# Table of contents

1 Foreword ................................................................. 5  
   1.1 Notes on the documentation ...................................... 5  
   1.2 Safety instructions .................................................. 6  
   1.3 Notes on information security ....................................... 7  
2 Introduction ............................................................. 8  
3 C++ API ........................................................................ 9  
   3.1 Functions ............................................................... 9  
      3.1.1 AdsGetDllVersion ............................................... 9  
      3.1.2 AdsPortOpen .................................................... 9  
      3.1.3 AdsPortClose ................................................... 9  
      3.1.4 AdsGetLocalAddress ......................................... 10  
      3.1.5 AdsSyncWriteReq .............................................. 10  
      3.1.6 AdsSyncReadReq .............................................. 11  
      3.1.7 AdsSyncReadReqEx ........................................... 11  
      3.1.8 AdsSyncReadWriteReq ....................................... 12  
      3.1.9 AdsSyncReadWriteReqEx .................................... 13  
      3.1.10 AdsSyncReadDeviceInfoReq ............................... 13  
      3.1.11 AdsSyncWriteControlReq ................................. 14  
      3.1.12 AdsSyncReadStateReq ...................................... 15  
      3.1.13 AdsSyncAddDeviceNotificationReq ........................ 15  
      3.1.14 AdsSyncDelDeviceNotificationReq ........................ 16  
      3.1.15 AdsSyncSetTimeout ......................................... 17  
      3.1.16 AdsAmsRegisterRouterNotification ........................ 17  
      3.1.17 AdsAmsUnRegisterRouterNotification ................... 18  
      3.1.18 PAdsRouterNotificationFuncEx ........................... 18  
      3.1.19 PAdsNotificationFuncEx ................................. 18  
      3.1.20 Extended Functions (for multithreaded applications) 18  
   3.2 Structures ............................................................. 27  
      3.2.1 AmsAddr ........................................................ 27  
      3.2.2 AmsNetId ....................................................... 27  
      3.2.3 AdsVersion ..................................................... 28  
      3.2.4 AdsNotificationAttrib ....................................... 28  
      3.2.5 AdsNotificationHeader ...................................... 29  
   3.3 Enums ................................................................. 30  
      3.3.1 ADSSTATE ....................................................... 30  
      3.3.2 ADSTransMode ................................................ 30  
4 COM ............................................................................ 31  
   4.1 Classes ................................................................. 31  
      4.1.1 TcAdsDll::Classes ........................................... 31  
      4.1.2 TcClient ......................................................... 31  
      4.1.3 TcAdsSync ...................................................... 31  
   4.2 Interfaces ............................................................. 32  
      4.2.1 ITcClient ....................................................... 32
1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

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

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

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

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

<table>
<thead>
<tr>
<th><strong>DANGER</strong></th>
<th>Serious risk of injury!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WARNING</strong></th>
<th>Risk of injury!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CAUTION</strong></th>
<th>Personal injuries!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>NOTE</strong></th>
<th>Damage to the environment or devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
<td></td>
</tr>
</tbody>
</table>

Tip or pointer
This symbol indicates information that contributes to better understanding.
1.3 Notes on information security

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

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

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

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

The TcAdsDll provides functions for communication with other ADS devices.

- Communication to local TwinCAT systems or remote TwinCAT systems via the TwinCAT Message router.
- Communication to remote TwinCAT systems via TCP/IP for Win32 Systems.

The TcAdsDll provides the TwinCAT Ads Client functions. These functions are provided in 2 different ways:

- By C API.
- By COM interfaces [31]

It is recommended to use the TwinCAT CP (royalty free) version with the library.
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 [43].

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 [27].

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 [27] 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.

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

#### Parameter
- **pAddr**
  - [in] Structure with NetId [27] 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.

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

#### Parameter
- **pAddr**
[in] Structure with NetId \[\text{27}\] and port number of the ADS server.

\textit{nIndexGroup}

[in] Index Group.

\textit{nIndexOffset}

[in] Index Offset.

\textit{nLength}

[in] Length of the data in bytes.

\textit{pData}

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

\textit{pcbReturn}

[out] Pointer to a variable. This variable returns the number of successfully read data bytes.

**Return value**

Returns the function's error status.

### 3.1.8 AdsSyncReadWriteReq

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

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

**Parameter**

\textit{pAddr}

[in] Structure with NetId \[\text{27}\] and port number of the ADS server.

\textit{nIndexGroup}

[in] Index Group.

\textit{nIndexOffset}

[in] Index Offset.

