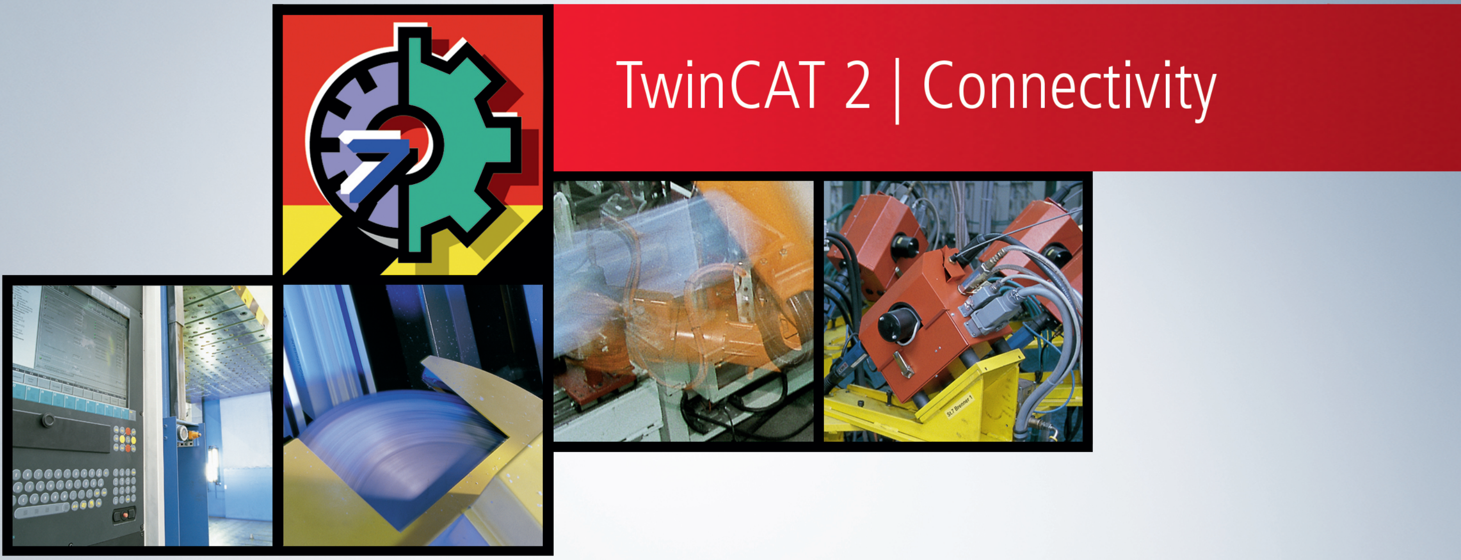
BECKHOFF New Automation Technology

Manual | EN

TX1000

TwinCAT 2 | ADS OCX



2022-10-06 | Version: 1.1

Table of contents

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:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
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!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

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 Access to the ADS devices

There are several methods of accessing the data of an ADS device:

- synchron
- asynchron
- cyclical

Depending on the application environment (communication medium, quantity of data, data transmission rate,...) each method has certain advantages, and these are explained in detail further below. There are two further variations that can be used to identify a variable in an ADS device.

By address

An address is given. The address is composed of the index group and the index offset. The address assignment is described in the corresponding documentation for the ADS device.

By variable name

As an alternative, the name of an ADS variable can be given when accessing an ADS device.

Synchron [▶ 11]

Once the write/read method has been called, the execution of the Visual Basic program is interrupted until the requested data is available. In the following instructions it is possible to continue working with the data immediately. The advantage of this access method is that very little programming effort has to be carried out in the Visual Basic program.
This access method is recommended if the Visual Basic program and the ADS device are on the same computer or are connected via a fast network so that the waiting time is very short.
Example: The operator is to enter various parameters in an input window. The data is to be written to the PLC when a button is clicked. Since the writing of the values is not performed cyclically, but in a manner that depends on the user's behavior, a synchronous write command should be used in this case.

Asynchron [▶ 11]

In the case of asynchronous access, the execution of the Visual Basic program is not interrupted, but continues immediately with the next command. When the requested data arrive at the ADS-OCX, an event function is triggered in the Visual Basic program, in which the value is passed as a parameter. By the fact that the Visual Basic program can receive its data at any time, a larger programming effort is necessary there than with the synchronous access method.
If the ADS server and the Visual Basic program are spatially separated from each other and the data transmission medium is very slow, e.g. modem or ISDN, then the asynchronous mode of operation makes sense.

Connect [▶ 11]

If values are to be transmitted continuously to a Visual Basic program, cyclical access, also known as 'by connect', is the easiest and most effective method. Calling the method results in the data from the ADS device being sent to the Visual Basic program cyclically or when there is a change, using an event function.

Example: The positions of multiple axes are to be shown in a display window, updated every 250 ms. Use the AdsReadVarConnectEx() [▶ 34] method, so that every 250 ms the AdsReadConnectUpdateEx() [▶ 53] event is triggered for each axis position. This principle can be further optimized, so that values are only transferred if the position of the axis changes (server on change)! There is a simple sample of this under 'Event-driven reading' [▶ 74].

Return values

All these methods return a value indicating whether the operation was carried out successfully, or whether an error occurred. It can generally be said that a return value of 0 indicates error-free execution. A detailed list of the possible return values and their meanings can be found under ADS error codes.

Alternatively the ADS-OCX can trigger an exception in case of an error. This requires the EnableErrorHandling [▶ 63] property to be set to TRUE. The cause of the error can then be determined via the *Err* object. The *Err* object is described in the Visual Basic documentation.

Method summary

		by address	**by variable name**
synchron	**Reading**	AdsSyncReadReq() [▶ 26] AdsSyncReadBoolReq() [▶ 27] AdsSyncReadIntegerReq() [▶ 27] AdsSyncReadLongReq() [▶ 27] AdsSyncReadSingleReq() [▶ 27] AdsSyncReadDoubleReq() [▶ 27] AdsSyncReadStringReq() [▶ 27]	- AdsSyncReadBoolVarReq() [▶ 24]AdsSyncReadIntegerVarReq() [▶ 24]AdsSyncReadLongVarReq() [▶ 24]AdsSyncReadSingleVarReq() [▶ 24]AdsSyncReadDoubleVarReq() [▶ 24]AdsSyncReadStringVarReq() [▶ 24]
	Writing	AdsSyncWriteReq() [▶ 29] AdsSyncWriteBoolReq() [▶ 30] AdsSyncWriteIntegerReq() [▶ 30] AdsSyncWriteLongReq() [▶ 30] AdsSyncWriteSingleReq() [▶ 30] AdsSyncWriteDoubleReq() [▶ 30] AdsSyncWriteStringReq() [▶ 30]	- AdsSyncWriteBoolVarReq() [▶ 28] AdsSyncWriteIntegerVarReq() [▶ 28] AdsSyncWriteLongVarReq() [▶ 28] AdsSyncWriteSingleVarReq() [▶ 28] AdsSyncWriteDoubleVarReq() [▶ 28]AdsSyncWriteStringVarReq() [▶ 28]
asynchron	**Reading**	AdsReadIntegerReq() [▶ 31] AdsReadLongReq() [▶ 31] AdsReadSingleReq() [▶ 31] AdsReadDoubleReq() [▶ 31] AdsReadStringReq() [▶ 31]	-
	Writing	AdsWriteIntegerReq() [▶ 33] AdsWriteLongReq() [▶ 33] AdsWriteSingleReq() [▶ 33] AdsWriteDoubleReq() [▶ 30] AdsWriteStringReq() [▶ 33]	-
connect	**Reading**	AdsReadConnect() [▶ 40] AdsReadBoolConnect() [▶ 42] AdsReadIntegerConnect() [▶ 42] AdsReadLongConnect() [▶ 42] AdsReadSingleConnect() [▶ 42] AdsReadDoubleConnect() [▶ 42] AdsReadStringConnect() [▶ 42]	AdsReadVarConnectEx() [▶ 34]
	Writing	AdsWriteConnect() [▶ 48] AdsWriteBoolConnect() [▶ 49] AdsWriteIntegerConnect() [▶ 49] AdsWriteLongConnect() [▶ 49] AdsWriteSingleConnect() [▶ 49] AdsWriteDoubleConnect() [▶ 49]	AdsWriteVarConnect() [▶ 45] AdsWriteBoolVarConnect() [▶ 47] AdsWriteIntegerVarConnect() [▶ 47] AdsWriteLongVarConnect() [▶ 47] AdsWriteSingleVarConnect() [▶ 47] AdsWriteDoubleVarConnect() [▶ 47]

3 Manual installation of the ADS OCX

The ADS OCX can be installed with Regsvr32.

✓ The path to the file that is to be registered must be stated.

1. Select **Start > Run**
2. Enter *Regsvr32 <path to AdsOcx file>\AdsOcs.ocx*.

⇨ ADS OCX has been inserted with Regsvr32.

It is not necessary to reboot the computer.

4 API

4.1 general

4.1.1 AboutBox

Displays an information window with the current version number and the copyright declaration of the ADS-OCX.

```
object.AboutBox( )
```

Parameter

-

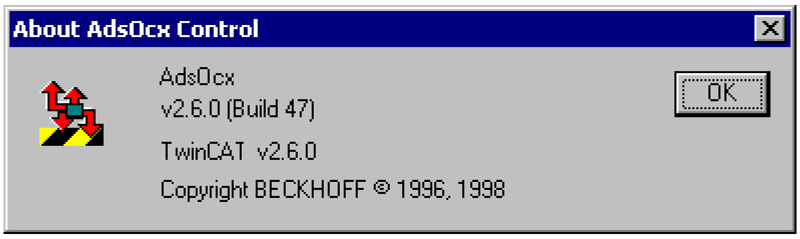
Return value

-

Comments

-

Example


About AdsOcx Control

AdsOcx
v2.6.0 (Build 47)
TwinCAT v2.6.0
Copyright BECKHOFF © 1996, 1998

4.1.2 AdsAmsDisconnect

This method is used to disconnect the ADS-OCX from the TwinCAT Router.

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

Parameter

-

Return value

-

Comments

All applications are closed when the present user logs out from Windows NT/2000/XP. If the program contains the ADS-OCX connected to the router, the program must disconnect from the TwinCAT Router. If this is not done, the program cannot be completely unloaded, and will still be seen in the NT Task Manager after a new login.

The disconnection from the TwinCAT Router is achieved through the *AdsAmsDisconnect()* method. This should be called in the *Form_Unload()* event.

Example

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

4.1.3 AdsAmsPortEnabled

This method can be used to determine whether the AMS port is available for communication.

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

Parameter

-

Return value

-

Comments

-

Example

The following sample illustrates a function in which messages are written to the Windows NT/2000/XP Event Viewer. The method AdsLogFmtString() [▶ 19] is used for this. Since the access to the Event Viewer is made via the TwinCAT Router, the method should only be used if the AMS port is active.

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

4.1.4 AdsCreateVarHandle

Generates a unique handle for an ADS variable.

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

Parameter

varName

[in] Name of the ADS variable

hVar

[out] Handle of the ADS variable

Return value

See ADS error codes

Comments on the PLC:

Enable Symbol download

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable 'SPSVar1' from the function 'Funk1' is to be accessed, then 'Funk1.SPSVar1' must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in '.SPSGlobVar'. The parameter 'varName' does not distinguish between upper and lower case letters. If only certain specific PLC variables are required in a form, the handle should only be created when the form is loaded, and should be released again when the form is closed. See also the AdsCreateVarHandle() [▶ 15] method.

Comments on the NC:

NOTE

Enable Symbol download at each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

Example

-

4.1.5 AdsDeleteVarHandle

Releases the handle of a PLC variable again.

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

Parameter

hVar

[in] Handle of the ADS variable

Return value

See ADS error codes

Comments

If the ADS variable that is referred to by a handle is no longer required, it should be released once more by means of the AdsDeleteVarHandle() method. If only certain specific ADS variables are required in a form, the handle should only be created when the form is loaded, and should be released again when the form is closed. See also the AdsCreateVarHandle() [▶ 14] method.

Example

-

4.1.6 AdsEnableLogNotification

Sets the filter for the reception of messages via the TwinCAT Router.

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

Parameter

nBasePort

[in] First port number for which the AdsLogNotification() [▶ 52] event is triggered

nPorts

[in] Number of ports starting from *nBasePort* for which the AdsLogNotification() [▶ 52] event is triggered

dwCtrlMask

[in] Filter mask for the kinds of messages that are to be reported (see ADSLOGMSGTYPE [▶ 65] data type)

Return value

See ADS error codes

Comments

ADS devices are able to send messages to other ADS devices via the TwinCAT Router. Before an ADS device is able to receive messages with the aid of the ADS-OCX, the AdsEnableLogNotification() method must be used to define a filter. This defines which messages are to be reported.

One of a filter's functions is to define a range of port numbers. All the messages from the ADS devices that lie within this range of port numbers will be reported by the AdsLogNotification() [▶ 52] event.

The second parameter with which messages can be filtered is the message type. A distinction is made between note, warning and error (see ADSLOGMSGTYPE [▶ 65]). Various other types of message can be received by using a OR combination.

Example

Visual Basic sample: 'Send/receive messages via the TwinCAT Router [▶ 81]'

4.1.7 AdsEnumSymbols

The list of declared variables can be read from an ADS device with this method.

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

Parameter

strSymbolName

[out] Name of the ADS variable

nSymbolType

[out] Data type of the ADS variable (see the ADSDATATYPEID [▶ 64] data type)

cbSymbolSize

[out] Data length of the ADS variable in bytes

strComment

[out] Comment following the ADS variable declaration

nIndexGroup

[out] Index group of the ADS variable

nIndexOffset

[out] Index offset of the ADS variable

bNext

[in] TRUE for the first ADS variable, FALSE for all those which follow

Return value

See ADS error codes

Comments

When the AdsEnumSymbols() method is first called, you must set the *bNext* parameter to FALSE. This causes all the information about the first variable to be read. Every time AdsEnumSymbols() is called after this, the parameter must be TRUE. This causes the information about the following variable to be read.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

Example

Visual Basic sample: 'Read PLC variable declaration [▶ 76]'

4.1.8 AdsSetFirstDynSymbol

The list of declared variables can be read from an ADS device with this method.

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

Parameter

bForceReload

[in] TRUE if a (new) loading of the symbol information from the server is desired. If no symbol information is available yet, it will be loaded independently from *bForceReload*.

Return value

See ADS error codes

Comments

On the method call of AdsSetFirstDynSymbol() the internal "pointer" to the current symbol, which can be loaded with AdsGetNextDynSymbol [▶ 18](), is set back to the beginning.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

Example

Visual Basic sample: 'Read PLC variable declaration [▶ 76]'

4.1.9 AdsGetNextDynSymbol

The list of declared variables can be read from an ADS device with this method.

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

Parameter

navType

[in] Navigation preset in the symbol tree (see data type ADSGETDYNSYMBOLTYPE [▶ 66])

bstrName

[out] Name of the symbol (short form without prefixed names of the parent)

bstrFullName

[out] Full name of the symbol

bstrType

[out] Name of the data type of the symbol

strComment

[out] Comment following the ADS variable declaration

adsType

[out] Data type of the ADS variable (see the ADSDATATYPEID [▶ 64] data type)

symbolSize

[out] byte length of the symbol

nIndexGroup

[out] Index group of the ADS variable

nIndexOffset

[out] Index offset of the ADS variable

Return value

See ADS error codes

Comments

At *navType* **ADSDYNSYM_GET_NEXT** the entire symbol tree is navigated. Hereby all symbols can be read out in a simple way. The other three *navTypes* can be used for controlled navigation through the symbol tree.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

Example

Visual Basic sample: 'Read PLC variable declaration [▶ 76]'

4.1.10 AdsLogFmtString

Issues a message via the TwinCAT Router.

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

Parameter

nMsgType

[in] Type of message (see the ADSLOGMSGTYPE [▶ 65] data type)

strFmt

[in] The message text that is to be issued

arg0

[in] 1th parameter in the message text

arg1

[in] 2nd parameter in the message text

arg2

[in] 3rd parameter in the message text

arg3

[in] 4th parameter in the message text

Return value

See ADS error codes

Comments

The issued message is reported to all the ADS devices in which the filter conditions are satisfied. The issued message is also written into the Windows NT/2000/XP Event Logger.

There are three types of messages: Note, Warning and Error. The message that is issued must belong to one of these three types. Up to four numeric parameters can be specified in the message string. The following letters can be used as placeholders:

Placeholder	Meaning
%d	Placeholder for a variable of type long/integer
%f	Placeholder for a variable of type single/double
%x	Placeholder for a variable of type hexadecimal
%X	Placeholder for a variable of type hexadecimal

The first placeholder is then occupied by the first parameter (arg0), the second placeholder with the second parameter (arg1), and so on.

NOTE

Too many messages in a short time

Make sure that not too many messages are transmitted in a short time, otherwise this could affect the overall system.

Log messages

If you want to keep a log of messages in your program (e.g. malfunctions in a machine) you should make use of the TwinCAT Event Logger. This is significantly more powerful than the Windows NT/2000/XP Event Logger, and is adapted to the requirements of automation technology.

Example

Visual Basic sample: 'Send/receive messages via the TwinCAT Router [▸ 81]'

4.1.11 AdsReadSymbolDesc

The AdsReadSymbolDesc() method can be used to obtain information about the individual symbols (variables) in ADS devices.

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

Parameter

strSymbolName

[in] Name of ADS variable from which the information is to be read

nSymbolType

[out] Data type of the ADS variable (see the ADSDATATYPID [▸ 64] data type)

cbSymbolSize

[out] Data length of the ADS variable in bytes

strComment

[out] Comment following the ADS variable declaration

nIndexGroup

[out] Index group of the ADS variable

nIndexOffset

[out] Index offset of the ADS variable

Return value

See ADS error codes

Comments

If you wish to read the information for all the ADS variables from an ADS device, you will find a relevant sample under 'Read PLC variable declaration' [▸ 76].

NOTE

Enable the Symbol download at PLC Control

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

Example

-

4.1.12 AdsReadSymbolInfo

The AdsReadSymbolInfo() method can be used to obtain information about the symbols (variables) in ADS devices.

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

Parameter

pSymbolsAvailable

[out] Number of symbols in the ADS device

pBufSizeNeeded

[out] Length of the data, in bytes, in which the symbol information is to be stored

Return value

See ADS error codes

Comments

Before the AdsEnumSymbols() [▸ 16] method can be used to read the symbol list, the method AdsReadSymbolInfo() must be used to find the number of symbols and the size of the symbol list.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

Example

Visual Basic sample: 'Read PLC variable declaration [▸ 76]'

4.1.13 AdsSyncWriteControlReq

Changes the state of an ADS device.

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

Parameter

ADSSTATE

[in] New state of the ADS device (see the ADSSTATE [▸ 65] data type)

deviceState

[in] Reserved

length

[in] Length of the data in bytes

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return parameter

See ADS error codes

Comments

As well as changing the ADS state, it is also possible to send data to the ADS device. Whether such data is evaluated, and how, depends on the individual ADS devices. The ADS devices supplied with TwinCAT (PLC, NC/NCI, camshaft controller, ...) do not evaluate such information.

Example

Visual Basic sample: 'Detect/alter state change in TwinCAT Router and the PLC [▶ 79]'

4.1.14 AdsWriteControlReq

Changes the ADS state and the device state of the ADS server.

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

Parameter

nInvokeId

[in] Job number for identification of the response

nAdsState

[in] New ADS state (see the ADSSTATE [▶ 65] data type)

nDeviceState

[in] New device state

cbLength

[in] Length of the data in bytes

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

In addition to changing the ADS state and the device state, it is also possible to send data to the ADS server in order to transfer further information. For the current ADS devices (PLC, NC, ...) this data is not evaluated further.
Each ADS device can communicate its current state to other ADS devices. A distinction is made between the state of the device itself (DeviceState) and the state 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

-

4.1.15 ShowPropertyPages

Displays the ADS-OCX properties window.

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

Parameter

-

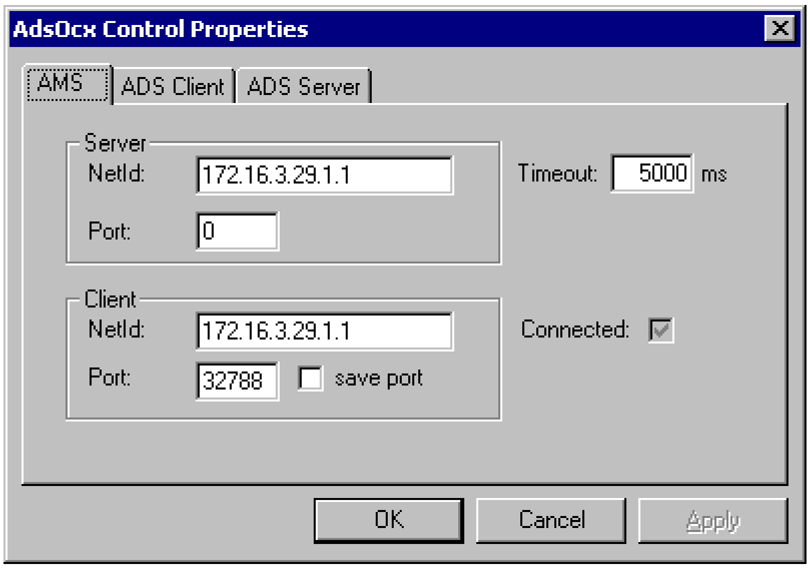
Return value

See ADS error codes

Comments

-

Example



Server
NetId: 172.16.3.29.1.1
Port: 0
Timeout: 5000 ms
Client
NetId: 172.16.3.29.1.1
Port: 32788
[ ] save port
Connected: [✓]

4.2 synchron

4.2.1 AdsSyncRead[Datatype]VarReq

AdsSyncReadBoolVarReq

AdsSyncReadIntegerVarReq

AdsSyncReadLongVarReq

AdsSyncReadSingleVarReq

AdsSyncReadDoubleVarReq

AdsSyncReadStringVarReq

Reads data synchronously from an ADS device, and writes it into a Visual Basic variable of type boolean, integer, long, single, double or string.

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

Parameter

hVar

[in] Handle of the ADS variable (see the AdsCreateVarHandle() [▸ 14] method)

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

NOTE

VB variable is set to "0

In case of an error the VB variable (pData), whose value should be written, is set to "0".

Comments

The execution of the Visual Basic program is stopped until the data from the ADS device is available or until the time in the property AdsAmsCommTimeout [▸ 59] is exceeded.
Note on the String data type: When specifying the length of the data, note that it refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

VB sample

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

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

Delphi sample

```
procedure TForm1.Button1Click(Sender: TObject);
var     res1, res2, res3 :integer;
    //handles
    hBoolean, hSmallint, hLongint, hSingle, hDouble, hString : integer;
    //read buffer
    vWordBool : WordBool;
    vSmallint : Smallint;
    vLongint : Longint;
    vSingle : Single;
    vDouble : Double;
    vString : WideString;
begin
    res1 := AdsOcx1.AdsCreateVarHandle( 'MAIN.vBOOL', hBoolean );
    res2 :=  AdsOcx1.AdsSyncReadBoolVarReq( hBoolean, sizeof(vWordBool), vWordBool );
    res3 := AdsOcx1.AdsDeleteVarHandle( hBoolean );
    Label1.Caption := Format('res1: %d, res2: %d, res3: %d,  Value: %s', [res1, res2, res3, BoolToSt
r(vWordBool, TRUE)]);
```

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

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

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

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

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

4.2.2 AdsSyncReadReq

Reads data of any type synchronously from an ADS device.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

NOTE

VB variable is set to "0

In case of an error the VB variable (pData), whose value should be written, is set to "0".

Comment

The execution of the Visual Basic program is stopped until the data from the ADS device is available or until the time in the property AdsAmsCommTimeout [▶ 59] is exceeded.
The Visual Basic variable must be declared as an array. The entire array is passed to the method.
The variable type string is not supported.

Example

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

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

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

4.2.3 AdsSyncRead[Datatype]Req

AdsSyncReadBoolReq

AdsSyncReadIntegerReq

AdsSyncReadLongReq

AdsSyncReadSingleReq

AdsSyncReadDoubleReq

AdsSyncReadStringReq

Reads data synchronously from an ADS device, and writes it into a Visual Basic variable of type boolean, integer, long, single, double or string.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

NOTE

VB variable is set to "0

In case of an error the VB variable (pData), whose value should be written, is set to "0".

Comments

The execution of the Visual Basic program is stopped until the data from the ADS device is available or until the time in the property AdsAmsCommTimeout [▸ 59] is exceeded.
Note on the String data type: When specifying the length of the data, note that it refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

VB sample

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

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

Delphi sample

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

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

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

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

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

    SetLength(vString,80{standard length of the PLC string variable});
    res :=  AdsOcx1.AdsSyncReadStringReq( $4020, 64, Length(vString)*2{byte length!}, vString );
    Label6.Caption := Format('res: %d,  Length: %d, Value: %s', [res, Length(vString), vString]);
end;
```

4.2.4 AdsSyncWrite[Datatype]VarReq

AdsSyncWriteBoolVarReq

AdsSyncWriteIntegerVarReq

AdsSyncWriteLongVarReq

AdsSyncWriteSingleVarReq

AdsSyncWriteDoubleVarReq

AdsSyncWriteStringVarReq

Requests data synchronously from an ADS device, and writes it into a Visual Basic variable of type boolean, integer, long, single, double or string.

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

Parameter

hVar

[in] Handle of the ADS variable (see the AdsCreateVarHandle() [▶ 14] method)

length

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

The execution of the Visual Basic program is stopped until the data from the ADS device is available or until the time in the property AdsAmsCommTimeout [▶ 59] is exceeded.
Note on the String data type: When specifying the length of the data, note that it refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

Example

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

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

4.2.5 AdsSyncWriteReq

Writes data of any type synchronously to an ADS device.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

The execution of the Visual Basic program is stopped until the ADS device has received the data or until the time in the AdsAmsCommTimeout [▸ 59] property is exceeded.
The Visual Basic variable must be declared as an array. The entire array is passed to the method.
The variable type string is not supported.

Example

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

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

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

4.2.6 AdsSyncWrite[Datatype]Req

AdsSyncWriteBoolReq

AdsSyncWriteIntegerReq

AdsSyncWriteLongReq

AdsSyncWriteSingleReq

AdsSyncWriteDoubleReq

AdsSyncWriteStringReq

Writes data synchronously from a Visual Basic variable of type boolean, integer, long, single, double or string into a data item of an ADS device.

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

Parameter

indexGroup

[in] Index group of the ADS variable

indexOffset

[in] Index offset of the ADS variable

length

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

The execution of the Visual Basic program is stopped until the ADS device has received the data or until the time in the property AdsAmsCommTimeout [▶ 59] is exceeded.
Note on the String data type: When considering the length of the data, note that it refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

Example

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

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

4.3 asynchron

4.3.1 AdsRead[Datatype]Req

AdsReadIntegerReq

AdsReadLongReq

AdsReadSingleReq

AdsReadDoubleReq

AdsReadStringReq

Issues a read request for a data item of type integer, long, single, double or string.

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

Parameter

nInvokeId

[in] Job number for identification of the response

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

Return value

See ADS error codes

Comments

Once a read request has been sent to the ADS device, execution of the Visual Basic program continues. As soon as the data is available, the ADS-OCX triggers the event function AdsRead[Datatype]Conf() [▸ 55] with which the requested data is transmitted.
When the read request is sent, an identification number must be specified, which is later returned when the event function is called. This allows an assignment between Read-Request and the event function.
Note on the data type String: It should be noted that the length of the data refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

Example

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

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

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

Private Sub AdsOcx1_AdsReadStringConf(ByVal nInvokeId As Long, ByVal nResult As Long, ByVal cbLength
 As Long, ByVal pData As String)
    If (nInvokeId = 1) And (nResult = 0) Then
    'Daten anzeigen
    Label1.Caption = pData
    End If
End Sub
```

4.3.2 AdsWrite[Datatype]Req

AdsWriteIntegerReq

AdsWriteLongReq

AdsWriteSingleReq

AdsWriteDoubleReq

AdsWriteStringReq

Issues a read request for a data item of type integer, long, single, double or string.

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

Parameter

nInvokeId

[in] Job number for identification of the response

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

Once the write request has been sent to the ADS device, execution of the Visual Basic program continues. As soon as the data has been written, the ADS-OCX triggers the AdsWriteConf() [▸ 58] event function.

When a write request is issued, an identification number, which is later returned when the event function is called, must also be provided. This makes it possible to assign the event function to the appropriate write request.

Note on the string data type: When specifying the length of the data, note that it refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

Example

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

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

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

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

4.4 connect

4.4.1 AdsReadVarConnectEx

Creates a fixed connection between a Visual Basic variable and a data item from an ADS device.

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

Parameter

adsVarName

[in] Name of the ADS variable

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▶ 65] data type)

nCycleTime

[in] Read cycle in ms

phConnect

[out] Contains a unique handle for the connection that has been established (this is not the handle of the ADS variable!).

hUser

[in] Optional: This value is passed when the AdsReadConnectUpdateEx() [▶ 53] event is called.

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be released using the AdsDisconnectEx() [▶ 40] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable *'SPSVar1'* from the function *'Funk1'* is to be accessed, then *'Funk1.SPSVar1'* must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in *'.SPSGlobVar'*. The parameter *adsVarName* does not distinguish between upper and lower case letters.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialogue for the axis under General. The 'Create symbols' box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

NOTE

Parameters not passed correctly under Borland Delphi

When calling the corresponding event function AdsReadConnectUpdateEx(), the OleVariant parameters are not passed to the Delphi application correctly. The method AdsReadVarConnectEx2() with its associated event function AdsReadConnectUpdateEx2() provides the same functionality as the method AdsReadVarConnectEx/AdsReadConnectUpdateEx. Please use this method/event in Delphi applications. In Visual Basic applications both methods can be used.

Example

Visual Basic sample: 'Event-driven reading' [▶ 74]

Also see about this

- AdsReadVarConnectEx2 [▶ 35]
- AdsReadConnectUpdateEx2 [▶ 54]
- AdsReadConnectUpdateEx [▶ 53]

4.4.2 AdsReadVarConnectEx2

Creates a fixed connection between a Visual Basic variable and a data item from an ADS device.

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

Parameter

adsVarName

[in] Name of the ADS variable

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▶ 65] data type)

nCycleTime

[in] Read cycle in ms

phConnect

[out] Contains a unique handle for the connection that has been established (this is not the handle of the ADS variable!).

hUser

[in] Optional: This value is passed when the AdsReadConnectUpdateEx2() [▶ 54] event is called.

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be released using the AdsDisconnectEx() [▶ 40] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable *'SPSVar1'* from the function *'Funk1'* is to be accessed, then *'Funk1.SPSVar1'* must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in *'.SPSGlobVar'*. The parameter *adsVarName* does not distinguish between upper and lower case letters.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialogue for the axis under General. The Create symbols box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

NOTE

Parameters not passed correctly under Borland Delphi

When calling the AdsReadConnectUpdateEx() event function, the OleVariant parameters are not passed to the Delphi application correctly. Please use the method AdsReadVarConnectEx2 and the corresponding event in Delphi applications. In Visual Basic applications both methods/events can be used.

Example

Visual Basic: 'Event-driven reading' [▶ 74]

4.4.3 AdsReadVarConvertConnect

From TwinCAT 2.8 Build > 743 and above.

This method creates a fixed connection to a variable in an ADS device. The *'usrConvertType'* parameter can be used to specify which data type (format) the incoming variable data should have in the event function. The *'usrConvertType'* parameter is passed by value, which means that the data type passed is only used as a "template" for the conversion. During the conversion, the appropriate quantity of data bytes is copied into the data type specified by the user.

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

Parameter

adsVarName

[in] Name of the ADS variable

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▶ 65] data type)

nCycleTime

[in] Read cycle in ms

phConnect

[out] Contains a unique handle for the connection that has been established (this is not the handle of the ADS variable!).

usrConverType

[in] Data type into which the event data is to be converted. The following table contains a list of the supported VB data types that can be passed as parameters.

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

* The string length must be set to the maximum number of characters (including the closing NULL) that the string variable can adopt. (VB string length + 1 byte (for null termination)) bytes are then copied from the event data into a string variable. After this, the length of the string is then shortened to the actual length. In other words, the string is truncated at the first null character. With appropriately set string length, string arrays can also be read from the PLC. E.g.:

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

in VB:

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

** During the conversion, one byte of event data at a time is converted to a 2-byte OleVariant data type. The following applies: TRUE when data <> 0 and FALSE when data = 0;

*** The OLE variant data type *Date* can only be used, for instance, to read PLC variables of type DATE_AND_TIME into a VB application. The local settings of the PC are taken into account during the conversion. Other PLC data types such as TIME or TOD are not supported, because they cannot be appropriately converted.

**** The variant variable must be initialized with a data type. VT_EMPTY or VT_NULL, for instance, are not allowed.

hUser

[in] Optional: This value is passed when the AdsReadConvertConnectUpdate() [▶ 55] event is called.

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be released using the AdsDisconnectEx() [▶ 40] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable *'SPSVar1'* from the function *'Funk1'* is to be accessed, then *'Funk1.SPSVar1'* must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in *'.SPSGlobVar'*. The parameter *adsVarName* does not distinguish between upper and lower case letters.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialogue for the axis under General. The 'Create symbols' box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

Example

Visual Basic sample: Event-driven reading (with conversion to another type) [▶ 84]

4.4.4 AdsRead[Datatype]VarConnect

AdsReadBoolVarConnect

AdsReadIntegerVarConnect

AdsReadLongVarConnect

AdsReadSingleVarConnect

AdsReadDoubleVarConnect

AdsReadStringVarConnect

Creates a fixed connection between a Visual Basic variable of type boolean, integer, long, single, double or string and a data item from an ADS device.

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

Parameter

adsVarName

[in] Name of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▶ 65] data type)

nCycleTime

[in] Read cycle in ms

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comment

When the PLC variable is changed, the event AdsReadConnectUpdate() [▶ 53] is triggered.

If the connection to an ADS variable is no longer required, it should be released using the AdsRead[DataType]Disconnect() [▶ 44] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

Only one handle is created per PLC variable, i.e. when connecting several variables to a PLC variable, the event AdsReadConnectUpdate() [▶ 53] is called accordingly several times with the same handle when changes are made.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable 'SPSVar1' from the function 'Funk1' is to be accessed, then 'Funk1.SPSVar1' must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in '.SPSGlobVar'. The parameter 'adsVarName' does not distinguish between upper and lower case letters.

