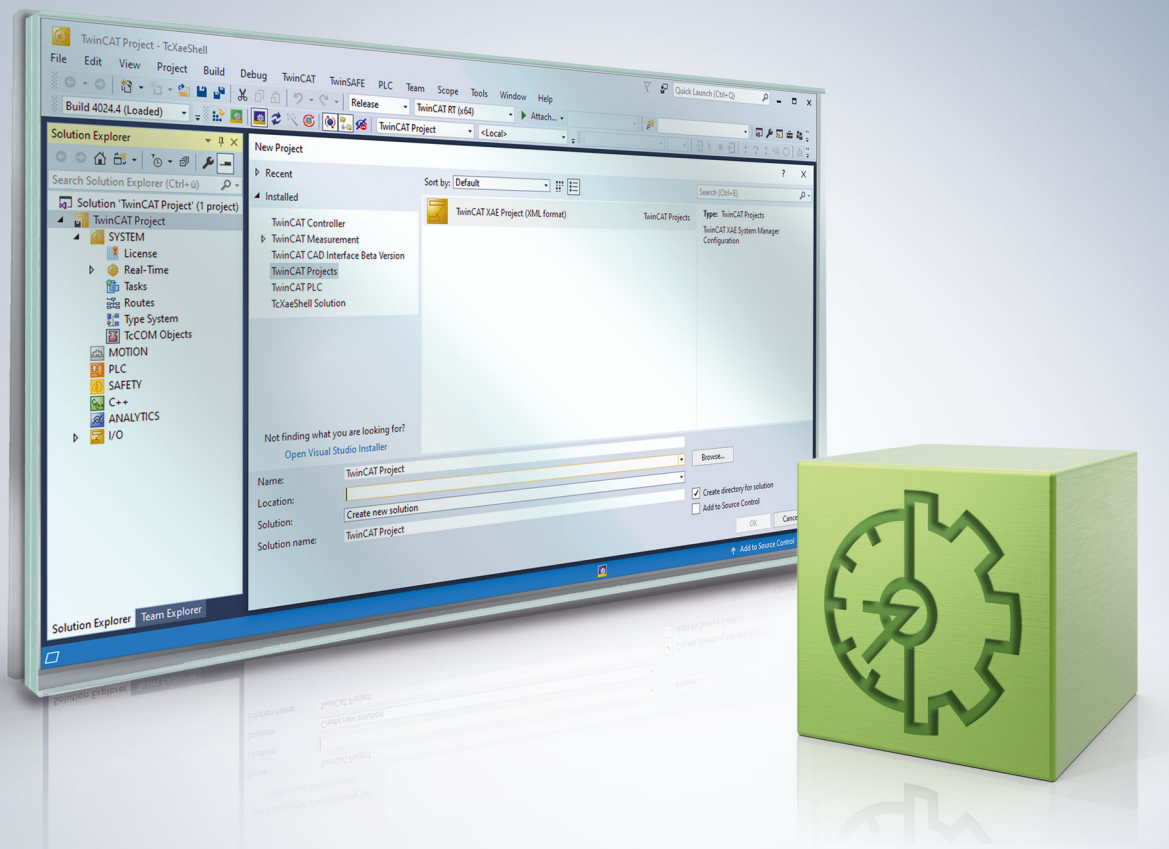
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | ADS-DLL C++



15.08.2023 | Version: 1.1.2

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® 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 Einführung

TcAdsDll stellt Funktionen für die Kommunikation mit anderen ADS Geräten zur Verfügung.

- Kommunikation mit dezentralen **TwinCAT-System**en oder mit Remote-**TwinCAT-Systemen** via **TwinCAT Message** Router.
- Kommunikation mit Remote-**TwinCAT-Systemen** via **TCP/IP** für **Win32 Systeme**.

TcAdsDll besitzt auch TwinCAT Ads Client Funktionen. Diese Funktionen werden auf 2 verschiedenen Wegen angeboten:

- **via C API**.
- oder via COM Interfaces [▶ 30]

Es wird empfohlen, die (kostenfreie) TwinCAT CP Version der Bibliothek zu nutzen.

3 C++ API

3.1 Functions

3.1.1 AdsGetDllVersion

Returns the version number, revision number and build number of the ADS-DLL.

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

Parameter

-

Return value

The return value, which is of type long, contains in coded form these three items related to the ADS-DLL.

Example

See example 1 [▶ 44].

3.1.2 AdsPortOpen

Establishes a connection (communication port) to the TwinCAT message router.

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

Parameter

-

Return value

A port number that has been assigned to the program by the ADS router is returned.

Example

See example 2 [▶ 44].

3.1.3 AdsPortClose

The connection (communication port) to the TwinCAT message router is closed.

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

Parameter

-

Return value

Returns the function's error status.

Example

See example 2 [▶ 44].

3.1.4 AdsGetLocalAddress

Returns the local NetId and port number.

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

Parameter

pAddr

[out] Pointer to the structure of type AmsAddr [▶ 26].

Return value

Returns the function's error status.

Example

See example 2 [▶ 44].

3.1.5 AdsSyncWriteReq

Writes data synchronously to an ADS device.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Length of the data, in bytes, written to the ADS server.

pData

[in] Pointer to the data written to the ADS server.

Return value

Returns the function's error status.

Example

See example 2 [▶ 44].

3.1.6 AdsSyncReadReq

Reads data synchronously from an ADS server.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Length of the data in bytes.

pData

[out] Pointer to a data buffer that will receive the data.

Return value

Returns the function's error status.

Example

See example 3 [▶ 44].

3.1.7 AdsSyncReadReqEx

Reads data synchronously from an ADS server.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Length of the data in bytes.

pData

[out] Pointer to a data buffer that will receive the data.
pcbReturn[out] Pointer to a variable. This variable returns the number of succesfully read data bytes.

Return value

Returns the function's error status.

3.1.8 AdsSyncReadWriteReq

Writes data synchronously into an ADS server and receives data back from the ADS device.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nReadLength

[in] Length of the data, in bytes, returned by the ADS device.

pReadData

[out] Buffer with data returned by the ADS device.

nWriteLength

[in] Length of the data, in bytes, written to the ADS device.

pWriteData

[out] Buffer with data written to the ADS device.

Return value

Returns the function's error status.

Example

See example 7 [▶ 47].

3.1.9 AdsSyncReadWriteReqEx

Writes data synchronously into an ADS server and receives data back from the ADS device.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nReadLength

[in] Length of the data, in bytes, returned by the ADS device.

pReadData

[out] Buffer with data returned by the ADS device.

nWriteLength

[in] Length of the data, in bytes, written to the ADS device.

pWriteData

[out] Buffer with data written to the ADS device.
pcbReturn[out] Pointer to a variable. This variable returns the number of successfully read data bytes.

Return value

Returns the function's error status.

3.1.10 AdsSyncReadDeviceInfoReq

Reads the identification and version number of an ADS server.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

pDevName

[out] Pointer to a character string that will receive the name of the ADS device.

pVersion

[out] Address of a variable of type AdsVersion [▶ 27], which will receive the version number, revision number and the build number.

Return value

Returns the function's error status.

Example

See example 5 [▶ 46].

3.1.11 AdsSyncWriteControlReq

Changes the ADS status and the device status of an ADS server.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nAdsState

[in] New ADS status.

nDeviceState

[in] New device status.

nLength

[in] Length of the data in bytes.

pData

[in] Pointer to data sent additionally to the ADS device.

Return value

Returns the function's error status.

Comments

In addition to changing the ADS status and the device status, it is also possible to send data to the ADS server in order to transfer further information. In the current ADS devices (PLC, NC, ...) this data has no further effect. Any ADS device can inform another ADS device of its current state. A distinction is drawn here between the status of the device itself (DeviceState) and the status of the ADS interface of the ADS device (AdsState). The states that the ADS interface can adopt are laid down in the ADS specification.