\textit{nReadLength}

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

\textit{pReadData}

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

\textit{nWriteLength}

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

\textit{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(
    PAmAddr pAddr,
    ULONG nIndexGroup,
    ULONG nIndexOffset,
    ULONG nReadLength,
    PVOID pReadData,
    ULONG nWriteLength,
    PVOID pWriteData,
    ULONG *pcbReturn
);  

Parameter

pAddr
[in] Structure with NetId [27] 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(
    PAmAddr pAddr,
    PCHAR pDevName,
    PAdsVersion pVersion
);
Parameter

pAddr

[in] Structure with NetId [27] 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 [28], 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(
    PAmAddr   pAddr,
    USHORT   nAdsState,
    USHORT   nDeviceState,
    ULONG    nLength,
    PVOID    pData
);

Parameter

pAddr

[in] Structure with NetId [27] and port number of the ADS server.

nAdsState


nDeviceState


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(
    PAmAddr pAddr,
    USHORT *pAdsState,
    USHORT pDeviceState
);

Parameter
pAddr
[in] Structure with NetId [27] and port number of the ADS server.

pAdsState
[out] Address of a variable that will receive the ADS status (see data type ADSSTATE [30]).

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(
    PAmAddr pAddr,
    ULONG nIndexGroup,
    ULONG nIndexOffset,
    PAdsNotificationAttrib pNoteAttrib,
    PAdsNotificationFuncEx pNoteFunc,
ULONG hUser,
PULONG pNotification
);

Parameter

pAddr

[in] Structure with NetId [\ref 27] and port number of the ADS server.

nIndexGroup

[in] IndexGroup.

nIndexOffset

[in] IndexOffset.

pNoteAttrib


pNoteFunc

[in] Name of the callback function [\ref 18].

hUser

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

pNotification

[ou] 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.

Example

See example 8 [\ref 48].

3.1.14 AdsSyncDelDeviceNotificationReq

A notification defined previously is deleted from an ADS server.

LONG AdsSyncDelDeviceNotificationReq(
    PAmAddr pAddr,
    ULONG hNotification
);

Parameter

pAddr

[in] Structure with NetId [\ref 27] and port number of the ADS server.

hNotification

[ou] 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.

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

Parameter

*nMs*  

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.

```c
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 AdsAmsUnRegisterRouterNotification() function. See also AdsAmsRegisterNotificationReq().

```c
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 PAmssRouterNotificationFuncEx

Type definition of the callback function required by the AdsAmsRegisterRouterNotification function.

```c
typedef void ( __stdcall *PAmssRouterNotificationFuncEx)( long nEvent );
```

3.1.19 PAdsNotificationFuncEx

Type definition of the callback function required by the AdsSyncAddDeviceNotificationReq function.

```c
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.

```c
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.

```c
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.

```c
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 [27].

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.

```c
LONG AdsSyncWriteReqEx(
    LONG port,
    PAmAddr 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 [27] 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.

```c
LONG AdsSyncReadReqEx2(
    LONG port,
    PAmAddr 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 [1.27] 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,
    PAmAddr 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 [1.18] or AdsPortOpen [1.9].

**pAddr**

[in] Structure with NetId [1.27] 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.

```c
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 [27] 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 [28], 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.

```c
LONG AdsSyncWriteControlReqEx(
    LONG port,
    PAmsAddr pAddr,
    USHORT nAdsState,
    USHORT nDeviceState,
```


### 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 [27] and port number of the ADS server.

**nAdsState**


**nDeviceState**


**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.

```c
LONG AdsSyncReadStateReqEx(
    LONG port,
    PAmsAddr pAddr,
    USHORT *pAdsState,
    USHORT *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 [27] and port number of the ADS server.

\textit{pAdsState}

[out] Address of a variable that will receive the ADS status (see data type \texttt{ADSSTATE [30]}).

\textit{pDeviceState}

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

\textbf{Return value}

Returns the function's error status.

\textbf{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.

\textbf{Example}

See example 4 [45].

\textbf{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).

\begin{verbatim}
LONG AdsSyncAddDeviceNotificationReqEx(
    LONG port,
    PAmsAddr pAddr,
    ULONG nIndexGroup,
    ULONG nIndexOffset,
    PAdsNotificationAttrib pNoteAttrib,
    PAdsNotificationFuncEx pNoteFunc,
    ULONG hUser,
    PULONG pNotification
);
\end{verbatim}

\textbf{Parameters}

\textit{port}

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

\textit{pAddr}

[in] Structure with NetId [27] and port number of the ADS server.

\textit{nIndexGroup}

[in] IndexGroup.

\textit{nIndexOffset}

[in] IndexOffset.

\textit{pNoteAttrib}


\textit{pNoteFunc}
[in] Name of the callback function \([\text{\ref{C_18}}]\).

\textit{hUser}

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

\textit{pNotification}

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

\textbf{Return value}

Returns the function's error status.

