhängig von der Delphi-Version müssen teilweise manuelle Anpassungen gemacht werden um diese fehlerfrei übersetzen zu können.

Dieses Beispiel-Programm demonstriert wie die COM-Schnittstellen der TcAdsDll von Delphi angesprochen werden können, um Variablen aus der SPS zu lesen bzw. in die SPS zu schreiben.

In der FormCreate-Ereignisfunktion wird mit *CoTcClient.Create* eine Verbindung zum Server auf dem lokalen System aufgebaut. Mit Hilfe der Methode Connect [▶ 35] wird dann eine Verbindung zum ersten Laufzeitsystem der SPS (port 801) auf dem lokalen Rechner (netid = '0.0.0.0.0.0') hergestellt.

Durch Betätigen des Read-Buttons werden die Variablen aus der SPS gelesen und beim Betätigen des Write-Buttons in die SPS geschrieben.



Das Delphi Programm

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, TcAdsDll_TLB, OleServer, ComObj;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
```

```
    Bevel1: TBevel;
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private-Deklarationen }
    ads     : ITcAdsSync;
    client  : ITcClient;
    netId   : AmsNetId;
    cbRead  : integer;
    valBool : Byte;     // PLCBoolVar
    valInt  : Smallint; // PLCIntVar
    valDint : Longint;  // PLCDIntVar
    valReal : Single;   // PLCRealVar
    valLReal : Double;  // PLCLRealVar
    valString : AnsiString;// PLCStringVar
  public
    { Public-Deklarationen }
  end;
var
  Form1: TForm1;

implementation
{$R *.DFM}
////////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
begin
    netId.b[0] := 0;  // 172
    netId.b[1] := 0;  // 16
    netId.b[2] := 0;  // 4
    netId.b[3] := 0;  // 23
    netId.b[4] := 0;  // 1
    netId.b[5] := 0;  // 1
    try
    client := CoTcClient.Create;
    try
        OleCheck(client.Connect( netid, 801, ads ));
    except
        ShowMessage('client.Connect() failed!');
    end;
    except
    ShowMessage('CoTcClient.Create() failed!');
    end;
end;
////////////////////////////////////////////////////////////////////////
procedure TForm1.Button1Click(Sender: TObject);
begin
    SetLength( valString, 256);// make sure the string >= SIZEOF(PLCStringVar) variable
    try
    OleCheck(ads.Read($4020, 0, sizeof(valBool), cbRead, PByte(@valBool)^ ));
    OleCheck(ads.Read($4020, 2, sizeof(valInt), cbRead, PByte(@valInt)^ ));
    OleCheck(ads.Read($4020, 4, sizeof(valDint), cbRead, PByte(@valDint)^ ));
    OleCheck(ads.Read($4020, 8, sizeof(valReal), cbRead, PByte(@valReal)^ ));
    OleCheck(ads.Read($4020, 12, sizeof(valLReal), cbRead, PByte(@valLReal)^ ));
    OleCheck(ads.Read($4020, 100, Length(valString) + 1, cbRead, PByte(@valString[1])^ ));
    except
    ShowMessage('ads.Read(...) failed!');
    end;
    Edit1.Text := IntToStr( valBool );
    Edit2.Text := IntToStr( valInt );
    Edit3.Text := IntToStr( valDint );
    Edit4.Text := FloatToStr( valReal );
    Edit5.Text := FloatToStr( valLReal );
    Edit6.Text := String(valString);
end;
////////////////////////////////////////////////////////////////////////
procedure TForm1.Button2Click(Sender: TObject);
begin
    valBool := StrToInt( Edit1.Text );
    valInt := StrToInt( Edit2.Text );
    valDint := StrToInt( Edit3.Text );
    valReal := StrToFloat( Edit4.Text );
    valLReal := StrToFloat( Edit5.Text );
    valString := AnsiString( Edit6.Text );
    if valString = '' then
    valString := #00;
    try
    OleCheck(ads.Write($4020, 0, sizeof(valBool), PByte(@valBool)^ ));
```

```
    OleCheck(ads.Write($4020, 2, sizeof(valInt), PByte(@valInt)^ ));
    OleCheck(ads.Write($4020, 4, sizeof(valDint), PByte(@valDint)^ ));
    OleCheck(ads.Write($4020, 8, sizeof(valReal), PByte(@valReal)^ ));
    OleCheck(ads.Write($4020, 12, sizeof(valLReal), PByte(@valLReal)^ ));
    OleCheck(ads.Write($4020, 100, Length(valString) + 1, PByte(@valString[1])^));
    except
    ShowMessage('ads.Write(...) failed!');
    end;
end;

initialization
    IsMultiThread := True;

end.
```

SPS-Programm

```
PROGRAM MAIN
VAR
  PLCBoolVar AT %MX0.0 : BOOL := TRUE;
  PLCIntVar AT %MB2 : INT := 75;
  PLCDIntVar AT %MB4 : DINT := 43234532;
  PLCRealVar AT %MB8 : REAL := 3.1415;
  PLCLRealVar AT %MB12 : LREAL := 3.141592654;
  PLCStringVar AT %MB100: STRING := '12345';
END_VAR