Example

See example 6 [▶ 46].

3.1.12 AdsSyncReadStateReq

Reads the ADS status and the device status from an ADS server.

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

pAdsState

[out] Address of a variable that will receive the ADS status (see data type ADSSTATE [▶ 29]).

pDeviceState

[out] Address of a variable that will receive the device status.

Return value

Returns the function's error status.

Remarks

Any ADS device can inform another ADS device of its current state. A distinction is drawn here between the status of the device itself (DeviceState) and the status of the ADS interface of the ADS device (AdsState). The states that the ADS interface can adopt are laid down in the ADS specification.

Example 11 [▶ 52] illustrates how the change can be detected with the aid of a callback function.

Example

See example 4 [▶ 45].

3.1.13 AdsSyncAddDeviceNotificationReq

A notification is defined within an ADS server (e.g. PLC). When a certain event occurs a function (the callback function) is invoked in the ADS client (C program).

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

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] IndexGroup.

nIndexOffset

[in] IndexOffset.

pNoteAttrib

[in] Pointer to the structure [▶ 27] that contains further information.

pNoteFunc

[in] Name of the callback function [▶ 18].

hUser

[in] 32-bit value that is passed to the callback function.

pNotification

[out] Address of the variable that will receive the handle of the notification.

Return value

Returns the function's error status.

Limitation:

Per ADS-Port a limitted number of 550 notifications are available.

Example

See example 8 [▶ 48].

3.1.14 AdsSyncDelDeviceNotificationReq

A notification defined previously is deleted from an ADS server.

```
LONG AdsSyncDelDeviceNotificationReq(
     PAmsAddr  pAddr,
     ULONG   hNotification
);
```

Parameter

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

hNotification

[out] Address of the variable that contains the handle of the notification.

Return value

Returns the function's error status.

Example

See example 8 [▶ 48].

3.1.15 AdsSyncSetTimeout

Alters the timeout for the ADS functions. The standard value is 5000 ms.

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

Parameter

nMs

[in] Timeout in ms.

Return value

Returns the function's error status.

Example

-

3.1.16 AdsAmsRegisterRouterNotification

The AdsAmsRegisterNotificationReq() function can be used to detect a change in the status of the TwinCAT router. The given callback function is invoked each time the status changes. Monitoring of the router's status is ended once more by the AdsAmsUnRegisterNotification() function.

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

Parameter

pNoteFunc

[in] Name of the callback function

Return value

Returns the function's error status.

Hints:

- Implemented from TcAdsDLL File Version: 2.8.0.21 (delivered with TwinCAT 2.9 Build > 941).
- A connection to the TwinCAT-Router can be done, if TwinCAT has been installed on the local PC. The function delivers an error on a system without TwinCAT.

Example

-

3.1.17 AdsAmsUnRegisterRouterNotification

Monitoring the router's status is ended by the AdsAmsUnRegisterNotification() function. See also AdsAmsRegisterNotificationReq().

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

Parameter

-

Return value

Returns the function's error status.

Hints:

- Implemented from TcAdsDLL File Version: 2.8.0.21 (delivered with TwinCAT 2.9 Build > 941).
- A connection to the TwinCAT-Router can be done, if TwinCAT has been installed on the local PC. The function delivers an error on a system without TwinCAT.

Example

-

3.1.18 PAmsRouterNotificationFuncEx

Type definition of the callback function required by the AdsAmsRegisterRouterNotification [▶ 17] function.

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

3.1.19 PAdsNotificationFuncEx

Type definition of the callback function required by the AdsSyncAddDeviceNotificationReq [▶ 15] function.

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

3.1.20 Extended Functions (for multithreaded applications)

With the existing functions only one ADS port could be created for each process. Particularly for multithreaded applications this is not sufficient, since the individual Ads commands would block each other. With the new functions it is now possible to use more than one port. This would enable a separate ADS port to be used for each thread, for example. New ports can be opened via the AdsPortOpenEx function. The returned port number is then transferred as parameter to the individual sync functions.

3.1.20.1 AdsPortOpenEx

Establishes a connection (communication port) to the TwinCAT message router. Unlike with AdsPortOpen, a new ADS port is opened each time. The extended Ads functions have to be used for communicating with this port. The port number returned by AdsPortOpenEx is transferred as parameter to these functions. If no TwinCAT MessageRouter is present, the AdsPortOpenEx function will fail.

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

Parameters

-

Return value

The number of the opened Ads port. A return value of 0 means the call has failed.

Example

See example 2 [▶ 44].

3.1.20.2 AdsPortCloseEx

The connection (communication port) to the TwinCAT message router is closed. The port to be closed must previously have been opened via an `AdsPortOpenEx` call.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18].

Return value

Returns the function's error status.

Example

See example 2 [▶ 44].

3.1.20.3 AdsGetLocalAddressEx

Returns the local NetId and port number.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[out] Pointer to the structure of type AmsAddr [▶ 26].

Return value

Returns the function's error status.

Example

See example 2 [▶ 44].

3.1.20.4 AdsSyncWriteReqEx

Writes data synchronously to an ADS device.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Length of the data, in bytes, written to the ADS server.

pData

[in] Pointer to the data written to the ADS server.

Return value

Returns the function's error status.

Example

See example 2 [▶ 44].

3.1.20.5 AdsSyncReadReqEx2

Reads data synchronously from an ADS server.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nLength

[in] Length of the data in bytes.

pData

[out] Pointer to a data buffer that will receive the data.

pcbReturn

[out] pointer to a variable. If successful, this variable will return the number of actually read data bytes.

Return value

Returns the function's error status.

3.1.20.6 AdsSyncReadWriteReqEx2

Writes data synchronously into an ADS server and receives data back from the ADS device.

```
LONG AdsSyncReadWriteReqEx2(
     LONG  port,
     PAmsAddr  pAddr,
     ULONG  nIndexGroup,
     ULONG  nIndexOffset,
     ULONG  nReadLength,
     PVOID  pReadData,
     ULONG  nWriteLength,
```

```
    PVOID  pWriteData,
    ULONG*  pcbReturn
);
```

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] Index Group.

nIndexOffset

[in] Index Offset.

nReadLength

[in] Length of the data, in bytes, returned by the ADS device.

pReadData

[out] Buffer with data returned by the ADS device.

nWriteLength

[in] Length of the data, in bytes, written to the ADS device.

pWriteData

[out] Buffer with data written to the ADS device.
pcbReturn

[out] pointer to a variable. If successful, this variable will return the number of actually read data bytes.

Return value

Returns the function's error status.

3.1.20.7 AdsSyncReadDeviceInfoReqEx

Reads the identification and version number of an ADS server.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

pDevName

[out] Pointer to a character string that will receive the name of the ADS device.

pVersion

[out] Address of a variable of type AdsVersion [▶ 27], which will receive the version number, revision number and the build number.

Return value

Returns the function's error status.

Example

See example 5 [▶ 46].

3.1.20.8 AdsSyncWriteControlReqEx

Changes the ADS status and the device status of an ADS server.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nAdsState

[in] New ADS status.

nDeviceState

[in] New device status.

nLength

[in] Length of the data in bytes.

pData

[in] Pointer to data sent additionally to the ADS device.

Return value

Returns the function's error status.

Comments

In addition to changing the ADS status and the device status, it is also possible to send data to the ADS server in order to transfer further information. In the current ADS devices (PLC, NC, ...) this data has no further effect. Any ADS device can inform another ADS device of its current state. A distinction is drawn here between the status of the device itself (DeviceState) and the status of the ADS interface of the ADS device (AdsState). The states that the ADS interface can adopt are laid down in the ADS specification.

Example

See example 6 [▶ 46].

3.1.20.9 AdsSyncReadStateReqEx

Reads the ADS status and the device status from an ADS server.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

pAdsState

[out] Address of a variable that will receive the ADS status (see data type ADSSTATE [▶ 29]).

pDeviceState

[out] Address of a variable that will receive the device status.