If a variable from the PLC is linked to a Visual Basic variable you must enter 2 for the length, since Visual Basic manages boolean variables internally using 2 bytes.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialog for the axis under 'General'. The 'Create symbols' box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

This method has been replaced by AdsReadVarConnectEx() [▸ 34]. In future, use AdsReadVarConnectEx(), since AdsReadBoolVarConnect() will no longer be maintained, and will only be included for reasons of compatibility.

Example

-

4.4.5 AdsDisconnectEx

Closes a fixed connection between a Visual Basic variable and a data item from an ADS device.

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

Parameter

hConnect

[in] Handle of the connection between the Visual Basic variable and the ADS variable

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be closed using the AdsDisconnectEx() method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed. See also the AdsReadVarConnectEx() [▸ 34] method.

Example

Visual Basic sample: 'Event-driven reading [▸ 74]'

4.4.6 AdsReadConnect

Creates a fixed connection between a Visual Basic variable and a data item from an ADS device.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▶ 65] data type)

nCycleTime

[in] Read cycle in ms

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be released using the AdsReadDisconnect() [▶ 42] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed. The variable type string is not supported by the AdsReadConnect() method.

Example

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

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

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

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

4.4.7 AdsReadDisconnect

Closes a fixed connection between a Visual Basic variable and a data item from an ADS device.

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

Parameter

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be closed using the AdsReadDisconnect() method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed. See also the AdsReadConnect() [▶ 40] method.

Example

-

4.4.8 AdsRead[Datatype]Connect

AdsReadBoolConnect

AdsReadIntegerConnect

AdsReadLongConnect

AdsReadSingleConnect

AdsReadDoubleConnect

AdsReadStringConnect

Creates a cyclic connection between a Visual Basic variable of type boolean, integer, long, single, double or string and a data item from an ADS device.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▶ 65] data type)

nCycleTime

[in] Read cycle in ms

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comments

If the connection to the ADS variable is no longer required, it should be released using the AdsRead[Datatype]Disconnect() [▶ 44] method. If only certain values are required in a form, the connection should only be created when the form is loaded and released again when the form is closed.
Note on the data type String: It should be noted that the length of the data refers to the length of the variable in the Visual Basic program. Since Visual Basic represents a character with 2 bytes, the length of the variable must be determined with LenB(), not with Len().

Example

```
Dim VBVar As Integer

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
    'Verbindung zwischen Merkerwort 0 der SPS und VBVar herstellen
    Call AdsOcx1.AdsReadIntegerConnect(&H4020&, 0&, 2&, 1, 110, VBVar)
End Sub

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

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

```
Dim VBVar As String

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

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

'wird bei Veränderung der SPS-Variablen vom ADS-OCX aufgerufen
```

```
Private Sub AdsOcx1_AdsReadConnectUpdate(ByVal nIndexGroup As Long, ByVal nIndexOffset As Long)
    If (nIndexGroup = &H4020) And (nIndexOffset = 0) Then
    'Variablen in Form anzeigen
    Label1.Caption = VBVar
    End If
End Sub
```

4.4.9 AdsRead[Datatype]Disconnect

AdsReadBoolDisconnect

AdsReadIntegerDisconnect

AdsReadLongDisconnect

AdsReadSingleDisconnect

AdsReadDoubleDisconnect

AdsReadStringDisconnect

Ends a fixed connection between a Visual Basic variable of type boolean, integer, long, single, double or string and a data item from an ADS device.

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

Parameter

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comments

If the value of an ADS variable is no longer required, the connection should be closed using the AdsReadBoolDisconnect() method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed. See also the AdsRead[Datatype]VarConnect() [▶ 38] and AdsRead[Datatype]Connect() [▶ 42] method.

Example

-

4.4.10 AdsWriteDisconnect

Closes a fixed connection between a Visual Basic variable and the data item in an ADS device.

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

Parameter

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comments

If the connection to the ADS variable is no longer required, it should be released using the AdsWriteDisconnect() method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

Example

-

4.4.11 AdsWrite[Datatype]Disconnect

AdsWriteBoolDisconnect

AdsWriteIntegerDisconnect

AdsWriteLongDisconnect

AdsWriteSingleDisconnect

AdsWriteDoubleDisconnect

Ends a fixed connection between a Visual Basic variable of type boolean, integer, long, single or double and a data item from an ADS device.

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

Parameter

pData

[in] Visual Basic variable into which the data is written from the ADS variable

Return value

See ADS error codes

Comments

If the connection to the ADS variable is no longer required, it should be released using the AdsWrite[*Datatype*]Disconnect() method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

Example

-

4.4.12 AdsWriteVarConnect

Creates a fixed connection of a Visual Basic variable and a variable from an ADS device.

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

Parameter

adsVarName

[in] Name of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▸ 65] data type). The method *AdsWriteVarConnect* supports (sensibly) only the **ADSTRANS_CLIENTCYCLE** mode. The value of the Visual Basic variable is written cyclically to the ADS device.

nCycleTime

[in] Write cycle in ms

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be released using the AdsWriteDisconnect() [▸ 44] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed. The string variable type is not supported.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable *'SPSVar1'* from the function *'Funk1'* is to be accessed, then *'Funk1.SPSVar1'* must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in *'.SPSGlobVar'*. The parameter *adsVarName* does not distinguish between upper and lower case letters.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialogue for the axis under General. The 'Create symbols' box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

Example

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

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

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

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

4.4.13 AdsWrite[Datatype]VarConnect

AdsWriteBoolVarConnect

AdsWriteIntegerVarConnect

AdsWriteLongVarConnect

AdsWriteSingleVarConnect

AdsWriteDoubleVarConnect

Creates a fixed connection between a Visual Basic variable of type boolean, integer, long, single or double and an ADS device.

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

Parameter

adsVarName

[in] Name of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▸ 65] data type)

nCycleTime

[in] Write cycle in ms

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

If the connection to an ADS variable is no longer required, it should be released using the AdsWrite[Datatype]Disconnect() [▸ 45] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

NOTE

Enable the Symbol download at the PLC

Ensure that 'Symbol download' is enabled in PLC Control under Project / Options / TwinCAT. You will find more detailed information in the PLC Control manual.

The method's first parameter is composed of the POE name and the PLC variable that is to be addressed. If, for instance, the variable *'SPSVar1'* from the function *'Funk1'* is to be accessed, then *'Funk1.SPSVar1'* must be supplied as the first parameter. When global variables are being accessed, the POE name is omitted, as, for instance, in *'.SPSGlobVar'*. The parameter *adsVarName* does not distinguish between upper and lower case letters.

NOTE

In the NC, enable the Symbol download for each axis

Symbol download must be enabled for each axis in the System Manager. This can be specified in the configuration dialogue for the axis under General. The 'Create symbols' box must be checked. See System Manager manual.

The symbolic names of the individual NC parameters have a fixed specification, and can be found in the NC documentation.

Example

```
Dim VBVar As Integer

'wird beim Starten des Programms aufgerufen
Private Sub Form_Load()
    'Verbindung zur Variable in der SPS herstellen
    Call AdsOcx1.AdsWriteIntegerVarConnect("MAIN.PLCVar", 2&, 1, 110, VBVar)
End Sub

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

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

4.4.14 AdsWriteConnect

Creates a fixed connection of a Visual Basic variable and an ADS device.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▶ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▸ 65] data type)

nCycleTime

[in] Write cycle in ms

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

If the connection to the ADS variable is no longer required, it should be released using the AdsWriteDisconnect() [▸ 44] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed. The string variable type is not supported.

Example

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

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

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

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

4.4.15 AdsWrite[Datatype]Connect

AdsWriteBoolConnect

AdsWriteIntegerConnect

AdsWriteLongConnect

AdsWriteSingleConnect

AdsWriteDoubleConnect

Creates a fixed connection between a Visual Basic variable of type boolean, integer, long, single or double and a data item from an ADS device.

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

Parameter

nIndexGroup

[in] Index group of the ADS variable

nIndexOffset

[in] Index offset of the ADS variable

cbLength

[in] Length of the data in bytes (see VB variable lengths [▸ 69])

nRefreshType

[in] Type of data exchange between VB variable and ADS variable (see the ADSOCXTRANSMODE [▸ 65] data type)

nCycleTime

[in] Write cycle in ms

pData

[in] Visual Basic variable from which the data is written into the ADS variable

Return value

See ADS error codes

Comments

If the connection to the ADS variable is no longer required, it should be released using the AdsWrite[Datatype]Disconnect() [▸ 45] method. If only certain specific values are required in a form, the connection should only be created when the form is loaded, and should be released again when the form is closed.

Example

```
Dim VBVar As Integer

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

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

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

4.5 Events

4.5.1 AdsAmsConnectTimeout

This event is called as soon as a timeout occurs for a variable connected "per Connect".

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

Parameter

nIndexGroup

[out] Index group of the ADS variable where the timeout occurred.

nIndexOffset

[out] Index offset of the ADS variable where the timeout occurred.

Comments

The AdsAmsConnectTimeout event is called only if the type of data exchange between VB variable and ADS variable is controlled by the client (ADSTRANS_CLIENTCYCLE [▶ 65]).

Example

-

4.5.2 AdsAmsTimeout

This event is called as soon as a timeout occurs during an asynchronous read/write request.

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

Parameter

nInvokeId

[out] Identification number of the request where the timeout occurred.

Comments

-

Example

-

4.5.3 AdsConnectError

If an error occurs in the server on a variable connected "by connect", this event is called.

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

Parameter

nIndexGroup

[out] Index group of the ADS variable where the error occurred.

nIndexOffset

[out] Index offset of the ADS variable where the error occurred.

errorCode

[out] Error state; see ADS error codes

Comments

-

Example

-

4.5.4 AdsLogNotification

This event is called as soon as an ADS device has issued a message and the previously defined filter conditions are satisfied.

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

Parameter

dateTime

[out] Date and time at which the message was issued by the ADS device

nMs

[out] Milliseconds as the message was issued by the ADS device

dwMsgCtrl

[out] Filter mask for the kinds of messages that are to be reported (see ADSLOGMSGTYPE [▶ 65] data type)

nServerPort

[out] Port number of the ADS device that issued the message

szDeviceName

[out] Name of the ADS device that issued the message

szLogMsg

[out] Message that was issued by the ADS device

Comments

As soon as an ADS device has issued a message, and the filter conditions defined by the AdsEnableLogNotificaion() [▶ 15] method are met, the AdsLogNotification() event is triggered. The message can be further evaluated by means of the parameters that are passed.

Example

Visual Basic sample: 'Send/receive messages via the TwinCAT Router [▶ 81]'

4.5.5 AdsReadConnectUpdate

This event is called if the AdsReadYY(Var)Connect() method has been called, and the value from the ADS device has been read or has changed.

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

Parameter

nIndexGroup

[out] Date and time at which the message was issued by the ADS device

nIndexOffset

[out] Milliseconds as the message was issued by the ADS device

Comments

With the AdsReadConnectUpdate() event, it is not necessary that the value is transmitted at the same time, since the ADS-OCX will be updating the Visual Basic variable in the background. To optimize write accesses to display objects on the form, the event function should query which variable has changed and update only the element on the form that displays the value. If a VB variable was connected to an ADS variable via VarConnect, the handle of the variable is passed in the parameter *nIndexOffset* in the event AdsReadConnectUpdate(). The constant value &HF005 is transferred to the parameter *nIndexGroup* in this case. In order to be able to evaluate the *nIndexOffset*, you must first use the AdsCreateVarHandle() [▶ 14] method to fetch the handle of the ADS variable. This can be done, for instance, in the form's load event. In the event AdsReadConnectUpdate() it is then queried which variable handle was transferred in the parameter *nIndexOffset*.
If the connection was not created with the variable name, but with the variable address, then in the parameters *nIndexGroup* and *nIndexOffset* the address of the variable is transferred, which has changed.
If the connection between VB variable and ADS variable is terminated, also the handle should be released again with the method AdsDeleteVarHandle() [▶ 15].

Example

-

4.5.6 AdsReadConnectUpdateEx

This event is called if the AdsReadVarConnectEx() [▶ 34] method has been called, and the value from the ADS device has been read or has changed.

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

Parameter

dateTime

[out] Timestamp

nHs

[out] Milliseconds of timestamp

hConnect

[out] Handle of the connection; is created by the AdsReadVarConnectEx() [▶ 34] method

data

[out] Value from the ADS device

hUser

[out] General purpose value; is passed when the AdsReadVarConnectEx() [▶ 34] method is called

Comments

-

Example

Visual Basic sample: 'Event-driven reading [▶ 74]'

4.5.7 AdsReadConnectUpdateEx2

This event is called if the AdsReadVarConnectEx2() [▶ 35] method has been called, and the value from the ADS device has been read or has changed.

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

Parameter

dateTime

[out] Timestamp

nHs

[out] Milliseconds of timestamp

hConnect

[out] Handle of the connection; is created by the AdsReadVarConnectEx2() [▶ 35] method

data

[out] Value from the ADS device

hUser

[out] General purpose value; is passed when the AdsReadVarConnectEx2() [▶ 35] method is called

Comments

The parameters *data* and *hUser* must be passed **ByRef** (necessary for use under Borland Delphi).

Example

Visual Basic: 'Event-driven reading' [▶ 74]

4.5.8 AdsReadConvertConnectUpdate

From TwinCAT 2.8 Build > 743 and above.

This event is called if the AdsReadVarConvertConnect() [▶ 36] method has been called, and the value from the ADS device has been read or has changed.

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

Parameter

dateTime

[out] Timestamp.

nHs

[out] Milliseconds of timestamp.

hConnect

[out] Handle of the connection; is created by the AdsReadVarConvertConnect() [▶ 36] method.

data

[out] Value from the ADS device. The data type of the variant variable is specified as a parameter when AdsReadVarConvertConnect() [▶ 36] is called.

hUser

[out] General purpose value; is passed when the AdsReadVarConvertConnect() [▶ 36] method is called.

Example

Visual Basic: Event-driven reading (with conversion to another type) [▶ 84]

4.5.9 AdsRead[Datatype]Conf

AdsReadIntegerConf

AdsReadLongConf

AdsReadSingleConf

AdsReadDoubleConf

AdsReadStringConf

Returns the result after the AdsRead[*Datatype*]Req() method has been called.

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

Parameter

nInvokeId

[out] Job number for identification of the response

nResult

[out] Error state; see ADS error codes

cbLength

[out] Length of the data in bytes

pData

[out] Data being read from the ADS device

Comments

Once a read request has been sent to the ADS device, execution of the Visual Basic program continues. As soon as the data is available, the ADS-OCX triggers the event function AdsRead[*Datatype*]Conf() with which the requested data is transmitted.
When the read request is sent, an identification number must be specified, which is later returned when the event function is called. This makes it possible to assign the event function to the appropriate read request.
See also AdsRead[Datatype]Req() [▶ 31].

Example

-

4.5.10 AdsRouterRemove

This event is triggered if the TwinCAT Router is completely removed from the operating system in the Windows NT/2000 Control Panel.

```
object_AdsRouterRemove()
```

Parameter

-

Comments

-

Example

Visual Basic: 'Detect/alter state change in TwinCAT Router and the PLC [▶ 79]'

4.5.11 AdsRouterShutdown

This event is triggered when the TwinCAT Router is stopped.

```
object_AdsRouterShutdown()
```

Parameter

-

Comments

-

Example

Visual Basic: 'Detect/alter state change in TwinCAT Router and the PLC [▶ 79]'

4.5.12 AdsRouterStart

This event is triggered when the TwinCAT Router is started.

```
object_AdsRouterStart()
```

Parameter

-

Comments

-

Example

Visual Basic: 'Detect/alter state change in TwinCAT Router and the PLC [▶ 79]'

4.5.13 AdsServerStateChanged

This event function is called when the state of the ADS device has changed.

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

Parameter

nAdsState

[out] New state of the ADS device (see the ADSSTATE [▶ 65] data type)

nDeviceState

[out] (not presently supported)

Comments

-

Example

Visual Basic: 'Detect/alter state change in TwinCAT Router and the PLC [▶ 79]'

Also see about this

- ADSSTATE [▶ 65]

4.5.14 AdsServerSymChanged

This event function is triggered when the symbol table in the ADS device has changed.

```
object_AdsServerSymChanged()
```

Parameter

-

Comments

Every ADS device stores its symbol names in an internal table. Each symbol is assigned a handle that can be read with the AdsCreateVarHandle() [▸ 14] method. This event is triggered if the symbol table changes, for instance because the number of variables has changed.

Example

Visual Basic: 'Detect/alter state change in TwinCAT Router and the PLC [▸ 79]'

4.5.15 AdsWriteConf

Confirms a write request.

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

Parameter

nInvokeId

[out] Job number for identification of the response

nResult

[out] Error state; see ADS error codes

Comments

Once a write request has been sent to the ADS device, execution of the Visual Basic program continues. As soon as the data has been written to the device, the ADS-OCX triggers the event function AdsWriteConf(). When the write request is issued, an identification number must be specified, which is later returned when the event function is called. This makes it possible to assign the event function to the appropriate write request. See also AdsWrite[Datatype]Req() [▸ 33].

Example

-

4.6 Properties

4.6.1 AdsAmsClientNetId

This property stores the NetId of the computer in which the Visual Basic program with the ADS-OCX is executing.

```
object.AdsAmsClientNetId As String
```

Comment

This is a read-only property, and can be changed neither within the Visual Basic development environment nor during the program's runtime.

The NetID can be set using the TwinCAT system control.

4.6.2 AdsAmsClientPort

The client port number is the port number with which other ADS devices can address the Visual Basic program.

```
object.AdsAmsClientPort As Long
```

Comment

If you do not prescribe a port number yourself, the ADS-OCX will automatically assign a port number. This will always be greater than 32767. Note that this port number is different after each start.

If your Visual Basic program is to receive a fixed port number, you must set the desired port number in the program using the AdsAmsClientPort property. In that case the value must be between 16000 and 32000.

You can also set the AdsAmsClientPort property by means of the ADS-OCX properties window during the development phase. SavePort must then be set to TRUE. This will cause the ADS-OCX not to change the port number.

See also the AdsAmsSaveClientPort [▶ 59] property.

4.6.3 AdsAmsCommTimeout

AdsAmsCommTimeOut provides a time in milliseconds within which the communication partner is expected to respond.

```
object.AdsAmsCommTimeout As Long
```

Comment

Allowed values: 1 to 2147483647 milliseconds. Negative values and the value zero are not accepted during an assignment. Default: 5000 milliseconds.

4.6.4 AdsAmsConnected

AdsAmsConnected can be used to determine the state of the connection between ADS-OCX and TwinCAT ADS routers

```
object.AdsAmsConnected As Boolean
```

Comments

If the connection to the TwinCAT ADS router exists, the value of the property is "TRUE", otherwise the value is "FALSE".

This property can only be read.

4.6.5 AdsAmsSaveClientPort

Prevents the ADS-OCX from assigning the client's port dynamically.

```
object.AdsAmsSaveClientPort As Boolean
```

Comment

If you do not prescribe a port number yourself, the ADS-OCX will automatically assign a port number. This will always be greater than 32767. Note that this port number is different after each start.

If your Visual Basic program is to receive a fixed port number, you must set the desired port number in the program using the AdsAmsClientPort property. In that case the value must be between 16000 and 32000.

You can also set the AdsAmsClientPort property by means of the ADS-OCX properties window during the development phase. SavePort must then be set to TRUE. This will cause the ADS-OCX not to change the port number.

See also the AdsAmsClientPort [▶ 59] property.

4.6.6 AdsAmsServerNetId

This property stores the NetId of the computer that the Visual Basic program with the ADS-OCX is to access.

```
object.AdsAmsServerNetId As String
```

Comment

ADS devices can be located on various computers within a network. Each computer must have a unique NetId within that network.

Enter the NetId of the computer in which the ADS device with which you want to communicate is located into this property. If this property contains an empty string, the ADS devices of the local computer are addressed. If, for instance, your Visual Basic program is always located on the same computer as the PLC, leave this property empty. This makes it easier for the Visual Basic program to be used on other computers, even when those computers have different NetIds.

The TwinCAT system control can be used to determine what NetID has been set.

4.6.7 AdsAmsServerPort

Contains the port number of the ADS device that is to be addressed with the ADS-OCX.

```
object.AdsAmsServerPort As Long
```

Comment

The port numbers of the individual ADS devices can be found in the corresponding documentation. The following table lists the most important ADS devices:

ADS device	Port number
NC / NCI	501
PLC runtime system 1	801
PLC runtime system 2	811
PLC runtime system 3	821
PLC runtime system 4	831
Cam controller	901

4.6.8 AdsClientAdsState

This property can be used to tell other communication partners what state the ADS device is in at the moment.

```
object.AdsClientAdsState As String
```

Comment

The states that are supported by an ADS device can be found in the documentation for the ADS device.

4.6.9 AdsClientBuild

Contains the build level of the ADS device.

```
object.AdsClientBuild As Integer
```

Comment

Every ADS device has properties from which the ADS device's version number and type identification can be read. The version number consists of:

- Version (see the AdsClientVersion [▶ 62] property)
- Revision (see the AdsClientRevision [▶ 61] property)
- Build (see the AdsClientBuild [▶ 61] property)

Whenever ADS devices are created you should ensure that these properties are set, so that the ADS device can be identified by other participating devices.

4.6.10 AdsClientRevision

Contains the revision level of the ADS device.

```
object.AdsClientRevision As Integer
```

Comment

Every ADS device has properties from which the ADS device's version number and type identification can be read. The version number consists of:

- Version (see the AdsClientVersion [▶ 62] property)
- Revision (see the AdsClientRevision [▶ 61] property)
- Build (see the AdsClientBuild [▶ 61] property)

Whenever ADS devices are created you should ensure that these properties are set, so that the ADS device can be identified by other participating devices.

4.6.11 AdsClientType

Contains the type identification of the ADS device.

```
object.AdsClientType As String
```

Comment

The type identification consists of a character string of indeterminate length. Whenever ADS devices are created you should ensure that these properties are set, so that this ADS device can be identified by other participating devices.

4.6.12 AdsClientVersion

Contains the version number of the ADS device.

```
object.AdsClientVersion As Integer
```

Comment

Every ADS device has properties from which the ADS device's version number and type identification can be read. The version number consists of:

- Version (see the AdsClientVersion [▶ 62] property)
- Revision (see the AdsClientRevision [▶ 61] property)
- Build (see the AdsClientBuild [▶ 61] property)

Whenever ADS devices are created you should ensure that these properties are set, so that the ADS device can be identified by other participating devices.

4.6.13 AdsServerAdsState

These properties can be used in order to query the state of the ADS device that is currently being addressed.

```
object.AdsServerAdsState As String
```

Comment

The states that are indicated by an ADS device through this property can be found in the documentation for the ADS device.
This property is read-only.

4.6.14 AdsServerBuild

This property can be used in order to query the version of the ADS device that is addressed.

```
object.AdsServerBuild As Integer
```

Comment

Every ADS device has properties from which the ADS device's version number and type identification can be read. The version number consists of:

· Version (see the AdsServerVersion [▶ 63] property)

· Revision (see the AdsServerRevision [▶ 62] property)

· Build (see the AdsServerBuild [▶ 62] property)

4.6.15 AdsServerRevision

This property can be used in order to query the revision level of the ADS device that is addressed.

```
object.AdsServerRevision As Integer
```

Comment

Every ADS device has properties from which the ADS device's version number and type identification can be read. The version number consists of:

· Version (see the AdsServerVersion [▶ 63] property)

- Revision (see the AdsServerRevision [▸ 62] property)
- Build (see the AdsServerBuild [▸ 62] property)

4.6.16 AdsServerType

This property can be used in order to query the type of the ADS device that is addressed.

```
object.AdsServerType As String
```

Comment

The table below lists the type identifications of the most important ADS devices:

ADS device	Name
I/O	I/O server
PLC	PLC server
NC / NCI	NC-ADS server

4.6.17 AdsServerVersion

This property can be used in order to query the version number of the ADS device that is addressed.

```
object.AdsServerVersion As Integer
```

Comment

Every ADS device has properties from which the ADS device's version number and type identification can be read. The version number consists of:

- Version (see the AdsServerVersion [▸ 63] property)
- Revision (see the AdsServerRevision [▸ 62] property)
- Build (see the AdsServerBuild [▸ 62] property)

4.6.18 EnableErrorHandling

Switches on the exception handling.

```
object.EnableErrorHandling As Boolean
```

Comment

If this property is TRUE, and an error occurs within a method, an exception is triggered. Using the *On Error Goto* instruction it is possible to trap the exception at a defined label, and to examine the *Err* object to determine the cause.

4.6.19 Index

Index within a control array.

```
object.Index As Integer
```

Comment

It is possible to create an array of more than one ADS-OCX. For this purpose, each ADS-OCX that is to belong to the array is given the same name. The individual ADS-OCX devices are distinguished by their index property. The index normally begins with 0. The individual objects are addressed by the array name followed by the index in parenthesis, e.g. *AdsOcxName(1)*.

4.6.20 Name

Unique name of the controller.

```
object.Name As String
```

Comment

The standard name for newly added ADS-OCX devices is the object type (AdsOcx) plus a unique integer. For example, the first ADS-OCX has the name *AdsOcx1*, the second *AdsOcx2* and the third *AdsOcx3*. The name must start with a letter and can be a maximum of 40 characters long. Underscore characters (_) and numbers are permitted within the name. The names of global system objects (Clipboard, Screen or App) should not be used, as it would then no longer be possible to address them.

4.6.21 Object

With the aid of the object property of an OLE container you can also utilize the properties and methods of the linked or embedded object.

```
object.Object As Object
```

Comment

This property is used in association with OLE (Object Linking and Embedding). See the Visual Basic programming manual for further information.

4.6.22 Parent

Returns a form, object or a collection in which the ADS-OCX is contained.

```
object.Parent As Object
```

Comment

In order, for instance, to find the name of the container, you must enter the following instruction: AdsOcx1.Parent.Name

4.6.23 Tag

Contains a string for general purpose use.

```
object.Tag As String
```

Comment

Any data you wish may be stored in this property. Such data is neither evaluated by Visual Basic nor by the ADS-OCX.

4.7 Enums

4.7.1 ADSDATATYPEID

```
ADST_BIT          = 33 (&H21)
ADST_INT8         = 16 (&H10)
ADST_INT16        = 2
ADST_INT32        = 3
```

```
ADST_INT64       = 20 (&H14)
ADST_UINT8       = 17 (&H11)
ADST_UINT16      = 18 (&H12)
ADST_UINT32      = 19 (&H13)
ADST_UINT64      = 21 (&H15)
ADST_REAL32      = 4
ADST_REAL64      = 5
ADST_REAL80      = 32 (&H20)
ADST_BIGTYPE     = 65 (&H41)
ADST_VOID        = 0
```

4.7.2 ADSLOGMSGTYPE

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

4.7.3 ADSOCXTRANSMODE

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

Description

Parameter	Description
ADSTRANS_CLIENTCYCLE	The ADS-OCX executes a write / read command cyclically. The cycle time is rounded up to a multiple of 55. The shortest time is 55 ms. The timer that initiates the read / write runs in Windows NT/2000/XP user mode, which means that the time behavior strongly depends on the loading of the system.
ADSTRANS_SERVERCYCLE (only when reading)	The ADS that has been addressed writes the data cyclically to the ADS-OCX. The smallest possible time is the cycle time of the ADS device; for the PLC, this is the task cycle time. The cycle time can be handled in 1 ms steps. If you enter 0 ms as the cycle time, then the data is sent to the ADS-OCX with every cycle of the ADS device task.
ADSTRANS_SERVERONCHA (only when reading)	The ADS device that has been addressed then only writes the data to the ADS-OCX if they have changed. The ADS device is sampled at the rate given by the cycle time. The cycle time can be handled in 1 ms steps. If you enter a cycle time of 0 ms, every change in the variables will be sent to the ADS-OCX. A longer cycle time can be used to reduce the number of data transmissions to the ADS-OCX.

The largest cycle time is 32767 ms.

NOTE

Too many write / read operations

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

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

4.7.4 ADSSTATE