\textbf{Limitation:}

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

\textbf{Remarks}

If the TwinCAT router is stopped and then started again, the notifications become invalid. You can trap this event with the \textit{AdsAmsRegisterRouterNotification()} \([\text{\ref{C_17}}]\) function.

\textbf{Example}

See example 8 \([\text{\ref{C_48}}]\).

3.1.20.11 \quad \textbf{AdsSyncDelDeviceNotificationReqEx}

A notification defined previously is deleted from an ADS server.

\begin{verbatim}
LONG AdsSyncDelDeviceNotificationReqEx(
    LONG  port,
    PAmsAddr  pAddr,
    ULONG  hNotification
);
\end{verbatim}

\textbf{Parameters}

\textit{port}

[in] port number of an Ads port that had previously been opened with \textit{AdsPortOpenEx} \([\text{\ref{C_18}}]\) or \textit{AdsPortOpen} \([\text{\ref{C_9}}]\).

\textit{pAddr}

[in] Structure with NetId \([\text{\ref{C_27}}]\) and port number of the ADS server.

\textit{hNotification}

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

\textbf{Return value}

Returns the function's error status.

\textbf{Example}

See example 8 \([\text{\ref{C_48}}]\).
3.1.20.12 AdsSyncSetTimeoutEx
Alters the timeout for the ADS functions. The standard value is 5000 ms.

```c
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`
  

**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.

```c
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.

```c
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.

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

Elements

NetId

NetId [27].

port

Port number.

3.2.2 AmsNetId

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

```c
typedef struct {
    UCHAR b[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 [30]

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

ADSTRANS_SERVERONCHANGE: 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.

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

Using ADS Notifications
- 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.

```c
typedef struct {
    ULONG hNotification;
    __int64 nTimeStamp;
    ULONG cbSampleSize;
    UCHAR data[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_POWERFAIL = 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 [31] 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 a synchronized by a marshaler. The marhsaler is compiled into the TcAdsDll. No additional proxy-stub-dll is needed.

The TcClient [31] returns for each connection to one particular ADS device an object of the Type TcAdsSync [31]. This class provides synchronous ADS communication to the ADS device. The Class TcAdsSync [31] provides the communication function through the default interface ITcAdsSync [33]. To receive Ads Notification from the TcAdsSync [31] object the user program has to implement and connect the Event Interface ITcAdsSyncEvent [38].

![Diagram of TcAdsDll and its components]

4.1 Classes

4.1.1 TcAdsDll::Classes

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

<table>
<thead>
<tr>
<th>CoClasses</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcClient [31]</td>
<td>The Main class of the TcAdsDll. Provides a class factory to establish a ADS client connection</td>
</tr>
</tbody>
</table>

4.1.2 TcClient

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

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITcClient [32]</td>
<td>Interface provides a class factory</td>
</tr>
</tbody>
</table>

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.
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.

<table>
<thead>
<tr>
<th>IUnknown Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns a pointer to the interface you query for</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments the reference counter</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference counter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ITcClient Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect [32]</td>
<td>Creates an object of the type ITcAdsSync that provides Ads synchronous communication to one particular ADS device.</td>
</tr>
</tbody>
</table>

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.

\[ \text{HRESULT Connect(AmsNetId* pAmsNetId, long nPort, ITcAdsSync** pipTcAdsSync);} \]

**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 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 [33] pointer that holds the object that is used for Ads communication.

**Return Values**

- **S_OK** The connect function was successfully called.
- **ADSERRORCODES [39]** 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.

<table>
<thead>
<tr>
<th>IUnknown Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns a pointer to the interface you query for</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments the reference counter</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference counter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ITcAdsSync Methods and Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>[33]</td>
</tr>
<tr>
<td>Read</td>
<td>[34]</td>
</tr>
<tr>
<td>ReadWrite</td>
<td>[34]</td>
</tr>
<tr>
<td>WriteControl</td>
<td>[35]</td>
</tr>
<tr>
<td>AddDeviceNotification</td>
<td>[35]</td>
</tr>
<tr>
<td>DelDeviceNotification</td>
<td>[36]</td>
</tr>
<tr>
<td>ReadDeviceInfo</td>
<td>[36]</td>
</tr>
<tr>
<td>ReadState</td>
<td>[37]</td>
</tr>
<tr>
<td>Timeout</td>
<td>[37]</td>
</tr>
</tbody>
</table>

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 [ 39] 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 [ 39] 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.

cyRdLen [in] Count of byte we want to read from the variable.

cybRead [out] Pointer to a variable the returns the count of bytes we really had read.

pcbRead [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.

cyWrLen [in] Count of byte we want to write to the variable.

pcbWrData [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 [39] An error occurs

4.2.2.4 ITcAdsSync::WriteControl

This method sets the state of the ADS system an devices

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

Parameters

adsState [in] The states as ADSSTATE [41] we want to set onto the ADS system.

deviceState [in] The states as ADSSTATE [41] we want to set onto the ADS device.

cyLen [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 [39] 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 ADTRANSMODE [41].

nMaxDelay  [in]  The time with a resolution of 100 ns after we want to receive an callback on the implemented _ITcAdsSyncEvent [38] 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 [39]  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 phNotification);

Parameters

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

Return Values

S_OK  The function was successfully called

ADSERRORCODES [39]  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` that holds the version number.

Return Values

**S_OK**
The function was successfully called

**ADSErrorCodes**
An error occurs

### 4.2.2.8 ITcAdsSync::ReadState

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

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

Parameters

**pAdsState**
[out] Pointer to variable of the type `ADSSTATE` that holds the state of the ADS system.

**pDeviceState**
[out] Pointer to variable of the type `ADSSTATE` that holds the state of the ADS device.

Return Values

**S_OK**
The function was successfully called

**ADSErrorCodes**
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` interface.

Retrieve the Timeout value