Return value

Returns the function's error status.

Remarks

Any ADS device can inform another ADS device of its current state. A distinction is drawn here between the status of the device itself (DeviceState) and the status of the ADS interface of the ADS device (AdsState). The states that the ADS interface can adopt are laid down in the ADS specification.
Example 11 [▶ 52] illustrates how the change can be detected with the aid of a callback function.

Example

See example 4 [▶ 45].

3.1.20.10 AdsSyncAddDeviceNotificationReqEx

A notification is defined within an ADS server (e.g. PLC). When a certain event occurs, a function (the callback function) is invoked in the ADS client (C program).

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

nIndexGroup

[in] IndexGroup.

nIndexOffset

[in] IndexOffset.

pNoteAttrib

[in] Pointer to the structure [▶ 27] that contains further information.

pNoteFunc

[in] Name of the callback function [▶ 18].

hUser

[in] 32-bit value that is passed to the callback function.

pNotification

[out] Address of the variable that will receive the handle of the notification.

Return value

Returns the function's error status.

Limitation:

Per ADS-Port a limited number of 550 notifications are available.

Remarks

If the TwinCAT router is stopped and then started again, the notifications become invalid. You can trap this event with the AdsAmsRegisterRouterNotification() [▶ 17] function.

Example

See example 8 [▶ 48].

3.1.20.11 AdsSyncDelDeviceNotificationReqEx

A notification defined previously is deleted from an ADS server.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pAddr

[in] Structure with NetId [▶ 26] and port number of the ADS server.

hNotification

[out] Address of the variable that contains the handle of the notification.

Return value

Returns the function's error status.

Example

See example 8 [▶ 48].

3.1.20.12 AdsSyncSetTimeoutEx

Alters the timeout for the ADS functions. The standard value is 5000 ms.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

nMs

[in] Timeout in ms.

Return value

Returns the function's error status.

Example

-

3.1.20.13 AdsSyncGetTimeoutEx

Returns the configured timeout for the ADS functions. The standard value is 5000 ms.

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

Parameters

port

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pnMs

[out] Buffer to store timeout value in ms.

Return value

Returns the function's error status.

Example

-

3.1.20.14 AdsAmsPortEnabledEx

Returns status of the ADS client connection.

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

Parameters

nPort

[in] port number of an Ads port that had previously been opened with AdsPortOpenEx [▶ 18] or AdsPortOpen [▶ 9].

pbEnabled

[out] buffer to store status value.

Return value

Returns the function's error status.

Example

-

3.2 Structures

3.2.1 AmsAddr

The complete address of an ADS device can be stored in this structure.

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

Elements

NetId

NetId [▶ 26].

port

Port number.

3.2.2 AmsNetId

The NetId of and ADS device can be represented in this structure.

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

Elements

b[6]

NetId, consisting of 6 digits.

Comment

The structure consists of an array with 6 elements of type UCHAR. Each element in the array may adopt a value from 1 to 255. The NetId is set with the aid of the TwinCAT system service.

3.2.3 AdsVersion

The structure contains the version number, revision number and build number.

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

Elements

version

Version number.

revision

Revision number.

build

Build number.

3.2.4 AdsNotificationAttrib

This structure contains all the attributes for the definition of a notification.

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

Elements

cbLength

Length of the data that is to be passed to the callback function.

nTransMode [▶ 29]

ADSTRANS_SERVERCYCLE: The notification's callback function is invoked cyclically.

ADSTRANS_SERVERONCHA: The notification's callback function is only invoked when the value changes.

nMaxDelay

The notification's callback function is invoked at the latest when this time has elapsed. The unit is 100 ns.

nCycleTime

The ADS server checks whether the variable has changed after this time interval. The unit is 100 ns.

Remarks

The ADS DLL is buffered from the real time transmission by a FIFO. TwinCAT first writes every value that is to be transmitted by means of the callback function into the FIFO. If the buffer is full, or if the nMaxDelay time has elapsed, then the callback function is invoked for each entry. The nTransMode parameter affects this process as follows:

ADSTRANS_SERVERCYCLE
The value is written cyclically into the FIFO at intervals of nCycleTime. The smallest possible value for nCycleTime is the cycle time of the ADS server; for the PLC, this is the task cycle time. The cycle time can be handled in 1ms steps. If you enter a cycle time of 0 ms, then the value is written into the FIFO with every task cycle.

ADSTRANS_SERVERONCHA
A value is only written into the FIFO if it has changed. The real-time sampling is executed in the time given in nCycleTime. The cycle time can be handled in 1ms steps. If you enter 0 ms as the cycle time, the variable is written into the FIFO every time it changes.

HINWEIS

Balance read operations

Too many read operations can load the system so heavily that the user interface becomes much slower.

Using ADS Notifications

Folgen

- Set the cycle time to the most appropriate values
- Always close connections when they are no longer required.

3.2.5 AdsNotificationHeader

This structure is also passed to the callback function.

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

Elements

hNotification

Handle for the notification. Is specified when the notification is defined;

nTimeStamp

Time stamp in FILETIME format.

cbSampleSize

Number of bytes transferred.

data[ANY_SIZE_ARRAY]

Array with the transferred data.

Comment

The time stamp is transferred in the FILETIME format. FILETIME is a 64-bit variable, representing the time and date in 100 ns steps, starting from 01.01.1601. Local time shift is not considered; coordinated universal time (UTC) is used. If you want access to the individual elements (day, month, year, hour, minute, second) you need to convert the time stamp from the FILETIME format to the SYSTEMTIME format, and then calculate the time, taking local time shifts into account.

Example

See example 8 [▶ 48].