```
ADSSTATE_INVALID     = 0
ADSSTATE_IDLE        = 1
ADSSTATE_RESET       = 2
```

```
ADSSTATE_INIT         = 3
ADSSTATE_START        = 4
ADSSTATE_RUN          = 5
ADSSTATE_STOP         = 6
ADSSTATE_SAVECFG      = 7
ADSSTATE_LOADCFG      = 8
ADSSTATE_POWERFAILURE    = 9
ADSSTATE_POWERGOOD       = 10 (&H0A)
ADSSTATE_ERROR        = 11 (&H0B)
ADSSTATE_SHUTDOWN     = 12 (&H0C)
ADSSTATE_SUSPEND      = 13 (&H0D)
ADSSTATE_RESUME       = 14 (&H0E)
ADSSTATE_CONFIG       = 15 (&H0F)'system is in config mode
ADSSTATE_RECONFIG     = 16 (&H10)'system should restart in config mode
ADSSTATE_MAXSTATES       = 17 (&H11)
```

4.7.5 ADSGETDYNSYMBOLTYPE

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

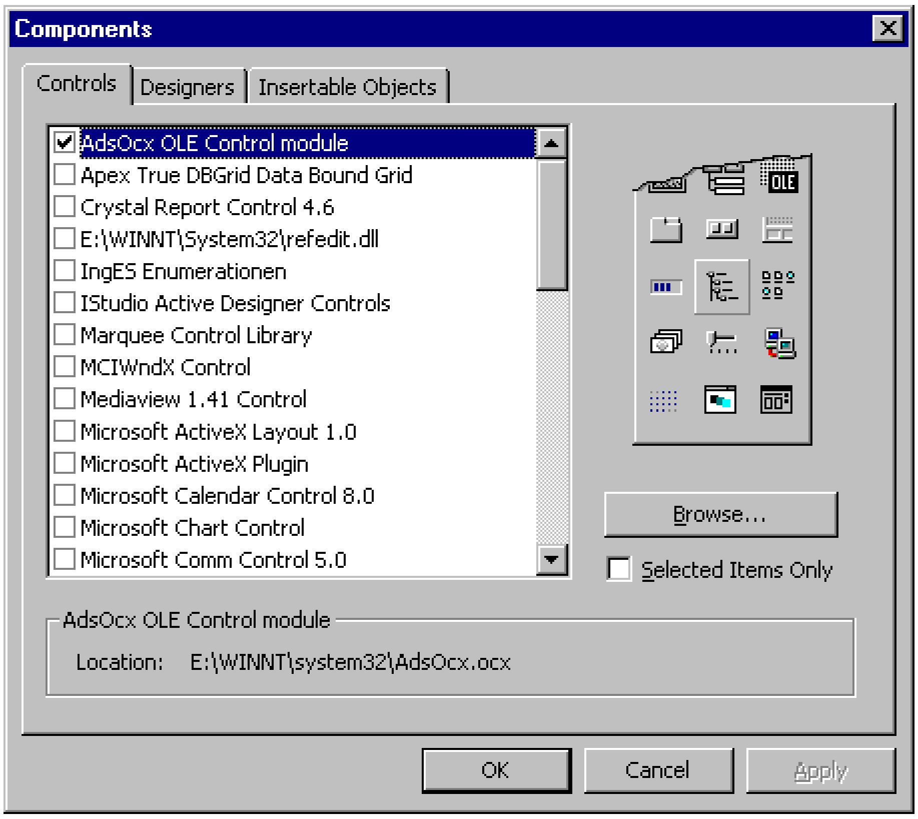
5 Samples

5.1 Visual Basic - samples

5.1.1 Linking into Visual Basic

Select the ADS-OCX

In order to select the ADS-OCX you must choose the command *Components...* under the *Project* menu item in Visual Basic, and mark the *AdsOcx OLE Control module* entry.



Components

Controls | Designers | Insertable Objects

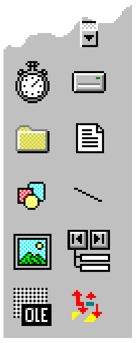
- [✓] AdsOcx OLE Control module
- [ ] Apex True DBGrid Data Bound Grid
- [ ] Crystal Report Control 4.6
- [ ] E:\WINNT\System32\refedit.dll
- [ ] IngES Enumerationen
- [ ] IStudio Active Designer Controls
- [ ] Marquee Control Library
- [ ] MCIWndX Control
- [ ] Mediaview 1.41 Control
- [ ] Microsoft ActiveX Layout 1.0
- [ ] Microsoft ActiveX Plugin
- [ ] Microsoft Calendar Control 8.0
- [ ] Microsoft Chart Control
- [ ] Microsoft Comm Control 5.0

[ ] Selected Items Only

AdsOcx OLE Control module

Location: E:\WINNT\system32\AdsOcx.ocx

The ADS-OCX then appears in the Visual Basic toolbox (bottom right).

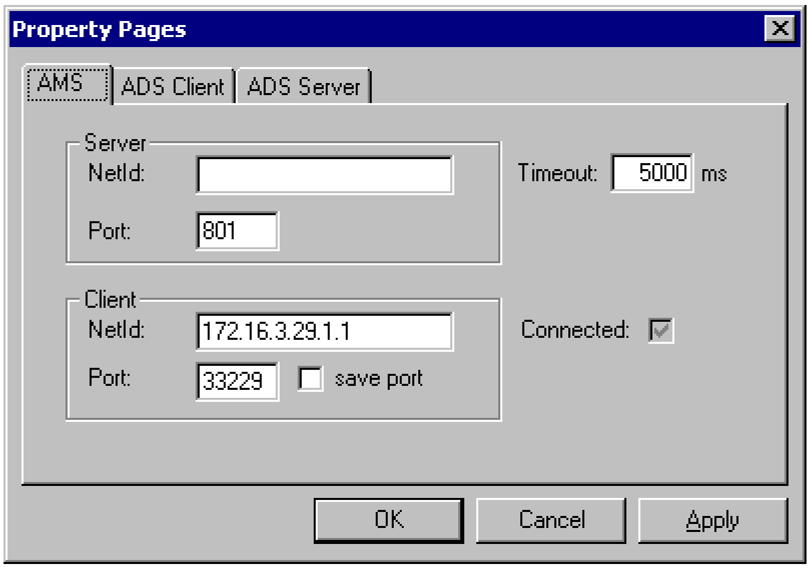


Define properties

Before you can use the ADS-OCX you must drag it onto a form and adjust the properties. The general properties can be configured or read using either the *properties pages* of the ADS-OCX or the Visual Basic *properties list*. The representation displayed on the *properties page* is particularly clear, since the properties are sorted here into groups. In order to make the Visual Basic program more readable, the names of (almost) all of the properties, methods and events begin with *Ads*.

AMS Properties

This page describes the communication channel between the ADS-OCX and the ADS device that is to be addressed. The terms ADS client and ADS server are also used here. The ADS client is the program that requests information or services from the ADS server. In our case, the Visual Basic program with the ADS-OCX is the ADS client. In most of the samples the TwinCAT PLC server is the ADS server.



Property Pages

AMS | ADS Client | ADS Server

Server
NetId:
Port: 801

Timeout: 5000 ms

Client
NetId: 172.16.3.29.1.1
Port: 33229 ☐ save port

Connected: ✓

OK | Cancel | Apply

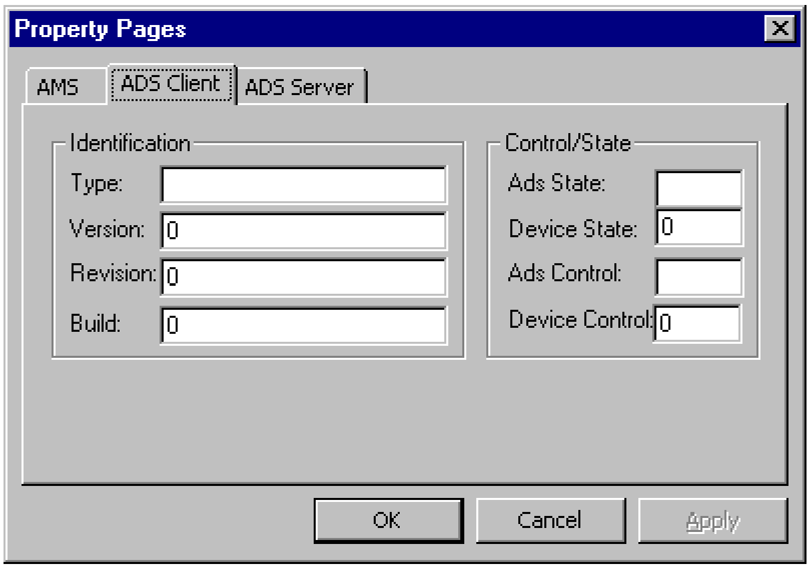
Each ADS device within TwinCAT has a unique address. This address is composed of a NetId and the port number. The NetId must be unique for each TwinCAT system within a network. The NetId of a computer can be read through the system control in the *TwinCAT system properties* dialog on the *AMS router* page. The NetId consists of 6 digits, separated from one another by a point. Do not provide a NetId when you want to address local ADS devices. In addition to the NetId, each ADS device is also addressed by a port number. Each port number may only exist once on a TwinCAT computer. Further information is to be found under TwinCAT ADS.

If, for instance, you want to address runtime system 1 in the PLC in the local computer, you should leave the server's *NetId* field empty, and enter port number 801 as the server *port*. The fields under *Client* can have different values in your case from those illustrated above.

If you want to address a number of ADS devices with the ADS-OCX (e.g. PLC and NC), you should use an ADS-OCX for each of these ADS devices. You should not alter the port number of the NetId while the system is running.

ADS Client Properties

This page shows the type and version of the ADS device with the ADS-OCX, as well as the current state of the ADS interface within the ADS-OCX.



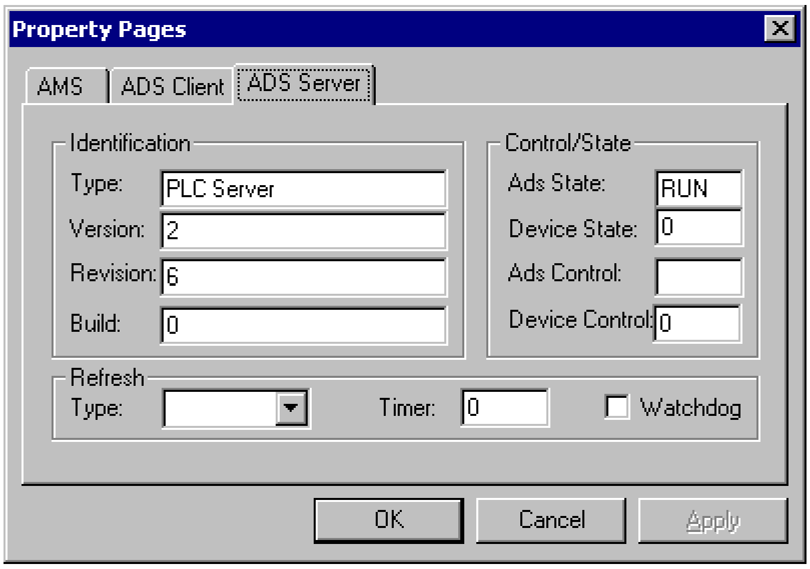
Identification
Type:
Version: 0
Revision: 0
Build: 0

Control/State
Ads State:
Device State: 0
Ads Control:
Device Control: 0

The *Identification* properties group shows the type, version, revision and build of the ADS-OCX. The other communication partner is able to interrogate these values in order to obtain more information about the ADS device. These fields can be released by the application.
In the property group *Control/State* the current state of the ADS interface within the ADS-OCX is displayed.

ADS Server Properties

This page shows the type and version of the ADS device that is to be addressed by the ADS-OCX. The current state of the ADS device is also displayed.



Identification
Type: PLC Server
Version: 2
Revision: 6
Build: 0

Control/State
Ads State: RUN
Device State: 0
Ads Control:
Device Control: 0

Refresh
Type:
Timer: 0
☐ Watchdog

The *Identification* properties group shows the type, version, revision and build of the ADS device that is addressed by the ADS-OCX. These properties are read-only.
The property group *Control/State* displays the current state of the ADS interface within the ADS device.
The property group *Refresh* is reserved and not yet supported.

The PLC is addressed by the ADS-OCX in the dialog shown above. This is in the RUN state, and is of version 2.6.0.

The ADS-OCX is now configured in such a way that it can address the PLC.

5.1.2 Visual Basic 6.0 variable lengths

VB variable type	Variable length in byte
Boolean	2

VB variable type	Variable length in byte
Integer	2
Long	4
Single	4
Double	8
String	Number of characters * 2
Byte	1

Array sizes

The length of an array is calculated from the number of individual array elements multiplied by the length of the variable type.

Example

If an array of 5 Long elements is to be read, the length is 20 bytes (5 elements * 4 bytes).

For a string with 25 characters, the length is 50 bytes (25 characters * 2 bytes).

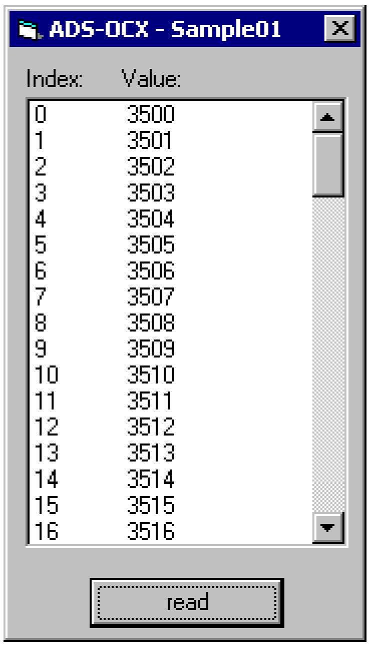
5.1.3 Accessing an array in the PLC

Task

The PLC contains an array that is to be read by Visual Basic using a read command.

Description

The PLC contains an array of 100 elements of type integer (2 bytes). The array is filled in the PLC with the values 3500 to 3599.
In the load event function of the Visual Basic program, the handle of the PLC variable is fetched first. When the program is terminated, this is released again in the Unload event function.
If the user presses the button on the form, the method AdsSyncRead[Datatype]VarReq() [▸ 24] reads the complete array from the PLC into the Visual Basic variable *Data*.
The variable *Data* must have the same structure as the corresponding variable in the PLC; 100 elements of type integer (2 bytes). The length specification in the method call is 200, because the length of the requested data is 200 bytes (100 elements with 2 bytes each).
In the following FOR loop, the array from the PLC is displayed in a list box control.



```
Index:  Value:
0       3500
1       3501
2       3502
3       3503
4       3504
5       3505
6       3506
7       3507
8       3508
9       3509
10      3510
11      3511
12      3512
13      3513
14      3514
15      3515
16      3516
```

Visual Basic 6 program

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

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

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

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

PLC program

```
PROGRAM MAIN
VAR
  PLCVar : ARRAY [0..99] OF INT;
  Index: BYTE;
END_VAR

FOR Index := 0 TO 99 DO
  PLCVar[Index] := 3500 + INDEX;
END_FOR
```

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463158027/.exe

5.1.4 Transmitting structures to the PLC

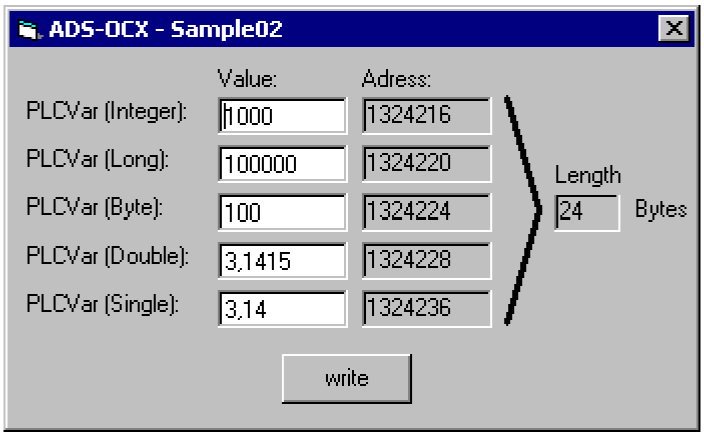
Task

A structure is to be written into the PLC by Visual Basic. The elements in the structure have different data types.

Description

In order for the CPU running under Windows NT/2000 to be able to access the variables more quickly, Visual Basic (as well as other programming languages) arranges them in the main memory according to certain rules. This arrangement of variables is called alignment. This can mean that 'memory gaps' occur within a structure. Since Visual Basic and IEC1131-3 have different guidelines for the alignment, these must be filled by dummy variables.
Unfortunately, no general rule for the alignment can be defined under Visual Basic. There are however two Visual Basic functions which allow the memory assignments of a structure to be analyzed. They are the functions *VarPtr()* and *LenB()*.
VarPtr() returns the address of a variable, *LenB()* the length in bytes that a variable (or a whole structure) occupies. The sample below illustrates the memory map of the structure in a form. This information can be

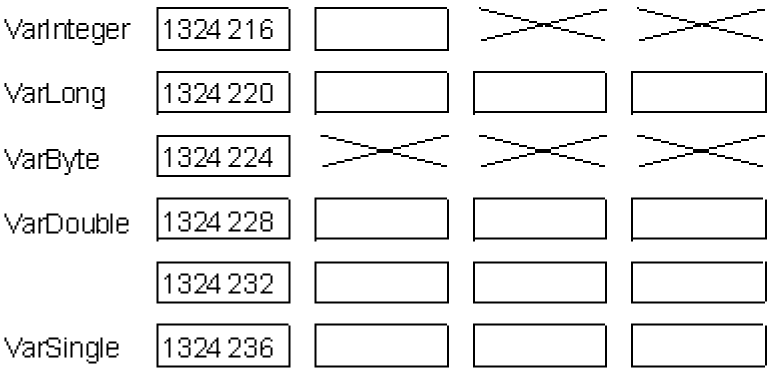
used to determine where the structure has 'memory gaps'. In the sample program these are filled by the variables *VarDummyX*.
The function *VarPtr()* is only available from Visual Basic 5.


ADS-OCX - Sample02

	Value:	Adress:
PLCVar (Integer):	1000	1324216
PLCVar (Long):	100000	1324220
PLCVar (Byte):	100	1324224
PLCVar (Double):	3,1415	1324228
PLCVar (Single):	3,14	1324236

Length 24 Bytes

The following sketch shows the memory allocation graphically once more:



A rectangle means that the variable occupies one byte at this location. A cross represents the location of a byte that is not used by any variables. In the sample program the crosses are filled by dummy variables.

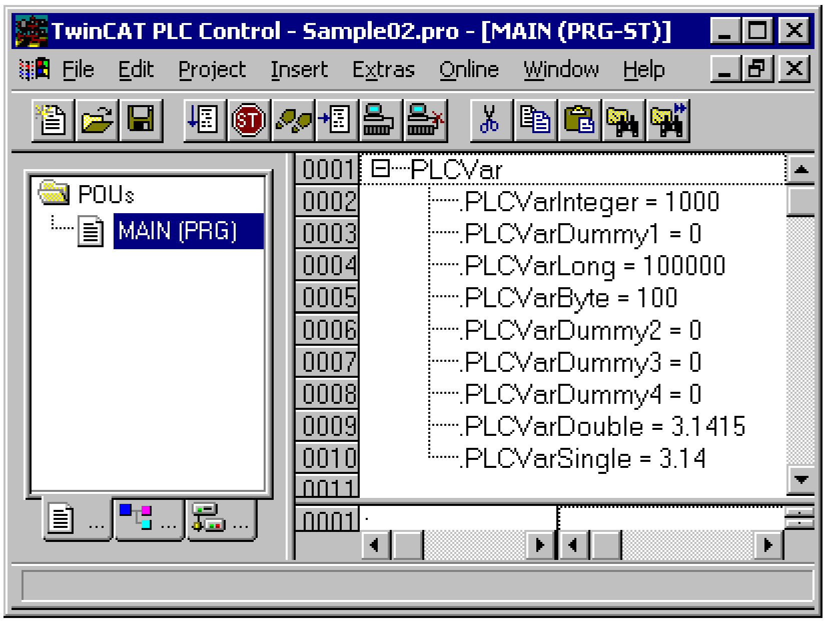
Structure declaration in Visual Basic

```
Type VBStruct
    VarInteger   As Integer
    VarDummy1    As Integer
    VarLong      As Long
    VarByte      As Byte
    VarDummy2    As Byte
    VarDummy3    As Byte
    VarDummy4    As Byte
    VarDouble    As Double
    VarSingle    As Single
End Type
```

Structure declaration in the PLC

After the structure in the Visual Basic program has been adapted to the alignment, the structure in the PLC program must also be supplemented:

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


TwinCAT PLC Control - Sample02.pro - [MAIN (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
MAIN (PRG)

```
PLCVar
    .PLCVarInteger = 1000
    .PLCVarDummy1 = 0
    .PLCVarLong = 100000
    .PLCVarByte = 100
    .PLCVarDummy2 = 0
    .PLCVarDummy3 = 0
    .PLCVarDummy4 = 0
    .PLCVarDouble = 3.1415
    .PLCVarSingle = 3.14
```

Visual Basic 6 program

```
Dim hVar As Long
Dim VBVar As VBStruct

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

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

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

PLC program

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

Optimizations

By a clever arrangement of the VBStruct member variables in the VB application the adding of the dummy bytes can be avoided. The following rule must be observed:

- Arrange the member variables in the VB structure according to the occupied memory size: first the largest and finally the smallest data types.
- The last bytes can (but do not have to) be padded to a full 4 bytes.

Optimized structure declaration in Visual Basic

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

Optimized structure declaration in the PLC

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

Our optimized VB structure now starts with a double, accordingly the VB program must be changed:

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

In addition to the changed method name, the data length to be written must be determined with the Len function and not with LenB. If you use LenB, the data will not be written to the PLC. The reason is that LenB returns a length = 20 bytes (including a padding byte in VB memory), but our structure in the PLC is only 19 bytes long.

Len vs. LenB

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

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463159435/.exe

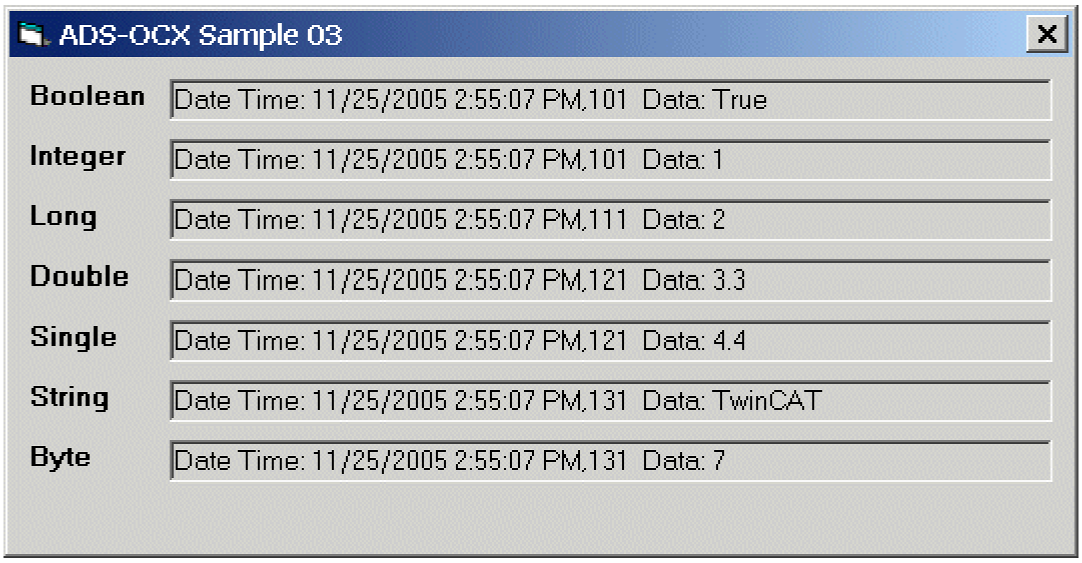
5.1.5 Event driven reading

Task

There are 7 global variables in the PLC. Each of these PLC variables is of a different data type. The values of the variables should be read in the most effective manner, and the value with its timestamp is to be displayed on a form in Visual Basic.

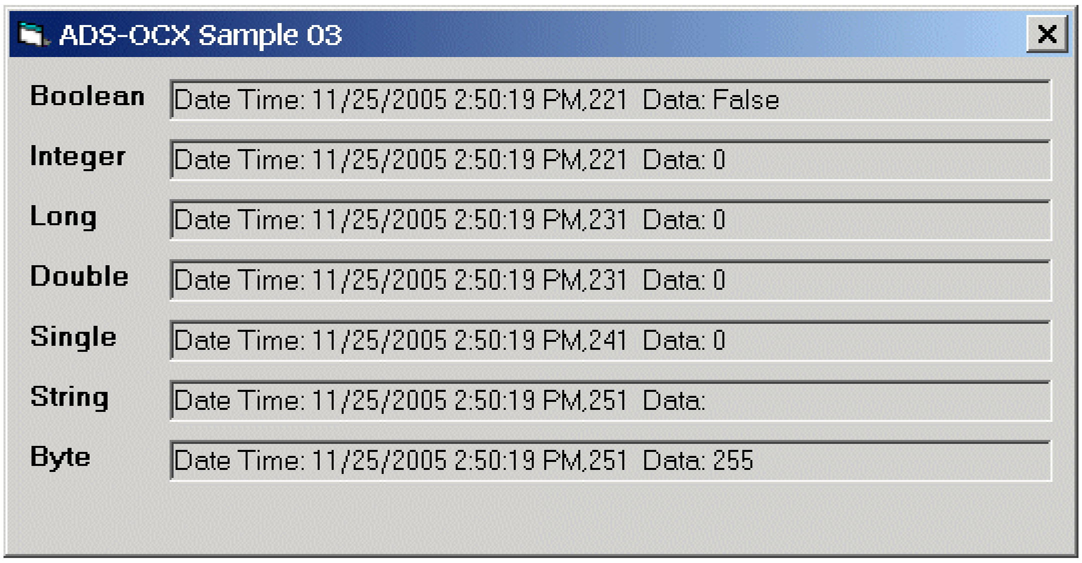
Description

In the form's load event, a connection to each of the PLC variables is created with the AdsReadVarConnectEx() [▶ 34] method. The handle of this connection is stored in a global array. The second parameter of the AdsReadVarConnectEx() method specifies the type of data exchange. ADSTRANS_SERVERONCHA has been selected here. This means that the value of the PLC variable is only transmitted if its value within the PLC has changed (see the ADSOCXTRANSMODE [▶ 65] data type). The third parameter indicates that the PLC is to check whether the corresponding variable has changed every 100 ms.



```
Boolean  Date Time: 11/25/2005 2:55:07 PM,101  Data: True
Integer  Date Time: 11/25/2005 2:55:07 PM,101  Data: 1
Long     Date Time: 11/25/2005 2:55:07 PM,111  Data: 2
Double   Date Time: 11/25/2005 2:55:07 PM,121  Data: 3.3
Single   Date Time: 11/25/2005 2:55:07 PM,121  Data: 4.4
String   Date Time: 11/25/2005 2:55:07 PM,131  Data: TwinCAT
Byte     Date Time: 11/25/2005 2:55:07 PM,131  Data: 7
```

When the PLC variable changes, the AdsReadConnectUpdateEx() [▶ 53] event is called. The timestamp, the handle, the value and a reference to the control in which the value is to be displayed are passed as parameters.
In the Unload event, the connections are released again with the method AdsDisconnectEx() [▶ 40]. You should pay attention to this, because every connection made with AdsReadVarConnectEx() consumes resources.
Also, set the CycleTime to reasonable values, since too many read/write operations can load the system so much that the user interface slows down considerably.



```
Boolean  Date Time: 11/25/2005 2:50:19 PM,221  Data: False
Integer  Date Time: 11/25/2005 2:50:19 PM,221  Data: 0
Long     Date Time: 11/25/2005 2:50:19 PM,231  Data: 0
Double   Date Time: 11/25/2005 2:50:19 PM,231  Data: 0
Single   Date Time: 11/25/2005 2:50:19 PM,241  Data: 0
String   Date Time: 11/25/2005 2:50:19 PM,251  Data:
Byte     Date Time: 11/25/2005 2:50:19 PM,251  Data: 255
```

Visual Basic 6 program

```
Option Explicit

Dim hConnect(0 to 6) As Long

Private Sub Form_Load()
    Dim nErr As Long

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

    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarInteger", ADSTRANS_SERVERONCHA, 100, hConnect(1), lbl
Integer)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarInteger: " & nErr)
    nErr = AdsOcx1.AdsReadVarConnectEx(".PLCVarLong", ADSTRANS_SERVERONCHA, 100, hConnect(2), lblLon
g)
    If (nErr > 0) Then Call MsgBox("Error AdsReadVarConnectEx -> .PLCVarLong: " & nErr)

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

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

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

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

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

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

PLC program

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

PROGRAM MAIN
VAR
    ;
END_VAR
```

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463160843/.exe

5.1.6 Read PLC variable declaration

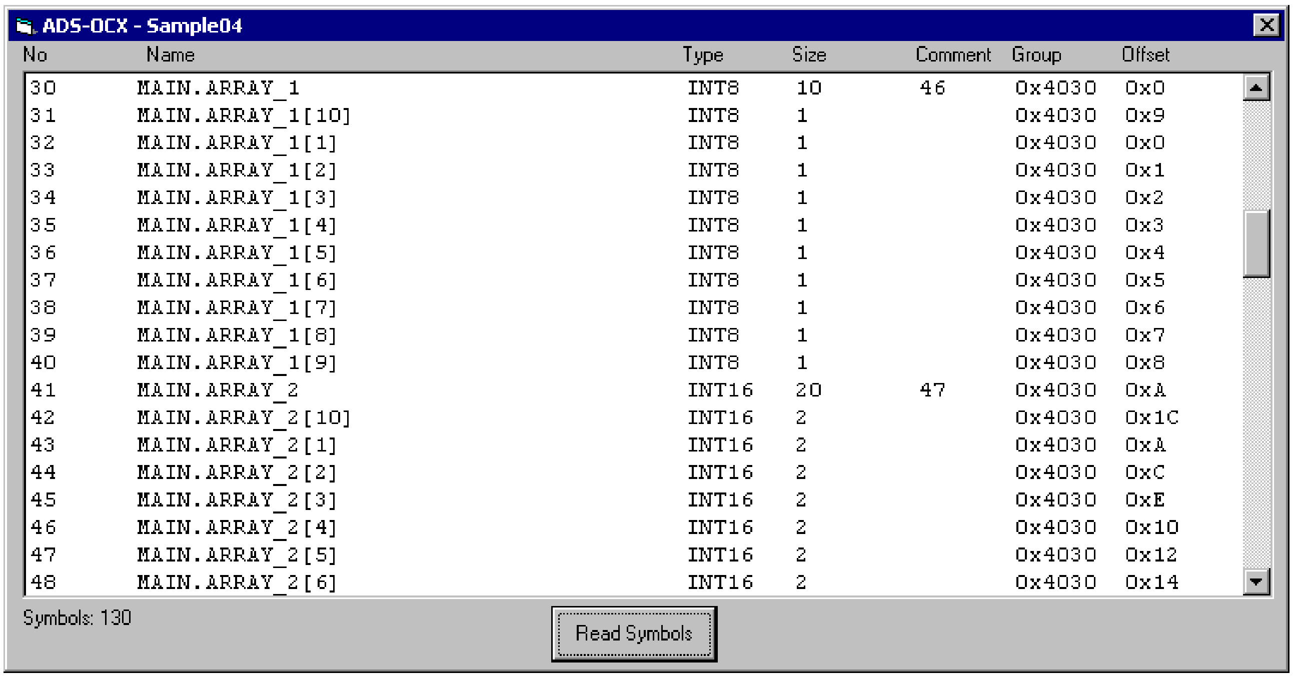
Task

All the information about the variables is to be read from the PLC (symbol upload).

Description

The *cmdReadSymbols_Click()* event function is called by clicking the button on the form. The AdsReadSymbolInfo() [▶ 21] method supplies the number of variables (symbols) and the length of the data in which the symbols are stored. The parameter *bNext* must be set to FALSE the first time the

AdsEnumSymbols() [▶ 16] method is called. This results in all the information about the first symbol being read. For every subsequent call, *bNext* is set to TRUE. In the FOR loop, AdsEnumSymbols() is called as many times as there are symbols in the PLC.



No	Name	Type	Size	Comment	Group	Offset
30	MAIN.ARRAY_1	INT8	10	46	0x4030	0x0
31	MAIN.ARRAY_1[10]	INT8	1		0x4030	0x9
32	MAIN.ARRAY_1[1]	INT8	1		0x4030	0x0
33	MAIN.ARRAY_1[2]	INT8	1		0x4030	0x1
34	MAIN.ARRAY_1[3]	INT8	1		0x4030	0x2
35	MAIN.ARRAY_1[4]	INT8	1		0x4030	0x3
36	MAIN.ARRAY_1[5]	INT8	1		0x4030	0x4
37	MAIN.ARRAY_1[6]	INT8	1		0x4030	0x5
38	MAIN.ARRAY_1[7]	INT8	1		0x4030	0x6
39	MAIN.ARRAY_1[8]	INT8	1		0x4030	0x7
40	MAIN.ARRAY_1[9]	INT8	1		0x4030	0x8
41	MAIN.ARRAY_2	INT16	20	47	0x4030	0xA
42	MAIN.ARRAY_2[10]	INT16	2		0x4030	0x1C
43	MAIN.ARRAY_2[1]	INT16	2		0x4030	0xA
44	MAIN.ARRAY_2[2]	INT16	2		0x4030	0xC
45	MAIN.ARRAY_2[3]	INT16	2		0x4030	0xE
46	MAIN.ARRAY_2[4]	INT16	2		0x4030	0x10
47	MAIN.ARRAY_2[5]	INT16	2		0x4030	0x12
48	MAIN.ARRAY_2[6]	INT16	2		0x4030	0x14

Symbols: 130

Visual Basic 6 program

```
Option Explicit

'--- wird beim Starten des Programms aufgerufen ---
Private Sub Form_Load()
    '--- Exception freigeben ---
    AdsOcx1.EnableErrorHandling = True
    '--- Anzeigeliste- und Felder löschen ---
    lstSymbols.Clear
    lblSymbols.Caption = "Symbols: "
End Sub

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

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

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

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

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

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

PLC program

```
PROGRAM MAIN
VAR
    REAL32_1 AT %MB0 : REAL;  (* 1 *)
    REAL32_2 AT %MB4 : REAL;  (* 2 *)
    REAL32_3 AT %MB8 : REAL;  (* 3 *)
    REAL32_4 AT %MB12: REAL;  (* 4 *)
    REAL32_5 AT %MB16: REAL;  (* 5 *)

    REAL64_1 AT %MB20 : LREAL;  (* 6 *)
    REAL64_2 AT %MB28 : LREAL;  (* 7 *)
    REAL64_3 AT %MB36 : LREAL;  (* 8 *)
    REAL64_4 AT %MB44 : LREAL;  (* 9 *)
    REAL64_5 AT %MB52 : LREAL;  (* 10 *)

    INT32_1 AT %MB60 : DINT;  (* 11 *)
    INT32_2 AT %MB64 : DINT;  (* 12 *)
    INT32_3 AT %MB68 : DINT;  (* 13 *)
    INT32_4 AT %MB72 : DINT;  (* 14 *)
    INT32_5 AT %MB76 : DINT;  (* 15 *)

    UINT32_1 AT %MB80 : UDINT;  (* 16 *)
    UINT32_2 AT %MB84 : UDINT;  (* 17 *)
    UINT32_3 AT %MB88 : UDINT;  (* 18 *)
    UINT32_4 AT %MB92 : UDINT;  (* 19 *)
    UINT32_5 AT %MB96 : UDINT;  (* 20 *)

    INT16_1 AT %MB100 : INT;  (* 21 *)
    INT16_2 AT %MB102 : INT;  (* 22 *)
    INT16_3 AT %MB104 : INT;  (* 23 *)
    INT16_4 AT %MB106 : INT;  (* 24 *)
    INT16_5 AT %MB108 : INT;  (* 25 *)

    UINT16_1 AT %MB110 : UINT;  (* 26 *)
    UINT16_2 AT %MB112 : UINT;  (* 27 *)
    UINT16_3 AT %MB114 : UINT;  (* 28 *)
    UINT16_4 AT %MB116 : UINT;  (* 29 *)
    UINT16_5 AT %MB118 : UINT;  (* 30 *)

    INT8_1 AT %MB120 : SINT;  (* 31 *)
    INT8_2 AT %MB121 : SINT;  (* 32 *)
    INT8_3 AT %MB122 : SINT;  (* 33 *)
    INT8_4 AT %MB123 : SINT;  (* 34 *)
    INT8_5 AT %MB124 : SINT;  (* 35 *)

    UINT8_1 AT %MB125 : USINT;  (* 36 *)
    UINT8_2 AT %MB126 : USINT;  (* 37 *)
    UINT8_3 AT %MB128 : USINT;  (* 38 *)
    UINT8_4 AT %MB129 : USINT;  (* 39 *)
    UINT8_5 AT %MB130 : USINT;  (* 40 *)

    BOOL_1 AT %MX131.0 : BOOL;  (* 41 *)
    BOOL_2 AT %MX131.1 : BOOL;  (* 42 *)
    BOOL_3 AT %MX131.2 : BOOL;  (* 43 *)
    BOOL_4 AT %MX131.3 : BOOL;  (* 44 *)
    BOOL_5 AT %MX131.4 : BOOL;  (* 45 *)

    ARRAY_1 : ARRAY[1 .. 10] OF SINT; (* 46 *)
    ARRAY_2 : ARRAY[1 .. 10] OF INT;  (* 47 *)
    ARRAY_3 : ARRAY[1 .. 10] OF DINT; (* 48 *)
    ARRAY_4 : ARRAY[1 .. 10] OF LREAL;(* 49 *)
    ARRAY_5 : ARRAY[1 .. 10] OF BOOL; (* 50 *)
END_VAR
```

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463802251/.exe

5.1.7 Detect/alter state of the router and the PLC

Task

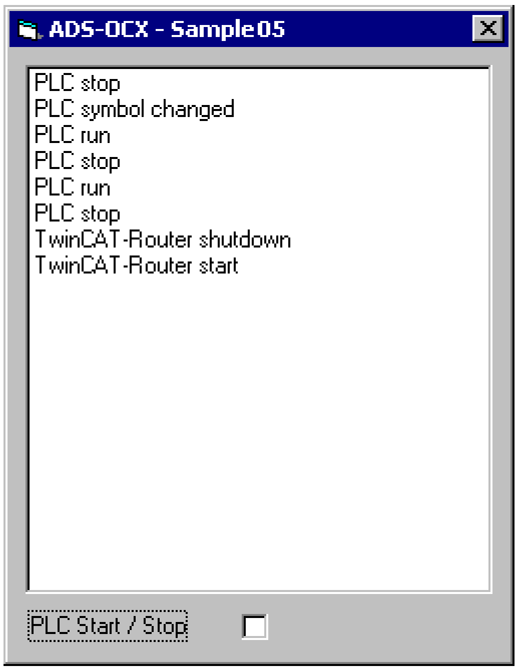
ADS-OCX provides methods for trapping state changes in the TwinCAT Router and the ADS devices. The events AdsRouterRemove() [▸ 56], AdsRouterShutdown() [▸ 56], AdsRouterStart() [▸ 57], AdsServerStateChanged() [▸ 57] and AdsServerSymChanged() [▸ 58] are available for this.

Description

If the AdsRouterShutdown() event is called when the TwinCAT Router is stopped, the affected program can react appropriately. When the TwinCAT Router is started, the AdsRouterStart() event is called, in which, for example, the AdsReadVarConnectEx() [▸ 34] method can be used to re-establish the connections to the ADS variables. If the TwinCAT router is completely removed from the operating system in the Windows NT/2000/XP control panel, the AdsRouterRemove() event is called.
In addition to state changes of the TwinCAT router, state changes in ADS devices can also be intercepted. This is particularly significant to the PLC. The AdsServerStateChanged() event can be used to establish whether the PLC has been started or stopped. Changes of the symbol table are reported by the AdsServerSymChanged() event. This happens, for instance, when the PLC program is recompiled and then transferred to the PLC.

Just as the state of an ADS device can be queried, it can also be changed. The AdsSyncWriteControlReq() [▸ 22] method makes this possible. The PLC can adopt the ADS states STOP and RUN. Using the check button in the sample program below, the user can switch between these two states.

Each change of state in the TwinCAT Router results in an appropriate entry in the listbox.