```c
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**
An Error occurs

Assign a new Timeout value

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

Table 1: Parameters

<table>
<thead>
<tr>
<th><strong>nTime</strong></th>
<th>[in] A variable that holds the new timeout value</th>
</tr>
</thead>
</table>
Table 2: Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The function was successfully called</td>
</tr>
<tr>
<td>ADSERRORCODES</td>
<td>An Error occurs</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>IUnknown Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInterface</td>
<td>Returns a pointer to the interface you query for</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments the reference counter</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference counter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_ITcAdsSyncEvent Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceNotification</td>
<td>This method is called from the server for all connected variables of this Ads device</td>
</tr>
</tbody>
</table>

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 before.

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

**Parameters**

- **pTime** [in] A pointer to a variable of the type TimeStamp that holds the exact time when the connected variable was collected.
- **hNotification** [in] The handle that identifies one particular connected variable. The handle was returned when the variable was connected by a call to method AddDeviceNotification before.
- **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** An error occurs

4.3 Structures

4.3.1 AdsVersion

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

```c
struct AdsVersion
{
    BYTE version;
};
```
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.

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

4.4 Enums

4.4.1 ADSERRORCODES

The enumeration type ADSERRORCODE describes Ads errors with the following values:
<table>
<thead>
<tr>
<th>Const</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS_E_ERROR</td>
<td>0x98117000</td>
<td>this is the offset of Ads Errors presented a COM HRESULT</td>
</tr>
<tr>
<td>ADS_E_SRVNOTSUPP</td>
<td>0x98117001</td>
<td>the requested service is not supported by the ADS device</td>
</tr>
<tr>
<td>ADS_E_INVALIDGRP</td>
<td>0x98117002</td>
<td>invalided index group</td>
</tr>
<tr>
<td>ADS_E_INVALIDOFFSET</td>
<td>0x98117003</td>
<td>invalid index offset</td>
</tr>
<tr>
<td>ADS_E_INVALIDACCESS</td>
<td>0x98117004</td>
<td>reading an writing not permitted</td>
</tr>
<tr>
<td>ADSERR_DEVICE_INVALIDSIZE</td>
<td>0x98117005</td>
<td>parameter size is not correct</td>
</tr>
<tr>
<td>ADS_E_INVALIDDATA</td>
<td>0x98117006</td>
<td>invalided data value(s)</td>
</tr>
<tr>
<td>ADS_E_NOTREADY</td>
<td>0x98117007</td>
<td>device is not in a ready state</td>
</tr>
<tr>
<td>ADS_E_BUSY</td>
<td>0x98117008</td>
<td>device is busy</td>
</tr>
<tr>
<td>ADS_E_INVALIDCONTEXT</td>
<td>0x98117009</td>
<td>invalid context</td>
</tr>
<tr>
<td>ADS_E_NOMEMORY</td>
<td>0x9811700A</td>
<td>out of memory</td>
</tr>
<tr>
<td>ADS_E_INVALIDPARAM</td>