;
```

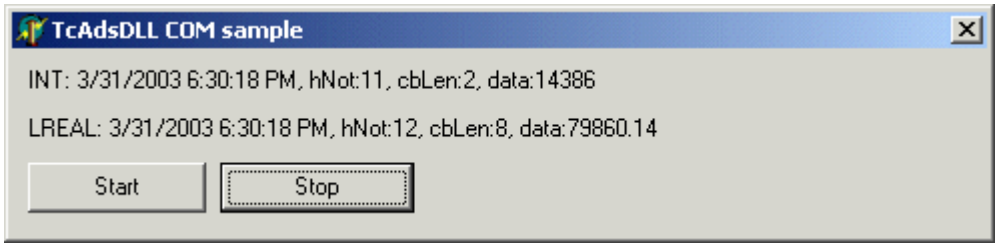
Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473519499.zip
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466742027.exe

5.4.2.4 Ereignisgesteuertes Lesen von SPS Variablen

Voraussetzungen:

- Delphi 5.0 oder höher;
- TwinCAT 2.9 oder höher;
- Die Typbibliothek der TcAdsDll.dll muss importiert werden (TcAdsDll_TLB.pas Unit). Abhängig von der Delphi-Version müssen teilweise manuelle Anpassungen gemacht werden um diese fehlerfrei übersetzen zu können.

In der FormCreate-Ereignisfunktion wird mit *CoTcClient.Create* eine Verbindung zum Server auf dem lokalen System aufgebaut. Mit Hilfe der Methode Connect [▶ 35] wird dann eine Verbindung zum ersten Laufzeitsystem der SPS (port 801) auf dem lokalen Rechner (netid = '0.0.0.0.0.0') hergestellt. Beim Betätigen des Start-Buttons wird mit der Methode AddDeviceNotification [▶ 39] eine Verbindung zu den SPS-Variablen hergestellt. Bei einer Änderung der SPS-Variablen wird von dem Server die Funktion DeviceNotification [▶ 41] der _ITcAdsSyncEvents-Schnittstelle aufgerufen. Damit Events an den Client gesendet werden können, muss diese Schnittstelle von dem Client implementiert werden. Ausserdem muss der Server über deren existenz informiert werden. Dies geschieht durch den Aufruf der InterfaceConnect-Prozedur. Beim Betätigen des Stop-Buttons werden die Verbindungen mit der Methode DelDeviceNotification [▶ 39] wieder aufgehoben.



Das Delphi Programm

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, TcAdsDll_TLB, OleServer, ComObj,activex, Math;

type
  TForm1 = class( TForm, IUnknown, _ITcAdsSyncEvents )
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    TcClient      : ITcClient;
    TcAdsSync     : ITcAdsSync;
    dwCookie      : integer;
    hInteger, hDouble   : integer;
    serverAddr      : AmsNetId;
    function  DeviceNotification(var pTime: TimeStamp; hNotification: Integer; cbLen: Integer; var p
Data: Byte): HResult; stdcall;
  end;
type
    PINT = ^Smallint;
    PLREAL= ^Double;
var
  Form1: TForm1;

implementation
{$R *.DFM}
////////////////////////////////////////////////////////////////////////
procedure TForm1.FormCreate(Sender: TObject);
begin
     serverAddr.b[0] := 0;
     serverAddr.b[1] := 0;
     serverAddr.b[2] := 0;
     serverAddr.b[3] := 0;
     serverAddr.b[4] := 0;
     serverAddr.b[5] := 0;
     Button1.Enabled := True;
     Button2.Enabled := False;

     try
    TcClient := CoTcClient.Create();
    try
        OleCheck(TcClient.Connect( serverAddr, 801,TcAdsSync));
        InterfaceConnect(TcAdsSync ,IID__ITcAdsSyncEvents,self as IUnknown, dwCookie);
    except
        ShowMessage('TcClient.Connect() failed!');
    end;
     except
    ShowMessage('CoTcClient.Create() failed!');
     end;
end;
////////////////////////////////////////////////////////////////////////
procedure TForm1.FormDestroy(Sender: TObject);
begin
    if hInteger <> 0 then
    OleCheck(TcAdsSync.DelDeviceNotification(hInteger));

    if hDouble <> 0 then
    OleCheck(TcAdsSync.DelDeviceNotification(hDouble));

     InterfaceDisconnect(TcAdsSync,IID__ITcAdsSyncEvents, dwCookie);
end;
////////////////////////////////////////////////////////////////////////
procedure TForm1.Button1Click(Sender: TObject);
begin
    hInteger := 0;
    hDouble := 0;
    try
    OleCheck(TcAdsSync.AddDeviceNotification($4020, 2, sizeof(Smallint), ADSTRANS_SERVERONCHA, 0, 10
00000, hInteger ));
```

```
    OleCheck(TcAdsSync.AddDeviceNotification($4020, 12, sizeof(Double), ADSTRANS_SERVERONCHA, 0, 100
0000, hDouble ));
    Button1.Enabled := False;
    Button2.Enabled := True;
    except
    ShowMessage('TcAdsSync.AddDeviceNotification() failed!');
    end;
end;
////////////////////////////////////////////////////////////////////////
procedure TForm1.Button2Click(Sender: TObject);
begin
    try
    OleCheck(TcAdsSync.DelDeviceNotification(hInteger));
    OleCheck(TcAdsSync.DelDeviceNotification(hDouble));
    Button1.Enabled := True;
    Button2.Enabled := False;
    except
    ShowMessage('TcAdsSync.DelDeviceNotification() failed!');
    end;
    hInteger := 0;
    hDouble := 0;
end;
////////////////////////////////////////////////////////////////////////
function  TForm1.DeviceNotification(var pTime: TimeStamp; hNotification: Integer; cbLen: Integer; va
r pData: Byte): HResult; stdcall;
var
    SystemTime, LocalTime   : _SYSTEMTIME;
    TimeZoneInformation     : _TIME_ZONE_INFORMATION;
    stamp           : TDateTime;
begin
    FileTimeToSystemTime(_FILETIME(pTime), SystemTime);
    GetTimeZoneInformation(TimeZoneInformation);
    SystemTimeToTzSpecificLocalTime(@TimeZoneInformation, SystemTime, LocalTime);
    stamp := SystemTimeToDateTime(LocalTime);

    if hNotification = hInteger then
       Label1.Caption := Format( 'PLCIntVar, %s %s, hNot:%d, cbLen:%d, data:%d',
[DateToStr(stamp), TimeToStr(stamp), hNotification, cbLen, PINT(Addr(pData))^] )
    else
       Label2.Caption := Format( 'PLCLRealVar, %s %s, hNot:%d, cbLen:%d, data:%f',
[DateToStr(stamp), TimeToStr(stamp), hNotification, cbLen, PLREAL(Addr(pData))^] );

    Result := S_OK;
end;

initialization
    IsMultiThread := True;


end.
```

SPS-Programm

```
PROGRAM MAIN
VAR
  PLCBoolVar AT %MX0.0 : BOOL := TRUE;
  PLCIntVar AT %MB2 : INT := 75;
  PLCDIntVar AT %MB4 : DINT := 43234532;
  PLCRealVar AT %MB8 : REAL := 3.1415;
  PLCLRealVar AT %MB12 : LREAL := 3.141592654;
END_VAR
```

```
PLCLRealVar := PLCLRealVar +1;
PLCIntVar :=PLCIntVar +1;
```

Sprache / IDE	Beispielprogram auspacken
Delphi XE2	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473521803.zip
Delphi 5 oder höher (classic)	https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12466743435.exe

5.5 Beispiele: NI LabVIEW

5.5.1 Einbinden in LabVIEW™

Nutzen Sie das TwinCAT 3 Interface for LabVIEW™

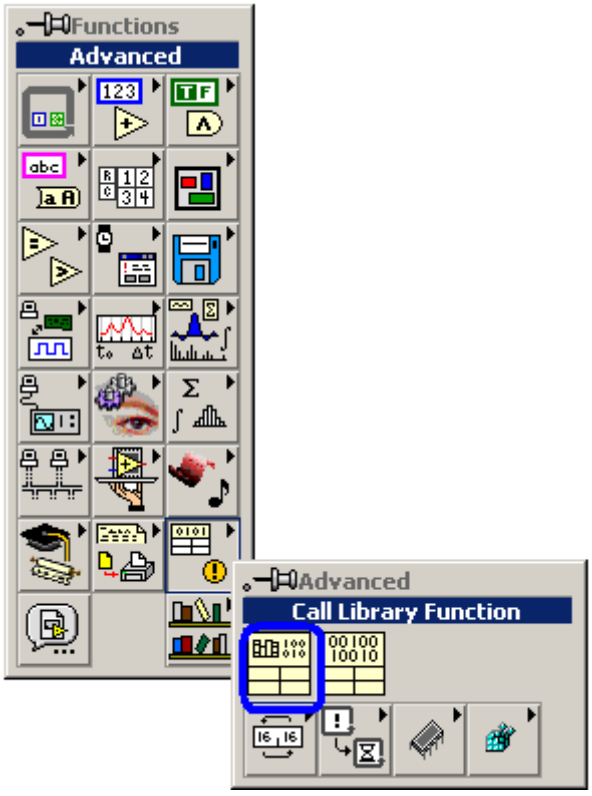
Wenn Sie eine ADS-Kommunikation zwischen LabVIEW™ und der TwinCAT 3 Laufzeit aufbauen möchten, nutzen Sie in jedem Fall das umfänglich supportete und dokumentierte Produkt TwinCAT 3 Interface for LabVIEW™, siehe TF3710. Die im Folgenden dargestellte händische Einbindung von kostenfreien ADS-Komponenten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

Notwendige Dateien

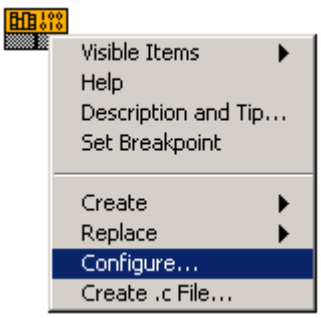
- TcAdsDll.dll - Dynamische Funktions-Bibliothek