3.3 Enums

3.3.1 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;
```

3.3.2 ADSTRANSMODE

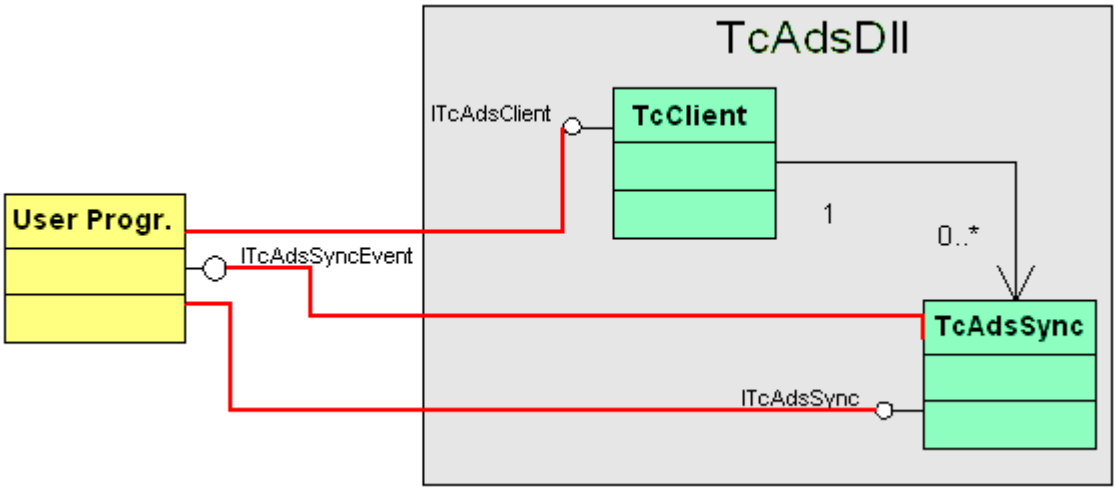
```
typedef enum nAdsTransMode {
  ADSTRANS_NOTRANS = 0,
  ADSTRANS_CLIENTCYCLE = 1,
  ADSTRANS_CLIENT1REQ = 2,
  ADSTRANS_SERVERCYCLE = 3,
  ADSTRANS_SERVERONCHA = 4
} ADSTRANSMODE;
```

4 COM

The TcAdsDll provides functions for communication with other ADS devices via the TwinCAT router through it's COM interface. You will find further information related to ADS under TwinCAT ADS

The COM Class TcClient [▶ 30] provides the user programs to establish a connection to ADS device to the local PC or to remote PC's. The TcAdsDll provides a multi-threaded threading model. It can be used by multi-threaded and single-threaded COM clients. If the TcAdsDll is used by single-threaded clients the method calls are synchronized by a marshaler. The marshaler is compiled into the TcAdsDll. No additional proxy-stub-dll is needed.

The TcClient [▶ 30] returns for each connection to one particular ADS device an object of the Type TcAdsSync [▶ 30]. This class provides synchronous ADS communication to the ADS device. The Class TcAdsSync [▶ 30] provides the communication function through the default interface ITcAdsSync [▶ 32]. To receive Ads Notification from the TcAdsSync [▶ 30] object the user program has to implement and connect the Event Interface ITcAdsSyncEvent [▶ 37].



4.1 Classes

4.1.1 TcAdsDll::Classes

The TcAdsDll provides interface to the outside by COM (Component Object Model).

CoClasses	Description
TcClient [▶ 30]	The Main class of the TcAdsDll. Provides a class factory to establish a ADS client connection

4.1.2 TcClient

The TcClient object provides a class factory to establish a ADS client connection.

Interface	Description
ITcClient [▶ 31]	Interface provides a class factory

4.1.3 TcAdsSync

The TcAdsSync object provides ADS communication to an ADS device. The TcAdsSync object has no class factory and can just be created by an call.

to ITcClient [▶ 30] ::ITcClient [▶ 30]Connect [▶ 31]

Interface	Description
ITcAdsSync [▶ 32]	Interface that provides the Ads communication functions.

4.2 Interfaces

4.2.1 ITcClient

The ITcClient interface provides a class factory to create an object to communicate with one Ads device. The interface derives from IUnknown.

IUnknown Methods	Description
QueryInterface	Returns a pointer to the interface you query for
AddRef	Increments the reference counter
Release	Decrements the reference counter

ITcClient Methodes	Description
Connect [▶ 31]	Creates an object of the type ITcAdsSync that provides Ads synchronous communication to one particular ADS device.

4.2.1.1 ITcClient::Connect

Creates a new Ads Client communication object for one particular Ads device by given AdsNet Id and Port Number.

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

Parameters

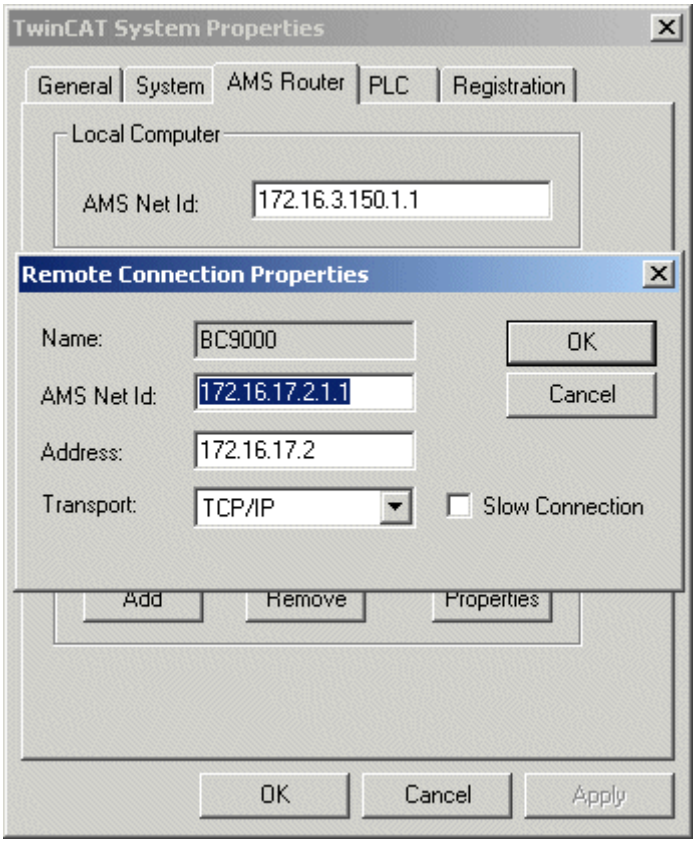
pAmsNetId	[in] variable presents the Ams Net Id by the structure type AmsNetId. If the Net Id is set to 0.0.0.0.0.0 the connection is made to the local TwinCAT system. If the client PC has no TwinCAT system installed the connection uses TCP/IP. The limitation on TCP/IP is that the client can just establish connections to the main remote device with the AMS Net Is = TCP/IP address + 1.1. .
nPort	[in] The Ads Port number of the Ads device we want to communicate with.
ITcAdsSync	[out, retval] Returns a pointer to an ITcAdsSync [▶ 32] pointer that holds the object that is used for Ads communication.

Return Values

S_OK	The connect function was successfully called.
ADSERRORCODES [▶ 38]	An error occurs

Remarks

To establish a connection to remote TwinCAT Systems, the remote device has to be added to the list of Remote Computer on the TwinCAT system. If on client PC and remote PC a TwinCAT system is installed, the client PC has to be added to the list of remote computer on the remote PC and vice versa. If the client PC does not have a TwinCAT system installed the client PC has just to be added to the list of remote computers on the remote PC.
If the Client PC has no TwinCAT system the AMS Net Id is just the TCP/IP address + 1.1. .



4.2.2 ITcAdsSync

The ITcClient interface provides a client the functionality to communicate to ads device. The interface derives from IUnknown.

IUnknown Methods	Description
QueryInterface	Returns a pointer to the interface you query for
AddRef	Increments the reference counter
Release	Decrements the reference counter

ITcAdsSync Methodes and Properties	Description
Write [▶ 32]	Writes a value to a variable as byte stream in an Ads device.
Read [▶ 33]	Reads the value of a variable as byte stream from an Ads device.
ReadWrite [▶ 33]	Writes a value to an Ads device and reads back the result in one step.
WriteControl [▶ 34]	Writes a control command to an Ads device.
AddDeviceNotification [▶ 34]	Connects a variable to the client. The client will be notified by an event.
DelDeviceNotification [▶ 35]	Removes the connection of a variable.
ReadDeviceInfo [▶ 35]	Reads Device Information from an Ads device.
ReadState [▶ 36]	Reads the state from an Ads device.
Timeout [▶ 36]	Sets the time after which the client will get a timeout warning on all other Ads commands

4.2.2.1 ITcAdsSync::Write

This method writes a value to a variable in an ADS device as byte stream

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

Parameters

indexGroup	[in] A variable of the type long that holds the index group of the variable we want to write to.
indexOffset	[in] A variable of the type long that holds the index offset of the variable we want to write to.
cbLen	[in] Count of byte we want to write to the variable.
pData	[in, size_is(cbLen)] A pointer to the first element of a byte array with the length **cbLen** of the data we want to write to a variable in an Ads device.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.2.2.2 ITcAdsSync::Read

This method reads a value of a variable from an ADS device as byte stream

