HAYES

CONTROL SYSTEMS

# Beckhoff TwinCAT

## Accessing TwinCAT Variables with VB6 and ActiveX

User Level: Beginner

Revision: 1.5
Updated: 13 April 2006

# Table Of Contents

| Version | Updated | Changes |
|---|---|---|
| V1.5 | 13th April 2006 | Info for “Connection Access” method using AdsReadVarConnectEx and AdsDisconnectEx commands and AdsReadConnectUpdateEx event added |
| V1.4 | 13th June 2005 | Info for accessing arrays and data structures from VB6 added |

# 1. Introduction

The Beckhoff ActiveX control (ADS-OCX) can be used to access variables from an ADS device within the TwinCAT system. This allows the user two possible approaches to controlling a machine or system:

1. Use TwinCAT PLC for real-time system and I/O control, with a Visual Basic 6 front-end program accessing PLC variables for data display, user control and data logging purposes.
2. Use TwinCAT I/O Server for I/O configuration and control only, and write the entire machine control system and data display functions in Visual Basic 6.

This application note will guide the user through the steps required to make a simple Visual Basic 6 program communicate with and control TwinCAT PLC variables, arrays, data structures and I/O terminals.

## 1.1. Data Access

There are 2 main methods of accessing TwinCAT variables using the ADS-OCX control:

| | | |
|---|---|---|
| 1. | Synchronous Access | driven by events in the Visual Basic program |
| 2. | Connection Access | driven by events in the ADS device |

To access data using Synchronous Access, the Visual Basic must issue a read or write command whenever new data is needed. This often required a TIMER event to ensure that data is monitored on a regular basis for changes.

Connection Access is driven by the Ads device itself. When data in the ADS device changes, it reports this to the ADS-OCX control generates an event in the Visual Basic program, which can take action as required.

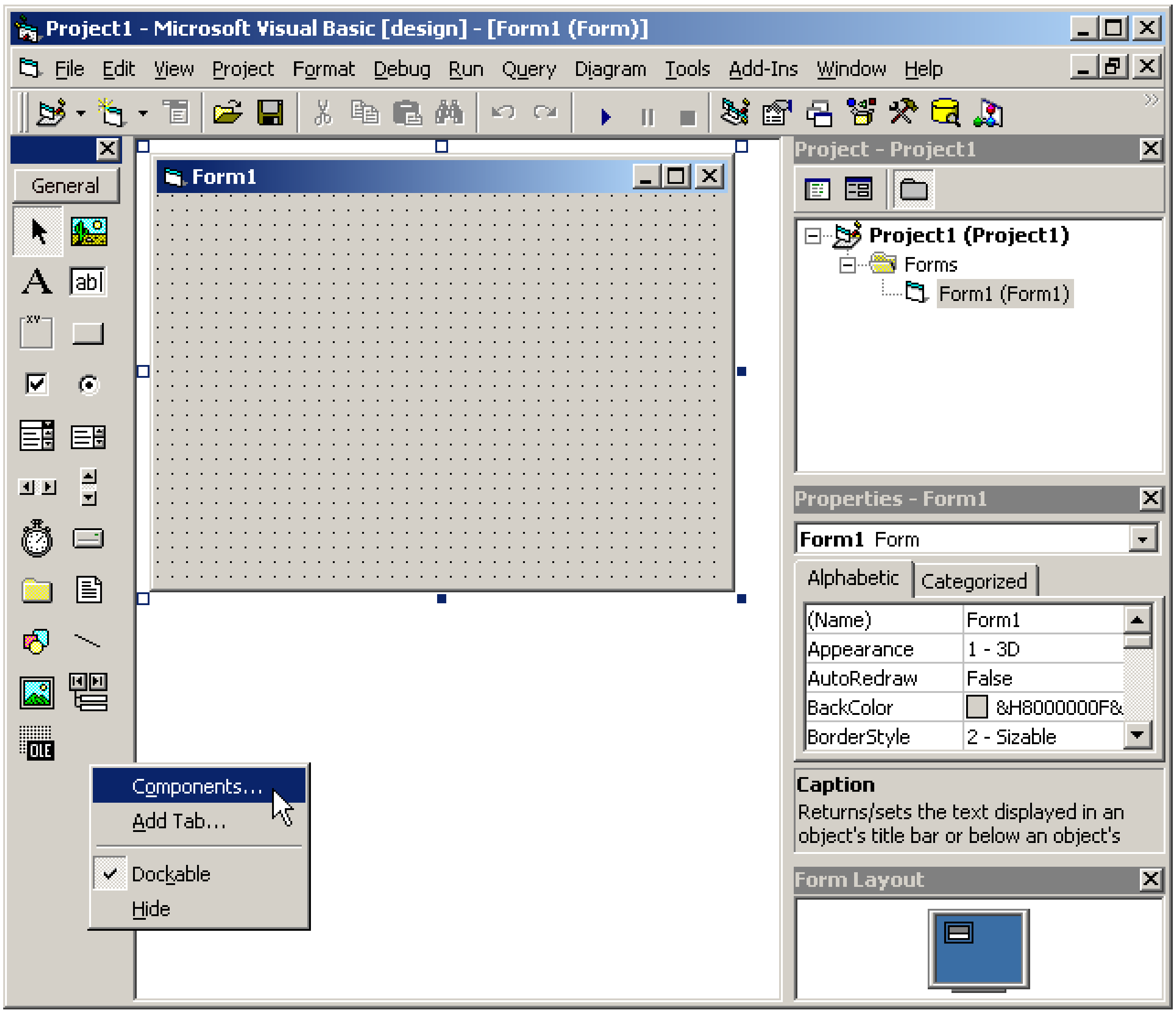
# 2. Beckhoff ADS-OCX Control with Visual Basic 6