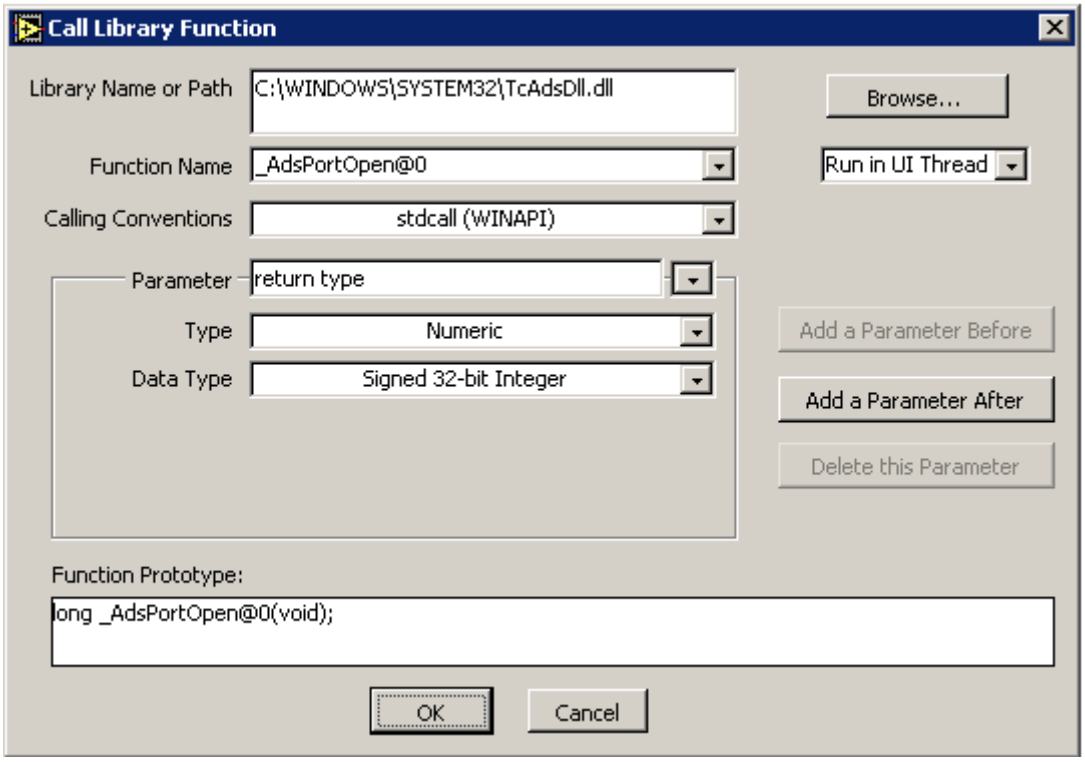
Die TcAdsDll.dll befindet sich im ‚System32'-Verzeichnis von Windows.

LabVIEW™-Funktion: Call Library Function



Konfiguration des Dll-Aufrufs





Library Name or Path

Über den Button "Browse..." oder direkt wird der Pfad der TcAdsDll eingetragen.

Function Name

Nach der Auswahl der TcAdsDll lässt sich die aufzurufende Dll-Funktion auswählen.

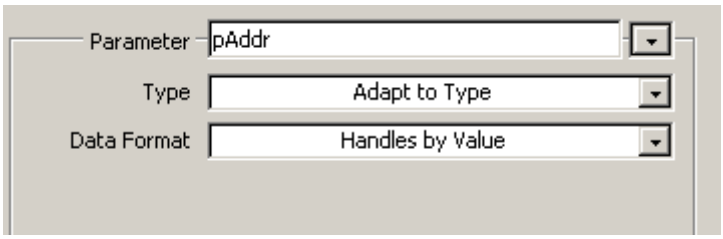
Calling Conventions

Es wird die Aufrufkonvention stdCall verwendet.

Parameter

Die Parameterliste muss an die aufzurufende Funktion angepasst werden. Über die Buttons 'Add a Parameter Before', 'Add a Parameter After' und 'Delete this Parameter' wird die Parameterliste des Aufrufs vervollständigt. Anschließend werden die Datentypen der Aufrufparameter angepasst. Bei Pointer-Übergabe von Strukturen wird der Typ 'Adapt to Type' und Format 'Handles by Value' ausgewählt.

Beispiel: Parameter pAddr der Funktion AdsSyncReadReq



Function Prototype

Der Funktionsprototyp der Aufgerufenen Funktion muss mit dem Funktionsprototypen der TcAdsDll übereinstimmen. Die Funktionsprototypen sind in der Dokumentation der TcAdsDll-Funktionen aufgeführt.

Beispiel für die Funktion **AdsSyncReadReq**

C Deklaration:

```
LONG AdsSyncReadReq(
  PAmsAddr pAddr,
  ULONGnIndexGroup,
  ULONGnIndexOffset,
  ULONGnLength,
  PVOIDpData);
```

LabVIEW™:

```
Function Prototype:
long _AdsSyncReadReq@20(void *pAddr, unsigned long nIndexGroup, unsigned long nIndexOffset,
unsigned long nLength, void *pData);
```

Strukturen

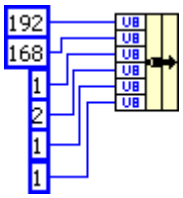
Für alle bei den Funktionsaufrufen verwendeten Strukturen werden im LabVIEW™ entsprechende Cluster gebildet.

AmsNetId

C Deklaration:

```
typedef struct {
  UCHARb[6];
} AmsNetId, *PAmsNetId;
```

Wird in LabVIEW™ als Cluster aus 6 x U8 Werten gebildet:

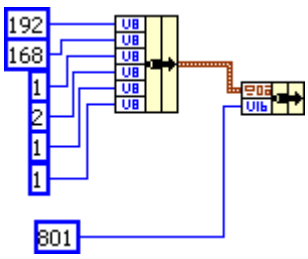


AmsAdr

C: Deklaration

```
typedef struct {
  AmsNetId netId;
  USHORTport;
} AmsAddr, *PAmsAddr;
```

Wird in LabVIEW™ als Cluster aus AmsNetId und einem U16 Wert gebildet:



Aus dem so gebildeten Cluster kann natürlich auch ein LabVIEW™-Control, z. B. für Adresseingaben gebildet werden:

Parameterübergabe

Bei der Parameterübergabe ist es sehr wichtig, dass an den Dll-Aufruf für alle Parameter Variablen übergeben werden. Insbesondere an die in der Dokumentation mit [out] gekennzeichneten Parameter müssen unbedingt entsprechende Platzhaltervariablen übergeben werden, damit entsprechend Speicher reserviert wird.

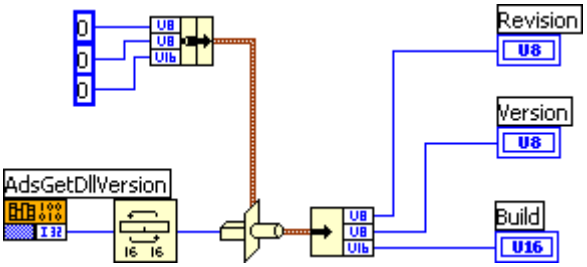
HINWEIS

Systemabsturz

Falsche Parameterübergabe verursacht Speicherschutzverletzungen und kann das LabVIEW™-System zum Absturz bringen!

5.5.2 DLL-Version auslesen

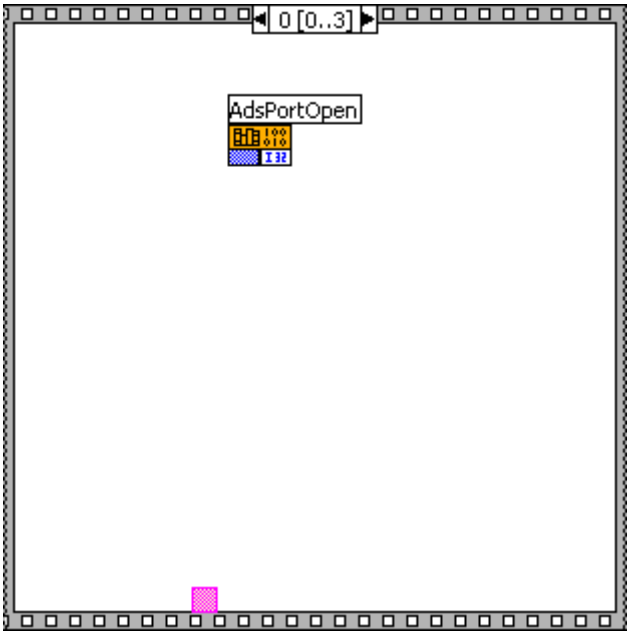
Dieses Programm ermittelt die Version der DLL-Datei.