```
HRESULT Read(
long indexGroup,
long indexOffset,
long cbLen,
long* pcbRead,
byte* pData);
```

Parameters

indexGroup	[in] A variable of the type long that holds the index group of the variable we want to read.
indexOffset	[in] A variable of the type long that holds the index offset of the variable we want to read.
cbLen	[in] Count of byte we want to read from the variable.
pcbRead	[out] Pointer to a variable the returns the count of bytes we really had read.
pData	[out, size_is(cbLen), length_is(*pcbRead)] A pointer to the first element of a byte array with the length **cbLen** of the data we want to read from a variable in a Ads device.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.2.2.3 ITcAdsSync::ReadWrite

This method writes a value to an ADS device and receive back the data device in one call.

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

Parameters

indexGroup	[in] A variable of the type long that holds the index group of the variable we want to read.
indexOffset	[in] A variable of the type long that holds the index offset of the variable we want to read.
cbRdLen	[in] Count of byte we want to read from the variable.
pcbRead	[out] Pointer to a variable the returns the count of bytes we really had read.
pRdData	[out, size_is(cbRdLen), length_is(*pcbRead)] A pointer to the first element of a byte array with the length **cbRdLen** of the data we want to read from a variable in an Ads device.
cbWrLen	[in] Count of byte we want to write to the variable.
pWrData	[in, size_is(pWrData)] A pointer to the first element of a byte array with the length **cbWrLen** of the data we want to write to a variable in an Ads device.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.2.2.4 ITcAdsSync::WriteControl

This method sets the state of the ADS system an devices

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

Parameters

adsState	[in] The states as ADSSTATE [▶ 40] we want to set onto the ADS system.
deviceState	[in] The states as ADSSTATE [▶ 40] we want to set onto the ADS device.
cbLen	[in] Count of byte we want to write to the variable.
pData	[in, size_is(cbLen)] A pointer to the first element of a byte array with the length **cbLen** of additional the data we want to write to the Ads device.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.2.2.5 ITcAdsSync::AddDeviceNotification

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

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

Parameters

indexGroup	[in] A variable of the type long that holds the index group of the variable we want to read.
indexOffset	[in] A variable of the type long that holds the index offset of the variable we want to read.
cbLenData	[in] Count of byte we want to read from the connected variable.
transMode	[out] The mode how the variable is connected with the type ADSTRANSMODE [▶ 40].
nMaxDelay	[in] The time with a resolution of 100 ns after we want to receive an callback on the implemented _ITcAdsSyncEvent [▶ 37] interface. • nCycleTime • [in] The time with a resolution of 100 ns how the variable should be collected
phNotification	[out, retval] A pointer to the handle that unique identifies the connection of our variable.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

Remarks

A nCycleTime= 10000 and nMaxDelay=100000 would receive every 10ms 10 values with the resolution of 1ms.

4.2.2.6 ITcAdsSync::DelDeviceNotification

This method removes the connection of a variable, that was connected before by an AddDeviceNotification

```
HRESULT DelDeviceNotification(
  long phNotication);
```

Parameters

phNotification	[in] The handle of the further established connection.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

Remarks

A nCycleTime= 10000 and nMaxDelay=100000 would receive every 10ms 10 values with the resolution of 1ms.

4.2.2.7 ITcAdsSync::ReadDeviceInfo

This method retrieves information about the Ads device.

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

Parameters

pName	[out] A variable that holds the BSTR string that describes the ADS device.
pVersion	[out] A pointer to a variable of the type AdsVersion [▶ 37] that holds the version number.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.2.2.8 ITcAdsSync::ReadState

This method retrieves information about the ADA device State and the ADS system state.

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

Parameters

pAdsState	[out] Pointer to variable of the type ADSSTATE [▶ 40] that holds the state of the ADS system.
pDeviceState	[out] Pointer to variable of the type ADSSTATE [▶ 40] that holds the state of the ADS device.

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.2.2.9 ITcAdsSync::Timeout

This property is used to assign or retrieve the timeout value in milliseconds for all other Ads functions of the ITcAdsSync [▶ 32] interface.

Retrieve the Timeout value

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

Parameters

pTime	[out, retval] Pointer to a variable that holds the current timeout value

Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An Error occurs

Assign a new Timeout value

```
HRESULT put_Timeout(long nTime);
```

Tab. 1: Parameters

nTime	[in] A variable that holds the new timeout value

Tab. 2: Return Values

S_OK	The function was successfully called
ADSERRORCODES [▶ 38]	An Error occurs

4.2.3 _ITcAdsSyncEvent

The **_ITcAdsSyncEvent** interface is the event Interface that a client has to implement if he wants to receive ADS Notification for connected variables. The interface derives from IUnknown.

IUnknown Methods	Description
QueryInterface	Returns a pointer to the interface you query for
AddRef	Increments the reference counter
Release	Decrements the reference counter

_ITcAdsSyncEvent Methodes	Description
DeviceNotification [▶ 37]	This method is called from the server for all connected variables of this Ads device

4.2.3.1 _ITcAdsSyncEvent::DeviceNotification

This method is called from the server for all connected variables of this Ads device. The Event occurs for those variables that had been connected by an AddDeviceNotification [▶ 34] before.