<td>0x9811700B</td>
<td>invalid parameter value(s)</td>
</tr>
<tr>
<td>ADS_E_NOTFOUND</td>
<td>0x9811700C</td>
<td>not found (files, ...)</td>
</tr>
<tr>
<td>ADS_E_SYNTAX</td>
<td>0x9811700D</td>
<td>syntax error in command or file</td>
</tr>
<tr>
<td>ADS_E_INCOMPATIBLE</td>
<td>0x9811700E</td>
<td>objects do not match</td>
</tr>
<tr>
<td>ADS_E_EXISTS</td>
<td>0x9811700F</td>
<td>object already exists</td>
</tr>
<tr>
<td>ADS_E_SYMBOLNOTFOUND</td>
<td>0x98117010</td>
<td>symbol not found</td>
</tr>
<tr>
<td>ADS_E_SYMBOLVERSIONINVALID</td>
<td>0x98117011</td>
<td>symbol version invalid</td>
</tr>
<tr>
<td>ADS_E_INVALIDSTATE</td>
<td>0x98117012</td>
<td>server is in invalid state</td>
</tr>
<tr>
<td>ADS_E_TRANSMODENOTSUPP</td>
<td>0x98117013</td>
<td>AdsTransMode not supported</td>
</tr>
<tr>
<td>ADS_E_NOTIFYHNDINVALID</td>
<td>0x98117014</td>
<td>Notification handle is invalid</td>
</tr>
<tr>
<td>ADS_E_CLIENTUNKNOWN</td>
<td>0x98117015</td>
<td>Notification client not registered</td>
</tr>
<tr>
<td>ADS_E_NOMOREHDLDS</td>
<td>0x98117016</td>
<td>no more notification handles</td>
</tr>
<tr>
<td>ADS_E_INVALIDWATCHSIZE</td>
<td>0x98117017</td>
<td>size for watch to big</td>
</tr>
<tr>
<td>ADS_E_NOTINIT</td>
<td>0x98117018</td>
<td>device not initialized</td>
</tr>
<tr>
<td>ADS_E_TIMEOUT</td>
<td>0x98117019</td>
<td>device has a timeout</td>
</tr>
<tr>
<td>ADS_E_NOINTERFACE</td>
<td>0x9811701A</td>
<td>query interface failed</td>
</tr>
<tr>
<td>ADS_E_INVALIDINTERFACE</td>
<td>0x9811701B</td>
<td>wrong interface required</td>
</tr>
<tr>
<td>ADS_E_INVALIDCLSID</td>
<td>0x9811701C</td>
<td>class ID is invalid</td>
</tr>
<tr>
<td>ADS_E_INVALIDOBJID</td>
<td>0x9811701D</td>
<td>object ID is invalid</td>
</tr>
<tr>
<td>ADS_E_CLIENT_ERROR</td>
<td>0x98117040</td>
<td>Error class: client error</td>
</tr>
<tr>
<td>ADS_E_CLIENT_INVALIDPARAM</td>
<td>0x98117041</td>
<td>invalid parameter at service call</td>
</tr>
<tr>
<td>ADS_E_CLIENT_LISTEMPTY</td>
<td>0x98117042</td>
<td>polling list is empty</td>
</tr>
<tr>
<td>ADS_E_CLIENT_VARUSED</td>
<td>0x98117043</td>
<td>var connection already in use</td>
</tr>
<tr>
<td>ADS_E_CLIENT_DUPINVOKED</td>
<td>0x98117044</td>
<td>invoke id in use</td>
</tr>
<tr>
<td>ADS_E_CLIENT_SYNCTIMEOUT</td>
<td>0x98117045</td>
<td>timeout elapsed</td>
</tr>
<tr>
<td>ADS_E_CLIENT_W32ERROR</td>
<td>0x98117046</td>
<td>error in win32 subsystem</td>
</tr>
<tr>
<td>ADS_E_CLIENT_TIMEOUTINVALID</td>
<td>0x98117047</td>
<td></td>
</tr>
<tr>
<td>ADS_E_CLIENT_PORTNOTOPEN</td>
<td>0x98117048</td>
<td></td>
</tr>
<tr>
<td>ADS_E_CLIENT_NOAMSADD</td>
<td>0x98117049</td>
<td></td>
</tr>
<tr>
<td>Const</td>
<td>Hex Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>ADS_E_CLIENT_SYNCINTERNAL</td>
<td>0x98117050</td>
<td>internal error in ads sync</td>
</tr>
<tr>
<td>ADS_E_CLIENT_ADDHASH</td>
<td>0x98117051</td>
<td>hash table overflow</td>
</tr>
<tr>
<td>ADS_E_CLIENT_REMOVEHASH</td>
<td>0x98117052</td>
<td>key not found in hash table</td>
</tr>
<tr>
<td>ADS_E_CLIENT_NOMORESYMBOLS</td>
<td>0x98117053</td>
<td>no more symbols in cache</td>
</tr>
</tbody>
</table>