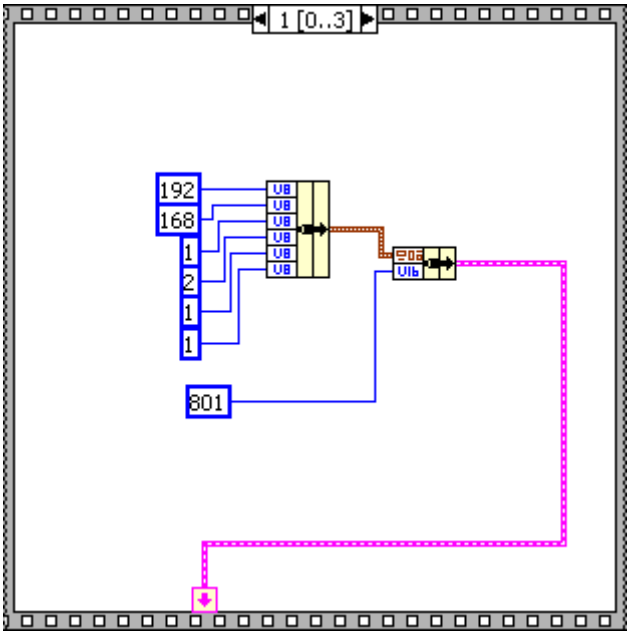


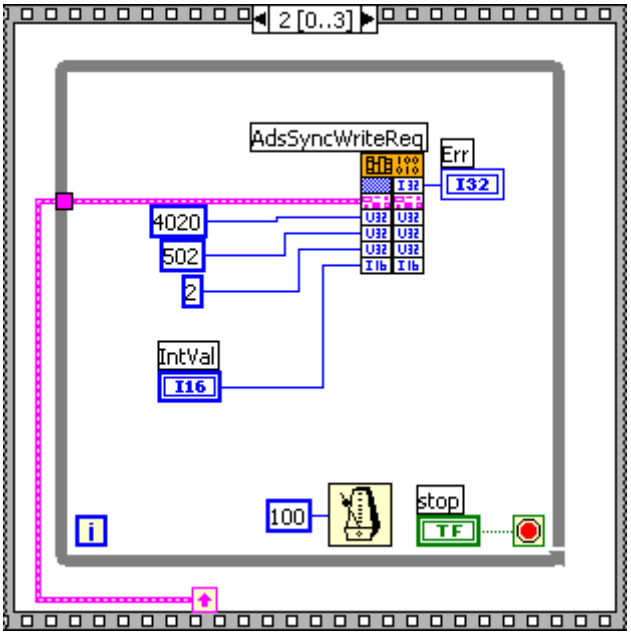
Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967679627.zip' entpacken.

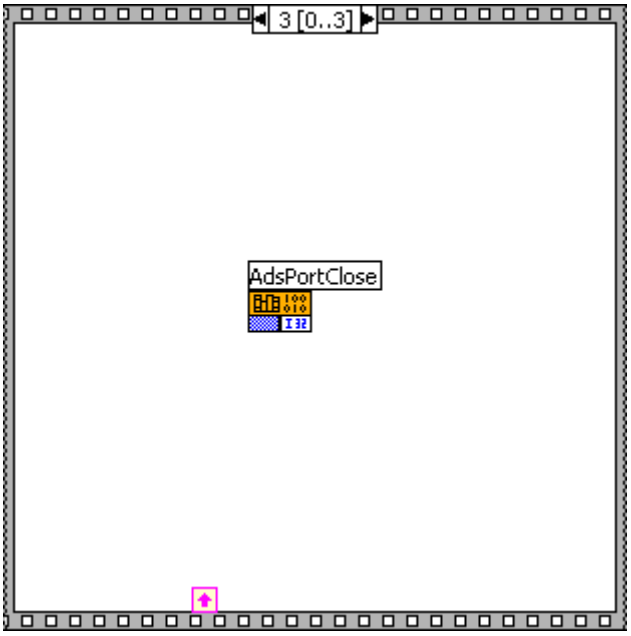
5.5.3 Merker synchron in die SPS schreiben

Bei diesem Beispielprogramm wird der Wert, den der Bediener eingegeben hat, in das Merkerwort 502 geschrieben.


0 [0..3]
AdsPortOpen


1 [0..3]
192
168
1
2
1
1
801

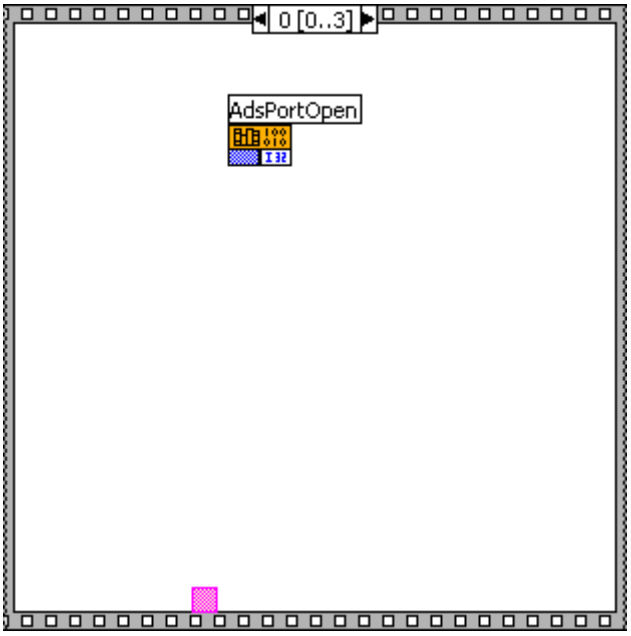


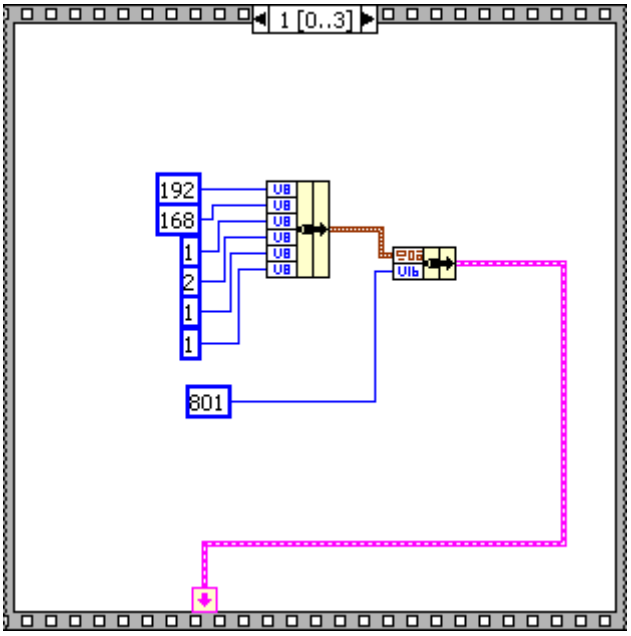


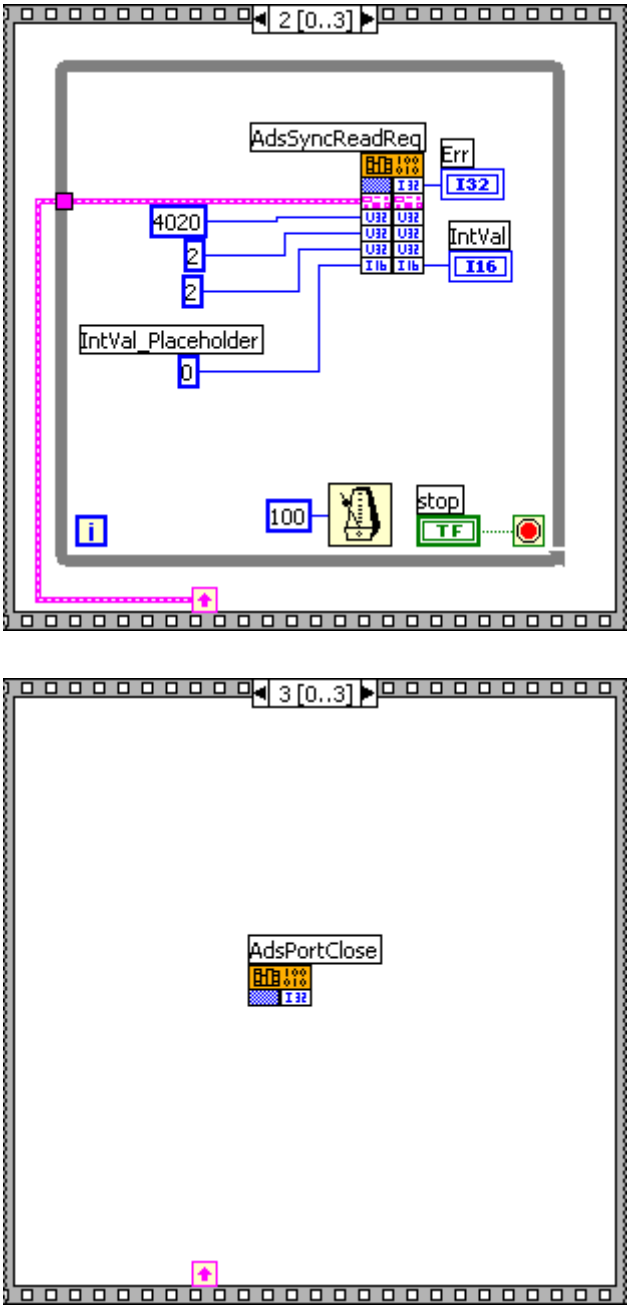
Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967681035.zip' entpacken.