```
HRESULT DeviceNotification(
  TimeStamp* pTime,
  long hNotification,
  long cbLen,
  byte* pData
);
```

Parameters

pTime	[in] A pointer to a variable of the type TimeStamp [▶ 38] that holds the exact time when the connected variable was collected.
hNotification	[in] The handle that identifies one particular connected variable. The handle was retuned when the variable was connected by a call to method AddDeviceNotification [▶ 34].
cbLen	[in] Count of data bytes received by this method.
pData	[in, size_is(cbLen)]. A pointer to the first element of a byte array with the size of **cbLen** that contains the data of the connected variable

Return Values

S_OK	The connect function was successfully called
ADSERRORCODES [▶ 38]	An error occurs

4.3 Structures

4.3.1 AdsVersion

The structure AdsVersion represents a version number spitted into version, revision and build number.

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

4.3.2 TimeStamp

The structure TimeStamp represents a windows **FILETIME** data structure. It is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601. It is the means by which Win32 determines the date and time.

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

4.4 Enums

4.4.1 ADSERRORCODES

The enumeration type **ADSERRORCODE** describes Ads errors with the following values:

Const	Hex Value	Description
ADS_E_ERROR	0x98117000	this is the offset of Ads Errors presented a COM HRESULT
ADS_E_SRVNOTSUPP	0x98117001	the requested service is not supported by the ADS device
ADS_E_INVALIDGRP	0x98117002	invalided index group
ADS_E_INVALIDOFFSET	0x98117003	invalid index offset
ADS_E_INVALIDACCESS	0x98117004	reading an writing not permitted
ADSERR_DEVICE_INVALIDSIZE	0x98117005	parameter size is not correct
ADS_E_INVALIDDATA	0x98117006	invalided data value(s)
ADS_E_NOTREADY	0x98117007	device is not in a ready state
ADS_E_BUSY	0x98117008	device is busy
ADS_E_INVALIDCONTEXT	0x98117009	invalid context
ADS_E_NOMEMORY	0x9811700A	out of memory
ADS_E_INVALIDPARM	0x9811700B	invalid parameter value(s)
ADS_E_NOTFOUND	0x9811700C	not found (files, ...)
ADS_E_SYNTAX	0x9811700D	syntax error in command or file
ADS_E_INCOMPATIBLE	0x9811700E	objects do not match
ADS_E_EXISTS	0x9811700F	object already exists
ADS_E_SYMBOLNOTFOUND	0x98117010	symbol not found
ADS_E_SYMBOLVERSIONINVALID	0x98117011	symbol version invalid
ADS_E_INVALIDSTATE	0x98117012	server is in invalid state
ADS_E_TRANSMODENOTSUPP	0x98117013	AdsTransMode not supported
ADS_E_NOTIFYHNDINVALID	0x98117014	Notification handle is invalid
ADS_E_CLIENTUNKNOWN	0x98117015	Notification client not registered
ADS_E_NOMOREHDLS	0x98117016	no more notification handles
ADS_E_INVALIDWATCHSIZE	0x98117017	size for watch to big
ADS_E_NOTINIT	0x98117018	device not initialized
ADS_E_TIMEOUT	0x98117019	device has a timeout
ADS_E_NOINTERFACE	0x9811701A	query interface failed
ADS_E_INVALIDINTERFACE	0x9811701B	wrong interface required
ADS_E_INVALIDCLSID	0x9811701C	class ID is invalid
ADS_E_INVALIDOBJID	0x9811701D	object ID is invalid
ADS_E_CLIENT_ERROR	0x98117040	Error class: client error
ADS_E_CLIENT_INVALIDPARM	0x98117041	invalid parameter at service call
ADS_E_CLIENT_LISTEMPTY	0x98117042	polling list is empty
ADS_E_CLIENT_VARUSED	0x98117043	var connection already in use
ADS_E_CLIENT_DUPLINVOKEID	0x98117044	invoke id in use
ADS_E_CLIENT_SYNCTIMEOUT	0x98117045	timeout elapsed
ADS_E_CLIENT_W32ERROR	0x98117046	error in win32 subsystem
ADS_E_CLIENT_TIMEOUTINVALID	0x98117047	
ADS_E_CLIENT_PORTNOTOPEN	0x98117048	
ADS_E_CLIENT_NOAMSADDR	0x98117049	

Const	Hex Value	Description
ADS_E_CLIENT_SYNCINTERNAL	0x98117050	internal error in ads sync
ADS_E_CLIENT_ADDHASH	0x98117051	hash table overflow
ADS_E_CLIENT_REMOVEHASH	0x98117052	key not found in hash table
ADS_E_CLIENT_NOMORESYM	0x98117053	no more symbols in cache

4.4.2 ADSSTATE

The enumeration type **ADSSTATE** describes the Ads state with the following values:

Const	Int Value	Description
ADSSTATE_INVALID	0	Invalided state
ADSSTATE_IDLE	1	Idles state
ADSSTATE_RESET	2	Reset state
ADSSTATE_INIT	3	initialized
ADSSTATE_START	4	started
ADSSTATE_RUN	5	running
ADSSTATE_STOP	6	stopped
ADSSTATE_SAVECFG	7	saved configuration
ADSSTATE_LOADCFG	8	load configuration
ADSSTATE_POWERFAILURE	9	power failure
ADSSTATE_POWERGOOD	10	power good
ADSSTATE_ERROR	11	error state
ADSSTATE_SHUTDOWN	12	shutting down
ADSSTATE_SUSPEND	13	suspended
ADSSTATE_RESUME	14	resumed
ADSSTATE_CONFIG	15	system is in config mode
ADSSTATE_RECONFIG	16	system should restart in config mode

4.4.3 ADSTRANSMODE

The enumeration type **ADSTRANSMODE** describes the mode of an device notification with the following values:

Const	Int Value	Description
ADSTRANS_NOTRANS	0	
ADSTRANS_CLIENTCYCLE	1	
ADSTRANS_CLIENTONCHA	2	
ADSTRANS_SERVERCYCLE	3	
ADSTRANS_SERVERONCHA	4	
ADSTRANS_CLIENT1REQ	5	

5 Integration

5.1 Linking C++ ADS library for TwinCAT 3 in Visual Studio

Necessary files

The ADS components are installed with TwinCAT 3 and are located in '**\TwinCAT\AdsApi'** directory.

Include the Header files.

To use the functionality of the TcAdsDll in your project you have to include the TcAdsApi.h and the TcAdsDef.h header files into your project.

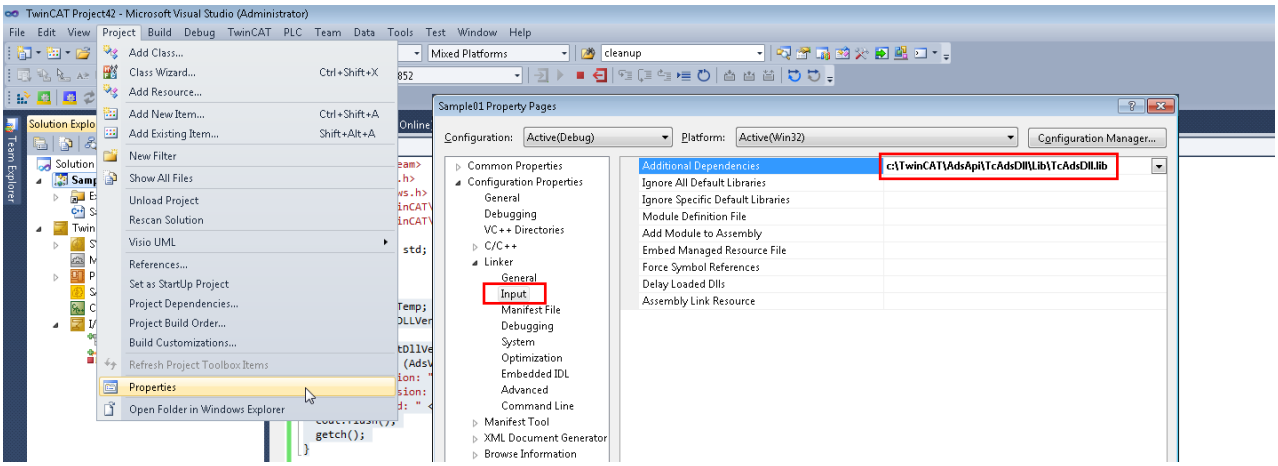
```
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"
```

Add the Library to your project

You have to include the **TcAdsDll.lib** library, to use the functionality of the TcAdsDll. The library can be found per default in following TwinCAT folder:

C:\TwinCAT\AdsApi\TcAdsDll\Lib\TcAdsDll.lib

In Visual Studio you have to select the menu item **Project|Properties**. On the project settings dialog you select the scope of the settings for: **Configuration Properties**. To include the library, you have to add the path to the TcAdsDll.Lib in the **Additional Dependencies** modules text box.



6 Samples

Description	Source text
Example 1: Read DLL version [▶ 44]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723063691.zip
Example 2: Write flag synchronously into PLC [▶ 44]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723065867.zip
Example 3: Read flag synchronously from the PLC [▶ 44]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723209099.zip
Example 4: Read ADS status [▶ 45]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723067531.zip
Example 5: Read ADS information [▶ 46]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723069195.zip
Example 6: Start/stop PLC [▶ 46]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723070859.zip
Example 7: Access an array in the PLC [▶ 47]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723072523.zip
Example 8: Event driven reading [▶ 48]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723074187.zip
Example 9: Access by variable name [▶ 50]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723075851.zip
Example 10: Read PLC variable declaration [▶ 51]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723090315.zip
Example 11: Detect status change in TwinCAT router and the PLC [▶ 52]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723094283.zip
Example 12: Event-Driven Detection of Changes to the Symbol Table [▶ 53]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723095947.zip
Example 13: Reserved	
Example 14: Reading the PLC variable declaration of an individual variable [▶ 54]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723097611.zip
Example 15: Reserved	
Example 16: Reserved	
Example 17: ADS-sum command: read or write [▶ 57]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723115403.zip
Example 18: ADS-sum command: Get and release [▶ 59]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723117067.zip
Example 19: Reserved	
Example 20: Transmitting structures to the PLC [▶ 62]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723118731.zip
Example 21: Reading and writing of TIME/DATE variables [▶ 63]	https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723120395.zip

6.1 Read DLL version

Download: 'https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723063691.zip'

This program determines the version of the DLL file.

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


// ADS headers for TwinCAT 3
#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();
}
```

6.2 Write flag synchronously into the PLC

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723065867.zip

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

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

// ADS headers for TwinCAT 3
#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;

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

// TwinCAT 3 PLC1 = 851
pAddr->port = 851;

// Read value from user that is to be written to the PLC
cout << "Value: ";
cin >> dwData;

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

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

6.3 Read flag synchronously from the PLC

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723209099.zip

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

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

// ADS headers for TwinCAT 3
#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;
// Open communication port on the ADS router
nPort = AdsPortOpen();
nErr = AdsGetLocalAddress(pAddr);
if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';

// TwinCAT3 PLC1 = 851
pAddr->port = 851;
// Read value from MD0 and display
do
{
nErr = AdsSyncReadReq(pAddr, 0x4020, 0x0, 0x4, &dwData);
if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
cout << dwData << '\n';
cout.flush();
}
while (getch() == '\r'); // Read the next value (use Carriage return as delimitter), stop otherwise
// Close communication port
nErr = AdsPortClose();
if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

6.4 Read ADS status

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723067531.zip

This program reads the status of the PLC. The variable of type ADSSTATE contains information such as, for example, whether the PLC is in the RUN or STOP state.