```
PLC stop
PLC symbol changed
PLC run
PLC stop
PLC run
PLC stop
TwinCAT-Router shutdown
TwinCAT-Router start
```

NOTE

Change of the symbol table

If a change in the symbol table is detected, it can be that a variable that was addressed by AdsReadVar-ConnectEx() has been deleted or renamed. When the AdsServerSymChanged() event occurs, all connects and handles should be deleted and then recreated.

Visual Basic 6 program

```
Option Explicit

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

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

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

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

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

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

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

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463803659/.exe

5.1.8 Send/receive messages via the router

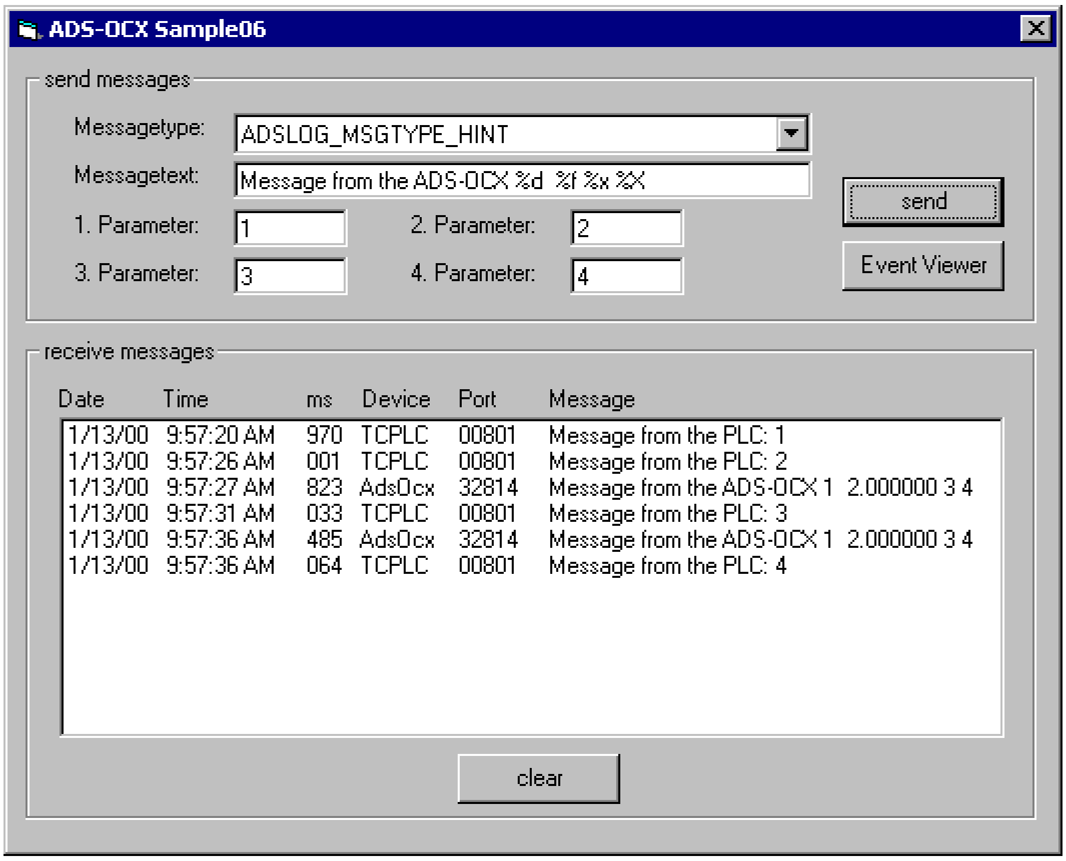
Task

ADS devices can send messages to other ADS devices via the TwinCAT Router. They can be received and evaluated there. It is also possible to write messages to the Windows NT/2000/XP Event Logger.
The following Visual Basic program receives messages from the PLC and displays them on the screen. It is also possible for messages to be written to the Windows NT/2000/XP Event Logger from the Visual Basic program.

Description

In order to be able to receive messages, a filter must first be defined using the AdsEnableLogNotification() [▸ 15] method. Here the range of port numbers from which ADS device messages are to be received is given. A second parameter additionally states the type of message (error, note or warning). The OR operator can also be used to combine several message types.
Each time a message is sent from an ADS device and the filter conditions are met, the AdsLogNotification() [▸ 52] event is called. Via the parameters the source, the type, the time and the message itself can be determined.
If a message is to be sent with the help of the ADS-OCX, the method AdsLogFmtString() [▸ 19] is used for this. The first parameter contains the message type (error, note or warning). The message text can contain up to four placeholders for numerical values. This allows, for instance, values that only become known at runtime to be transmitted. All messages that are sent with ADS-OCX are automatically written in the Windows NT/2000/XP Event Logger.

The Windows NT/2000/XP Event Viewer can be called from the Visual Basic program. This allows all the messages that are in the Event Logger to be examined. The Event Logger and the Event Viewer are standard elements in the Windows NT/2000/XP suites. You will find more details in the Windows NT/2000/XP documentation.



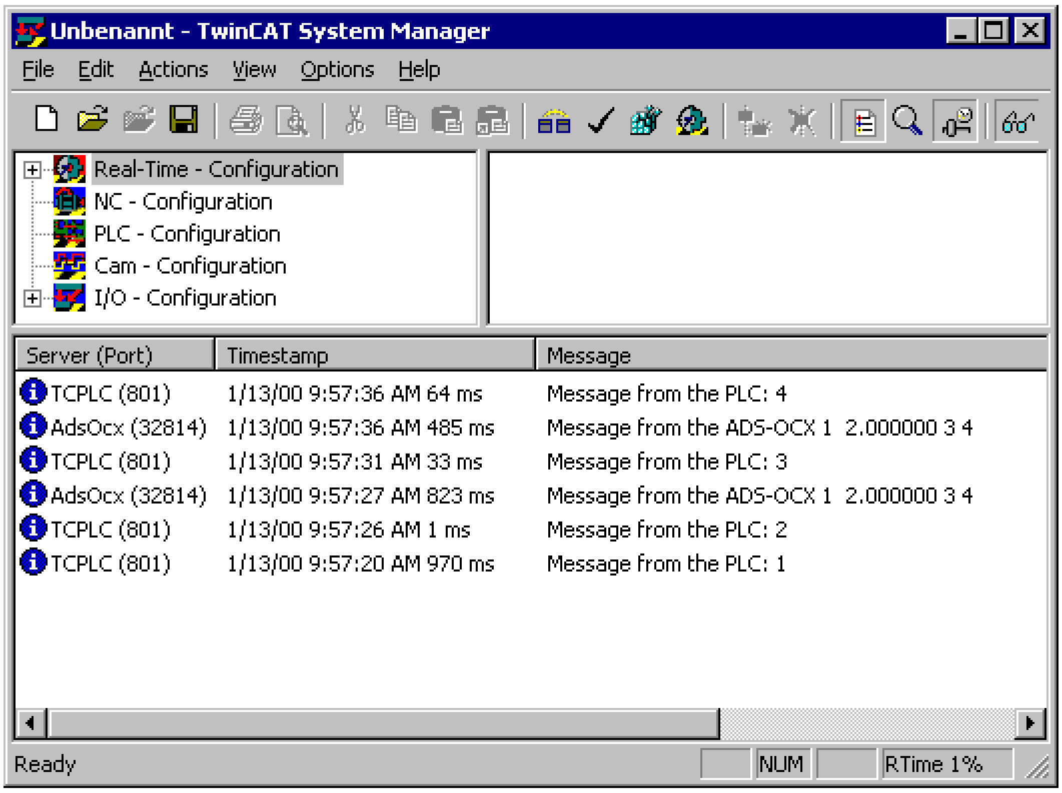
Messagetype: ADSLOG_MSGTYPE_HINT
Messagetext: Message from the ADS-OCX %d %f %x %X
1. Parameter: 1
2. Parameter: 2
3. Parameter: 3
4. Parameter: 4

```
Date     Time        ms   Device  Port   Message
1/13/00  9:57:20 AM  970  TCPLC   00801  Message from the PLC: 1
1/13/00  9:57:26 AM  001  TCPLC   00801  Message from the PLC: 2
1/13/00  9:57:27 AM  823  AdsOcx  32814  Message from the ADS-OCX 1  2.000000 3 4
1/13/00  9:57:31 AM  033  TCPLC   00801  Message from the PLC: 3
1/13/00  9:57:36 AM  485  AdsOcx  32814  Message from the ADS-OCX 1  2.000000 3 4
1/13/00  9:57:36 AM  064  TCPLC   00801  Message from the PLC: 4
```

The PLC program sends a note cyclically every 5 seconds. This note is also written into the Windows NT/2000/XP Event Logger. The ADSLOGDINT() PLC function is described in the PLC documentation.

Monitor messages online

All messages that are sent via the TwinCAT Router can be monitored online in the System Manager. This requires Logger output to be enabled in the View menu.


Unbenannt - TwinCAT System Manager
File Edit Actions View Options Help
Real-Time - Configuration
NC - Configuration
PLC - Configuration
Cam - Configuration
I/O - Configuration
Ready
NUM
RTime 1%

```
Server (Port)    Timestamp                      Message
TCPLC (801)      1/13/00 9:57:36 AM 64 ms       Message from the PLC: 4
AdsOcx (32814)   1/13/00 9:57:36 AM 485 ms      Message from the ADS-OCX 1  2.000000 3 4
TCPLC (801)      1/13/00 9:57:31 AM 33 ms       Message from the PLC: 3
AdsOcx (32814)   1/13/00 9:57:27 AM 823 ms      Message from the ADS-OCX 1  2.000000 3 4
TCPLC (801)      1/13/00 9:57:26 AM 1 ms        Message from the PLC: 2
TCPLC (801)      1/13/00 9:57:20 AM 970 ms      Message from the PLC: 1
```

Windows NT/2000/XP contains API functions with which the messages that have been saved in the Event Logger can be read again. Unfortunately, Visual Basic does not (yet) have any components with which the messages saved in the Event Logger can be accessed. On pp. 56 ff of issue 6/98 of the basic*pro* magazine, published by Steingräberverlag (http://www.basicpro.de), there is an article that illustrates the utilization of the API functions under Visual Basic.

Application

This technique has been found very useful in the development and debugging of applications. A PLC program, for example, or a Visual Basic program can indicate certain internal program states (in the System Manager) and save them (in the Windows NT/2000 Event Logger). For this kind of program tracking it is not necessary to install the development environment (e.g. on a machine computer).

NOTE

Too many messages in a short time

Make sure that not too many messages are transmitted in a short time, otherwise this could affect the overall system.

Log messages

If you want to keep a log of messages in your program (e.g. malfunctions in a machine) you should make use of the TwinCAT Event Logger. This is significantly more powerful than the Windows NT/2000/XP Event Logger, and is adapted to the requirements of automation technology.

Visual Basic 6 program

```
Option Explicit

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

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

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

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

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

PLC program

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

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

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463805067/.exe

5.1.9 Delete handle of a PLC variable

This sample shows how to delete the handle of a PLC variable:

Visual Basic 6 program

```
Dim handle As Long

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

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

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

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463806475/.exe

5.1.10 Event-driven reading (with conversion to another type)

From TwinCAT 2.8 Build > 743 and above

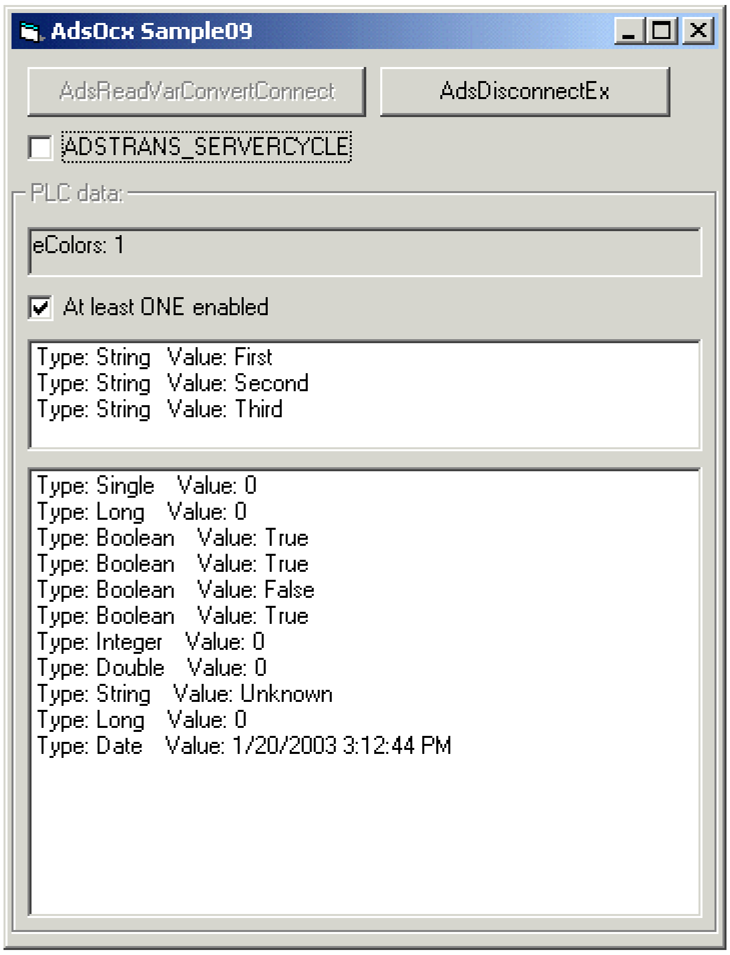
Task

There are 4 variables of different types in the PLC. The variables are to be read out in the most effective way and the values are to be displayed on a Visual Basic form. A checkbox can be used to switch between two different connection modes (ADSTRANS_SERVERCYCLE or ADSTRANS_SERVERONCHA). Two buttons can be used to establish or break the connection to the PLC variables.

The PLC variables are structured data types. The PLC sends these as a data block, for instance, to the AdsOcx client. However, the AdsOcx can pass to the VB event routine only variables of the specific data type as parameters, including the variant type. With the method AdsReadVarConvertConnect [▶ 36] the type of the variant variable in the VB event routine can be set by the user beforehand. The event data is then copied into the variant variable by AdsOcx, and so passed to the VB event routine. A variant array can also represent a complex structure in the PLC. How much data can be copied into the individual variant elements depends on the type of the individual elements. A few exceptions need to be considered, such as occur with strings (the string length must be appropriately set beforehand) and boolean variables (where a 2-byte VB boolean is formed from 1 byte of data).

The following PLC variables are to be displayed on the form:

- The value of an enumeration type (enum) is to be read into a long variable and displayed in the label.
- The value of a structured data type (a structure containing four booleans) is to be read into a long variable and displayed in a checkbox. The checkbox is to be selected if one of the boolean variables in the PLC is TRUE.
- The value of a string array is to be displayed in a listbox.
- The value of a structured data type is to be read into a variant array and displayed in a further listbox.



```
eColors: 1
```

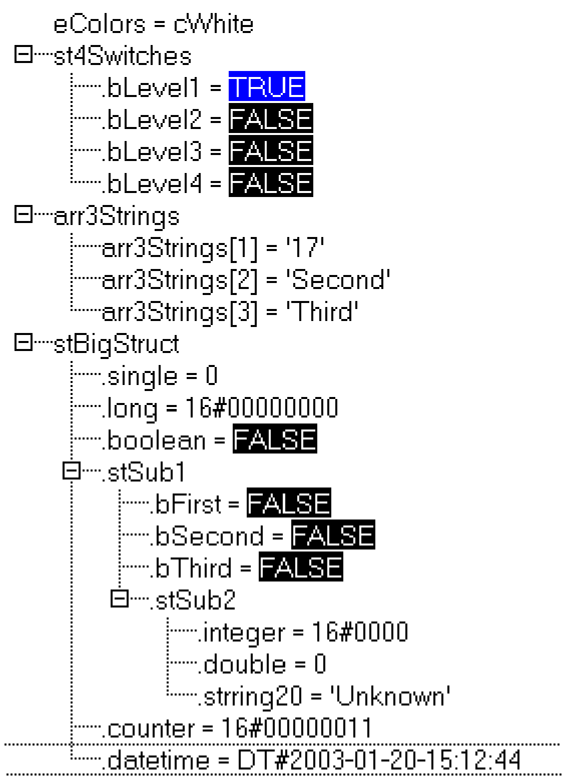
```
Type: String  Value: First
Type: String  Value: Second
Type: String  Value: Third
```

```
Type: Single   Value: 0
Type: Long   Value: 0
Type: Boolean   Value: True
Type: Boolean   Value: True
Type: Boolean   Value: False
Type: Boolean   Value: True
Type: Integer   Value: 0
Type: Double   Value: 0
Type: String   Value: Unknown
Type: Long   Value: 0
Type: Date   Value: 1/20/2003 3:12:44 PM
```

The PLC application

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

Online display of the PLC data:



```
     eColors = cWhite
⊟st4Switches
        .bLevel1 = TRUE
        .bLevel2 = FALSE
        .bLevel3 = FALSE
        .bLevel4 = FALSE
⊟arr3Strings
        arr3Strings[1] = '17'
        arr3Strings[2] = 'Second'
        arr3Strings[3] = 'Third'
⊟stBigStruct
        .single = 0
        .long = 16#00000000
        .boolean = FALSE
    ⊟.stSub1
            .bFirst = FALSE
            .bSecond = FALSE
            .bThird = FALSE
        ⊟.stSub2
                .integer = 16#0000
                .double = 0
                .strring20 = 'Unknown'
        .counter = 16#00000011
        .datetime = DT#2003-01-20-15:12:44
```

The definition of the enumeration type:

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

The definition of the structure with four boolean variables:

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

The definition of the structured data type:

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

This, in turn, has two substructures:

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

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

Visual Basic 6 program

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

The connection to the first PLC runtime system is established as the form is loaded:

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

A mouse click on the *AdsReadVarConvertConnect* button establishes a connection to the PLC variables. When successful, the AdsReadVarConvertConnect method returns a handle. Only via this handle the connection is identified and can be terminated later.

1. The enumeration type in the PLC only occupies 2 bytes of memory. These two bytes are read into a long variable (four bytes) and are returned in the event function. Using the VB integer data type would be just as effective.

2. The 4 boolean values in the structure variable occupy 4 individual bytes of PLC memory in the PLC. These are read into a long variable, and returned as a long variable in the event function.

3. The strings in the array occupy a total of 243 bytes of memory in the PLC (defined string length + 1 byte for the null termination) *3. The length of the individual VB strings must correspond to the length of the PLC strings in order to be able to separate the individual strings. If the string has zero length, no event data is copied into a string variable.

4. The structure variable can be read into a one-dimensional variant array. The individual array elements can be of different types. Before establishing the connection, however, the individual array elements must be initialized with the appropriate type.

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

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

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

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


    'Connects to struct variable
    Dim convertedBigStruct(1 To 11) As Variant
    convertedBigStruct(1) = CSng(0)         'stBigStruct.single
    convertedBigStruct(2) = CLng(0)     'stBigStruct.long
    convertedBigStruct(3) = CBool(False)    'stBigStruct.boolean
    convertedBigStruct(4) = CBool(False)    'stBigStruct.stSub1.bFirst
    convertedBigStruct(5) = CBool(False)    'stBigStruct.stSub1.bSecond
    convertedBigStruct(6) = CBool(False)    'stBigStruct.stSub1.bThird
    convertedBigStruct(7) = CInt(0)     'stBigStruct.stSub1.stSub2.integer
    convertedBigStruct(8) = CDbl(0)     'stBigStruct.stSub1.stSub2.double
    convertedBigStruct(9) = CStr(String(21, "*"))'stBigStruct.stSub1.stSub2.string20
    convertedBigStruct(10) = CLng(0)        'stBigStruct.counter
    convertedBigStruct(11) = CDate(0)           'stBigStruct.datetime
    adsErr = AdsOcx1.AdsReadVarConvertConnect(".stBigStruct", adsTransMode, 300, hConnect_BigStruct,
 convertedBigStruct, lstBigStruct)

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

A mouse click on the AdsDisconnectEx [▶ 40] button will break the connections to the PLC variables:

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

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

The AdsReadConvertConnectUpdate [▶ 55] event routine. This event routine is called cyclically (if ADSTRANS_SERVERCYCLE is selected) or is only called when the value of the PLC variable has changed (if ADSTRANS_SERVERONCHA is selected). The hUser parameter can be used to be able to assign the event data to the appropriate control (label, checkbox, listbox).

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

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

    ElseIf TypeOf hUser Is ListBox Then

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

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

Language / IDE	Unpack sample program
Visual Basic 6	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12463807883/.exe

5.2 Delphi - samples

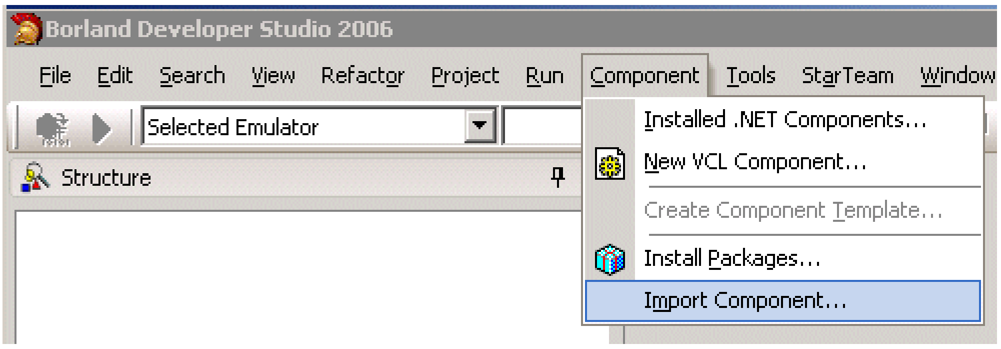
5.2.1 Integration in Delphi

5.2.1.1 Linking to Borland Developer Studio 2006 (VCL for Delphi Win32)

These instructions can also be used for linking ADS-OCX in **Borland Delphi 2005**. The differences compared with **"Borland Delphi 2006" or "Delphi XE2"** are only marginal.

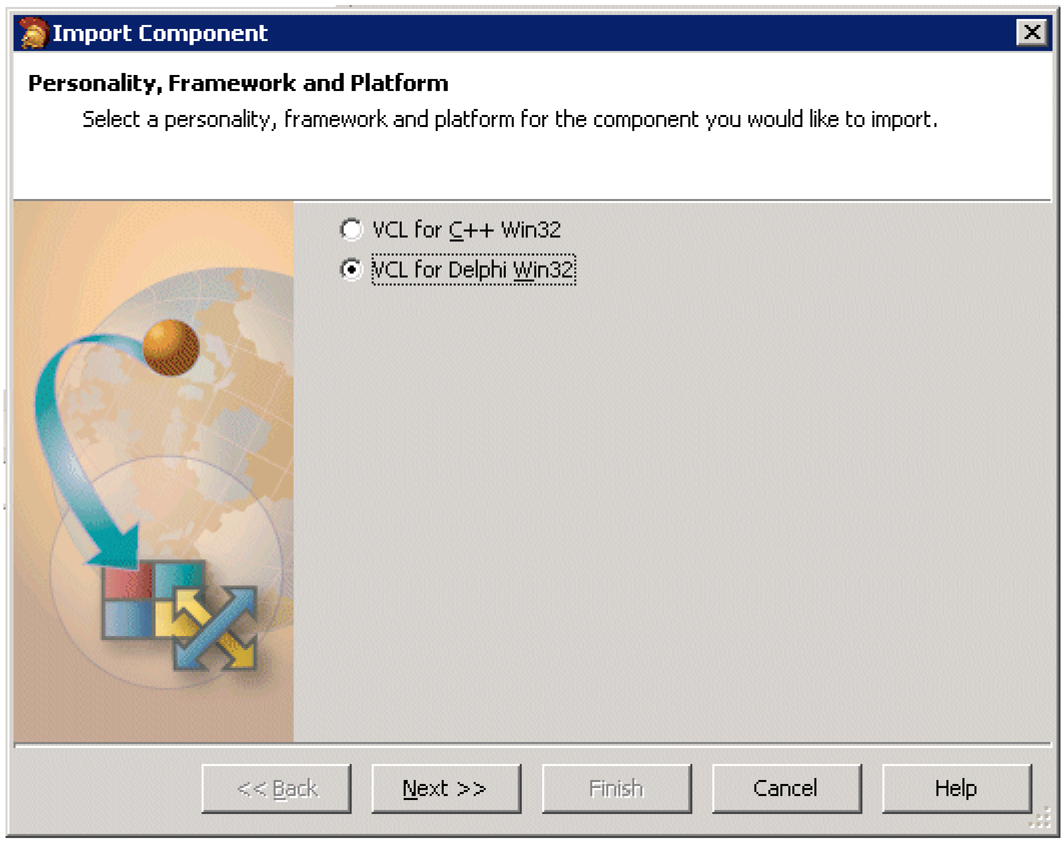
Step 1

First a Delphi unit has to be derived from the ActiveX Control. Select *"Import component..."* under *"Component"*



Step 2

The Component Wizard opens. Select *"VCL for Delphi Win32"* and confirm with *"Next>>"*.


Import Component

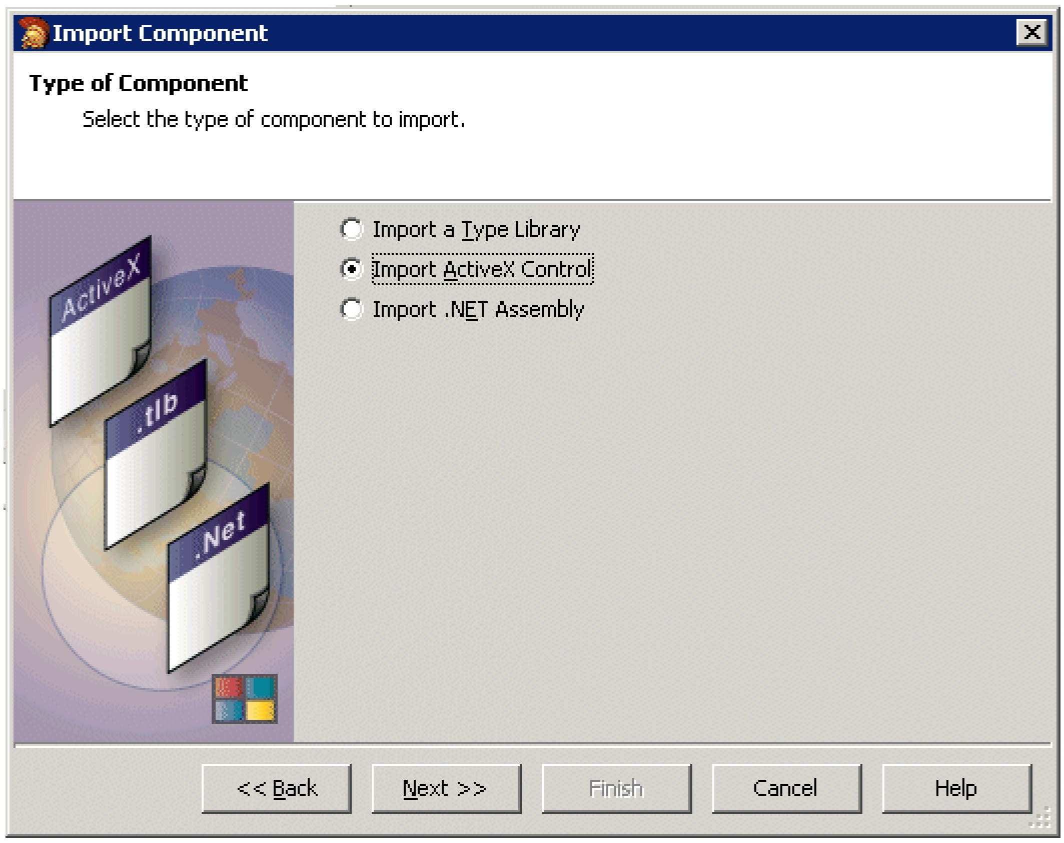
Personality, Framework and Platform

Select a personality, framework and platform for the component you would like to import.

VCL for C++ Win32 [✓ VCL for Delphi Win32]

Step 3

In next dialog select *"Import ActiveX Control"* and click on *"Next>>"*.



Import Component

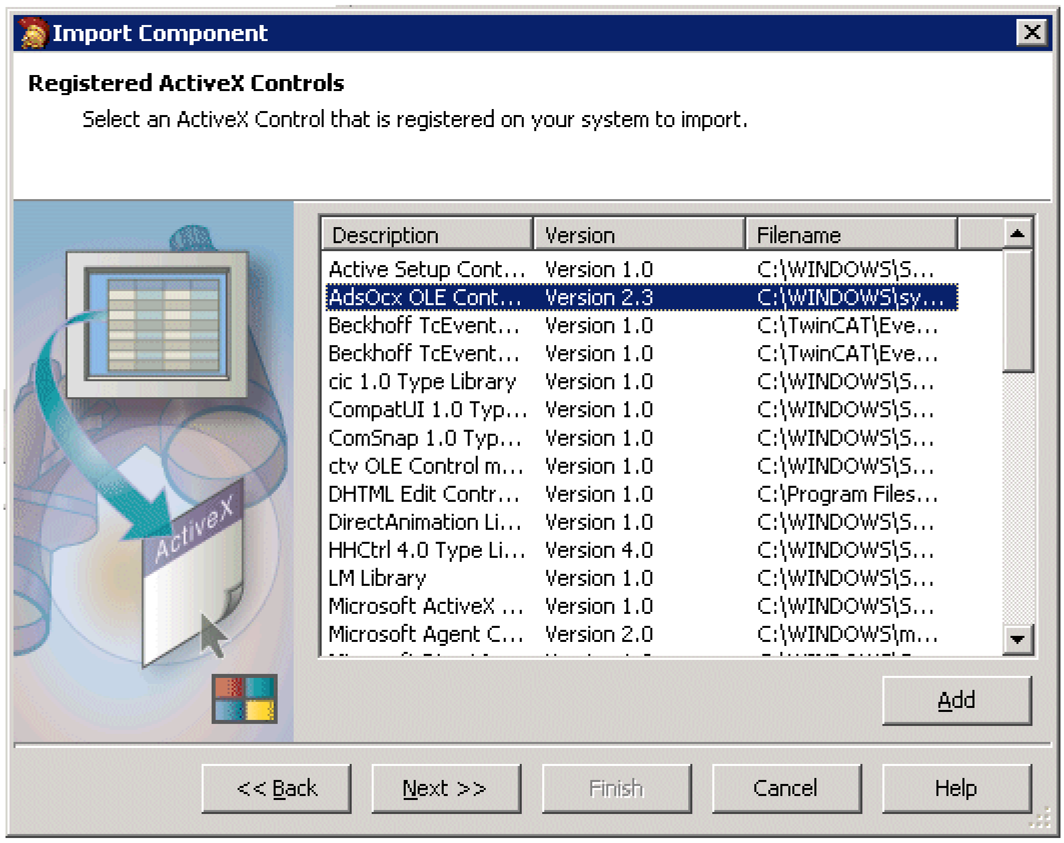
Type of Component

Select the type of component to import.

( ) Import a Type Library
[✓ Import ActiveX Control]
( ) Import .NET Assembly

Step 4

Select the required component from the list of registered ActiveX Controls (AdsOcx OLE Control Module). If the component does not appear in the list of registered elements, it has to be registered and integrated via the *"Add"* button. Then click on *"Next"*.


Import Component

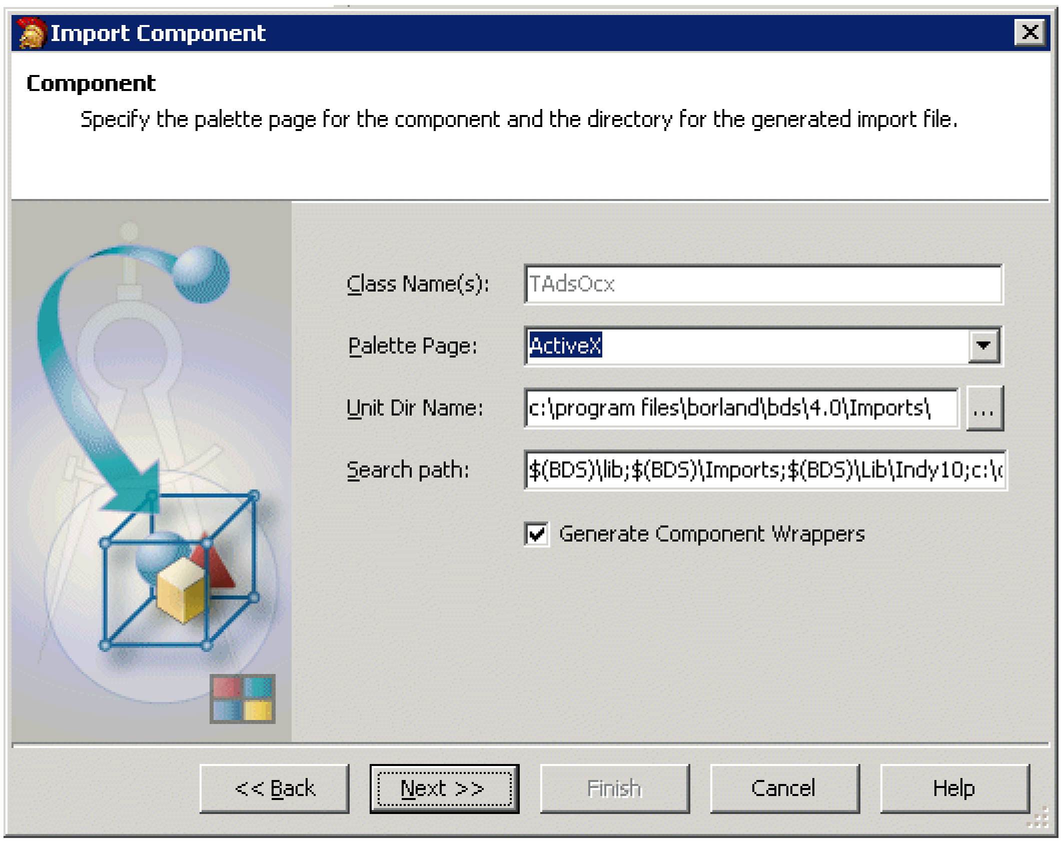
Registered ActiveX Controls

Select an ActiveX Control that is registered on your system to import.

Description	Version	Filename
Active Setup Cont…	Version 1.0	C:\WINDOWS\S…
AdsOcx OLE Cont…	Version 2.3	C:\WINDOWS\sy…
Beckhoff TcEvent…	Version 1.0	C:\TwinCAT\Eve…
Beckhoff TcEvent…	Version 1.0	C:\TwinCAT\Eve…
cic 1.0 Type Library	Version 1.0	C:\WINDOWS\S…
CompatUI 1.0 Typ…	Version 1.0	C:\WINDOWS\S…
ComSnap 1.0 Typ…	Version 1.0	C:\WINDOWS\S…
ctv OLE Control m…	Version 1.0	C:\WINDOWS\S…
DHTML Edit Contr…	Version 1.0	C:\Program Files…
DirectAnimation Li…	Version 1.0	C:\WINDOWS\S…
HHCtrl 4.0 Type Li…	Version 4.0	C:\WINDOWS\S…
LM Library	Version 1.0	C:\WINDOWS\S…
Microsoft ActiveX …	Version 1.0	C:\WINDOWS\S…
Microsoft Agent C…	Version 2.0	C:\WINDOWS\m…

Step 5

In next Wizard window specify the VCL pallete page and the directory for the newly created unit (Default: *C:\program files\borland\bds\4.0\Imports\*).
Confirm with "Next".