5.5.4 Merker synchron aus der SPS lesen

Bei diesem Beispielprogramm wird der Wert aus dem Merkerwort 2 der SPS ausgelesen und am Bildschirm angezeigt.


0 [0..3]
AdsPortOpen

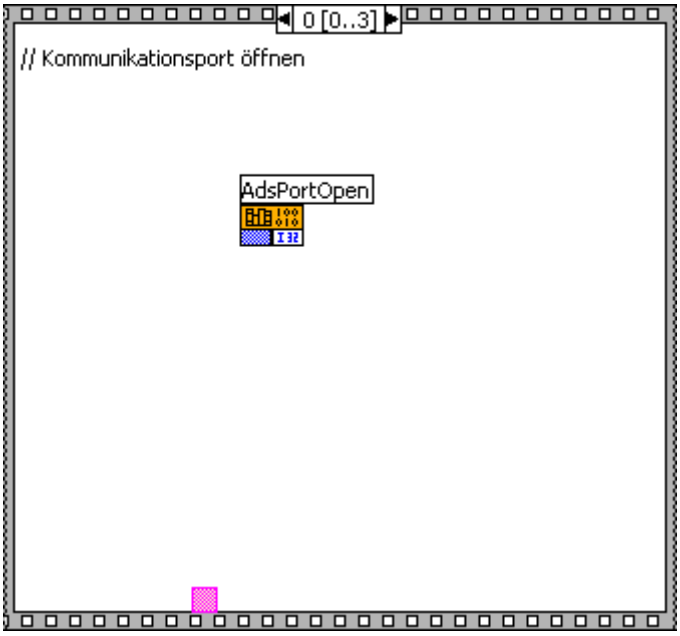

1 [0..3]
192
168
1
2
1
1
801

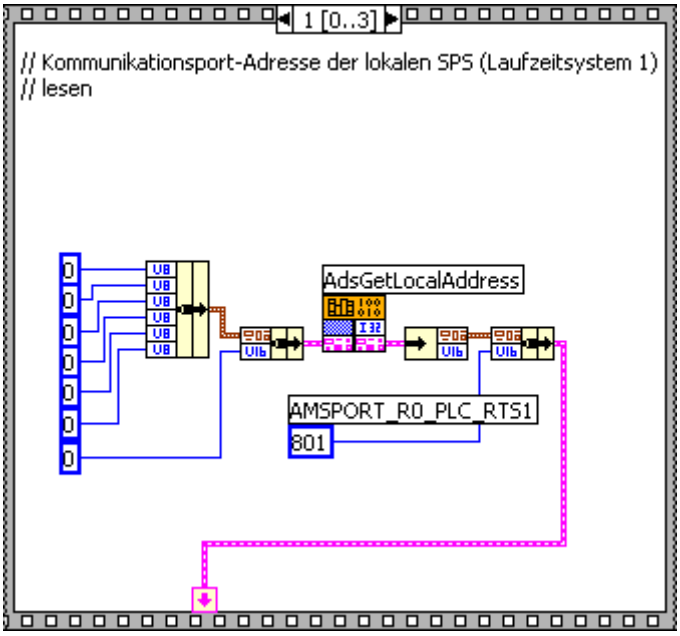


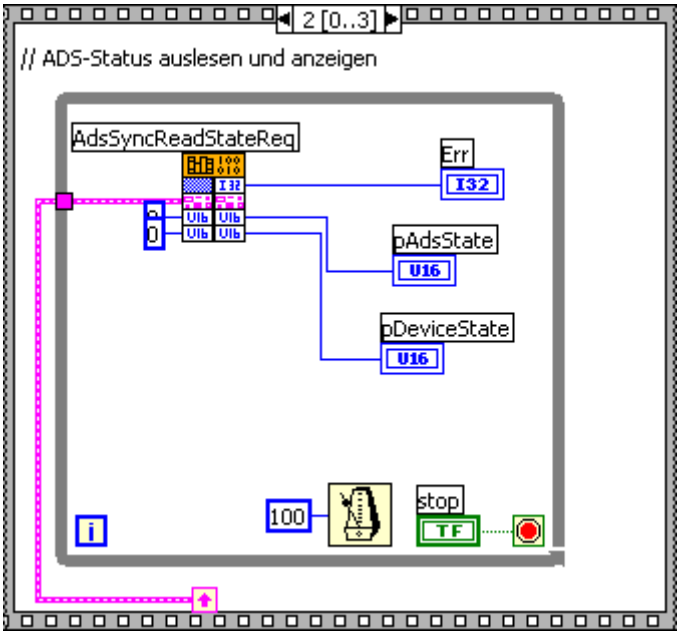
Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967682443.zip' entpacken.

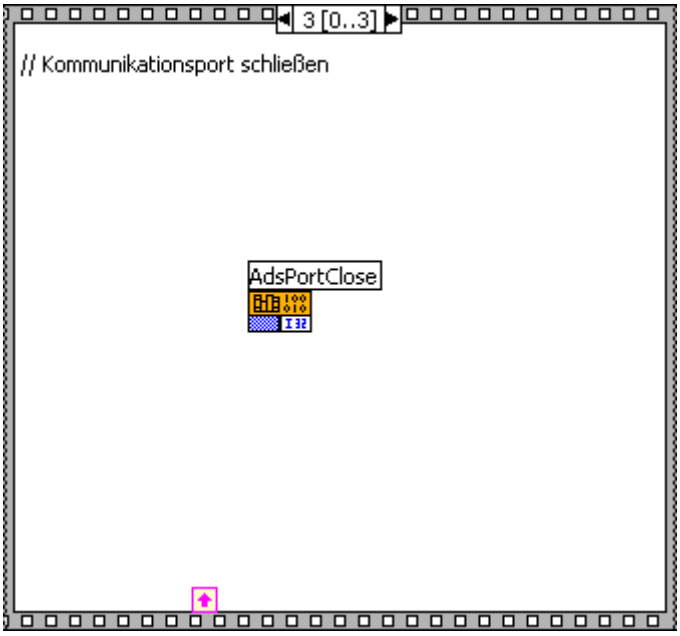
5.5.5 ADS-Status auslesen

Dieses Programm ließ den Status der SPS aus. Die Variable vom Typ ADSSTATE enthält Informationen, ob sich z.B. die SPS im RUN oder STOP Zustand befindet.


0 [0..3]
// Kommunikationsport öffnen
AdsPortOpen


1 [0..3]
// Kommunikationsport-Adresse der lokalen SPS (Laufzeitsystem 1)
// lesen
AdsGetLocalAddress
AMSPORT_R0_PLC_RTS1
801