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

// ADS headers for TwinCAT 3
#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;

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

// TwinCAT3 PLC1 = 851
pAddr->port = 851;

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'); // continue on a carriage return, finish for any other key

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

6.5 Read ADS information

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723069195.zip

Each ADS device contains a version number and an identification. The example program reads this information from the PLC and displays it on the screen.

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


// ADS headers for TwinCAT 3
#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;

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

// TwinCAT3 PLC1 = 851
pAddr->port = 851;

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();

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

6.6 Start/stop PLC

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723070859.zip.

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

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

// ADS headers for TwinCAT 3
#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;

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

// TwinCAT 3 PLC1 = 851
pAddr->port = 851;

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);
}

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

6.7 Access an array in the PLC

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723072523.zip

An array, located in the PLC, is to be read by means of a read command. The variable is addressed here by its handle. The length of the whole array is provided as the length for the function AdsSyncReadReq(). The address of the first array element is given as variable.

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

// ADS headers for TwinCAT 3
#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';

// Select Port: TwinCAT 3 PLC1 = 851
pAddr->port = 851;
```

```
 // 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';
}
```

6.8 Event driven reading

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723074187.zip

If values from a PLC or NC are to be displayed continuously on a user interface, then it is very inefficient to use AdsSyncReadReq() [▶ 11], since this function must be called cyclically. By defining what are known as notifications (messages), a TwinCAT server can be made to transmit values via ADS to another ADS device. A distinction is drawn between whether the TwinCAT server is to transmit the values cyclically, or only when the values change.

A notification is begun with the AdsSyncAddDeviceNotificationReq() [▶ 15] function. After this, the callback function is automatically invoked by TwinCAT. AdsSyncDelDeviceNotificationReq() [▶ 16] is used to halt the notification again. Since the number of notifications is limited, you should ensure the notifications no longer required by your program are deleted. You will find further information under the description of the AdsNotificationAttrib [▶ 27] structure.

The following program starts a notification on a variable handle in the PLC. Each time the PLC variable changes, the callback function is invoked. The callback function receives a variable of type AdsNotificationHeader() [▶ 28] as one of its parameters. This structure contains all the necessary information (value, time stamp, ...).

Efficient Usage

- Don't use time intensive executions in callbacks.
- Remind to sync your callback and your mainthread, if you access each other (e.g. critical sections, mutex, events).

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

// ADS headers for TwinCAT 3
#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';

// TwinCAT 3 RTS1 Port = 851
pAddr->port = 851;

// set the attributes of the notification
adsNotificationAttrib.cbLength = 4;
adsNotificationAttrib.nTransMode = ADSTRANS_SERVERONCHA;
adsNotificationAttrib.nMaxDelay = 0;
adsNotificationAttrib.nCycleTime = 10000000; // 1sec

// get 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 << "Notification: " << hNotification << "\n\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 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 << ". Call:\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 << '.' <<
LocalTime.wMilliseconds <<
" den: " << LocalTime.wDay << '.' << LocalTime.wMonth << '.' << LocalTime.wYear << '\n';

// Größe des Buffers in Byte
cout << "SampleSize: " << pNotification->cbSampleSize << '\n';

// 32-Bit Variable (auch Zeiger), die beim AddNotification gesetzt wurde // (siehe main)
cout << "hUser: " << hUser << '\n';

// Print out the ADS-address of the sender
cout << "ServerNetId: ";
```

```
for (nIndex = 0; nIndex < 6; nIndex++)
cout << (int)pAddr->netId.b[nIndex] << ".";
cout << "\nPort: " << pAddr->port << "\n\n";
cout.flush();
}
```

6.9 Access by variable name

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723075851.zip

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

All data that ADS devices make available to the outside is organised by means of IndexGroups and IndexOffset. An IndexGroup can be thought of as a table, with each entry being addressed by the IndexOffset. The TwinCAT PLC has, for example, IndexGroups in which the variables that belong to the input/output or flags regions are stored. IndexGroups are also available to the TwinCAT PLC through which system functions may be addressed.

The IndexGroups ADSIGRP_SYM_HNDBYNAME and ADSIGRP_ SYM_VALBYHND are important for the example program. The IndexGroup ADSIGRP_SYM_HNDBYNAME is used to request a handle from a PLC variable identified by name. The variable can be accessed with the aid of this handle and the IndexGroup ADSIGRP_SYM_VALBYHND. The variable's handle is passed as the IndexOffset.

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


// ADS headers for TwinCAT 3
#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;
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';

// TwinCAT 3 PLC1 = 851
pAddr->port = 851;

// Fetch handle for an <szVar> PLC variable
nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlVar), &lHdlVar,
sizeof(szVar), szVar);
if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';
do
{
// Read value of a PLC variable (by handle)
nErr = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nData );
if (nErr)
cerr << "Fehler: AdsSyncReadReq: " << nErr << '\n';
else
cout << "Wert: " << nData << '\n';
cout.flush();
if (Data > 10)
{
// Reset 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'); // read next value with RETURN, else end


//Release handle of plc variable
nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlVar), &lHdlVar);
```

```
if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

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

6.10 Read PLC variable declaration

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723090315.zip

The following information is transferred when accessing the variable declaration:

- Variable name
- Data type
- Length
- Address (IndexGroup / IndexOffset)
- Comment

All the information listed above is transmitted in a data stream. Before this can be read, the first AdsSyncReadReq() [▶ 11] is used to obtain the length. The data itself is transferred with the second AdsSyncReadReq() [▶ 11]. The *pchSymbols* variable is a pointer, pointing to this region. The FOR-loop copies the corresponding data region into the *pAdsSymbolEntry* structure for each individual PLC variable. The individual information items in the PLC variables are stored in this structure. The macros PADSSYMBOLNAME, PADSSYMBOLTYPE and PADSSYMBOLCOMMENT simplify the evaluation of this data.

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

// ADS headers for TwinCAT 3
#include "C:\TwinCAT\3.0\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\3.0\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';

// Select Port: TwinCAT 3 PLC1 = 851
pAddr->port = 851;

// Read the length of the variable declaration
nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_UPLOADINFO, 0x0, sizeof(tAdsSymbolUploadInfo),
&tAdsSymbolUploadInfo);
if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
pchSymbols = new char[tAdsSymbolUploadInfo.nSymSize];
assert(pchSymbols);

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

// Output information about the PLC variables
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();
}
getch();

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

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

6.11 Detect status change in TwinCAT router and the PLC

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723094283.zip

When an application is actually running it is often important to interrogate the status of TwinCAT and/or of its components; e.g., whether the PLC is in the RUN state. To avoid the need to repeatedly issue this inquiry, changes in the status can be detected very effectively with the aid of callback functions.
The following example program monitors the status of the PLC (run-time system 1) and of the TwinCAT router.
By invoking the AdsAmsRegisterRouterNotification() function, the given callback function will be invoked every time the status of the TwinCAT router changes. The current status can be interrogated by means of the parameters that are transferred.
The AdsSyncAddDeviceNotificationReq() is used to monitor the status of the PLC. The data that is passed to the callback function represents the current status of the PLC.