Import Component

Component

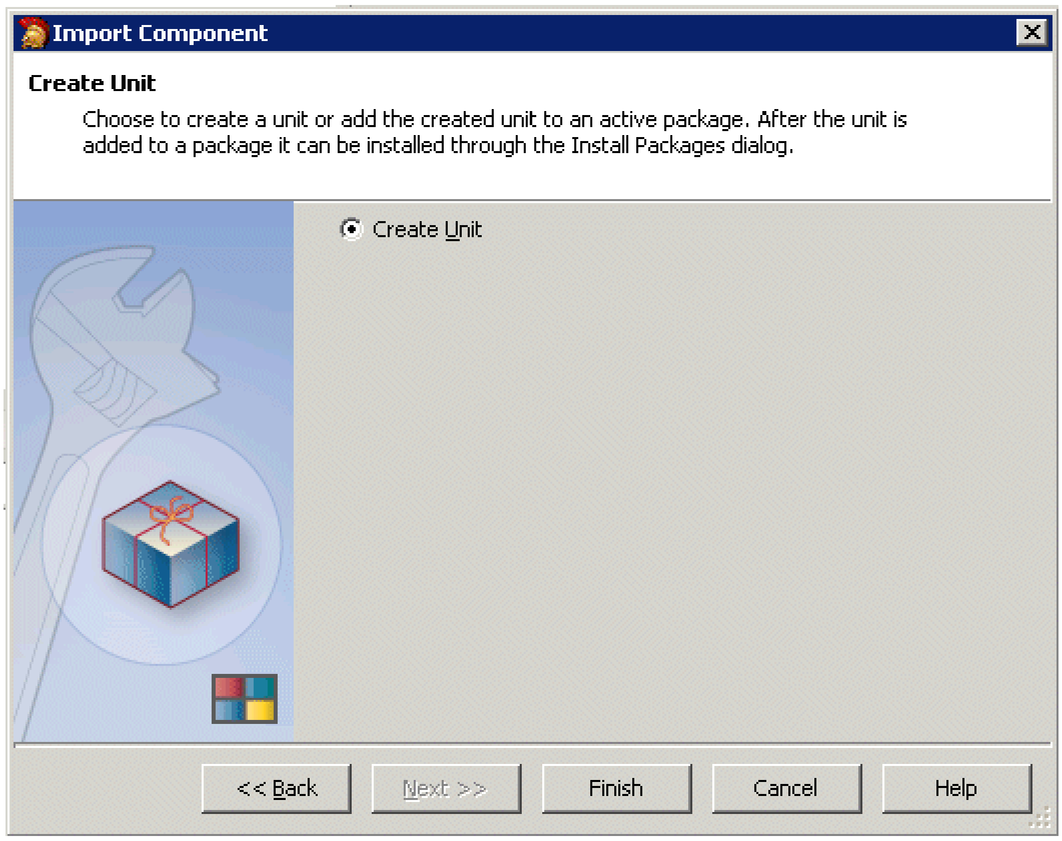
Specify the palette page for the component and the directory for the generated import file.

Class Name(s):	TAdsOcx
Palette Page:	ActiveX
Unit Dir Name:	c:\program files\borland\bds\4.0\Imports\
Search path:	$(BDS)\lib;$(BDS)\Imports;$(BDS)\Lib\Indy10;c:\(

☑ Generate Component Wrappers

Step 6

In the next step a unit for the ActiveX Component is generated. Confirm with "Finish".

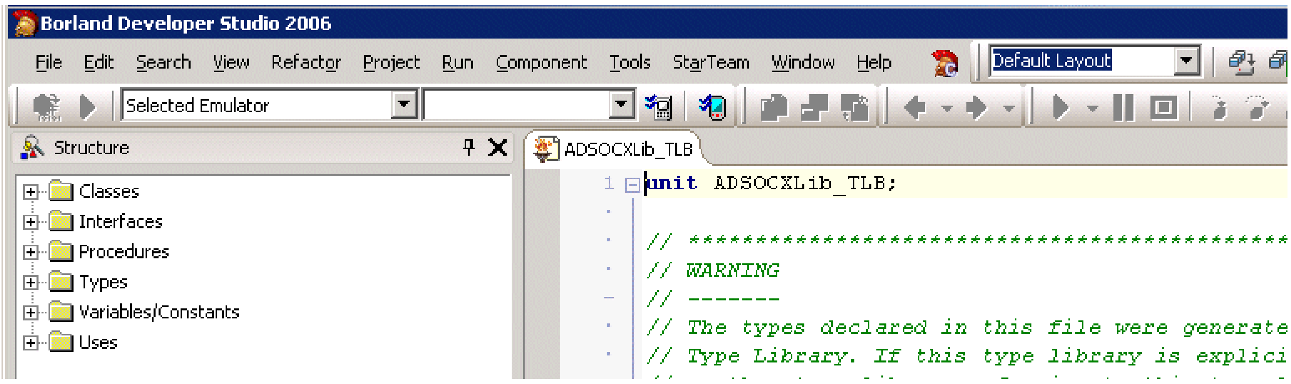


Create Unit

Choose to create a unit or add the created unit to an active package. After the unit is added to a package it can be installed through the Install Packages dialog.

(●) Create Unit

The generated unit is opened automatically for verification:

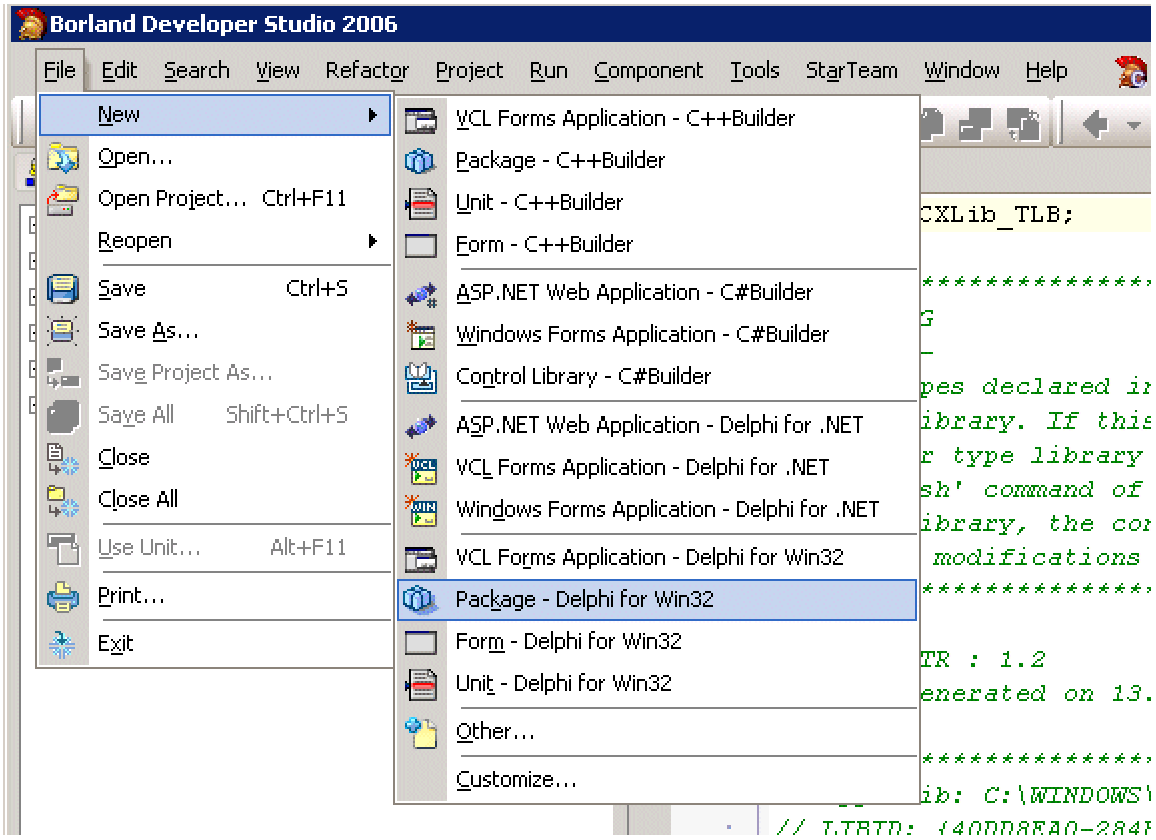


```
unit ADSOCXLib_TLB;

// ************************************************
// WARNING
// -------
// The types declared in this file were generate
// Type Library. If this type library is explici
```

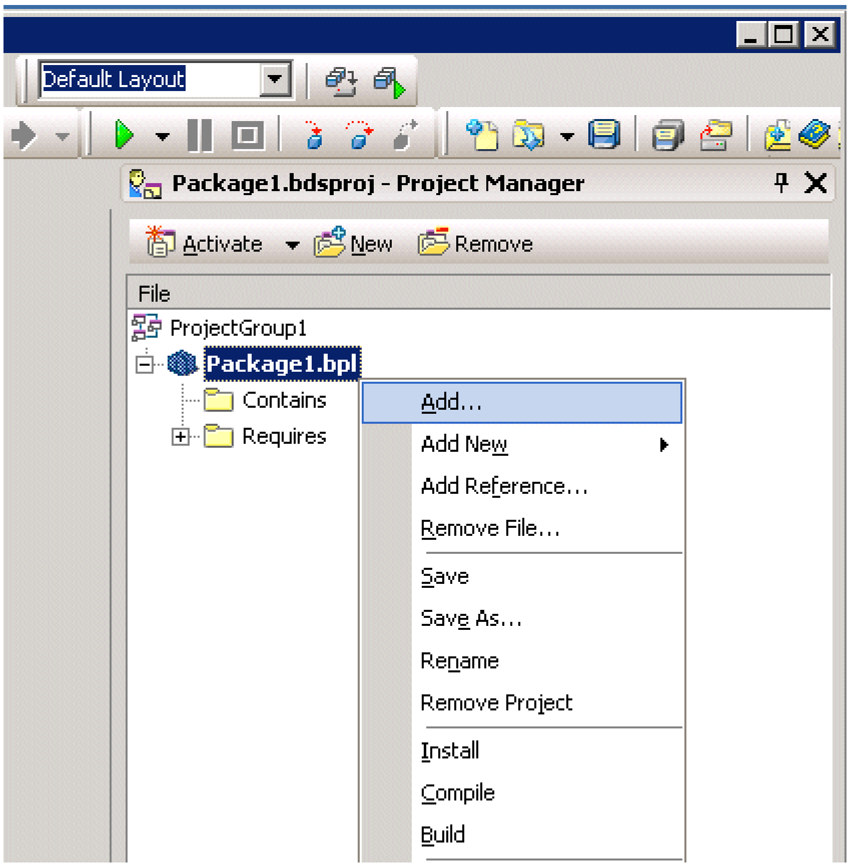
Step 7

In the next step a new package has to be generated. Click on *"File -> New -> Package"* in the main menu.


Borland Developer Studio 2006
File Edit Search View Refactor Project Run Component Tools StarTeam Window Help
New
Open...
Open Project... Ctrl+F11
Reopen
Save Ctrl+S
Save As...
Save Project As...
Save All Shift+Ctrl+S
Close
Close All
Use Unit... Alt+F11
Print...
Exit
VCL Forms Application - C++Builder
Package - C++Builder
Unit - C++Builder
Form - C++Builder
ASP.NET Web Application - C#Builder
Windows Forms Application - C#Builder
Control Library - C#Builder
ASP.NET Web Application - Delphi for .NET
VCL Forms Application - Delphi for .NET
Windows Forms Application - Delphi for .NET
VCL Forms Application - Delphi for Win32
Package - Delphi for Win32
Form - Delphi for Win32
Unit - Delphi for Win32
Other...
Customize...

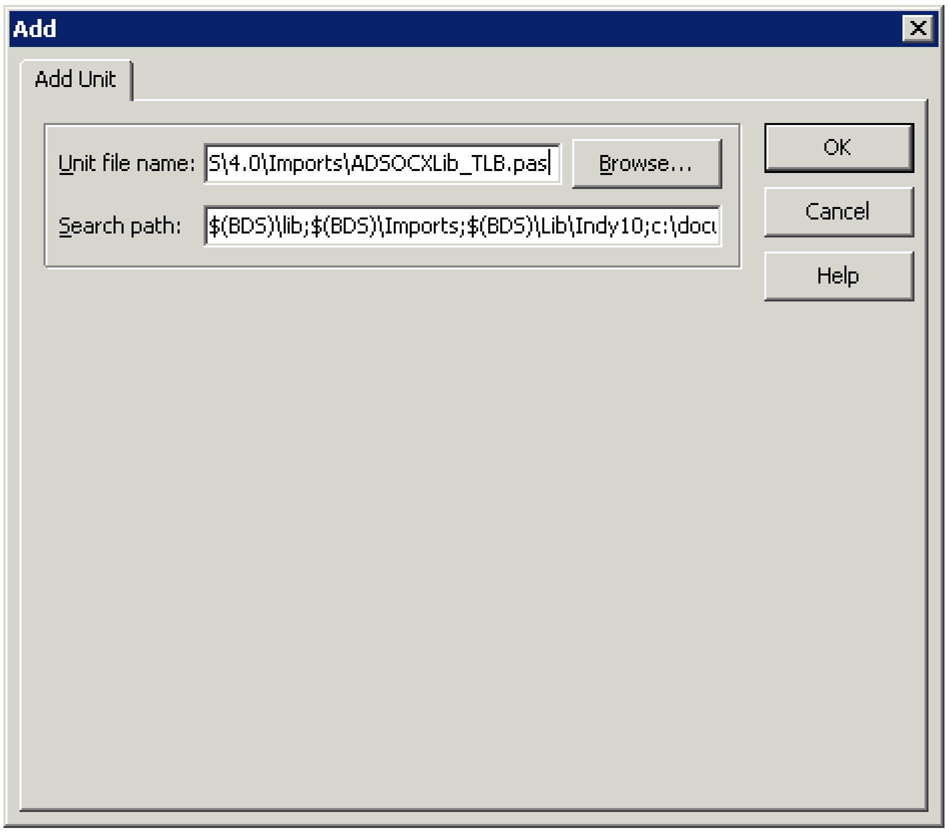
Step 8

The previously generated unit must now be inserted into the newly created package.
Click the right mouse button in the project manager and select the entry *"Package1.bpl"* and in the context menu that opens *"Add"*.



Step 9

In the *"Add"* window enter the storage location for the unit previously generated for the ActiveX Component (Default: *C:\program files\borland\bds\4.0\Imports\ADSOCXLib_TLB.pas*).

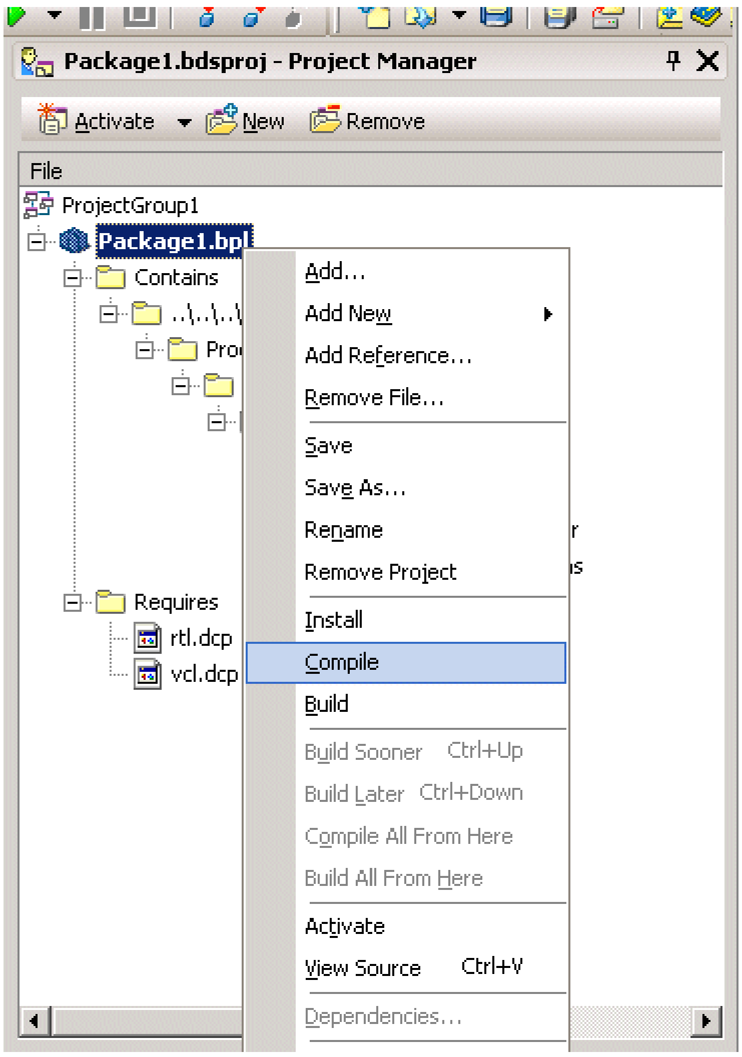


Unit file name: S\4.0\Imports\ADSOCXLib_TLB.pas

Search path: $(BDS)\lib;$(BDS)\Imports;$(BDS)\Lib\Indy10;c:\docu

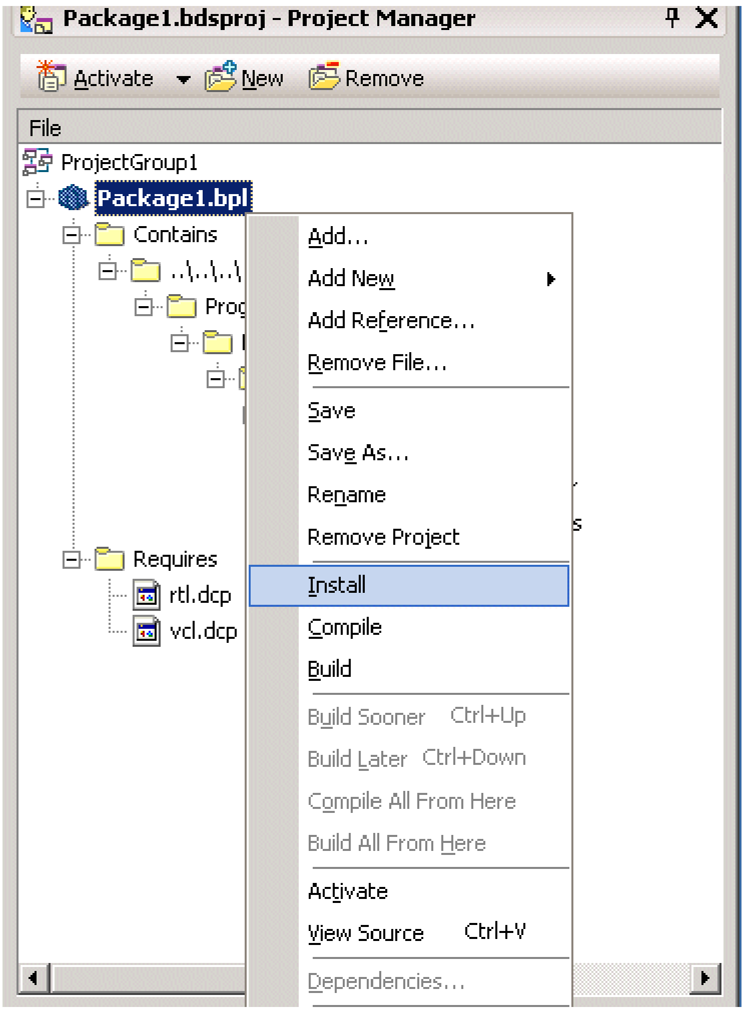
Step 10

For compiling the package right-click on *"Package1.bpl"* and select *"Compile"* in the context menu.

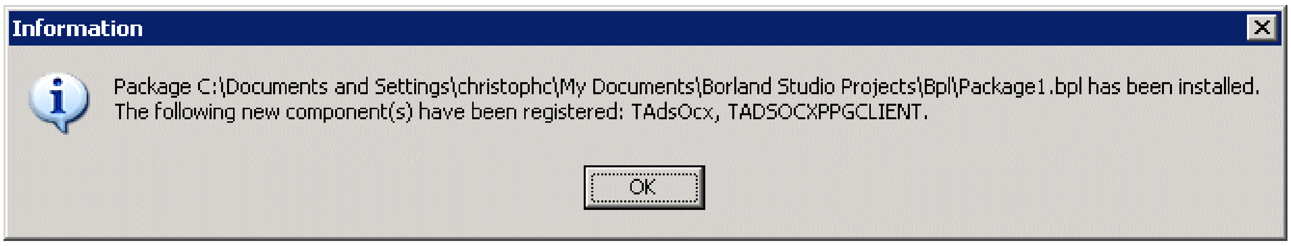


Step 11

After the new package has been compiled select *"Install"* in the context menu.



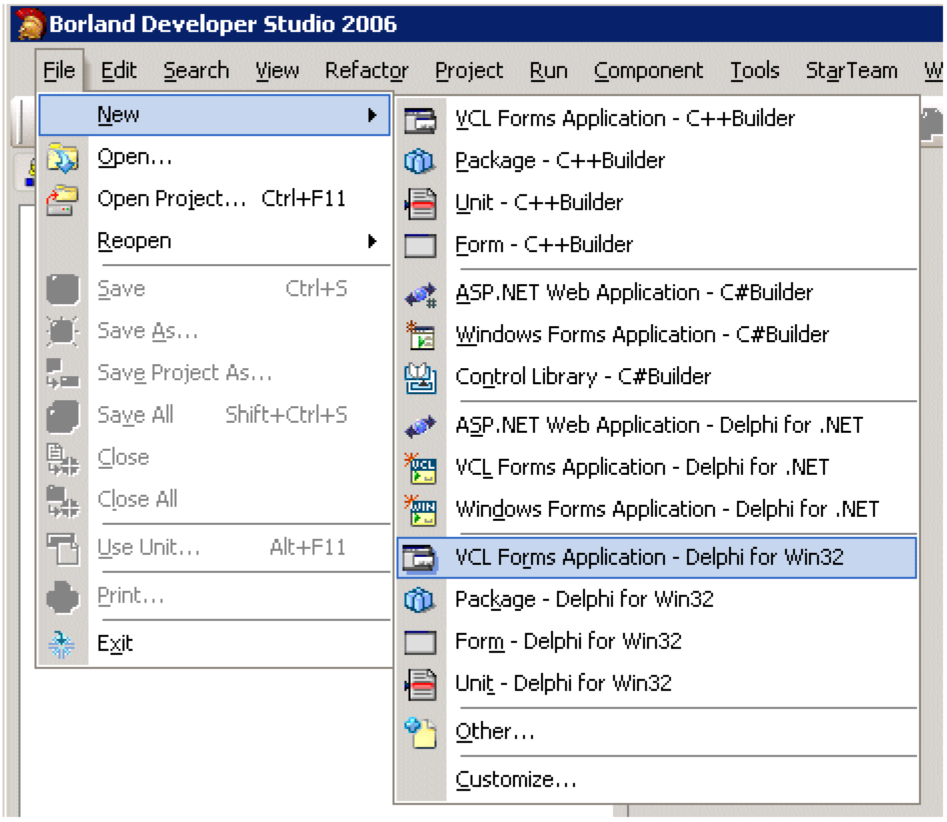
The installation is now complete. The following message appears:

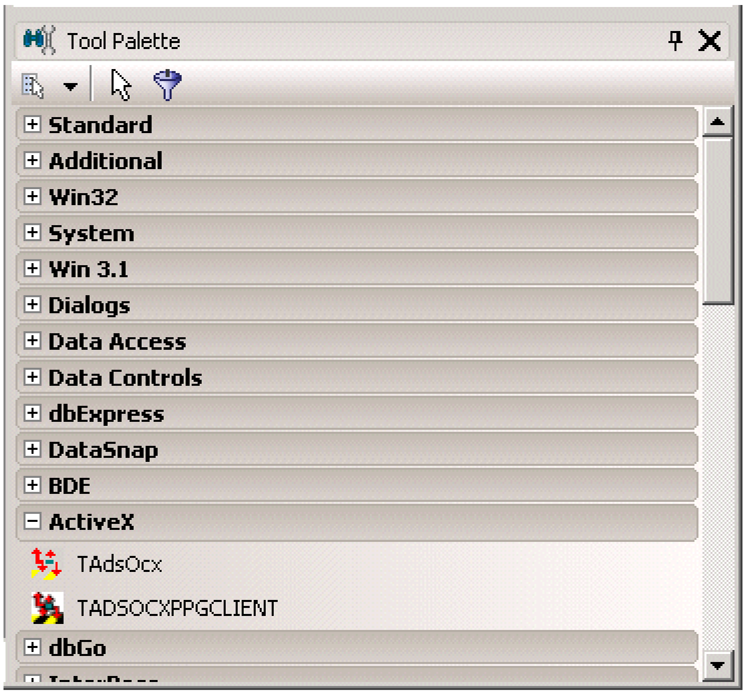

Information

Package C:\Documents and Settings\christophc\My Documents\Borland Studio Projects\Bpl\Package1.bpl has been installed.
The following new component(s) have been registered: TAdsOcx, TADSOCXPPGCLIENT.

After the installation

The ActiveX Component appears in the specified category when a new *"VCL Forms Application - Delphi for Win32"* is created, for example.


Borland Developer Studio 2006
File Edit Search View Refactor Project Run Component Tools StarTeam W
New
Open...
Open Project... Ctrl+F11
Reopen
Save Ctrl+S
Save As...
Save Project As...
Save All Shift+Ctrl+S
Close
Close All
Use Unit... Alt+F11
Print...
Exit
VCL Forms Application - C++Builder
Package - C++Builder
Unit - C++Builder
Form - C++Builder
ASP.NET Web Application - C#Builder
Windows Forms Application - C#Builder
Control Library - C#Builder
ASP.NET Web Application - Delphi for .NET
VCL Forms Application - Delphi for .NET
Windows Forms Application - Delphi for .NET
VCL Forms Application - Delphi for Win32
Package - Delphi for Win32
Form - Delphi for Win32
Unit - Delphi for Win32
Other...
Customize...



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

Please note the information about restrictions and limitations [▶ 104] when using AdsOcx in Delphi applications.

ActiveX controls can be integrated into Delphi in two ways:

1. Implementing via the import of the ActiveX control [▶ 98]

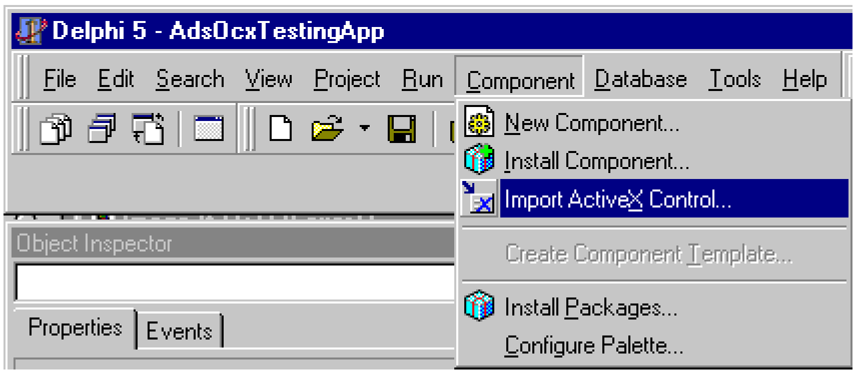
2. Implementing via the import of the type library of the ActiveX control [▶ 100]

With the older versions of Delphi you still have to

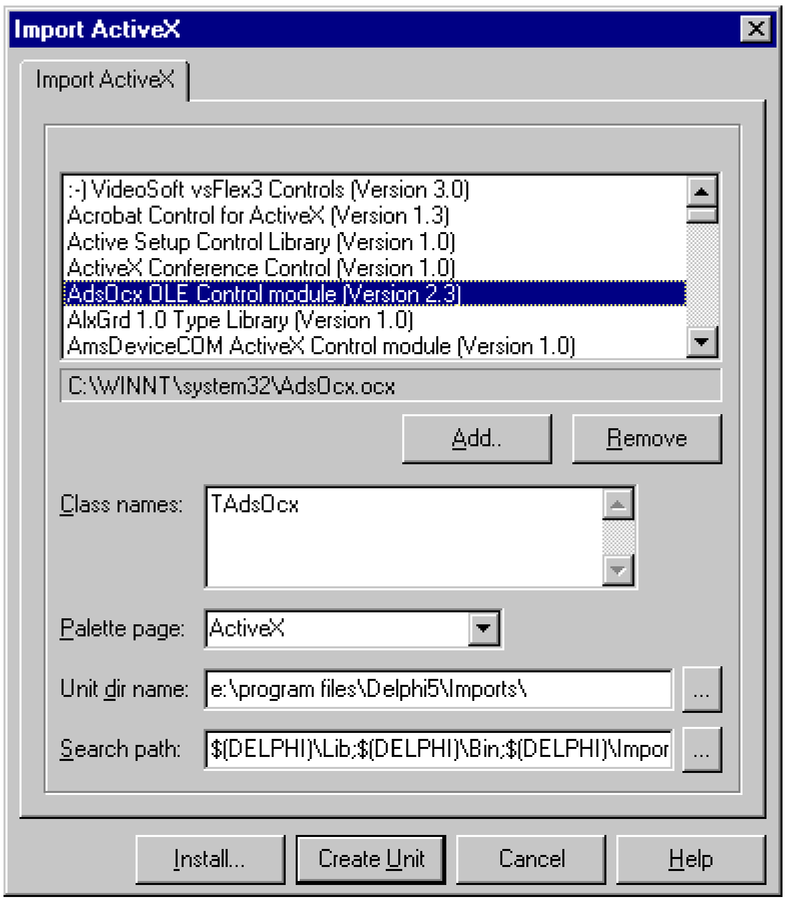
3. Install AdsOcx via the generated type library in the component palette [▶ 102]

1. Implementing via the import of the ActiveX control

1.1 Use the menu command *Component->Import ActiveX Control* to open the Import ActiveX dialog box.


Delphi 5 - AdsOcxTestingApp
File Edit Search View Project Run Component Database Tools Help
New Component...
Install Component...
Import ActiveX Control...
Create Component Template...
Install Packages...
Configure Palette...
Object Inspector
Properties
Events

1.2 In the dialog box, select the *AdsOcx OLE Control module* from the list of ActiveX controls and confirm with a mouse click on *Install.....* If the AdsOcx control is not in the list, you can add it using the *Add...* command. By default, the AdsOcx is located in the *.../WinNT/System32* folder.


Import ActiveX

:-) VideoSoft vsFlex3 Controls (Version 3.0)
Acrobat Control for ActiveX (Version 1.3)
Active Setup Control Library (Version 1.0)
ActiveX Conference Control (Version 1.0)
AdsOcx OLE Control module (Version 2.3)
AlxGrd 1.0 Type Library (Version 1.0)
AmsDeviceCOM ActiveX Control module (Version 1.0)

C:\WINNT\system32\AdsOcx.ocx

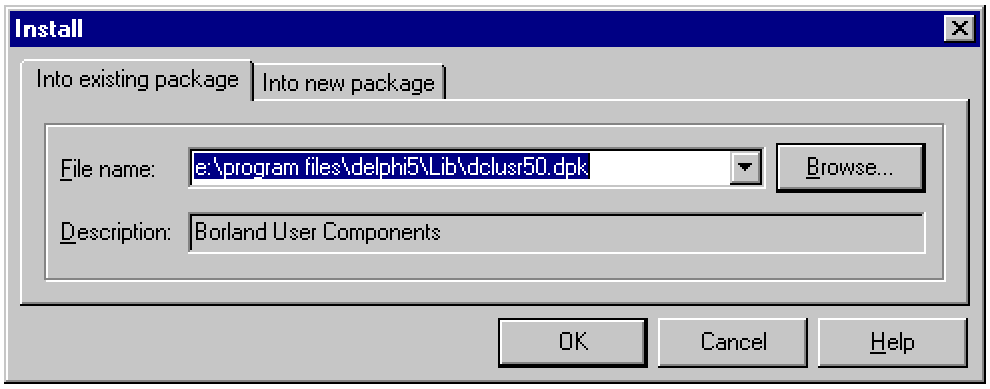
Class names: TAdsOcx

Palette page: ActiveX

Unit dir name: e:\program files\Delphi5\Imports\

Search path: $(DELPHI)\Lib;$(DELPHI)\Bin;$(DELPHI)\Impor

1.3 In the Install dialog box, confirm with OK.

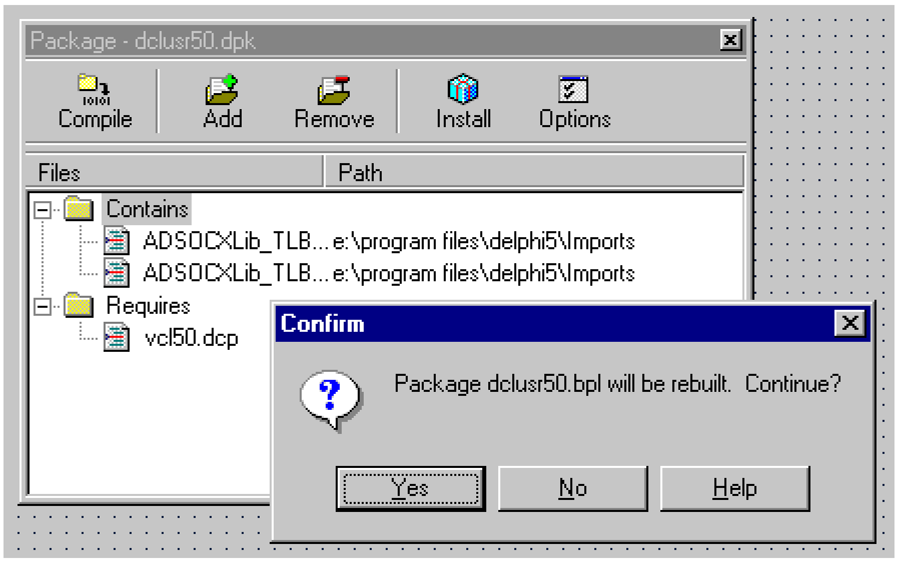

Install

Into existing package | Into new package

File name: e:\program files\delphi5\Lib\dclusr50.dpk

Description: Borland User Components

1.4 The package with user-defined components must be rebuilt. Confirm with *Yes*.

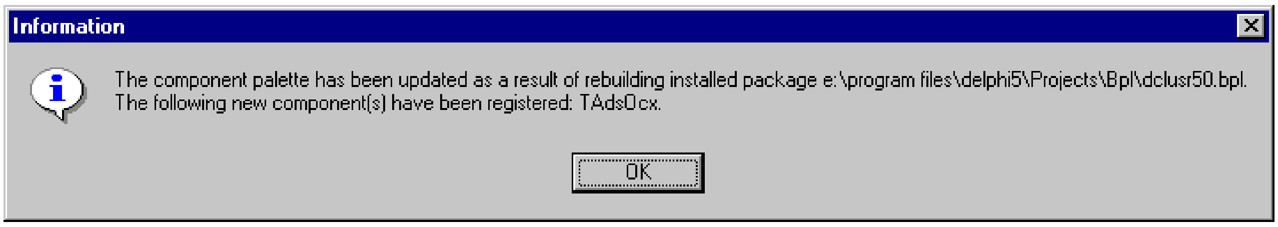

Package - dclusr50.dpk

Files	Path
Contains	
ADSOCXLib_TLB...	e:\program files\delphi5\Imports
ADSOCXLib_TLB...	e:\program files\delphi5\Imports
Requires	
vcl50.dcp	

Confirm

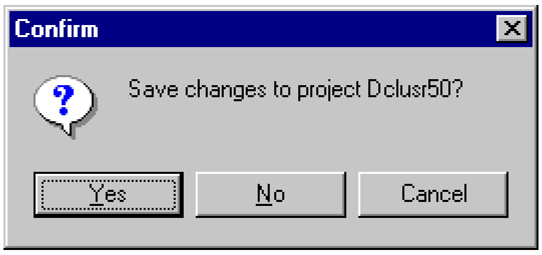
Package dclusr50.bpl will be rebuilt. Continue?