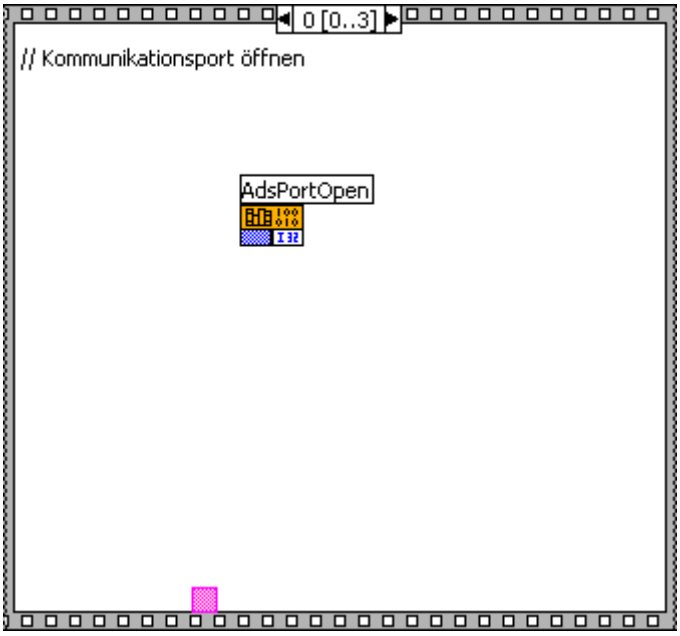


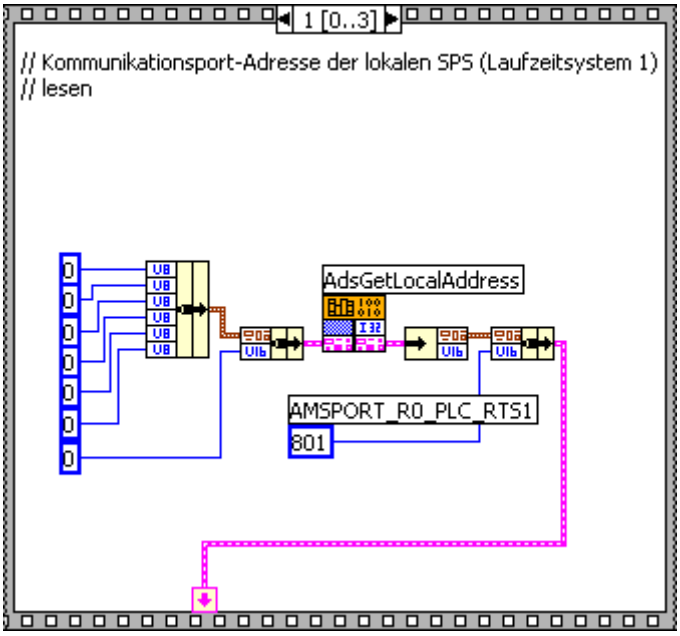


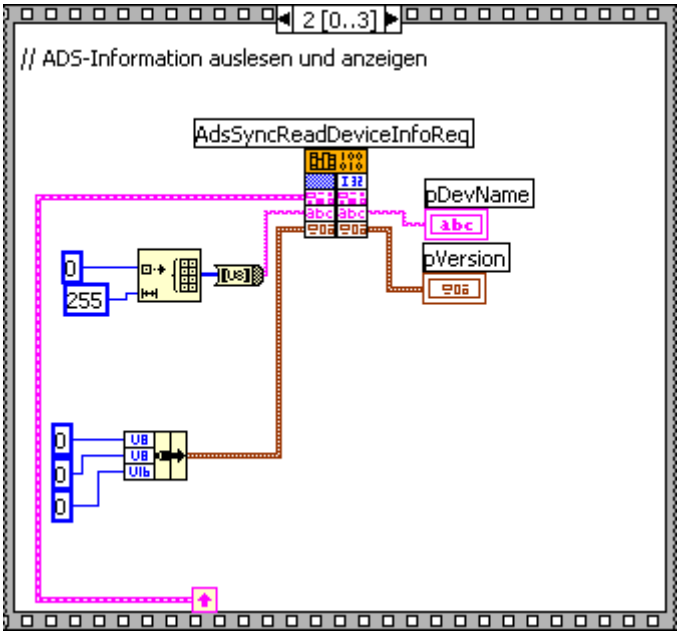
Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967683851.zip' entpacken.

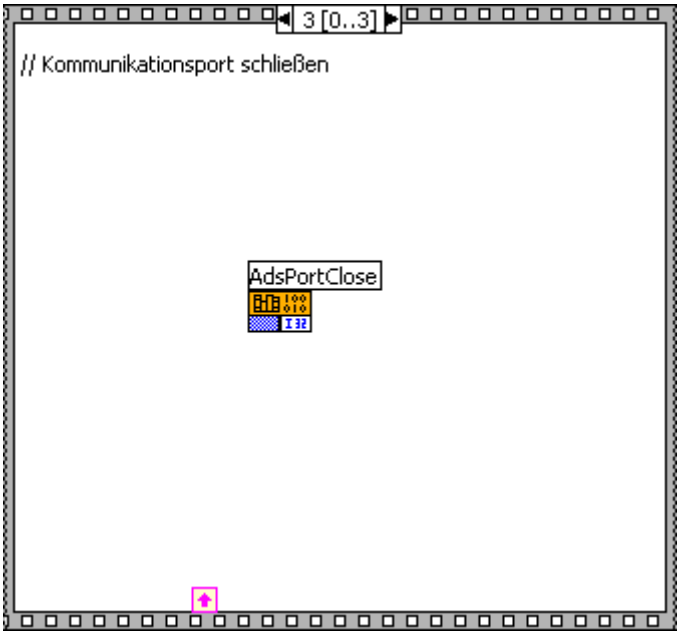
5.5.6 ADS-Informationen auslesen

Jedes ADS-Gerät enthält eine Versionsnummer und eine Bezeichnung. Das Beispielprogramm liest diese Informationen der SPS aus und zeigt sie am Bildschirm an.


0 [0..3]
// Kommunikationsport öffnen
AdsPortOpen


1 [0..3]
// Kommunikationsport-Adresse der lokalen SPS (Laufzeitsystem 1)
// lesen
AdsGetLocalAddress
AMSPORT_R0_PLC_RTS1
801

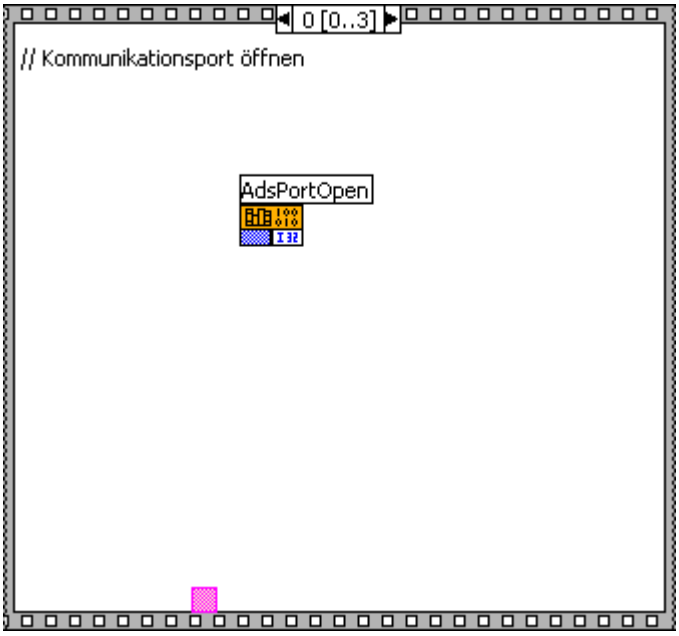


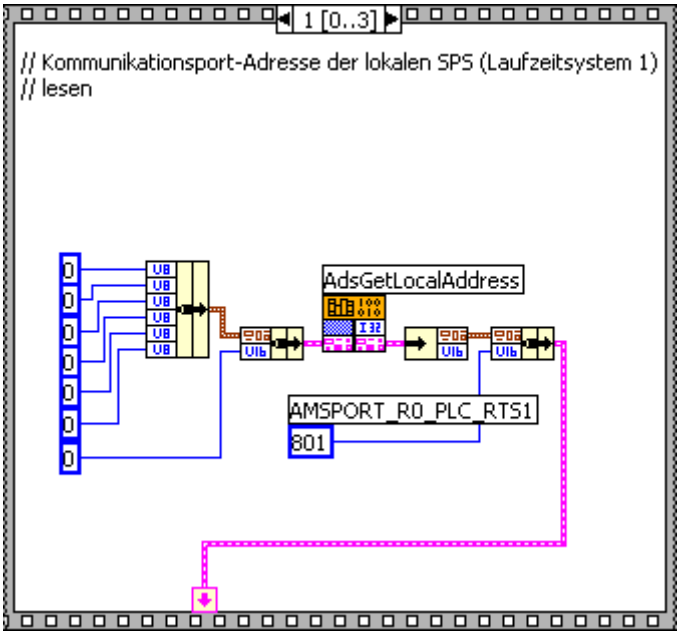


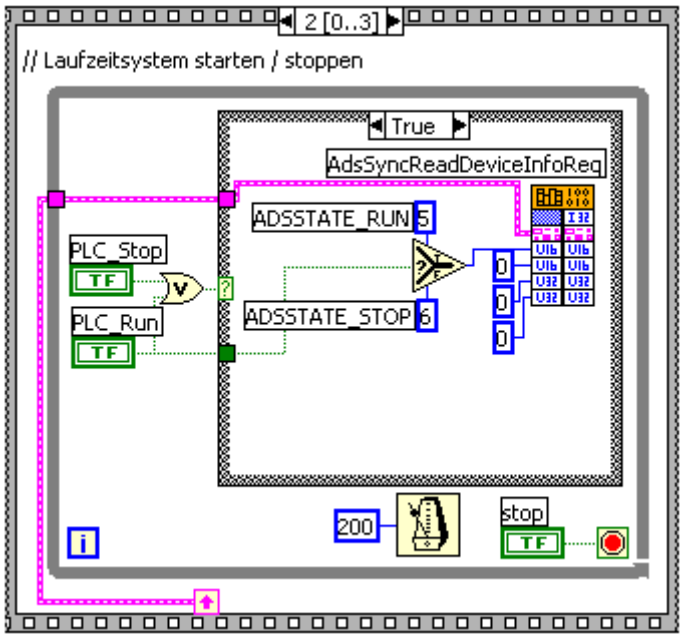
Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967685259.zip' entpacken.