### 4.4.2 ADSSTATE

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

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

```cpp
#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

<table>
<thead>
<tr>
<th>Description</th>
<th>Source text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1: Read DLL version</td>
<td>ADS-DLL_Sample01.zip (Resources/zip/7723063691.zip)</td>
</tr>
<tr>
<td>Example 2: Write flag synchronously into PLC</td>
<td>ADS-DLL_Sample02.zip (Resources/zip/7723065867.zip)</td>
</tr>
<tr>
<td>Example 3: Read flag synchronously from the PLC</td>
<td>ADS-DLL_Sample03.zip (Resources/zip/7723090999.zip)</td>
</tr>
<tr>
<td>Example 4: Read ADS status</td>
<td>ADS-DLL_Sample04.zip (Resources/zip/7723067531.zip)</td>
</tr>
<tr>
<td>Example 5: Read ADS information</td>
<td>ADS-DLL_Sample05.zip (Resources/zip/7723069195.zip)</td>
</tr>
<tr>
<td>Example 6: Start/stop PLC</td>
<td>ADS-DLL_Sample06.zip (Resources/zip/7723069195.zip)</td>
</tr>
<tr>
<td>Example 7: Access an array in the PLC</td>
<td>ADS-DLL_Sample07.zip (Resources/zip/7723072523.zip)</td>
</tr>
<tr>
<td>Example 8: Event driven reading</td>
<td>ADS-DLL_Sample08.zip (Resources/zip/7723074187.zip)</td>
</tr>
<tr>
<td>Example 9: Access by variable name</td>
<td>ADS-DLL_Sample09.zip (Resources/zip/7723075851.zip)</td>
</tr>
<tr>
<td>Example 10: Read PLC variable declaration</td>
<td>ADS-DLL_Sample10.zip (Resources/zip/7723090315.zip)</td>
</tr>
<tr>
<td>Example 11: Detect status change in TwinCAT router and the PLC</td>
<td>ADS-DLL_Sample11.zip (Resources/zip/7723094283.zip)</td>
</tr>
<tr>
<td>Example 12: Event-Driven Detection of Changes to the Symbol Table</td>
<td>ADS-DLL_Sample12.zip (Resources/zip/7723095947.zip)</td>
</tr>
<tr>
<td>Example 13: Reserved</td>
<td></td>
</tr>
<tr>
<td>Example 14: Reading the PLC variable declaration of an individual variable</td>
<td>ADS-DLL_Sample14.zip (Resources/zip/7723097611.zip)</td>
</tr>
<tr>
<td>Example 15: Reserved</td>
<td></td>
</tr>
<tr>
<td>Example 16: Reserved</td>
<td></td>
</tr>
<tr>
<td>Example 17: ADS-sum command: read or write</td>
<td>ADS-DLL_Sample17.zip (Resources/zip/7723115403.zip)</td>
</tr>
<tr>
<td>Example 18: ADS-sum command: Get and release</td>
<td>ADS-DLL_Sample18.zip (Resources/zip/7723117067.zip)</td>
</tr>
<tr>
<td>Example 19: Reserved</td>
<td></td>
</tr>
<tr>
<td>Example 20: Transmitting structures to the PLC</td>
<td>ADS-DLL_Sample20.zip (Resources/zip/7723118731.zip)</td>
</tr>
<tr>
<td>Example 21: Reading and writing of TIME/DATE variables</td>
<td>ADS-DLL_Sample21.zip (Resources/zip/7723120395.zip)</td>
</tr>
</tbody>
</table>

### 6.1 Read DLL version

Download: ‘ADS-DLL_Sample01.zip (Resources/zip/7723063691.zip)’

This program determines the version of the DLL file.

```cpp
#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 << '
';
    cout << "Revision: " << (int)pDLLVersion->revision << '
';
    cout << "Build: " << pDLLVersion->build << '
';
    cout.flush();
    getch();
}

6.2 Write flag synchronously into the PLC

Download: ADS-DLL_Sample02.zip (Resources/zip/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 << '
';

    // 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 << '
';

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

6.3 Read flag synchronously from the PLC

Download: ADS-DLL_Sample03.zip (Resources/zip/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;

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

    // Read value from PLC
    nErr = AdsSyncReadReq( pAddr, 0x4020, 0x0, 0x4, &dwData );
    if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '
';

    // Display value
    cout << "Value: ";
    cout << (int)dwData << '
';

    // Close communication port
    nErr = AdsPortClose();
    if (nErr) cerr << "Error: AdsPortClose: " << nErr << '
';
}
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() == ''); // Read the next value (use Carriage return as delimiter), stop otherwise
// Close communication port
nErr = AdsPortClose();
if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';

6.4  Read ADS status

Download: ADS-DLL_Sample04.zip (Resources/zip/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: ADS-DLL_Sample05.zip (Resources/zip/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.

```c
#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;
    PAmssAddr 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: ADS-DLL_Sample06.zip (Resources/zip/7723070859.zip).

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

```c
#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;
    PAmssAddr 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";
c = getch();
c = toupper(c);
while ((c == 'R') || (c == 'S'))
{
    switch (c)
    {
        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';
c = getch();
c = toupper(c);
}