1.5 If successful, the AdsOcx component is registered. Confirm with *OK*.


Information

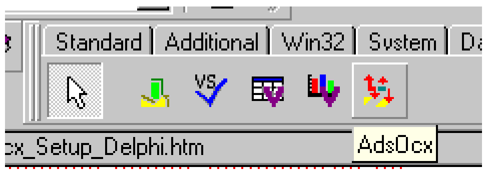
The component palette has been updated as a result of rebuilding installed package e:\program files\delphi5\Projects\Bpl\dclusr50.bpl.
The following new component(s) have been registered: TAdsOcx.

1.6 Close the Package Editor and save the changes with *Yes*.


Confirm

Save changes to project Dclusr50?

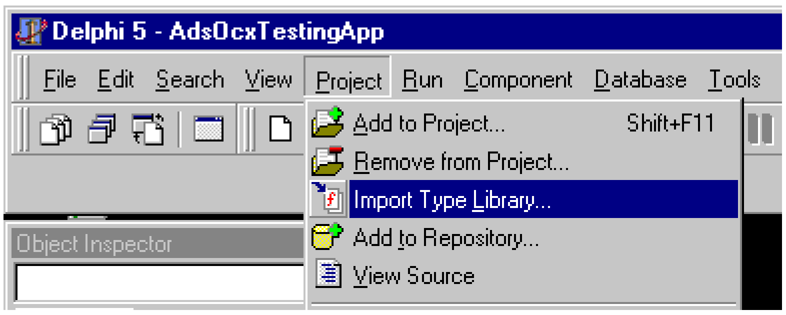
From now on you can use the AdsOcx component from the ActiveX components palette in a new project.



2. Implementing via the import of the type library of the ActiveX control

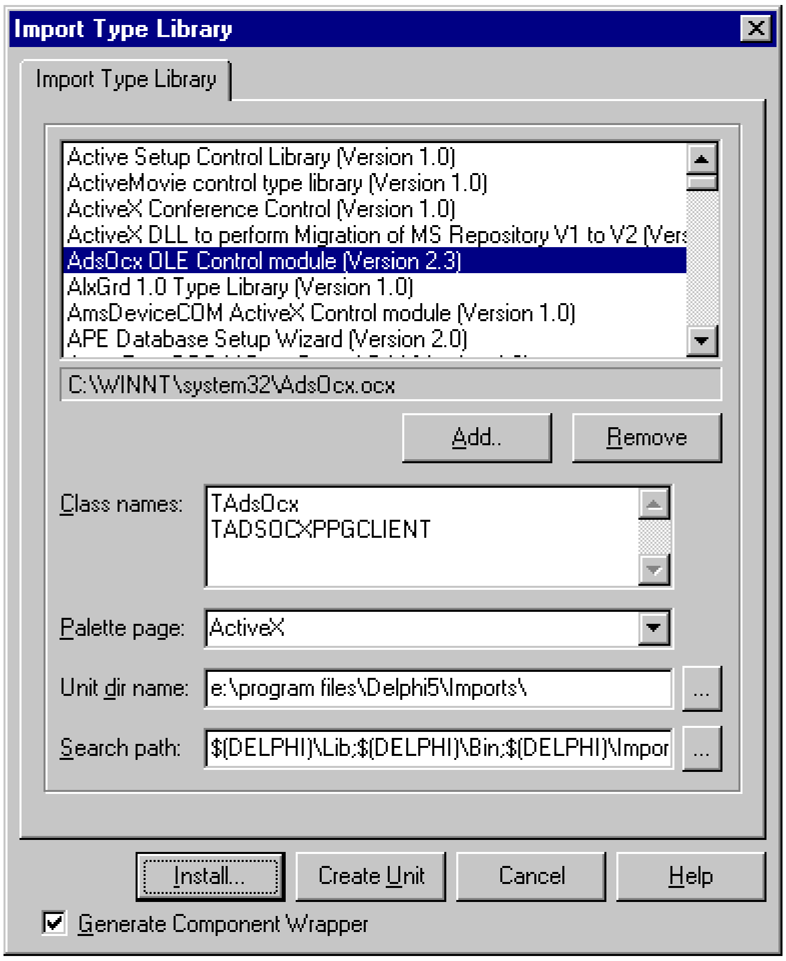
2.1 In order to be able to link the AdsOcx into Delphi's component palette, it is first necessary to generate a type library (with the prototypes for the functions, procedures and data type definitions of the ActiveX control).

The type library can be generated via *Project -> Import Type Library*.



2.2 In the dialog box that opens, select the *AdsOcx OLE Control module* from the list of ActiveX controls and confirm with *Install.....*

If the AdsOcx is not in the selection list, you can add it using the *Add...* command. The AdsOcx is normally located in the *.../WinNT/System32* folder, to which it is copied during the installation of TwinCAT.


Import Type Library

Import Type Library

Active Setup Control Library (Version 1.0)
ActiveMovie control type library (Version 1.0)
ActiveX Conference Control (Version 1.0)
ActiveX DLL to perform Migration of MS Repository V1 to V2 (Vers
AdsOcx OLE Control module (Version 2.3)
AlxGrd 1.0 Type Library (Version 1.0)
AmsDeviceCOM ActiveX Control module (Version 1.0)
APE Database Setup Wizard (Version 2.0)

C:\WINNT\system32\AdsOcx.ocx

Add.. Remove

Class names: TAdsOcx
TADSOCXPPGCLIENT

Palette page: ActiveX

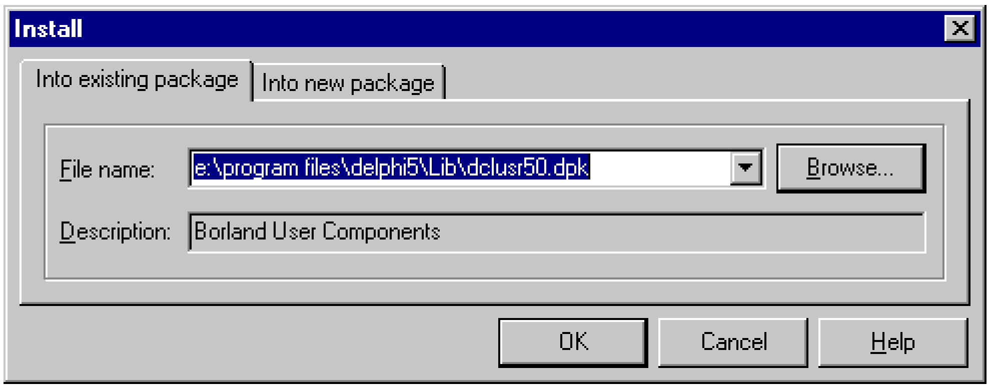
Unit dir name: e:\program files\Delphi5\Imports\

Search path: $(DELPHI)\Lib;$(DELPHI)\Bin;$(DELPHI)\Impor

Install... Create Unit Cancel Help

☑ Generate Component Wrapper

2.3 In the newer versions of Delphi (e.g. Delphi 5.0), the imported type library is immediately added to the component palette. For the older versions only the type library (e.g. in the folder .../Delphi 3/Imports) is generated and you have to install the AdsOcx via the generated type library into the component palette [▶ 102]. If you have a newer version, confirm in the following dialog with *OK*.


Install

Into existing package | Into new package

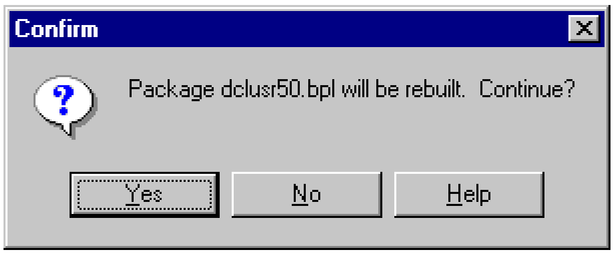
File name: e:\program files\delphi5\Lib\dclusr50.dpk

Browse...

Description: Borland User Components

OK Cancel Help

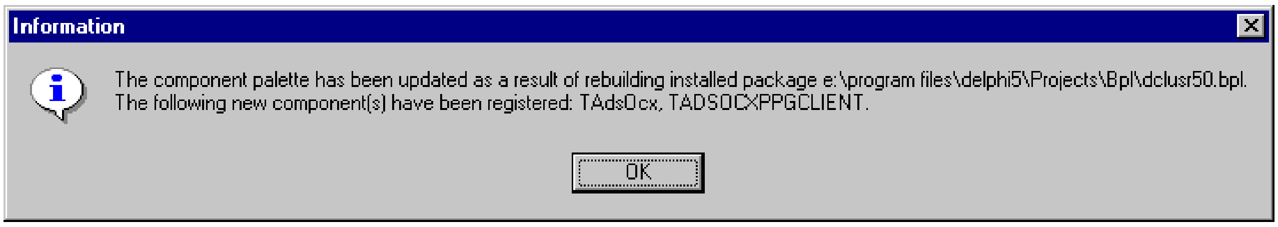
2.4 The package with user-defined components must be rebuilt. Confirm with *Yes*.


Confirm

Package dclusr50.bpl will be rebuilt. Continue?

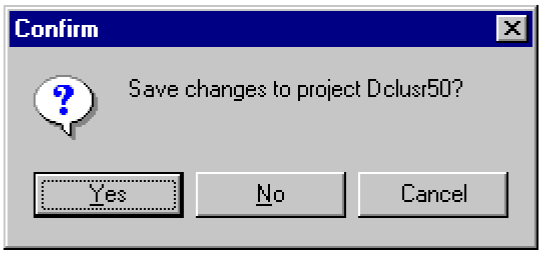
Yes No Help

2.5 On success, the AdsOcx component is registered. Confirm with *OK*.


Information

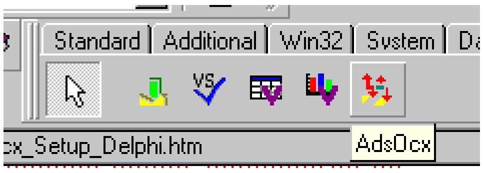
The component palette has been updated as a result of rebuilding installed package e:\program files\delphi5\Projects\Bpl\dclusr50.bpl.
The following new component(s) have been registered: TAdsOcx, TADSOCXPPGCLIENT.

2.6 Close the Package Editor and save the changes with *Yes*.


Confirm

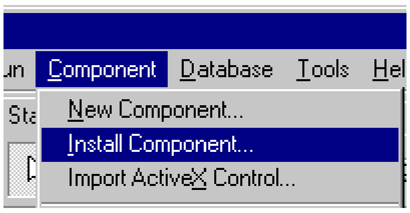
Save changes to project Dclusr50?

From now on you can use the AdsOcx component from the ActiveX components palette in a new project.

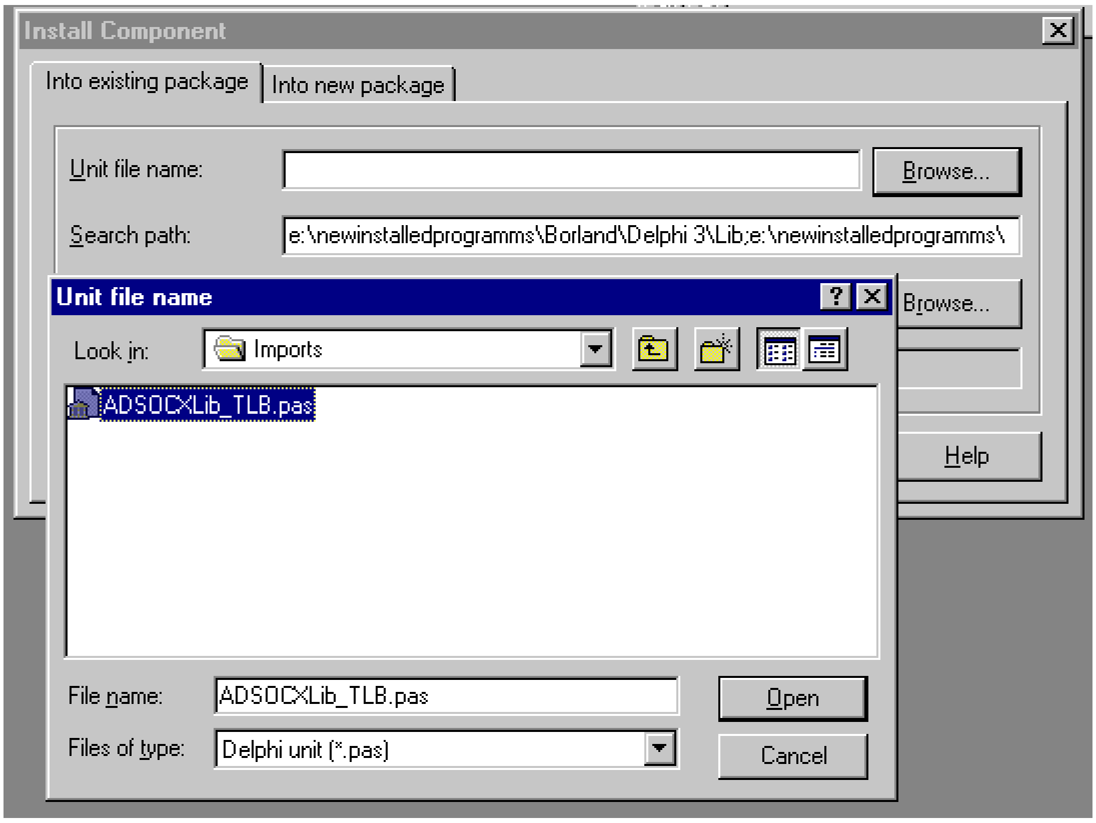


3. Install AdsOcx via the generated type library in the component palette

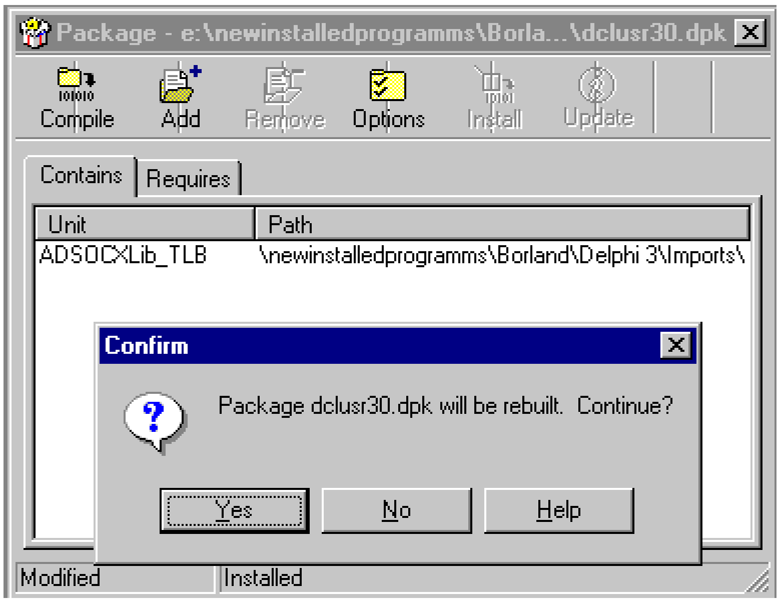
3.1 After the type library has been generated, the AdsOcx can be added as a new component to the component palette from the Pascal file generated in the process (by default, the ADSOCXLib_TLB.pas file is generated). For this purpose you must select the menu command: *Component -> Install Component...*.



3.2 The *Browse...* command must be used in the dialog box to select the type library that was created beforehand. The type libraries that are generated are usually located in the *.../Delphi 3/Imports/* folder. Select the type library, and confirm with *Open*.


Install Component
Into existing package | Into new package
Unit file name:
Search path: e:\newinstalledprogramms\Borland\Delphi 3\Lib;e:\newinstalledprogramms\
Unit file name
Look in: Imports
ADSOCXLib_TLB.pas
File name: ADSOCXLib_TLB.pas
Files of type: Delphi unit (*.pas)

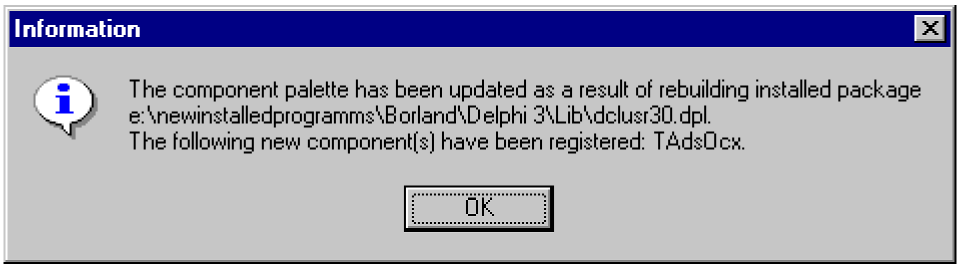
3.3 After this the component palette must be rebuilt. Confirm with *Yes*.


Package - e:\newinstalledprogramms\Borla...\dclusr30.dpk

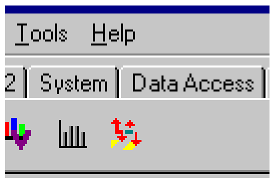
Unit	Path
ADSOCXLib_TLB	\newinstalledprogramms\Borland\Delphi 3\Imports\

Confirm
Package dclusr30.dpk will be rebuilt. Continue?

3.4 The ActiveX control is registered after successful rebuilding. Confirm with *OK*.


Information
The component palette has been updated as a result of rebuilding installed package e:\newinstalledprogramms\Borland\Delphi 3\Lib\dclusr30.dpl.
The following new component(s) have been registered: TAdsOcx.

3.5 The changes to the component package must be saved when closing. The AdsOcx ActiveX Control can now be dragged onto the form from the component palette and used similarly to all the other Delphi components.



5.2.1.3 ADS-OCX limitations in Delphi applications

Delphi's Memory Manager

In the AdsOcx application you have to make sure that the system variable: **IsMultiThread** is set to True in any case. The Memory Manager is "thread-safe" only if this variable is set. Only then will access to shared resources be locked. Often the Memory Manager of Delphi does not set this variable if an included DLL or control starts own threads.

Add the following line to the initialization section of your application:

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

Methods/properties

The following properties, methods and events cause errors in Delphi applications, and must not be used. As can be seen in the table, either the latest version of Delphi should be used, or certain functionalities must be omitted. There are a variety of a updates for the Delphi versions listed, and these may correct certain errors.

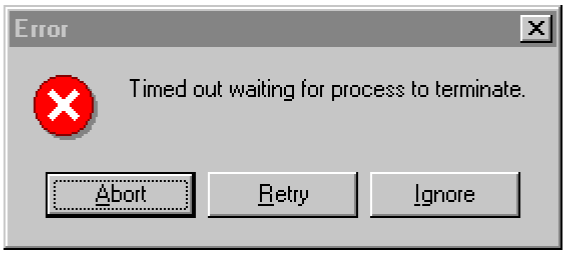
	Error description	Workaround	Delphi version			
Properties			**3.0**	**4.0**	**5.0**	**6.0**
AdsClientType	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsClientAdsState	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsClientAdsControl	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsServerAdsControl	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	

	Error description	Workaround	Delphi version			
AdsServerAdsState	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsServerType	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsServerLastMessage	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsAmsClientNetId	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
AdsAmsServerNetId	There is a **memory leak** when accessing this property. Memory for the returned string is not returned correctly.	n/a	Bug	?	Fixed	
Methods						
All methods	The functions of the generated type library ADSOCXLib_TLB return undefined return parameters.	Please install the Delphi 6 **Update Pack 2** and rebind the ADSOCX.	-	-	**no bug**	Bug

	Error description	Workaround	Delphi version			
AdsSyncReadReq *AdsSyncWriteReq*	These methods allow variables of any type to be transferred to the PLC or to be read from the PLC. The OleVariant parameters, however, are passed by value and not by reference by the *AdsSyncReadReq* method. This means that the method cannot alter the value of the *data* parameter during the call. Although it is true that the PLC variables are copied into a corresponding OleVariant variable during the call, that variable is only a copy of the actual variable from the current parameter list. The method prototypes for the ADS-OCX are generated automatically by the Delphi development environment when the ADS-OCX is linked, and cannot be modified.	**Use the "released" methods** to have synchronous access the PLC variables (e.g. AdsSyncReadIntegerReq() etc.).	Bug	?	Bug	
AdsReadVarConnectEx	Similarly to the process for the AdsSync methods, the OleVariant parameters in the event functions are passed by value and not by reference.	Use the **AdsReadVarConnectEx2** method	Bug	?	Bug	
Events						
AdsReadConnectUpdateEx	An **access violation** is generated when the event function is called.	Use the **AdsReadConnectUpdateEx2** event function	Bug	?	Bug	

5.2.1.4 Reset ADS-OCX application

After a program error, it is often not possible to terminate the application via the operating function "Start -> Program Reset", as is usually the case. The following message of the debugger is the consequence:



Error

Timed out waiting for process to terminate.

The cause of this is that using the ADS-OCX generates a client-server connection to the TwinCAT router, and this must be closed when the application stops. The Delphi application cannot be closed using the "Program Reset" menu command, because at this point there is a connection to the TwinCAT router. Their connection is generated in the application through the assignment of the AdsAmsNetId and the port number.

The following methods may be used to close the application without having to restart the computer:

- First confirm the runtime error with OK, then stop the TwinCAT system via the taskbar, and then reset the Delphi application. This causes existing connections to the clients to be closed. The disadvantage is that the TwinCAT system and the PLC must then be restarted;
- First confirm the runtime error with OK, then call the Router Cleanup via the taskbar, and then reset the Delphi application;
- Make use of exception handling. The AdsAmsDisconnect() can be used to explicitly close the connection to the router;

  try n:=8; Switch[n].Tag:=0; // This index is invalid except on EAccessViolation do begin AdsOcxSPS.AdsAmsDisconnect(); Application.Terminate(); end; end;

5.2.2 Accessing PLC variables in synchronous/asynchronous/ connected modes

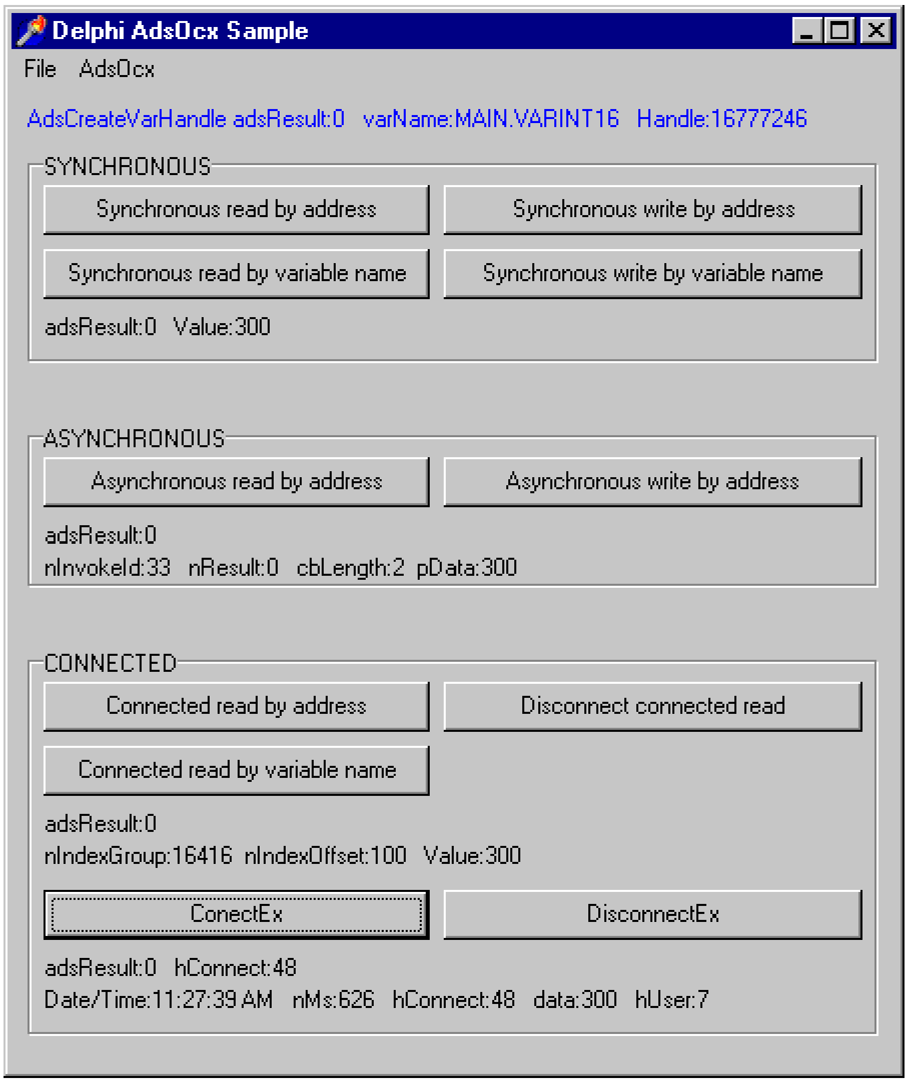
System requirements:

- Delphi 5.0 or higher;
- TwinCAT v2.9 or higher

Task

The sample program shows how AdsOcx methods and events can be used in a Delphi application. The various access types (synchronous/asynchronous/connected) are applied to the PLC variables. The PLC program defines an integer variable at address 100 in the process data flags area. The PLC variable is to be accessed for reading or writing from the Delphi application, using the various access modes.

Description


Delphi AdsOcx Sample
File
AdsOcx
SYNCHRONOUS
Synchronous read by address
Synchronous write by address
Synchronous read by variable name
Synchronous write by variable name
ASYNCHRONOUS
Asynchronous read by address
Asynchronous write by address
CONNECTED
Connected read by address
Disconnect connected read
Connected read by variable name
ConectEx
DisconnectEx

AdsCreateVarHandle adsResult:0 varName:MAIN.VARINT16 Handle:16777246

adsResult:0 Value:300

adsResult:0
nInvokeId:33 nResult:0 cbLength:2 pData:300

adsResult:0
nIndexGroup:16416 nIndexOffset:100 Value:300

adsResult:0 hConnect:48
Date/Time:11:27:39 AM nMs:626 hConnect:48 data:300 hUser:7

Synchronous, asynchronous or connected access to the PLC variables is possible by means of the AdsOcx. In a synchronous access the application is stopped until the requested data has arrived. In an asynchronous access, a request is sent to the PLC, after which execution of the Windows application continues. A callback function is then activated in the Windows application when the requested data has arrived. Under the connected access mode, an event function is called in the Windows application whenever the value of the PLC variable has changed.

Delphi 5 program

In the event function *OnFormCreate*, the AdsCreateVarHandle [▶ 14] method requests a handle for the symbol name of the PLC variable. The handle is then used in the sample application for read or write access to the PLC variable. The *OnDestroy* event function releases the handle once more using the AdsDeleteVarHandle [▶ 15] method when the application is closed.

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

implementation

{$R *.DFM}

procedure TForm1.OnFormCreate(Sender: TObject);
```

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

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

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

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

Synchronous access

A mouse click on one of the buttons in the SYNCHRONOUS group will cause the value of the PLC variable to be read or written synchronously, and to be displayed as text on the form. The PLC variable can be accessed in two ways: via the variable name or via the variable address.

Access by means of the variable address

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

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

Access by means of the variable name

In the case of access using the variable name, the corresponding handle of the PLC variable is used as a parameter in the AdsSyncReadIntegerVarReq [▸ 24] or AdsSyncWriteIntegerVarReq [▸ 28] methods. The handle of the PLC variable is requested in the *OnCreate* event function when the application starts.

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

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

Asynchronous access

The PLC variable can be accessed asynchronously by means of the AdsReadIntegerReq [▸ 31] and AdsWriteIntegerReq [▸ 33] methods.

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

```
end;

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

After an asynchronous access the execution of the Delphi application is continued, and an event function is called in the Windows application once the return parameter is available. In our sample, the event function AdsReadIntegerConf [▶ 55] is called when reading the PLC variable, while for writing the PLC variable the event function called is AdsWriteConf [▶ 58].

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

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

Connected access

In the connected access mode, a "connection" to the PLC variable is established. Depending on the parameters (ADSTRANS_SERVERCYCLE or ADSTRANS_SERVERONCHA), the event functions are called either cyclically or when the PLC variable changes.

In the sample application, clicking on the *Connected read by address* button calls the AdsReadIntegerConnect [▶ 42] method, while a click on the *Connected read by variable name* button calls the AdsReadIntegerVarConnect [▶ 38] method.

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

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

When successful, the AdsReadConnectUpdate [▶ 53] event function is called in the Delphi application, regardless of which of the two methods is used to establish the connection.

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

The AdsReadIntegerDisconnect [▶ 44] method can be used to remove the connection to the PLC variable.

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

ConnectEx methods (connected access with a user handle)

The ConnectEx methods can be used, in a manner similar to that of the Connect methods, to establish connected access to the PLC variables. The ConnectEx methods have the advantage that a user-defined handle can be passed as a parameter in the connect method when the connection is established. This handle can then be evaluated in the event function, and used to identify the PLC variable for which the event function has been called.

Clicking on the *ConnectEx* button will call the AdsReadVarConnectEx2 [▶ 35] method in the *OnConnectExClick* routine.

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

     hUser := 7;    {create user handle}

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

If the connection is successfully established, then the parameters will be displayed as text on the form in the event function AdsReadConnectUpdateEx2 [▶ 54].

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

Clicking with the mouse on the *DisconnectEx* button will call the AdsDisconnectEx [▶ 40] method, and the connection to the PLC variable will be removed.

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

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

Comment

In the course of linking the ADS-OCX into Delphi applications it has been found that the Delphi development environment generates faulty prototypes (more precisely: faulty parameter passing of OleVariant types) for the AdsReadConnectUpdateEx [▶ 53] event function. For this reason, the ADS-OCX has been supplemented with a new AdsReadVarConnectEx2 method and associated AdsReadConnectUpdateEx2 event function. In the new event function the OleVariant parameter is passed by reference instead of by value.

Other

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

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

```
end;

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

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

PLC program

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

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466730763/.exe
Delphi 5 or higher (classic)	

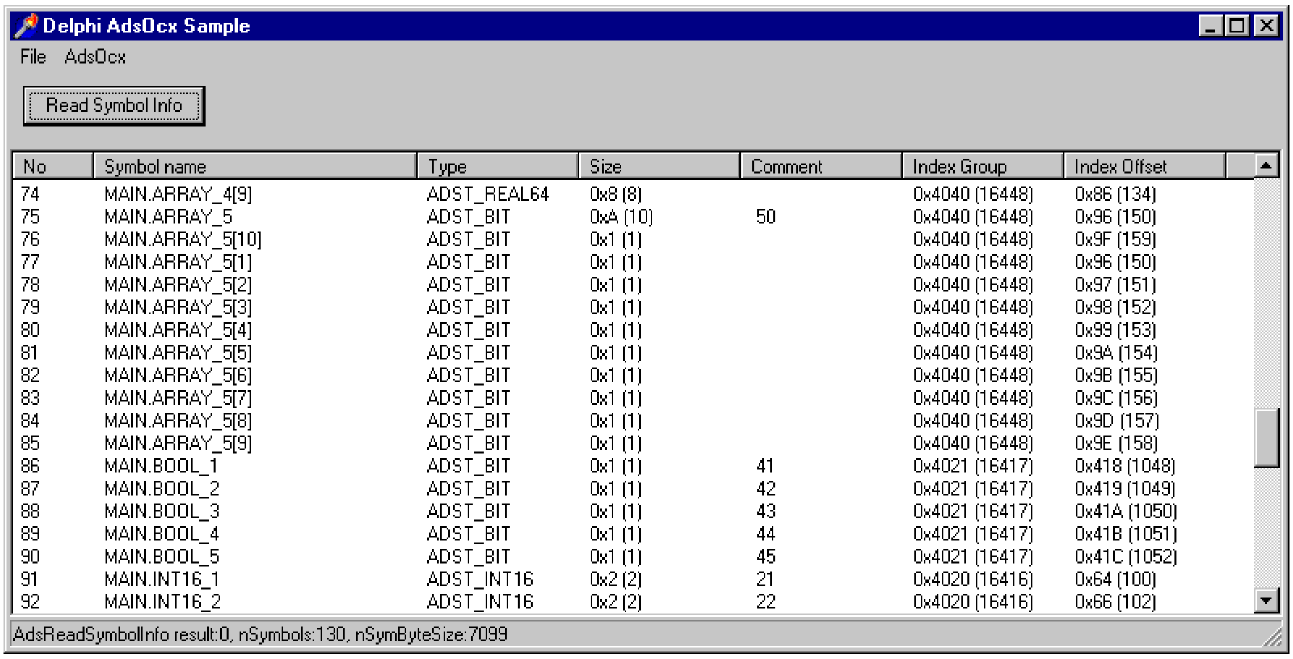
5.2.3 Read the List of an ADS Device's Declared Variables

System requirements:

- Delphi 5.0 or higher;
- TwinCAT v2.9 or higher

Task

The sample program illustrates how the AdsReadSymbolInfo [▶ 21] and AdsEnumSymbols [▶ 16] methods can be used to read the list of declared variables of an ADS device. By clicking on the *Read Symbol Info* button, the symbol information of the first PLC runtime system (port 801), or of an additional task in the TwinCAT System Manager (port 301), is read and displayed in a table.



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

AdsReadSymbolInfo result:0, nSymbols:130, nSymByteSize:7099

Description

Symbol Configuration for the PLC Runtime System

To be able to access the symbol information for a PLC runtime system, it is necessary to activate the symbol generation for the PLC variables or structures, and for the symbol information to be loaded into the PLC runtime system during the project download. The settings necessary for the symbol download can be made in the option dialog for the TwinCAT category in TwinCAT PLC Control. The first PLC runtime system is addressed via port number 801.

Symbol Configuration of the Additional Task in the TwinCAT System Manager

An additional task can be inserted and configured in the TwinCAT System Manager. The variables of the additional task can be linked to other variables (e.g. with the PLC variables, or the I/O variables of a Bus Terminal Controller). To be able to access the additional task's symbol information, the checkbox for symbol generation must be activated in the Task settings configuration dialog. The additional task is addressed via port number 301.

Delphi 5 program

The connection to the first runtime system of the PLC (port 801) on the local PC is established in the *OnFormCreate* event function. At the same time the ListView component and the necessary variables are initialized. The *ReadSymInfoButtonClick* method is called by clicking the *Read Symbol Info* button. In this method, the AdsReadSymbolInfo method is first called to determine the number of available symbols, after which a for-loop is used to read the symbol information for each individual symbol variable. The values are then added to the ListView component by means of the supplementary *AddListViewItem* procedure. The AdsEnumSymbols method possesses a boolean flag, *bNext*. If this flag is set to FALSE, the symbol information of the first symbol is read, but if bNext=TRUE then all the other symbols are read. In order to be able to read the symbol information of the additional task in the TwinCAT System Manager, the *AdsAmsServerPort* property of the AdsOcx component must be set to 301. The port number can be set at runtime using the AdsOcx component's properties page. The properties page can be called in the sample application via the *AdsOcx->Properties* menu.