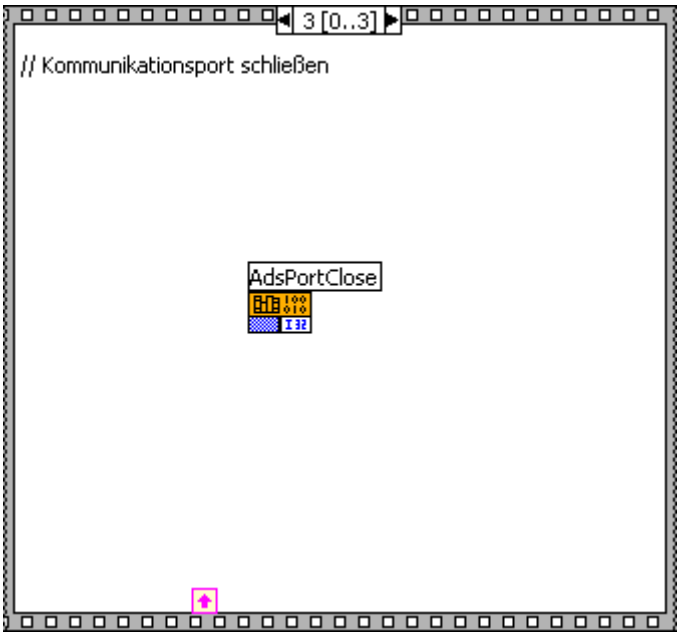
5.5.7 SPS starten/stoppen

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


0 [0..3]
// Kommunikationsport öffnen
AdsPortOpen


1 [0..3]
// Kommunikationsport-Adresse der lokalen SPS (Laufzeitsystem 1)
// lesen
AdsGetLocalAddress
AMSPORT_R0_PLC_RTS1
801

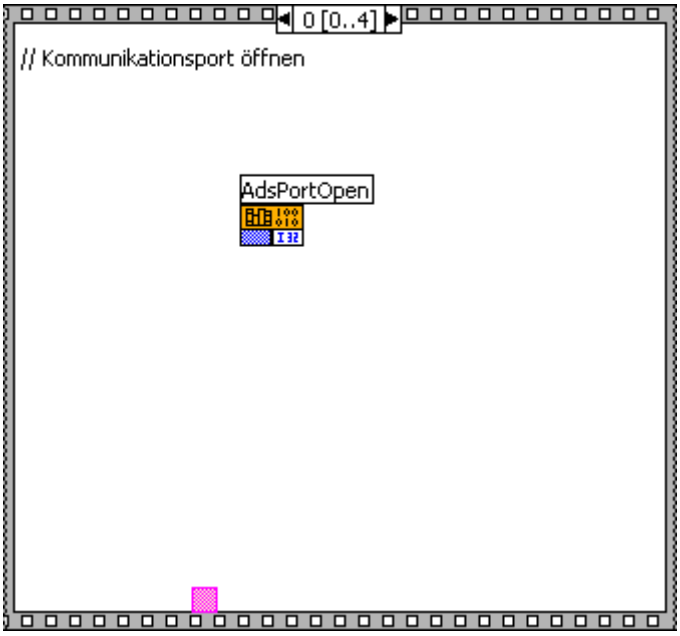


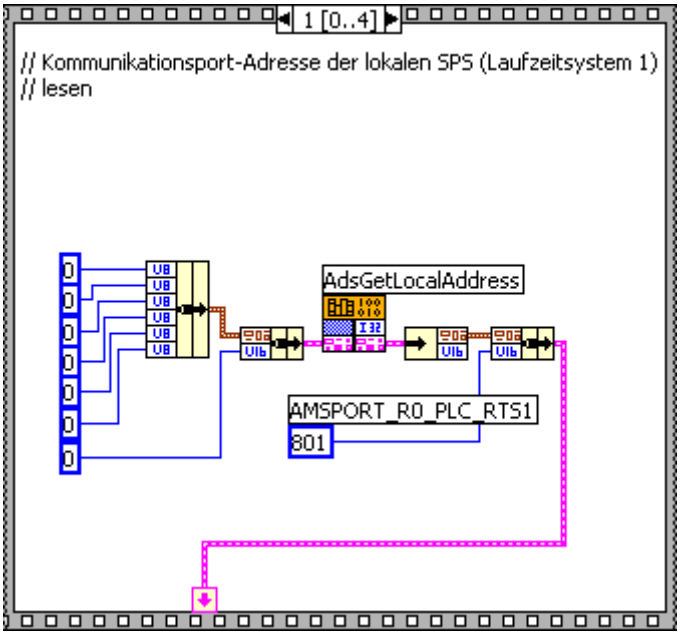


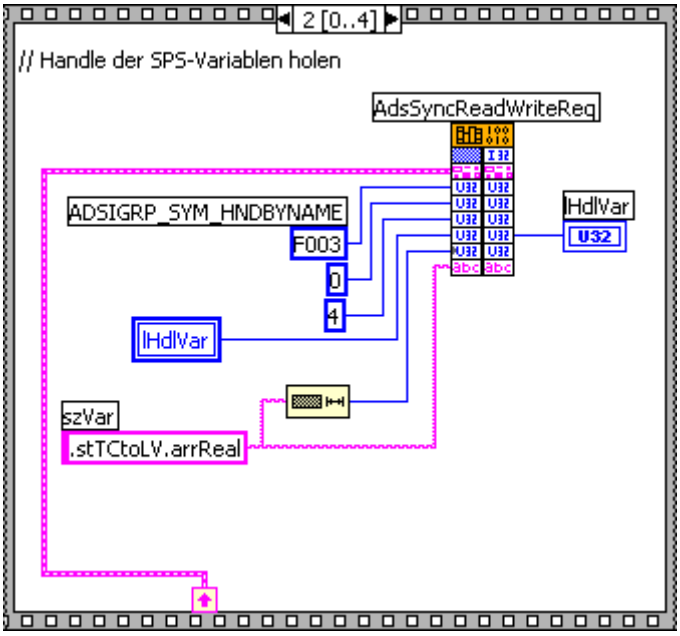
Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967686667.zip' entpacken.

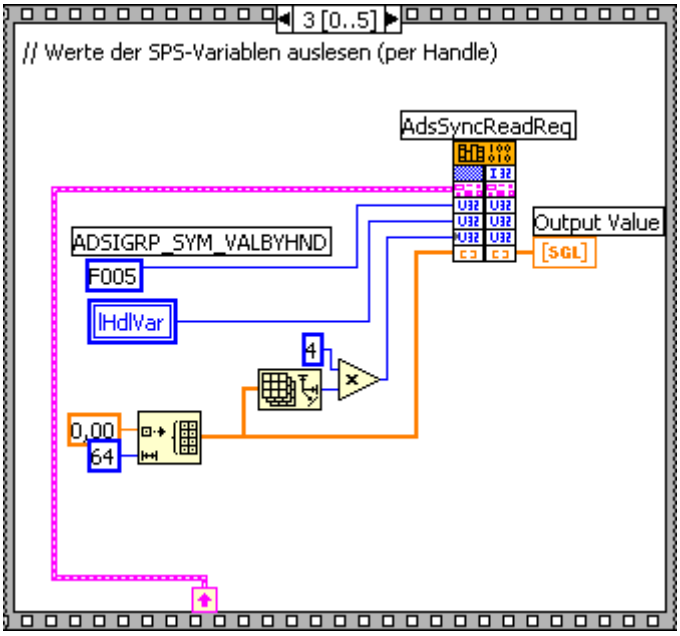
5.5.8 Zugriff per Name auf ein Array in der SPS