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

6.7 Access an array in the PLC

Download: ADS-DLL_Sample07.zip (Resources/zip/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\AdsAppl\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsAppl\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, ADSGRP_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, ADSGRP_SYM_VALBYHND, lHdlVar, sizeof(Data), &Data[0]);
    if (nErr)
cerr << "Error: AdsSyncReadReq: " << nErr << '
';
else
{
    for (nIndex = 0; nIndex < 10; nIndex++)
        cout << "Data[" << nIndex << "]": " << Data[nIndex] << '
';
}
cout.flush();
getch();
// Close communication port
nErr = AdsPortClose();
if (nErr) cerr << "Error: AdsPortClose: " << nErr << '
';
}

6.8 Event driven reading

Download: ADS-DLL_Sample08.zip (Resources/zip/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(), 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 [28] 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() [29] 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\AdApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdApi\TcAdsDll\Include\TcAdsAPI.h"

using namespace std;

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

void main()
{
    long nErr, nPort;
    AmsAddr Addr;
    PAddr 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 << '
';

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

    // set the attributes of the notification
Samples

```cpp
adsNotificationAttrib.cbLength = 4;
adsNotificationAttrib.nTransMode = ADSTRANS_SERVERONCHA;
adsNotificationAttrib.nMaxDelay = 0;
adsNotificationAttrib.nCycleTime = 1000000; // 1sec

// get handle
nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(hUser), &hUser, sizeof(szVar), szVar);
if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '"' << 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 << '"' << nErr << '\n';
cout << "Notification: " << hNotification << '"' << nErr << '\n';
cout.flush();

// wait for user interaction (keystroke)
getch();

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

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

// Close the communication port
nErr = AdsPortClose();
if (nErr) cerr << "Error: AdsPortClose: " << nErr << '"' << 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 << '"' << nErr << '\n';
    cout << "Notification: " << pNotification->hNotification << '"' << nErr << '\n';

    // Convert the timestamp into SYSTEMTIME
    LargeInteger.QuadPart = pNotification->nTimeStamp;
    FileTime.dwLowDateTime = (DWORD)LargeInteger.LowPart;
    FileTime.dwHighDateTime = (DWORD)LargeInteger.HighPart;
    FileTimeToFileTime(&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 << '"' << nErr << '\n';

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

    // 32-Bit Variable (auch Zeiger), die beim AddNotification gesetzt wurde
    cout << "hUser: " << hUser << '"' << nErr << '\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 << '"' << nErr << '\n';
cout.flush();
```
6.9 Access by variable name

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.

```c
#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 << '
';

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

    // Fetch handle for an <szVar> PLC variable
    nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlVar), &lHdlVar,
        szVar, sizeof(szVar));
    if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '
';
    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 << '
';
        else
        { cout << "Wert: " << nData << '
';
            if (Data > 10)
                do
                {
                    // 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 << '
';
                }
            while ( getch() == ''); // 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 << '
';

            // Close communication port
            nErr = AdsPortClose();
            if (nErr)
                cerr << "Error: AdsPortClose: " << nErr << '
';
    }
}
```
6.10 Read PLC variable declaration

Download: ADS-DLL_Sample10.zip (Resources/zip/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() is used to obtain the length. The data itself is transferred with the second AdsSyncReadReq(). 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.

```c++
#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;
  PAddr pAddr = &Addr;
  AdsSymbolUploadInfo tAdsSymbolUploadInfo;
  PAdsSymbolEntry pAdsSymbolEntry;

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

  // 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 << '
';
  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 << '
';

  // Output information about the PLC variables
  pAdsSymbolEntry = (PAdsSymbolEntry)pchSymbols;
  for (uiIndex = 0; uiIndex < tAdsSymbolUploadInfo.nSymbols; uiIndex++)
    {
      cout << PADSSYMBOLNAME(pAdsSymbolEntry) << "\t"
           << pAdsSymbolEntry->iGroup << "\t"
           << pAdsSymbolEntry->iOffs << "\t"
           << pAdsSymbolEntry->size << "\t"
           << PADSSYMBOLTYPE(pAdsSymbolEntry) << "\n"
           << PADSSYMBOLCOMMENT(pAdsSymbolEntry) << "";
      pAdsSymbolEntry = PADSNEXTSYMBOLENTRY(pAdsSymbolEntry);
      cout.flush();
    }

  // Close communication port
}
```
6.11 Detect status change in TwinCAT router and the PLC

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.

```c
#include <iostream.h>
#include <conio.h>
#include <windows.h>
#include <winbase.h>
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsAPI.h"

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

// ADS headers for TwinCAT 3

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; // jde 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;
}
```
6.12 Event-Driven Detection of Changes to the Symbol Table

Download: ADS-DLL_Sample12.zip (Resources/zip/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 [p. 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.

```cpp
#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;
    PAmAddr pAddr = &Addr;
    ULONG hNotification;
    AdsNotificationAttrib adsNotificationAttrib;

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

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

// Specify attributes of the notification
adsNotificationAttrib.chLength = 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 << '
';

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

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

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

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

6.13 Reading the PLC variable declaration of an individual variable

Download: ADS-DLL_Sample14.zip (Resources/zip/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.