```
unit SampleUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, OleCtrls, ADSOCXLib_TLB, ExtCtrls, ComCtrls, Menus;

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

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

implementation

{$R *.DFM}

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

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

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

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

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

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

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


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

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

     CreateColumns(Sender);
end;



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

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

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

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

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

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

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

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

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

     ListItem.SubItems.Add(strSymbolName);

     case  nSymbolType of
       0:  strAdsType := 'ADST_VOID';
       16:  strAdsType := 'ADST_INT8';
       17:  strAdsType := 'ADST_UINT8';
       2:  strAdsType := 'ADST_INT16';
       18:  strAdsType := 'ADST_UINT16';
       3:  strAdsType := 'ADST_INT32';
       19:  strAdsType := 'ADST_UINT32';
       20:  strAdsType := 'ADST_INT64';
       21:  strAdsType := 'ADST_UINT64';
       4:  strAdsType := 'ADST_REAL32';
       5:  strAdsType := 'ADST_REAL64';
       65:  strAdsType := 'ADST_BIGTYPE';
       30:  strAdsType := 'ADST_STRING';
       31:  strAdsType := 'ADST_WSTRING';
       32:  strAdsType := 'ADST_REAL80';
       33:  strAdsType := 'ADST_BIT';
       34:  strAdsType := 'ADST_MAXTYPES';
     end;

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


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

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

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

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


end.
```

PLC program

```
PROGRAM MAIN
VAR
    REAL32_1 AT %MB0 : REAL;   (* 1 *)
    REAL32_2 AT %MB4 : REAL;   (* 2 *)
    REAL32_3 AT %MB8 : REAL;   (* 3 *)
    REAL32_4 AT %MB12: REAL;   (* 4 *)
    REAL32_5 AT %MB16: REAL;   (* 5 *)

    REAL64_1 AT %MB20 : LREAL;   (* 6 *)
    REAL64_2 AT %MB28 : LREAL;   (* 7 *)
    REAL64_3 AT %MB36 : LREAL;   (* 8 *)
    REAL64_4 AT %MB44 : LREAL;   (* 9 *)
    REAL64_5 AT %MB52 : LREAL;   (* 10 *)

    INT32_1 AT %MB60 : DINT;   (* 11 *)
    INT32_2 AT %MB64 : DINT;   (* 12 *)
    INT32_3 AT %MB68 : DINT;   (* 13 *)
    INT32_4 AT %MB72 : DINT;   (* 14 *)
    INT32_5 AT %MB76 : DINT;   (* 15 *)

    UINT32_1 AT %MB80 : UDINT;   (* 16 *)
    UINT32_2 AT %MB84 : UDINT;   (* 17 *)
    UINT32_3 AT %MB88 : UDINT;   (* 18 *)
    UINT32_4 AT %MB92 : UDINT;   (* 19 *)
    UINT32_5 AT %MB96 : UDINT;   (* 20 *)

    INT16_1 AT %MB100 : INT;   (* 21 *)
    INT16_2 AT %MB102 : INT;   (* 22 *)
    INT16_3 AT %MB104 : INT;   (* 23 *)
    INT16_4 AT %MB106 : INT;   (* 24 *)
    INT16_5 AT %MB108 : INT;   (* 25 *)

    UINT16_1 AT %MB110 : UINT;   (* 26 *)
    UINT16_2 AT %MB112 : UINT;   (* 27 *)
    UINT16_3 AT %MB114 : UINT;   (* 28 *)
    UINT16_4 AT %MB116 : UINT;   (* 29 *)
    UINT16_5 AT %MB118 : UINT;   (* 30 *)

    INT8_1 AT %MB120 : SINT;   (* 31 *)
    INT8_2 AT %MB121 : SINT;   (* 32 *)
    INT8_3 AT %MB122 : SINT;   (* 33 *)
    INT8_4 AT %MB123 : SINT;   (* 34 *)
    INT8_5 AT %MB124 : SINT;   (* 35 *)

    UINT8_1 AT %MB125 : USINT;   (* 36 *)
    UINT8_2 AT %MB126 : USINT;   (* 37 *)
    UINT8_3 AT %MB128 : USINT;   (* 38 *)
    UINT8_4 AT %MB129 : USINT;   (* 39 *)
    UINT8_5 AT %MB130 : USINT;   (* 40 *)

    BOOL_1 AT %MX131.0 : BOOL;   (* 41 *)
    BOOL_2 AT %MX131.1 : BOOL;   (* 42 *)
    BOOL_3 AT %MX131.2 : BOOL;   (* 43 *)
    BOOL_4 AT %MX131.3 : BOOL;   (* 44 *)
    BOOL_5 AT %MX131.4 : BOOL;   (* 45 *)

    ARRAY_1 : ARRAY[1 .. 10] OF SINT;  (* 46 *)
    ARRAY_2 : ARRAY[1 .. 10] OF INT;   (* 47 *)
    ARRAY_3 : ARRAY[1 .. 10] OF DINT;  (* 48 *)
    ARRAY_4 : ARRAY[1 .. 10] OF LREAL; (* 49 *)
    ARRAY_5 : ARRAY[1 .. 10] OF BOOL;  (* 50 *)
END_VAR
```

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466732171/.exe
Delphi 5 or higher (classic)	

5.2.4 Write array to PLC or read array from PLC

System requirements:

- Delphi 6.0 or higher;
- TwinCAT v2.10 or higher

Use the "released" methods for particular data types. If you want, for example, to read an array in the PLC of type INT, then the following methods may be used, depending on the access type:

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

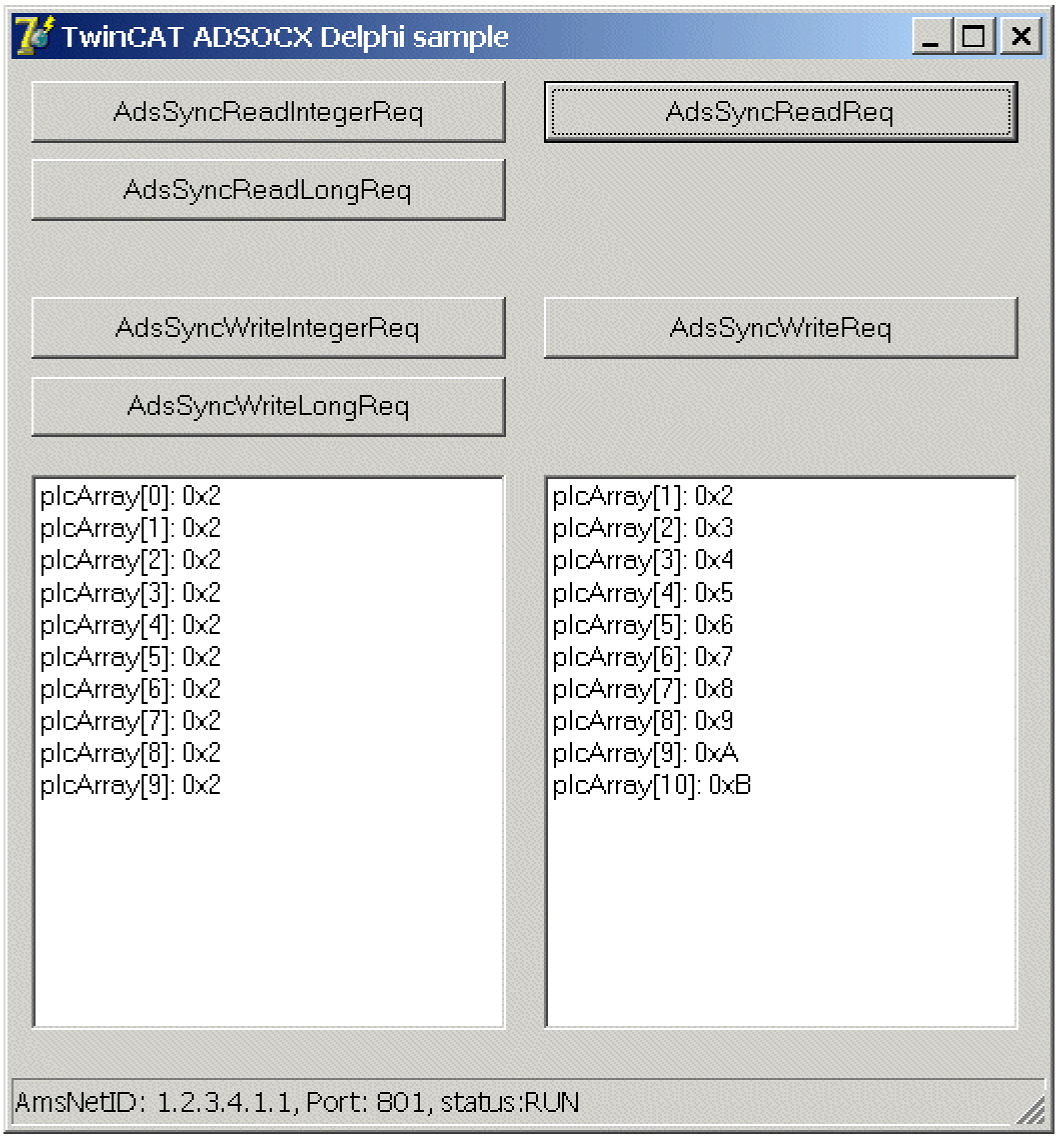
AdsSyncReadIntegerReq(indexGroup : Integer, indexOffset : Integer, length : Integer, var pData : Smallint)

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

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

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

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



```
plcArray[0]: 0x2
plcArray[1]: 0x2
plcArray[2]: 0x2
plcArray[3]: 0x2
plcArray[4]: 0x2
plcArray[5]: 0x2
plcArray[6]: 0x2
plcArray[7]: 0x2
plcArray[8]: 0x2
plcArray[9]: 0x2
```

```
plcArray[1]: 0x2
plcArray[2]: 0x3
plcArray[3]: 0x4
plcArray[4]: 0x5
plcArray[5]: 0x6
plcArray[6]: 0x7
plcArray[7]: 0x8
plcArray[8]: 0x9
plcArray[9]: 0xA
plcArray[10]: 0xB
```

AmsNetID: 1.2.3.4.1.1, Port: 801, status:RUN

String arrays cannot be accessed in this way. The length of the data to be read or written is determined by the number of elements to be read or written multiplied by the byte size of an element. This length must be passed in the *length* or *cbLength* parameters. The parameter *pData* is used to pass the first element of the Delphi array.

Sample:

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

Delphi 6 program:

Reading an array from the PLC:

```
procedure TForm1.SyncReadArrayVarButtonClick(Sender: TObject);
var i, hVar, AdsResult:integer;
    varIntArray       : ARRAY[1..9] OF Smallint;
begin
    AdsResult := AdsOcx1.AdsCreateVarHandle( 'MAIN.VARINTARRAY', hVar );
    if AdsResult = 0 then
    begin
    AdsResult := AdsOcx1.AdsSyncReadIntegerVarReq( hVar, sizeof(varIntArray), varIntArray[1] );
    if AdsResult = 0 then
    begin
        ListBox1.Clear();
        for i:=1 to 9 do
        ListBox1.Items.Add( Format('varIntArray[%d] = %d', [i, varIntArray[i]] ) );
    end
    else Label1.Caption := Format('AdsSyncReadIntegerVarReq error:%d', [AdsResult] );
    AdsOcx1.AdsDeleteVarHandle( hVar );
    end
    else Label1.Caption := Format('AdsCreateVarHandle error:%d', [AdsResult] );
end;
```

Writing an array into the PLC:

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

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

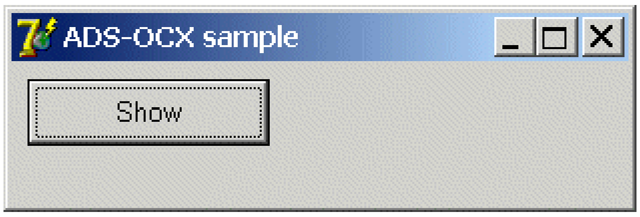
Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466733579/.exe
Delphi 6 or higher (classic)	

5.2.5 Call ADS-OCX property page

System requirements:

- Delphi 7.0 or higher;
- TwinCAT v2.9 or higher

Description



The ADS-OCX properties page is opened under Delphi as follows:

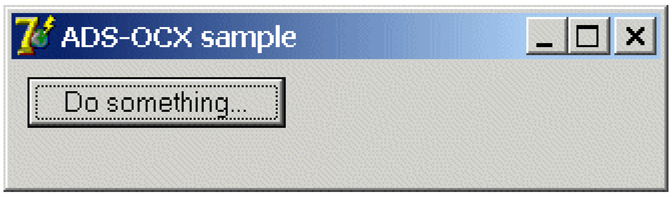
```
procedure TForm1.btnShowPropertyPageClick(Sender: TObject);
begin
    AdsOcx1.BrowseProperties();
end;
```

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466734987/.exe
Delphi 7 or higher (classic)	

5.2.6 Working with handles of PLC variables

System requirements:

- Delphi 7.0 or higher;
- TwinCAT v2.9 or higher



All the required handles can be fetched once at the start of the application, and released again when the application is closed. Continuously requesting and releasing handles places unnecessary loading on the system.

Handles that have already been requested become invalid when TwinCAT is restarted, and must be requested again. The same applies after 'Rebuild All' in the PLC. 'Rebuild All' causes a complete new program to be loaded into the runtime system, so that any handles that have already been requested are invalid, and are automatically released by TwinCAT. The handles that are no longer required must always be released. This can, however, only be done if the TwinCAT system is still running. If the TwinCAT system has already stopped, then all the handles are automatically released.

Connect with the first runtime system on the local PC and fetch the handle of the PLC variables:

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

Release the handle when the application is closed:

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

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466736395/.exe
Delphi 7 or higher (classic)	

5.2.7 Write string to PLC or read array from PLC

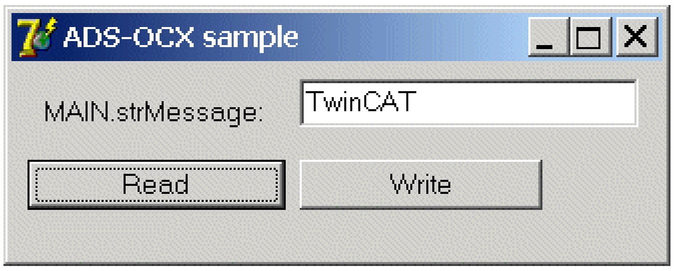
System requirements:

- Delphi 7.0 or higher;
- TwinCAT v2.11 Build 2034 or higher;

Task

A string is to be written to or read from the PLC.

Description



So that a string can be written to or read from the PLC, you need the length of the PLC string. The actual length of a PLC string can be determined using the PLC operator SIZEOF. In the PLC, the strings are terminated with a null and the actual string length is calculated from the defined length plus 1. If no length was specified during the string definition, then the string has an actual length of 81 characters including the terminating null.

PLC program

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

strColor has a length of 11 characters;

strState has a length of 21 characters;

strMessage has a length of 81 characters;

Delphi 7 program

```
unit Unit1;

interface

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

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

var
   Form1: TForm1;

implementation

{$R *.dfm}
```

Reading a string from the PLC

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

In the sample above, seven characters of a PLC string were read in Delphi. The dynamic string types have a length of zero immediately after initialization. The Delphi string variable must first be allocated the correct length if the ADS-OCX is to be able to copy the PLC string into the Delphi string variable. In Delphi, a WideString variable requires two bytes for each character. The Length function returns the localized number of characters in the string. However, the *Length* parameter in the method call requires the byte length, so the length determined with Length function is doubled.

Writing a string into the PLC

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

Establish connection to PLC, fetch variable handle

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

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

Release resources (variable handle)

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

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

end.
```

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466737803/.exe
Delphi 7 or higher (classic)	

Documents about this

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

5.2.8 Read multiple boolean variables into an array with one access

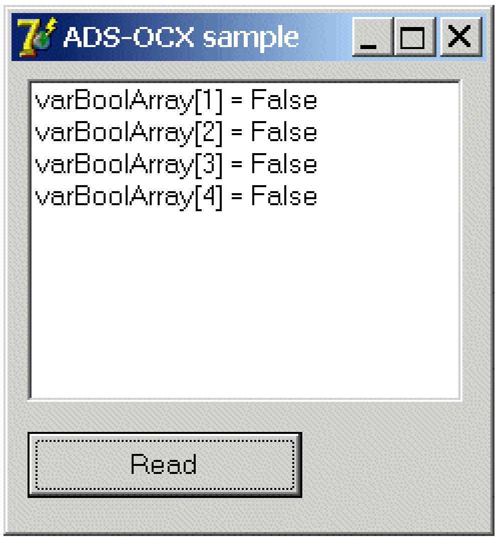
System requirements:

- Delphi 7.0 or higher;
- TwinCAT v2.11 Build 2034 or higher;

Task

Multiple boolean PLC variables can be read into Delphi applications with one access, provided the variables are stored at addresses that are sequentially located in the memory. It is, however, important that the first variable is located at a byte address.

Description



```
varBoolArray[1] = False
varBoolArray[2] = False
varBoolArray[3] = False
varBoolArray[4] = False
```

PLC program

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

Delphi 7 program

```
unit Unit1;
interface

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

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

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

implementation
{$R *.dfm}

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

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

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

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

end.
```

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466739211/.exe
Delphi 7 or higher (classic)	

5.2.9 Transmitting structures to/from the PLC

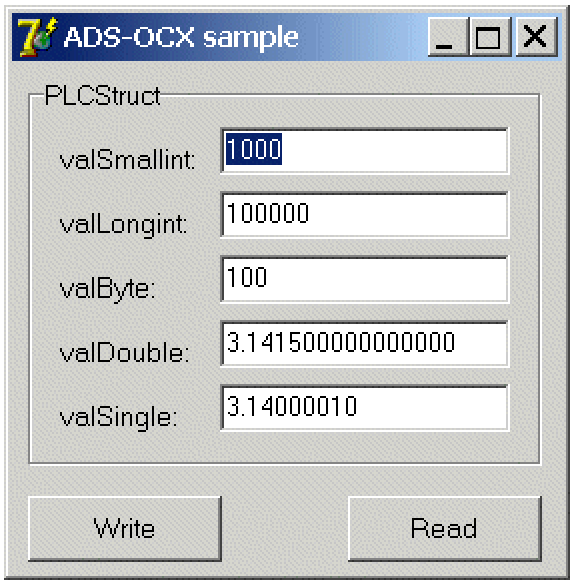
System requirements:

- Delphi 7.0 or higher;
- TwinCAT v2.11 Build 2034 or higher;

Task

A structure is to be written to or read from the PLC by the Delphi application. The elements in the structure have different data types.

Description



valSmallint: 1000
valLongint: 100000
valByte: 100
valDouble: 3.1415000000000000
valSingle: 3.14000010

Structure declaration in the PLC

```
TYPE PLCStruct
STRUCT
    valSmallint     : INT;
    valLongint      : DINT;
    valByte         : BYTE;
    valDouble       : LREAL;
    valSingle       : REAL;
END_STRUCT
END_TYPE
```

PLC program

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

```
;
```

Structure declaration in Delphi

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

Delphi 7 program

```
unit Unit1;

interface

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

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

var
  Form1: TForm1;
  hVar : Integer;

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

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

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

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

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

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

end.
```

Language / IDE	Unpack sample program
Delphi XE2	https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/12466740619/.exe
Delphi 7 or higher (classic)	

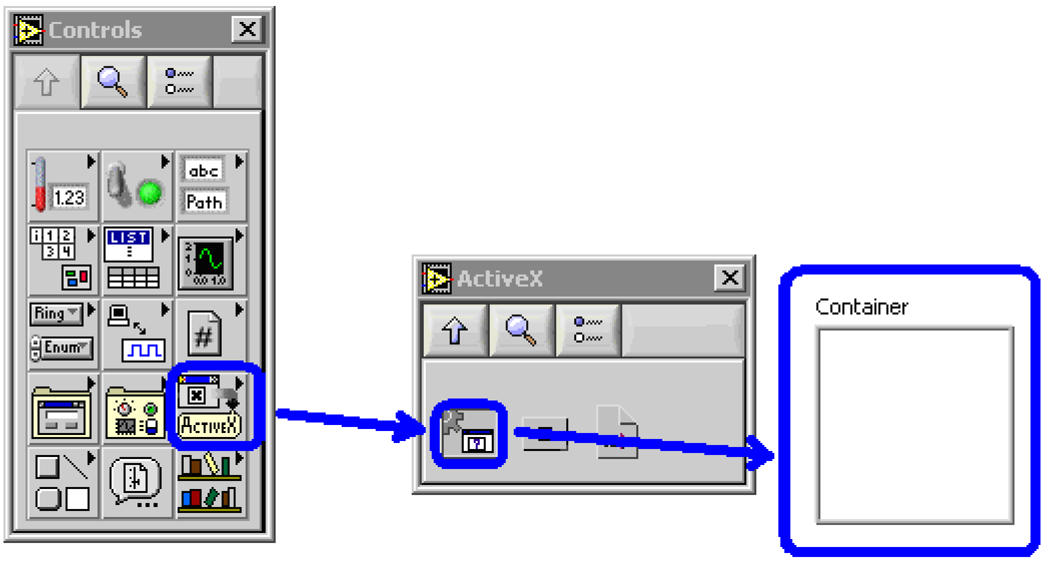
5.3 TwinCAT ADS OCX

5.3.1 Integration in LabVIEW™

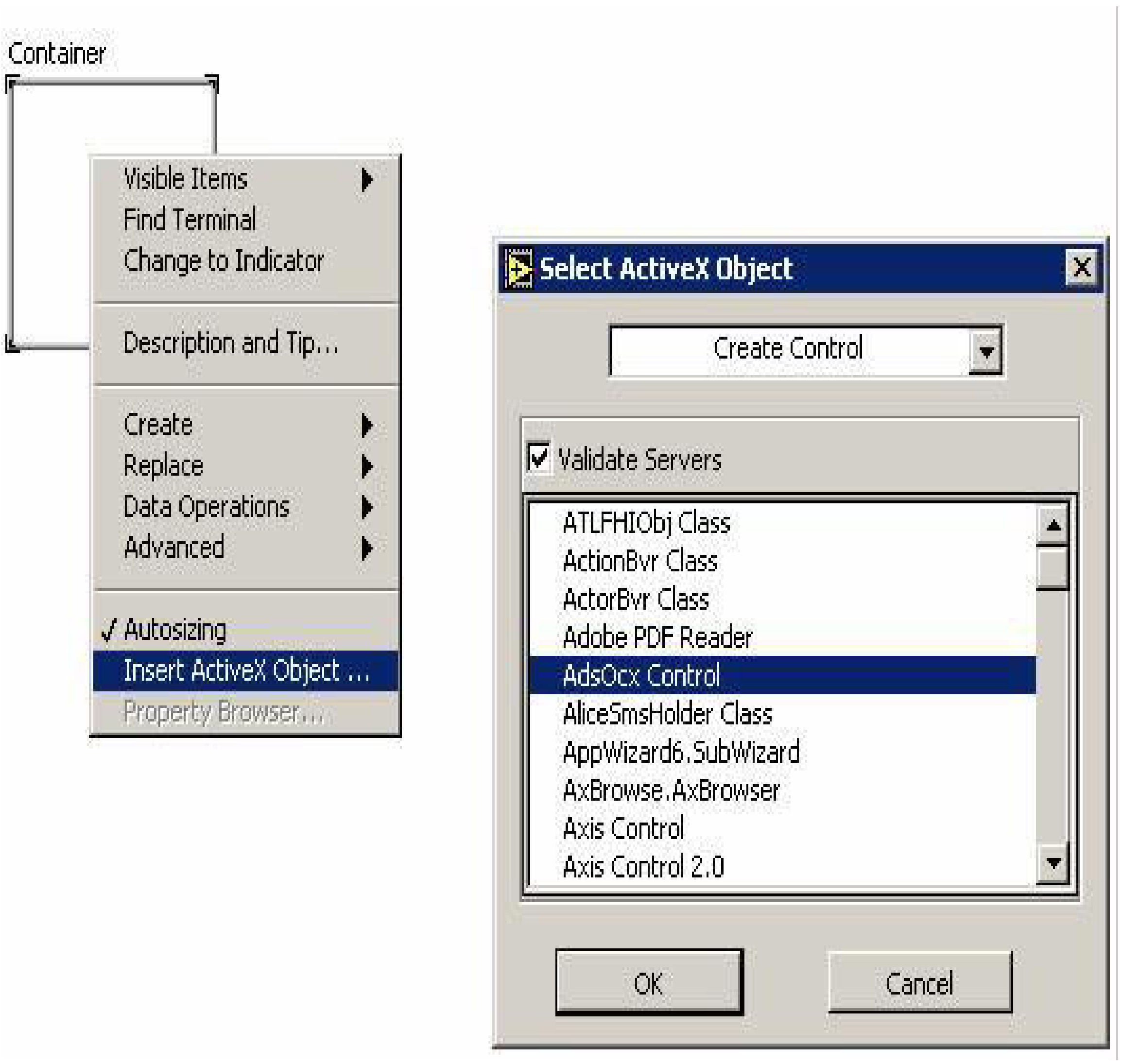
Use the TwinCAT 3 Interface for LabVIEW™

If you want to establish an ADS communication between LabVIEW™ and the TwinCAT 3 runtime, use in any case the extensively supported and documented product TwinCAT 3 Interface for LabVIEW™, see TF3710. The manual integration of free ADS components presented in the following are only application examples. These are not subject to Beckhoff support.

1. Create ActiveX Container



2. Insert ActiveX Object



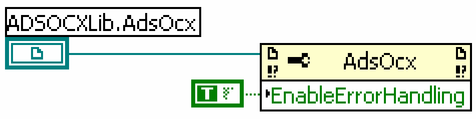
3. AdsOcx element in LabVIEW™

   Panel:

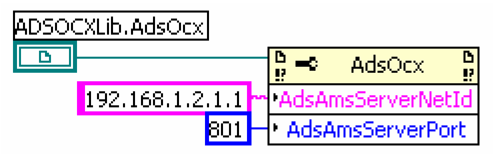
Diagram:

5.3.2 Samples using AdsOcx properties

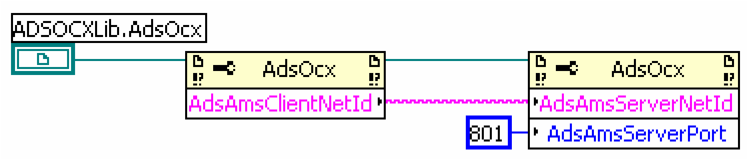
1. Set EnableErrorHandling true.



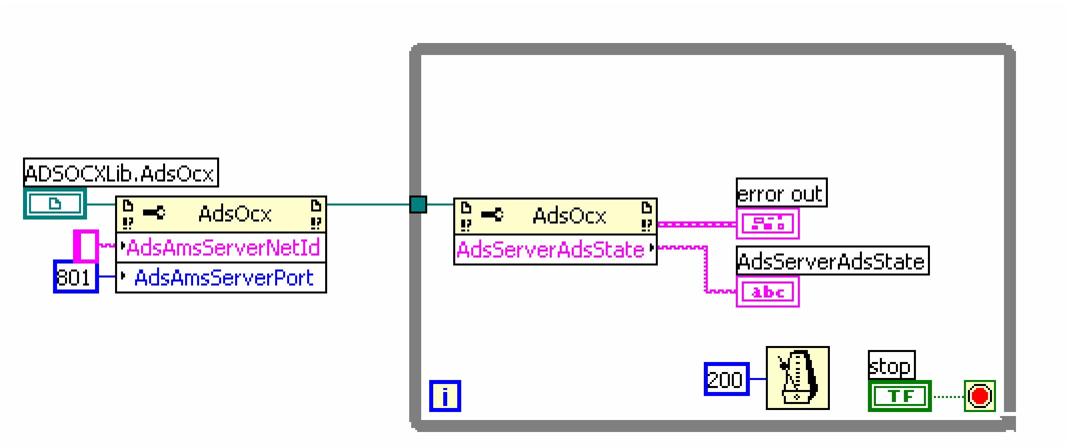
2. Set AdsAmsServerNetId and AdsAmsServerPort to fix values.