## 2.1. Add Beckhoff ADS-OCX to Component Toolbar

The Beckhoff ActiveX control (called ADS-OCX) must be added to the list of available controls in the Visual Basic 6 form design screen before it can be inserted into a form.

In the VB project,

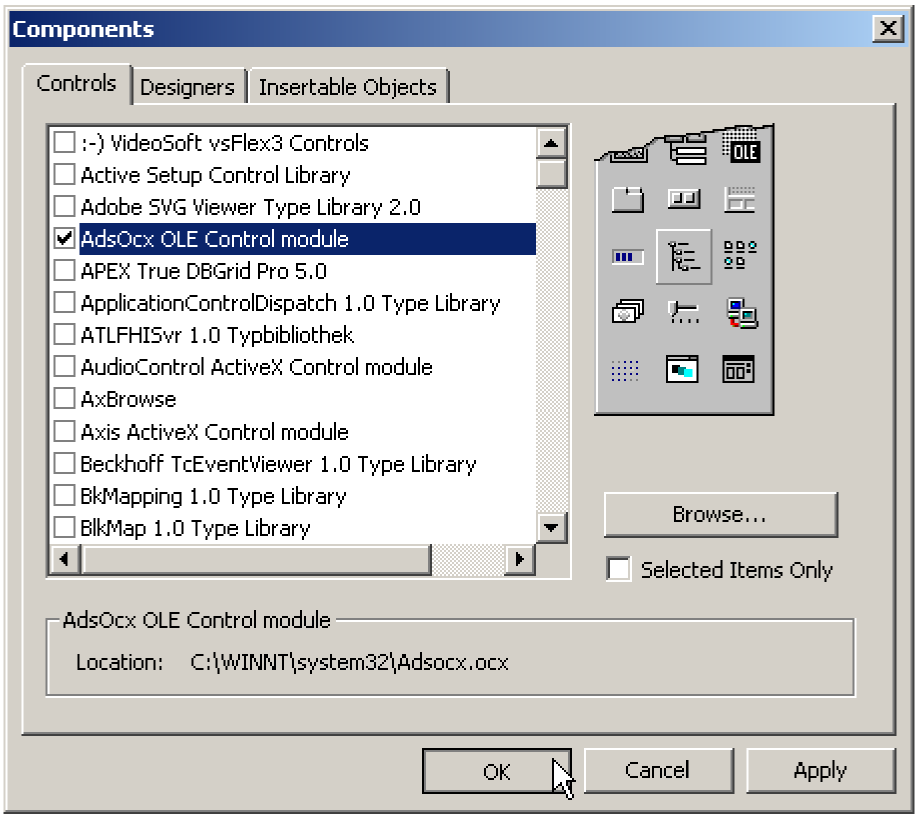
1. Right click on the controls toolbar (usually down the left hand side of the screen)
2. Select “Components…”. This will display a list of all installed ActiveX components that may be used in the current Visual Basic 6 project.



|  |  |
|---|---|
| (Name) | Form1 |
| Appearance | 1 - 3D |
| AutoRedraw | False |
| BackColor | &H8000000F& |
| BorderStyle | 2 - Sizable |

**Caption**
Returns/sets the text displayed in an object's title bar or below an object's

3. Find and select the control listed as "AdsOcx OLE Control module". Alternatively, it may appear as "C:\WINNT\system32\ADS-OCX.ocx" on some systems.
4. Click OK to add the ADS-OCX control to the controls toolbar.