```c
#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,
```
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" <<endl;
        AdsSyncReadReq(pAddr, pAdsSymbolEntry->iGroup, pAdsSymbolEntry->iOffs, pAdsSymbolEntry->size, pVal);
        if( nErr )
        {
            cerr << "Error: AdsSyncReadReq: Unable to read Value" << nErr << endl;
        }
        else
        {
            cout << "Value: ";
            for( int i=0; i<nElements; i++ )
            {
                cout << pVal[i] << 't';
            }
            cout << endl;
        }
        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 <<endl;
                break;
            }
        }
        if( i == nType )
        {
            cout << "Datatype:" << "Unknown datatype:" << pAdsSymbolEntry->dataType <<endl;
        }
        break;
    }
}( getch() == 'y' )
break;
// Close communication port
nErr = AdsPortClose();
if (nErr) cerr << "Fehler: AdsPortClose: " << nErr << endl;
}

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

Download: ADS-DLL_Sample15.zip (Resources/zip/7723099275.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

NOTE

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":

```c
// Read major symbol information via ADS from device
AdsSymbolUploadInfo2 info;
if (nResult == ADS_ERR_NOERR)
{
    // size of symbol information
    PBYTE pSym = new BYTE[info.nSymSize];
    if (pSym)
    {
        // upload symbols (instances)
        nResult = AdsSyncReadReq(m_amsAddr, ADSIGRP_SYM_UPLOAD, 0, info.nSymSize, pSym);
        if (nResult == ADSERR_NOERR)
        {
            // get size of datatype description
            PBYTE pDT = new BYTE[info.nDatatypeSize];
            if (pDT)
            {
                // upload datatype-descriptions
                nResult = AdsSyncReadReq(m_amsAddr, ADSIGRP_SYM_DT_UPLOAD, 0, info.nDatatypeSize, 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: ADS-DLL_Sample17.zip (Resources/zip/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 single ADS call multiple variables which are not structured within a linear memory.

As a result the ADS caller application (like scada Systems etc.) can extremely 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!

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

Note

// 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

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

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
&parReq);
6.16 ADS-sum command: Get and release several handles

Download: ADS-DLL_Sample18.zip (Resources/zip/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.

   ```cpp
   #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.

   ```cpp
   // Structure declaration for values
typedef struct dataReq
   {
     unsigned long indexGroup; // index group in ADS server interface
     unsigned long indexOffset; // index offset in ADS server interface
     unsigned long rlength; // count of bytes to read
     unsigned long wlength; // count of bytes to write
   }TDataReq, *PTDataReq;

   // Variables declaration
   AmsAddr Addr;
   LONG nErr, nPort;
   PAmssAddr 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.

   ```cpp
   // 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;
   ```
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.

2. Release handles

Define a structure, declare variables and allocate memory again.
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";
}
6.17 Transmitting structures to the PLC

Download: ADS-DLL_Sample20.zip (Resources/zip/7723118731.zip)

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

```c
#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 += 1;
    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();

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

    cout.flush();
    // Wait for key press
    getch();
    }
```
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
0x13, // 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

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:

```c
#include <stdio.h>
#include <windows.h>
#include <tchar.h>
#include <time.h>
#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
&wTime);
if(lErr) printf_s((char*)"Error: Read time variable: 0x%i 
", 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;

wprintf_s(szTime, TIME_LENGHT);  
wsprintf(szTime, L"Time from PLC: %dd %dh %dm %ds %dms 
", 
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, 
&wTime);
if(lErr) printf_s((char*)"Error: Write time variable: 0x%i 
", lErr);

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

//Convert long to date
ttime_t tDate(dwDate);
tm tmDate;
gmtime_s(&tmDate, &tDate);

wprintf_s(szDate, DATE_LENGHT);  
wsprintf(szDate, L"Date from PLC: %d/%d/%d %d:%d:%d 
", 
tmDate.tm_mon + MON_START, 
tmDate.tm_year + YEAR_START, 
tmDate.tm_min, 
tmDate.tm_mday, 
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, 
&wDate);
if(lErr) printf_s((char*)L"Error: Write time variable: 0x%i 
", 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();
}