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


// ADS headers for TwinCAT 3
#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: TwinCAT 3 PLC1 = 851
pAddr->port = 851;

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

// Invoke notification
adsNotificationAttrib.cbLength = sizeof(short);
adsNotificationAttrib.nTransMode = ADSTRANS_SERVERONCHA;
adsNotificationAttrib.nMaxDelay = 0; // jede Aenderung sofort melden
adsNotificationAttrib.dwChangeFilter = 0; //
nErr = AdsSyncAddDeviceNotificationReq(pAddr, ADSIGRP_DEVICE_DATA, ADSIOFFS_DEVDATA_ADSSTATE,
&adsNotificationAttrib, 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 ();
}
```

6.12 Event-Driven Detection of Changes to the Symbol Table

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723095947.zip.

ADS devices that support symbol names (PLC, NC, ...) store those names in an internal table. A handle is assigned here to each symbol. The symbol handle is necessary in order to be able to access the variables (see also Example 9 [▶ 50]). If the symbol table changes because, for instance, a new PLC program is written into the controller, the handles must be ascertained once again. The example below illustrates how changes to the symbol table can be detected.

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

// ADS headers for TwinCAT 3
#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: TwinCAT 3 PLC1 = 851
pAddr->port = 851;

// 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,
SymbolChanged, 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();
}
```

6.13 Reading the PLC variable declaration of an individual variable

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723097611.zip

The following information is transferred when accessing the variable declaration:

- Variable name
- Data type
- Length
- Address (IndexGroup / IndexOffset)
- Comment

The AdsSyncReadWriteReq() call is used to read the variable information. The variable name is transferred to the function via parameter *pWriteData*. After the call the requested information is contained in variable *pAdsSymbolEntry* . The individual information items in the PLC variables are stored in this structure. The macros PADSSYMBOLNAME, PADSSYMBOLTYPE and PADSSYMBOLCOMMENT simplify the evaluation of this data. In the next step, the data type of the variable is evaluated via *pAdsSymbolEntry->dataType*. If the data type is UDINT or ARRAY OF UDINT, the value of this variable is also read.

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

// ADS headers for TwinCAT 3
#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: TwinCAT 3 PLC1 = 851
pAddr->port = 851;

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

nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_INFOBYNAMEEX, 0, sizeof(buffer), buffer,
strlen(szVariable)+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';
}
```

6.14 Upload PLC-variabledeclaration (dynamic) (2/2)

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/12470837515.zip'

This sample describes how to upload the PLC symbol information in a more efficient dynamic way.

The PLC symbol information contain the following parts:

- variable name
- data type
- length
- address (IndexGroup / IndexOffset)
- comment

HINWEIS

We highly recommend to NOT work with this IndexGroup/IndexOffset for ADS communication but instead use handles of symbols for ADS communication.

After uploading the information "name", "datatype" and "length" it makes sense to request a handle for this symbol.

Read the major information via ADS into internal class **"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 datatye-descriptions
nResult = AdsSyncReadReq(&m_amsAddr, ADSIGRP_SYM_DT_UPLOAD, 0, info.nDatatypeSize, pDT);
if ( nResult == ADSERR_NOERR )
{
// create class-object for each datatype-description
m_pDynSymbols = new CAdsParseSymbols(pSym, info.nSymSize, pDT, info.nDatatypeSize);
if ( m_pDynSymbols == NULL )
nResult = ADSERR_DEVICE_NOMEMORY;
}
delete [] pDT;
}
}
delete [] pSym;
}
}
}
```

Get Parent :
The routine Get Parent will jump NOT jump to the direct parent of a child. Instead this command will jump to the next entry of the direct parent.

Get Sibling:
Selecting this option will return the next symbol within the current hierarchy level. Symbols containing child information will be displayed, but the child elements will not be displayed.

Get Child:
If the current symbol contains child-symbol information (so the current symbol is an instance of a datatype description), this command will enter the next hierarchy and return the information about first child object.

Get Next:
Clicking this button the next symbol will be extracted from internal class **"CAdsParseSymbols"** and be displayed.
Selecting always just this option allows to navigate from first ADS-symbol through the hierarchy symbol tree to the end of list.

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

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723115403.zip

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

Mind the ADS requirements

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

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 each one single ADS-API call.
This new ADS command offers to read with one singale 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 these 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 completely work on this single ADS request BEFORE starting 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!

Hinweis **We highly recommend to not request more than 500 Ads-Sub commands.**

```
// This code snippet using 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
answer

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 using ADSIGRP_SUMUP_WRITE with IndexGroup 0xF081 and IndexOffset as number of
ADS-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 "Read" 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
```

6.16 ADS-sum command: Get and release several handles

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723117067.zip

This sample shows how to get and release several handles with the ADS-sum command. It's constructed like the AdsSyncReadWriteRequest and is used as container to transport the sub commands.

1. Get handles

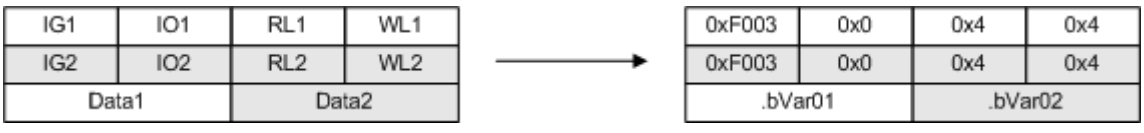
First, all necessary headers have to be included.

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

// ADS headers for TwinCAT 3
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsAPI.h"
```

Next step is to define a structure, declare variables and allocate memory.

```
// 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; // count 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;
```

The values which can be transferred are written behind the last structure.




```
// 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) );
```

For the communication a open port is necessary. After that the local address is handed over. If it comes to transmission the port is assigned to the address of the run time system 1 first.The parameters for the sum command consist of IndexGroup (0xf082) - call the sum command, *IndexOffset (0x2)*- count of sub commands, *ReadLength (0x18)*- size of the data which can be read, *ReadData (pBuffRes)*- memory which read data assumes, *WriteLength (cbReq)*- size of the data which can be send and *WriteLength (pBuffReq)*- memory which contains data that can be sent.

```
// Open communication port on the ADS router
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, WLen2, 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';};
```

2. Release handles Define a structure, declare variables and allocate memory again.

```
// 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;
```

The values which can be transferred are written behind the last structure again.


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

// Skip to next structure
pDataRel++;

// pDataReq-> structure 2
pDataRel->indexGroup = ADSIGRP_IOIMAGE_RWIB;
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) );
```

The existing connection is used to release the handles.The parameters for the sum command consist of IndexGroup (0xf081) - call the sum command, *IndexOffset (0x2)*- count of sub commands, *ReadLength (cbRelRes)*- size of the data which can be read, *ReadData (pBuffRelRes)*- memory which read data assumes, *WriteLength (cbRel)*- size of the data which can be send and *WriteLength (pBuffRel)*- memory which contains data that can be sent.Last thing left is releasing the handles and closing the port.

```
// 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();
}
```

6.17 Transmitting structures to the PLC

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723118731.zip

This example shows how to write a structure to the PLC via ADS. The elements in the structure have different data types:

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

// ADS headers for TwinCAT 3
#include "C:\TwinCAT\3.0\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\3.0\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);

// TwinCAT 3 PLC1 = 851
pAddr->port = 851;

// 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);

// 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();
}
```

6.18 Reading and writing of TIME/DATE variables

Download: https://infosys.beckhoff.com/content/1031/tc3_adsdll2/Resources/7723120395.zip

The PLC contains the TIME variable MAIN.Time1 and the DT variable MAIN.Date1. This example shows how to read, write and display those variables:

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

// ADS headers for TwinCAT 3
#include "C:\TwinCAT\3.0\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\3.0\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);
pAddr->port = AMSPORT_R0_PLC_RTS1;
```

```
// Get variable handle
lErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlTime), &lHdlTime,
sizeof(szPlcTime), szPlcTime);
lErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlDate), &lHdlDate,
sizeof(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();
}
```

Mehr Informationen:
www.beckhoff.de/te1000

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