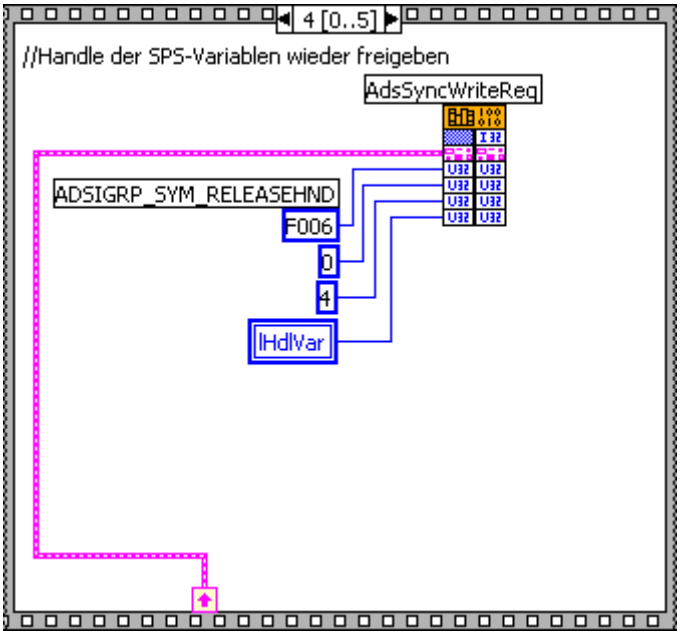
Ein Array, welches sich in der SPS befindet, soll mit einen Lesebefehl ausgelesen werden. Die Variable wird hierbei per Variablenname angesprochen. Die Vorgehensweise weicht nur geringfügig von dem Auslesen diskreter Variablen ab. Bei der Funktion AdsSyncReadReq() wird als Länge die Länge des gesamten Arrays angegeben.

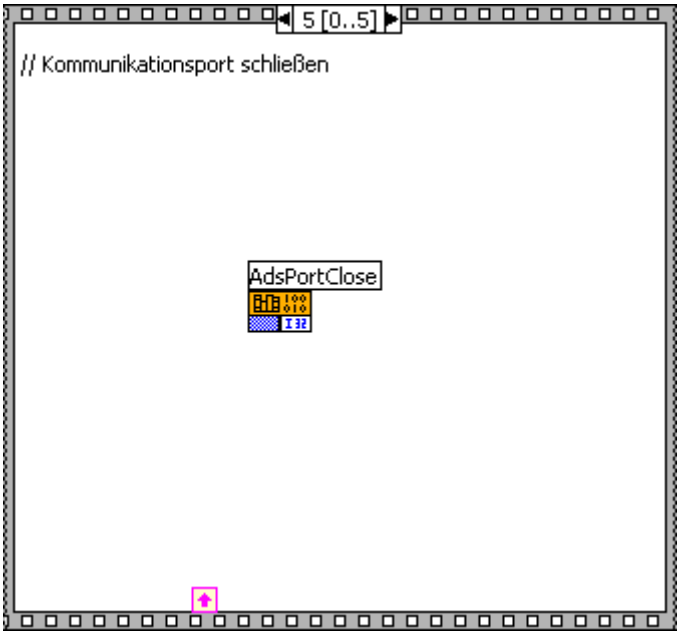

0 [0..4]
// Kommunikationsport öffnen
AdsPortOpen


1 [0..4]
// Kommunikationsport-Adresse der lokalen SPS (Laufzeitsystem 1)
// lesen
AdsGetLocalAddress
AMSPORT_R0_PLC_RTS1
801


2 [0..4]
// Handle der SPS-Variablen holen
AdsSyncReadWriteReq
ADSIGRP_SYM_HNDBYNAME
F003
0
4
lHdlVar
lHdlVar
U32
szVar
.stTCtoLV.arrReal


3 [0..5]
// Werte der SPS-Variablen auslesen (per Handle)
AdsSyncReadReq
ADSIGRP_SYM_VALBYHND
F005
lHdlVar
Output Value
SGL
4
0,00
64





Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/11967688075.zip' entpacken.

5.6 Beispiele: NI Measurement Studio

Um die TcAdsDll.dll im Measurement Studio zu verwenden, muss die Header-Datei def.h eingebunden werden. Diese Datei enthält fehlende Definitionen und die Struktur __int64, die vom Measurement Studio nicht unterstützt werden. Zur Demonstration sind hier die Notwendigen Änderungen an einem kleinen Beispiel dargestellt. Als Vorlage wurde das Beispiel Ereignisgesteuertes Lesen [▶ 51] verwendet.

Header-Datei def.h:

```
#define __inline
#define _declspec __declspec

typedef struct
{
   unsigned long LowPart;
   unsigned long HighPart;
} __int64;
```

Beispiel-Progamm:

```
#include <ansi_c.h>
#include <windows.h>
#include <winbase.h>
#include <utility.h>
#include "def.h"

#include "c:\twincat\adsapi\tcadsdll\include\tcadsdef.h"
#include "c:\twincat\adsapi\tcadsdll\include\tcadsapi.h"

static void __stdcall NotificationCallback(AmsAddr*pAddr,
AdsNotificationHeader* pNotification, unsigned long hUser)
{
   SYSTEMTIME SystemTime, LocalTime;
   FILETIME FileTime;
   TIME_ZONE_INFORMATION TimeZoneInformation;

   FileTime.dwLowDateTime = pNotification->nTimeStamp.LowPart;
   FileTime.dwHighDateTime = pNotification->nTimeStamp.HighPart;
   FileTimeToSystemTime(&FileTime, &SystemTime);

   // Zeit umrechnen in lokaler Zeiteinheit
   GetTimeZoneInformation(&TimeZoneInformation);
   SystemTimeToTzSpecificLocalTime(&TimeZoneInformation, &SystemTime, &LocalTime);

   // Zeitstempel ausgeben
   printf("%i:%i:%i.%i den: %i.%i.
%i\n",LocalTime.wHour, LocalTime.wMinute, LocalTime.wSecond, LocalTime.wSecond, LocalTime.wDay, Loca
lTime.wMonth, LocalTime.wYear);

   printf("NotificationCallback: nValue = %i\n", *(int *)pNotification->data);
}


int main(int argc, char *argv[])
{
   AdsNotificationAttrib adsNotificationAttrib;
   int nValue = 0;
   int nErr = ADSERR_NOERR;
   int port = AdsPortOpen();
   int hNotification;
   unsigned long hUser;

   AmsNetId netId = {172,16,8,56,1,1};
   AmsAddr addr;
   addr.netId = netId;
   addr.port = 801;

   nErr = AdsSyncReadReq( &addr, 0x4020, 0x0,
sizeof(nValue), &nValue);

   if( nErr != ADSERR_NOERR )
      printf("ErrorAdsSyncRead:%i\n", nErr);
   else
      printf("nValue = %i\n",nValue);

   adsNotificationAttrib.cbLength = 4;
   adsNotificationAttrib.nTransMode = ADSTRANS_SERVERONCHA;
   adsNotificationAttrib.nMaxDelay = 20000000; // 2sec
   adsNotificationAttrib.nCycleTime = 10000000; // 1sec

   hUser = 3474573467;

   nErr = AdsSyncAddDeviceNotificationReq( &addr, 0x4020, 0x0, &adsNotificationAttrib,
   NotificationCallback, hUser, &hNotification);

   if( nErr != ADSERR_NOERR )
   {
```

```
      printf("ErrorAdsSyncAddDeviceNotificationReq:%i\n", nErr);
      return 0;
   }


   while(!KeyHit());


   nErr = AdsSyncDelDeviceNotificationReq( &addr,
hNotification);

   if( nErr != ADSERR_NOERR )

      printf("Error
AdsSyncDelDeviceNotificationReq:%i\n", nErr);


   AdsPortClose();
   return 0;
}
```

Beispielprogramm 'https://infosys.beckhoff.com/content/1031/tcadsdll2/Resources/12473771403.exe' entpacken.

6 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 172]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 172]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 173]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 174]... (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.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

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.

Hex	Dec	HRESULT	Name	Beschreibung
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
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.com
www.beckhoff.com