3. Access to local PLC by reading and setting the AmsNetId



4. Monitoring the status of the ADS device (https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967690891/.zip)



5.3.3 synchron methods: Read via address

AdsSyncReadBoolReq,

AdsSyncReadIntegerReq,

AdsSyncReadLongReq,

AdsSyncReadSingleReq,

AdsSyncReadDoubleReq,

AdsSyncReadStringReq

Sample: **AdsSyncReadBoolReq**

PLC declaration:

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

LabVIEW™ (see https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967692299/.zip):

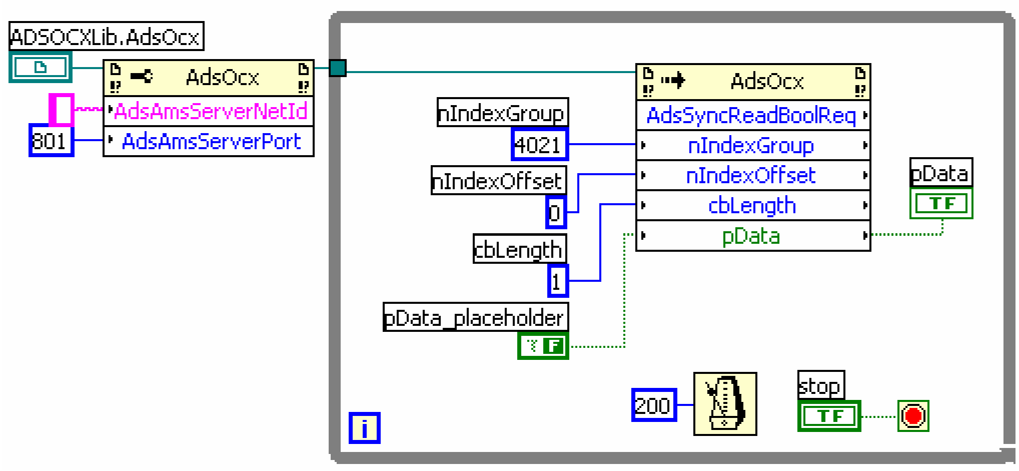


Fig. 1: TcAdsO35

5.3.4 synchron methods: Read via name

AdsSyncReadBoolVarReq

AdsSyncReadIntegerVarReq

AdsSyncReadLongVarReq

AdsSyncReadSingleVarReq

AdsSyncReadDoubleVarReq

AdsSyncReadStringVarReq

Sample: **AdsSyncReadBoolVarReq**

PLC declaration:

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

LabVIEW™: (see https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967693707/.zip)

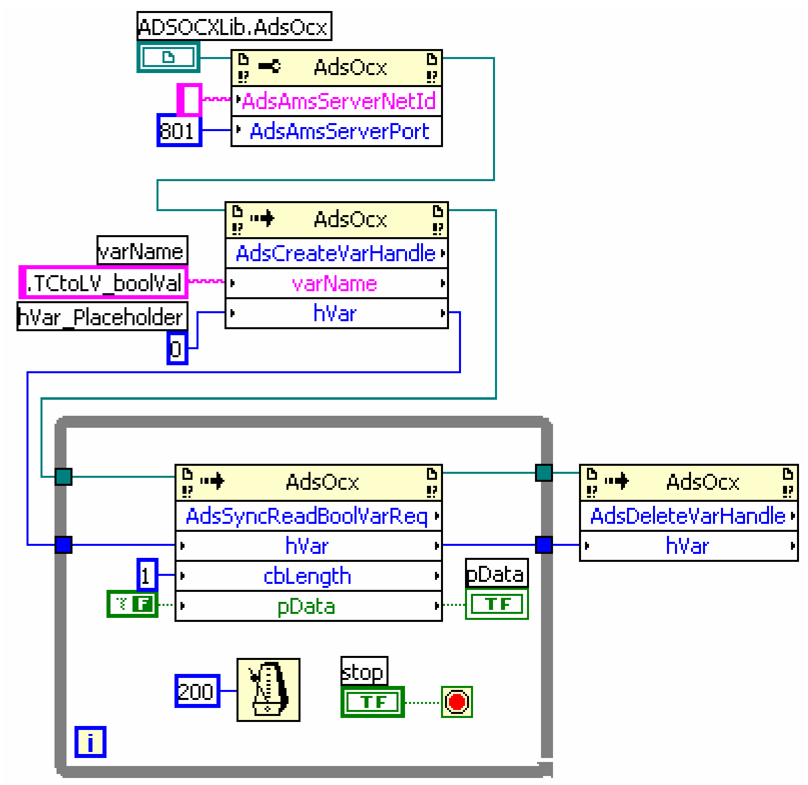


Fig. 2: TcAdsO36

5.3.5 synchron methods: Write via address

AdsSyncWriteBoolReq

AdsSyncWriteIntegerReq

AdsSyncWriteLongReq

AdsSyncWriteSingleReq

AdsSyncWriteDoubleReq

AdsSyncWriteStringReq

Sample: **AdsSyncWriteBoolReq**

PLC declaration:

```
    LVtoTC_boolVal AT%MX500.0: BOOL;
```

LabVIEW™: (see https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967695115/.zip)

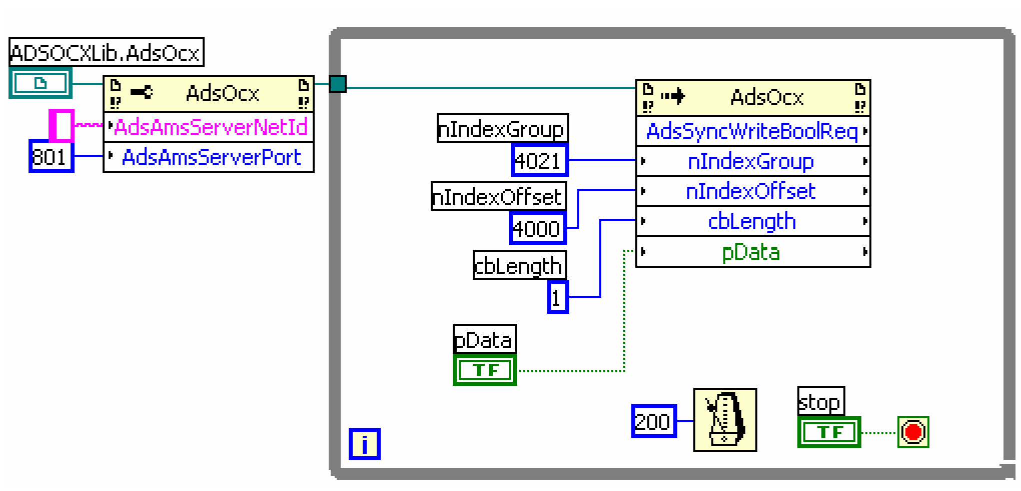


Fig. 3: TcAdsO37

5.3.6 synchron methods: Write via name

AdsSyncWriteBoolVarReq

AdsSyncWriteIntegerVarReq

AdsSyncWriteLongVarReq

AdsSyncWriteSingleVarReq

AdsSyncWriteDoubleVarReq

AdsSyncWriteStringVarReq

Sample: **AdsSyncWriteBoolVarReq**

PLC declaration:

```
    LVtoTC_boolVal AT%MX500.0: BOOL;
```

LabVIEW™: (see https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967696523/.zip)

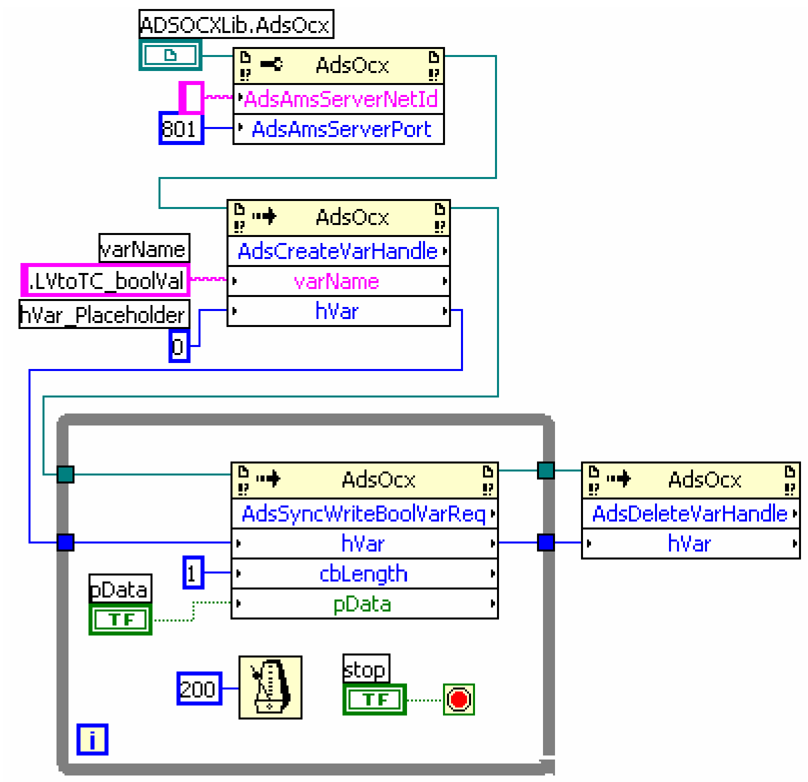


Fig. 4: TcAdsO38

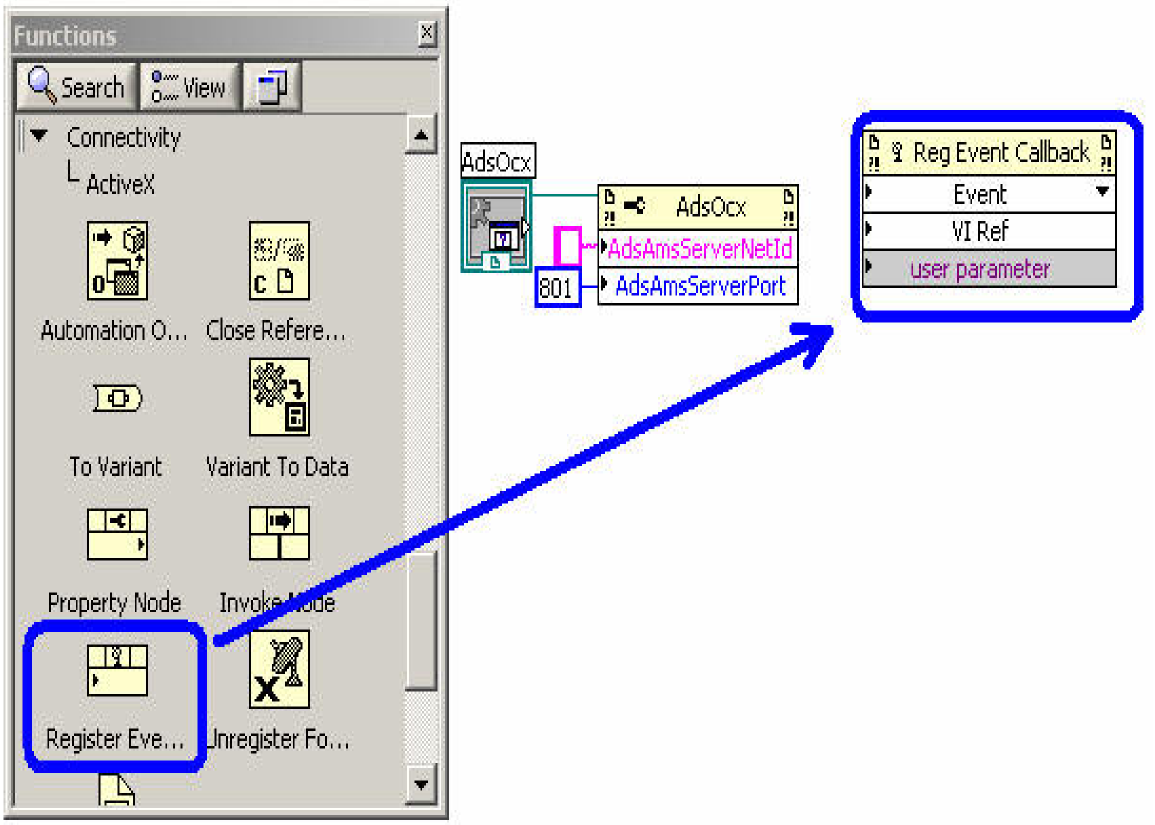
Documents about this

🗎 sample_dll_005_adsinforead.zip (Resources/zip/11967685259.zip)

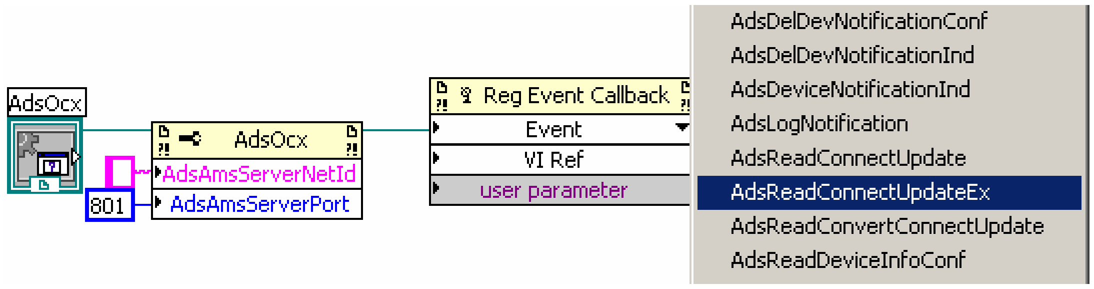
5.3.7 Event driven reading, registering Callback-vi

Sample files: https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967697931/.zip,

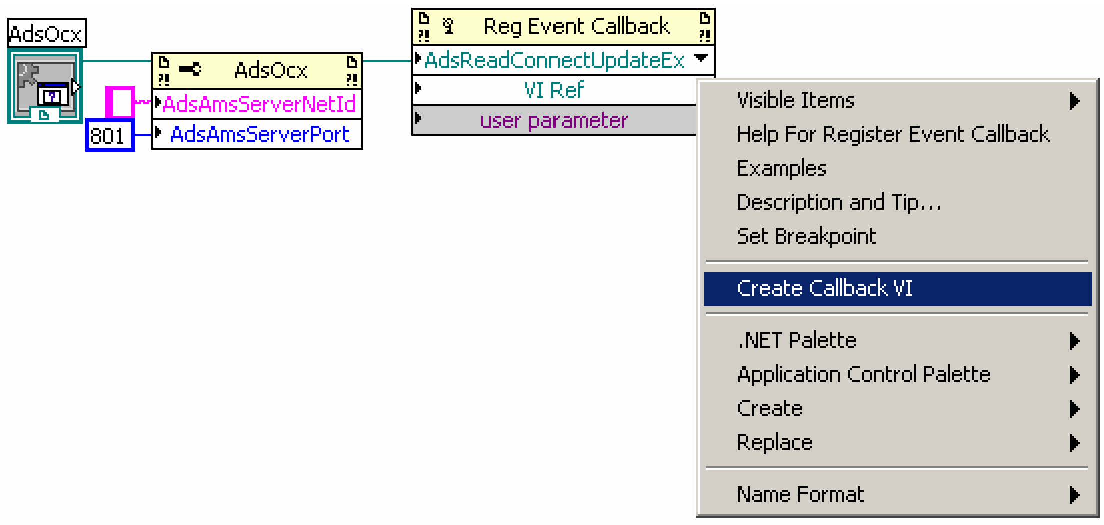
1. To use an asynchronous method, a callback VI is registered that is called by the AdsOcx.



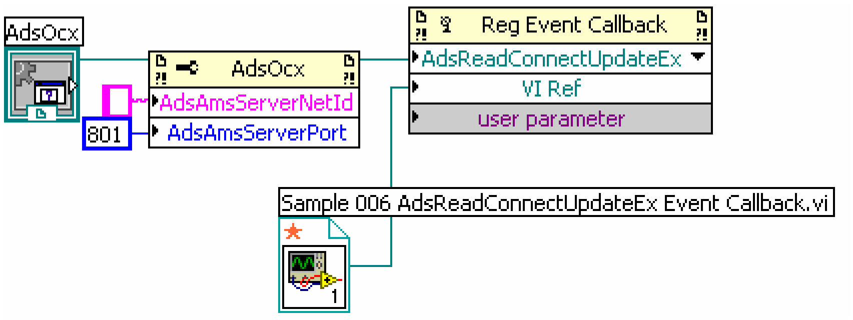
2. The event element is linked to the AdsOcx reference and the event to be called is selected.



3. The callback VI must have a very specific parameter structure. You can have LabVIEW™ create the callback VI.



4. The event callback VI should then be saved under a unique name.



5.3.8 Event driven reading, simple data types

Method: **AdsReadVarConnectEx**

Sample:

https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967699339/.zip

PLC declaration:

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

A callback VI is registered for the event **AdsReadConnectUpdateEx**.

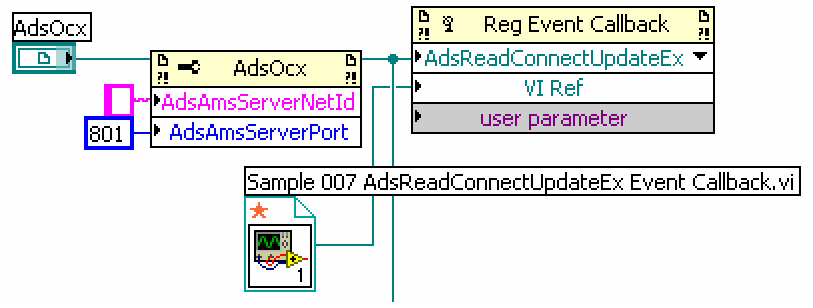


Fig. 5: TcAdsO44

The method **AdsReadVarConnectEx** establishes a fixed connection between LabVIEW™ and a PLC variable. The returned handle identifies the connection. When the connection is no longer needed, it is disconnected using **AdsDisconnectEx**.

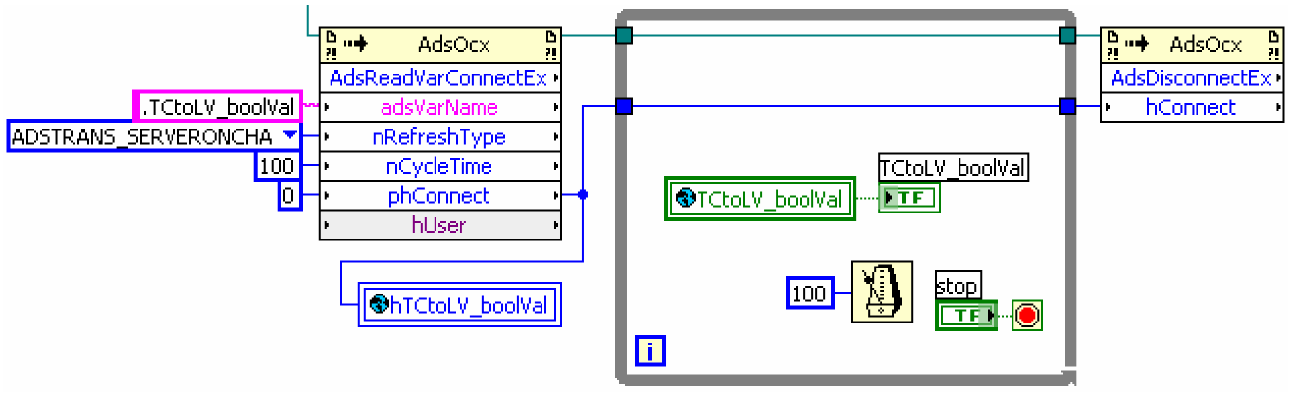


Fig. 6: TcAdsO45

The data is transferred as a variant when the callback VI is called. Using the handle, the variables can be converted to the correct type and assigned to the correct LabVIEW™ global variable.

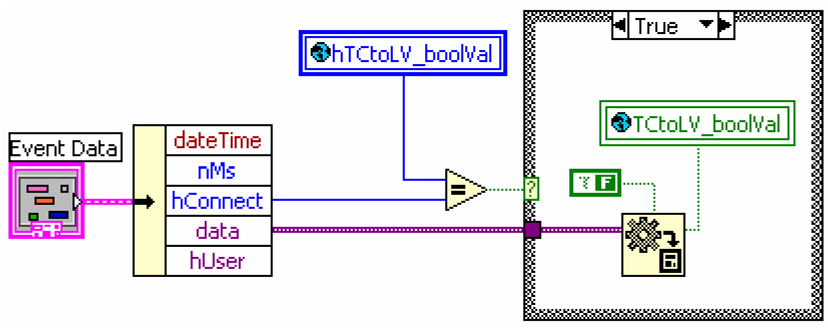


Fig. 7: TcAdsO46

5.3.9 Event driven reading, structure variables

Method: **AdsReadVarConvertConnect**

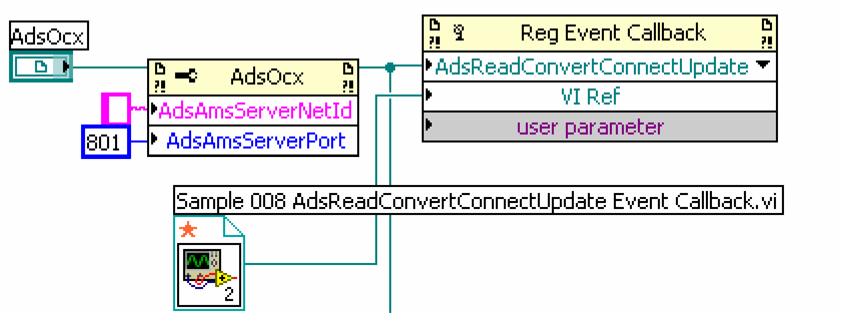
Sample : https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967700747/.zip

TwinCAT declaration:

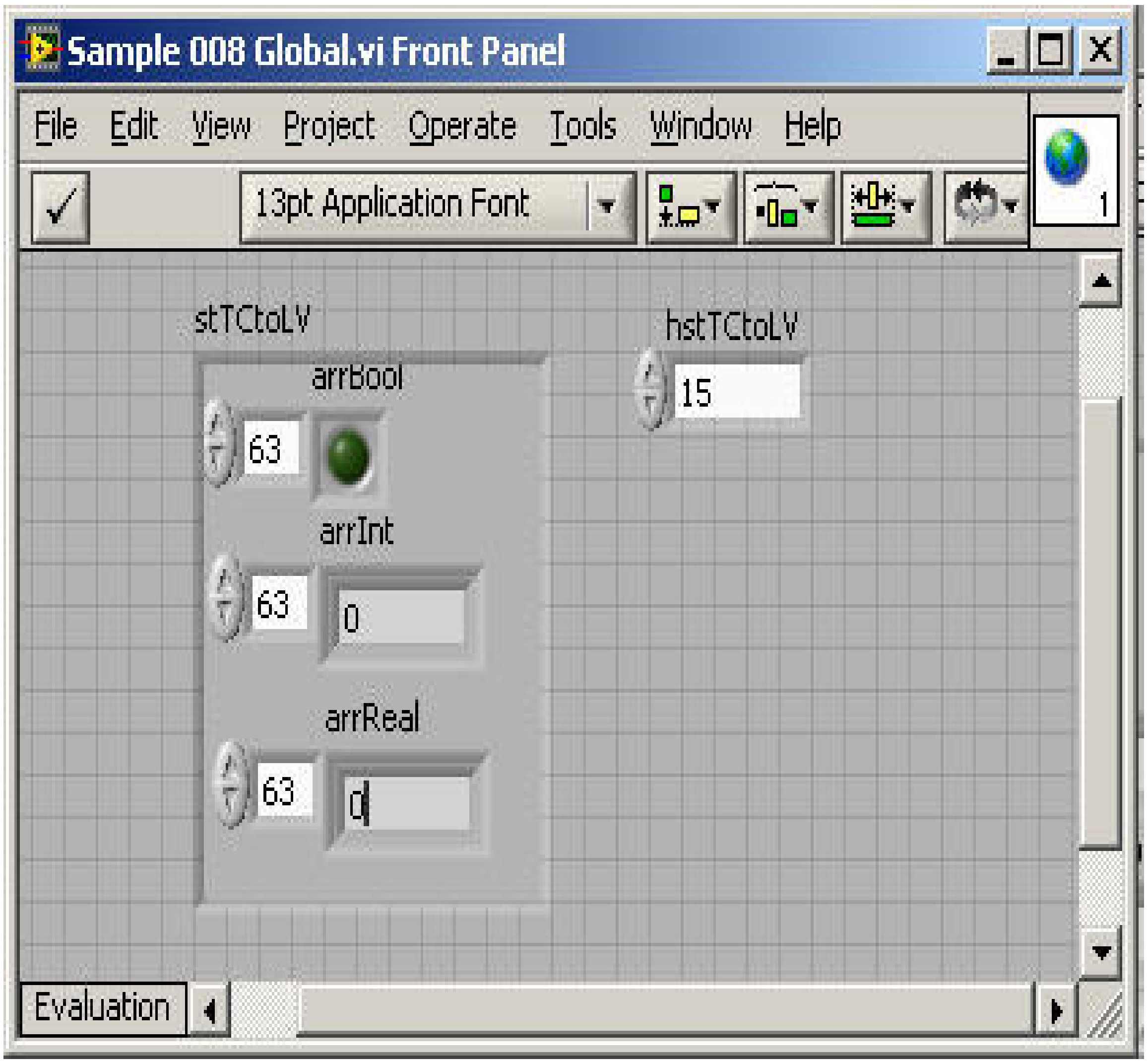
```
   TYPE ST_DataExchange :
     STRUCT arrBool:
       ARRAY[0..63] OF BOOL;
       arrInt  : ARRAY[0..63] OF INT;
       arrReal : ARRAY[0..63] OF REAL;
     END_STRUCT
END_TYPE
```

```
    stTCtoLV AT%MB1000: ST_DataExchange;
```

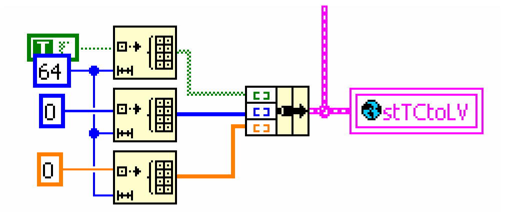
1. Registrate Callback-Vi fotr the event method **AdsReadConvertConnectUpdate**.



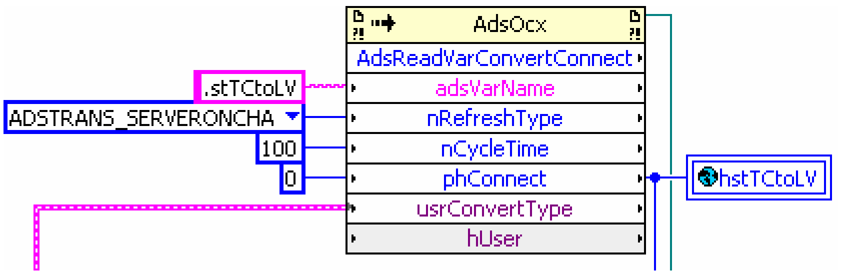
2. Global variables:
   - create cluster variable as illustration of the TwinCAT structure.
   - create global handle variable for the determination of the events.


Sample 008 Global.vi Front Panel
File Edit View Project Operate Tools Window Help
13pt Application Font
stTCtoLV
arrBool
63
arrInt
63
0
arrReal
63
0
hstTCtoLV
15
Evaluation

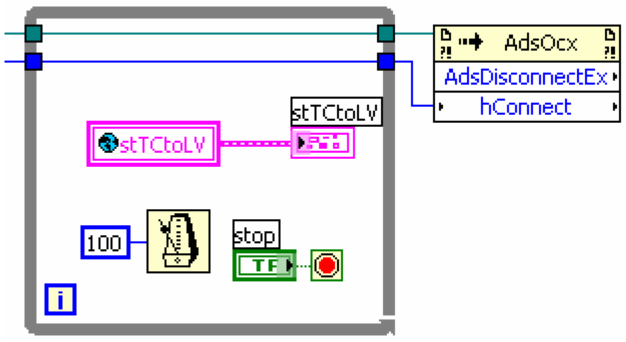
3. Initialise the data structure as illustration of the TwinCAT structure



4. Create data connection and store the connection handle

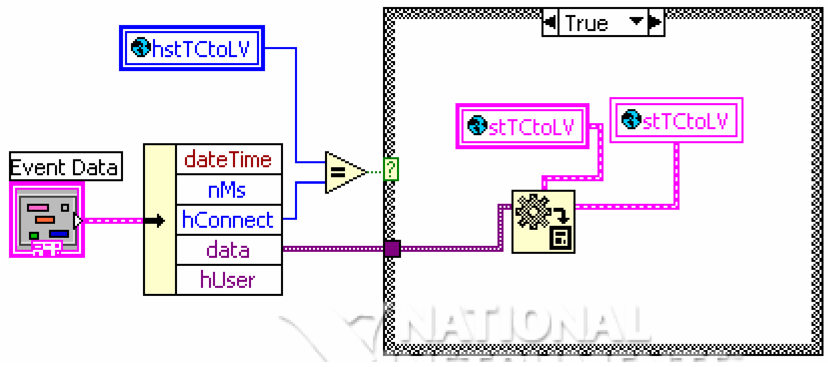


5. Cyclic access to global data and disconnect the connection



6. Event handling in the Callback-Vi

   With the passed handle hConnect the Callback-Vi decides for which variable the event has been called. It assigns the in data passed value to the right variable.

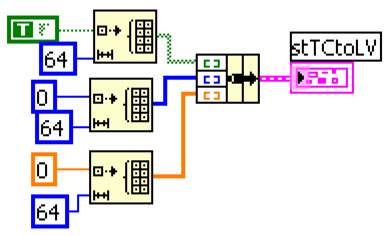


5.3.10 Event driven reading with data reference passing to Callback-vi

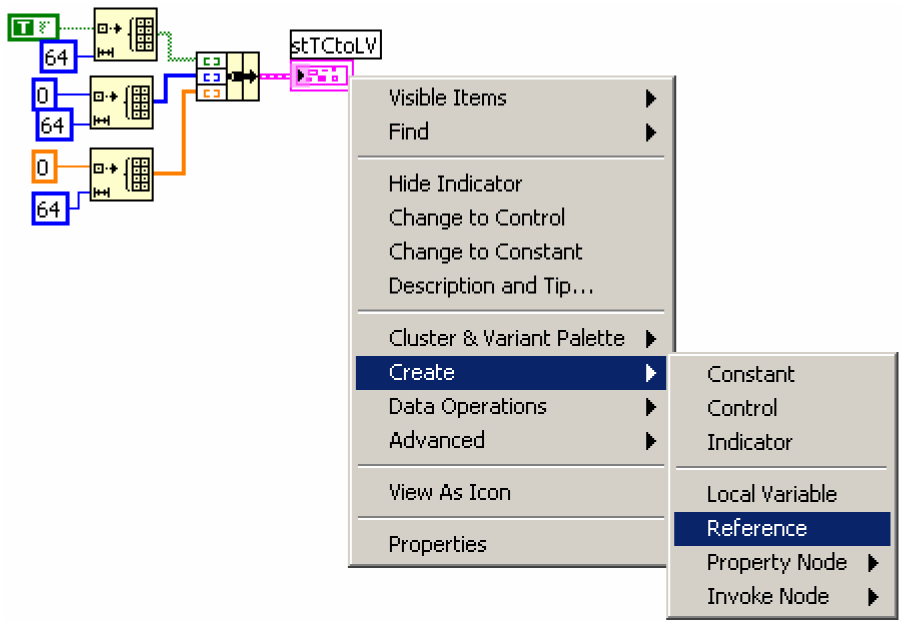
If only one variable is read via connect, the reference to the variable can be passed to the callback VI. This eliminates the need to use global variables. The callback VI writes directly to the variable of the calling VI by reference.

Sample files: https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967702155/.zip

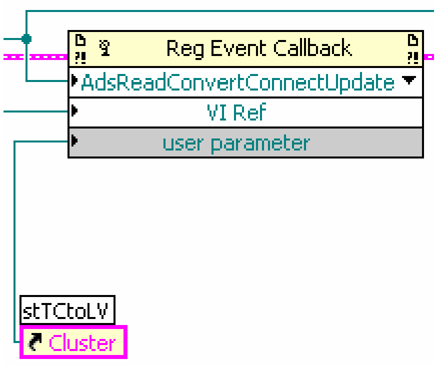
1. A LabVIEW™ display element of the correct type is created and initialized



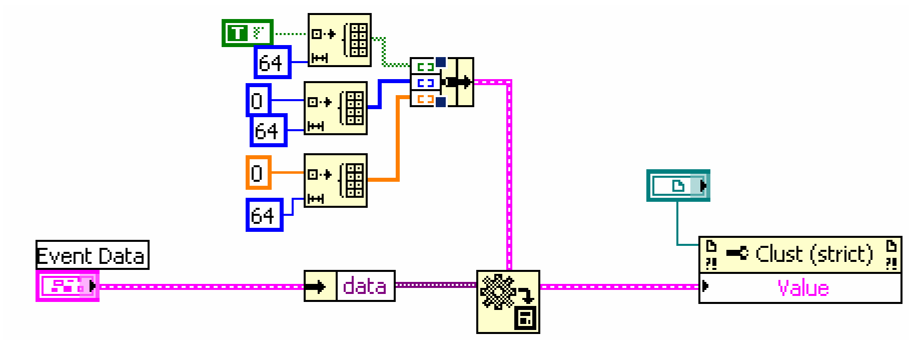
2. Creating the reference to the element



3. Passing the reference to the callback VI



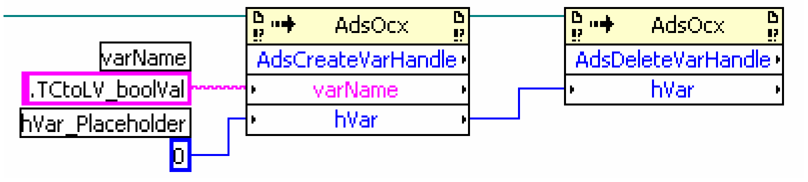
4. Accessing the reference variable in the callback VI
   The typeless variant variable must be converted to the correct data type, and then passed to the reference variable.



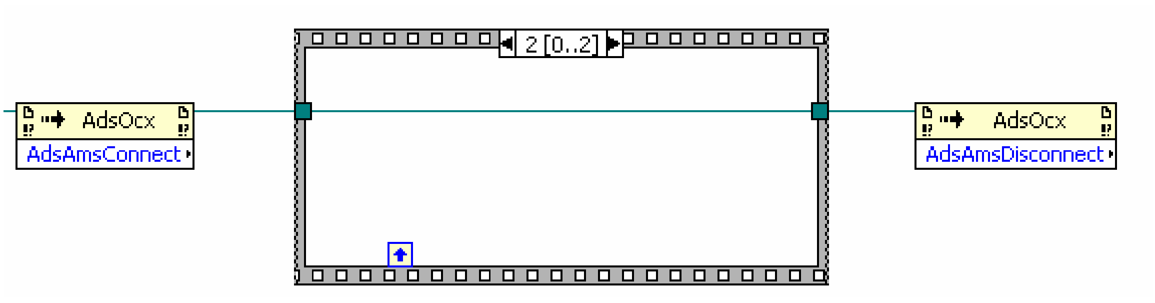
5.3.11 General Methods

1. Methods **AdsCreateVarHandle** and **AdsDeleteVarHandle** are used to access PLC variables by name
   PLC declaration:

TCtoLV_boolVal AT%MX0.0: BOOL;
LabVIEW™

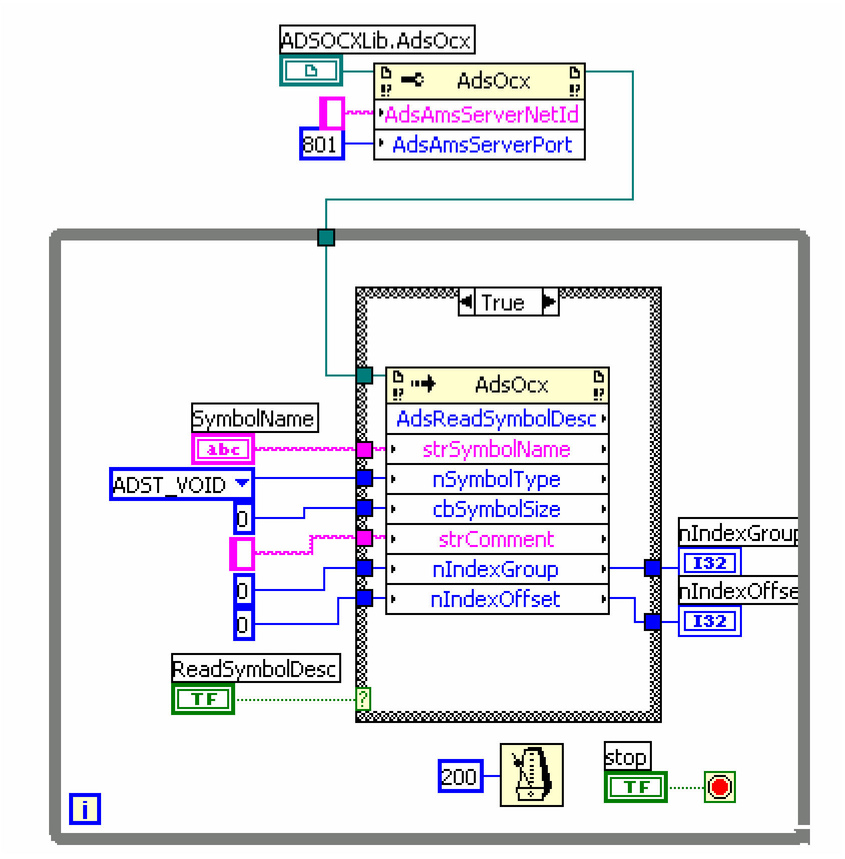


2. Methods **AdsAmsConnect** and **AdsAmsDisconnect**
   Are called in the start and end phase respectively to connect and disconnect the AdsOcx to/from the router.



   If the AdsOcx was disconnected from the router via AdsAmsDisconnect, AdsAmsConnect must be called or LabVIEW™ must be restarted before the next call of an AdsOcx method.

3. The method **AdsReadSymbolDesc**
   The method AdsReadSymbolDesc can be used to read information about a named PLC variable at runtime. For example, the address data nIndexGroup and nIndexOffset can be read to then access the variable by address (possibly also with the TcAdsDll). (see https://infosys.beckhoff.com/content/1033/tcadsocx/Resources/11967703563/.zip)


ADSOCXLib.AdsOcx
AdsOcx
AdsAmsServerNetId
801
AdsAmsServerPort
True
AdsOcx
AdsReadSymbolDesc
SymbolName
abc
strSymbolName
ADST_VOID
nSymbolType
0
cbSymbolSize
strComment
0
nIndexGroup
0
nIndexOffset
nIndexGroup
I32
nIndexOffse
I32
ReadSymbolDesc
TF
200
stop
TF
i

6 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▸ 141]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▸ 141]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▸ 142]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▸ 143]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.

Hex	Dec	HRESULT	Name	Description
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking syn-chronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.

Hex	Dec	HRESULT	Name	Description
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maxi-mum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

More Information:

www.beckhoff.com/automation

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