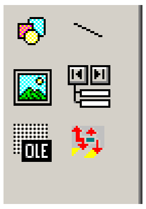


- [ ] :-) VideoSoft vsFlex3 Controls
- [ ] Active Setup Control Library
- [ ] Adobe SVG Viewer Type Library 2.0
- [x] AdsOcx OLE Control module
- [ ] APEX True DBGrid Pro 5.0
- [ ] ApplicationControlDispatch 1.0 Type Library
- [ ] ATLFHISvr 1.0 Typbibliothek
- [ ] AudioControl ActiveX Control module
- [ ] AxBrowse
- [ ] Axis ActiveX Control module
- [ ] Beckhoff TcEventViewer 1.0 Type Library
- [ ] BkMapping 1.0 Type Library
- [ ] BlkMap 1.0 Type Library

AdsOcx OLE Control module

Location: C:\WINNT\system32\Adsocx.ocx

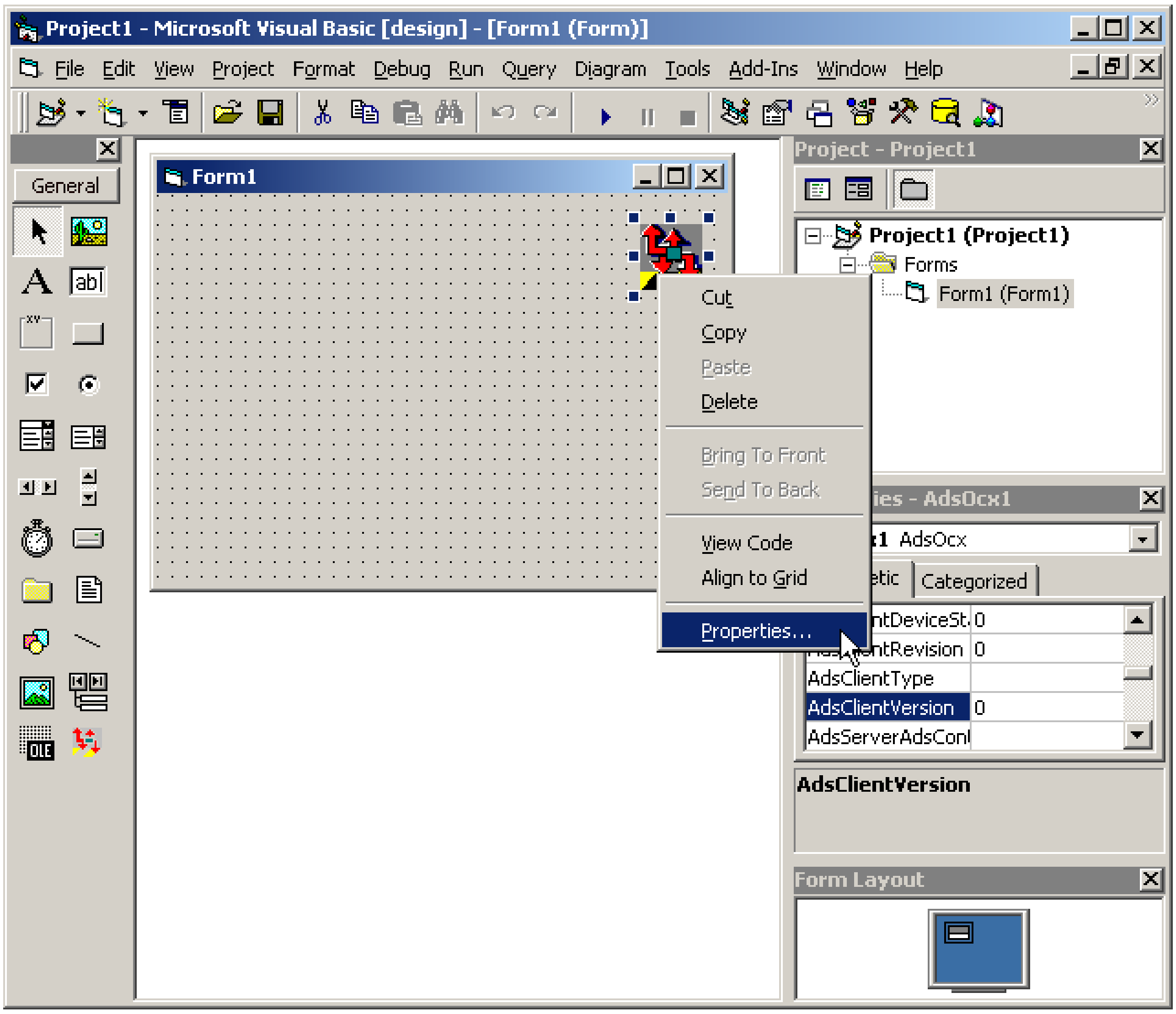
5. The ADS-OCX control will now appear somewhere on the Controls toolbar.



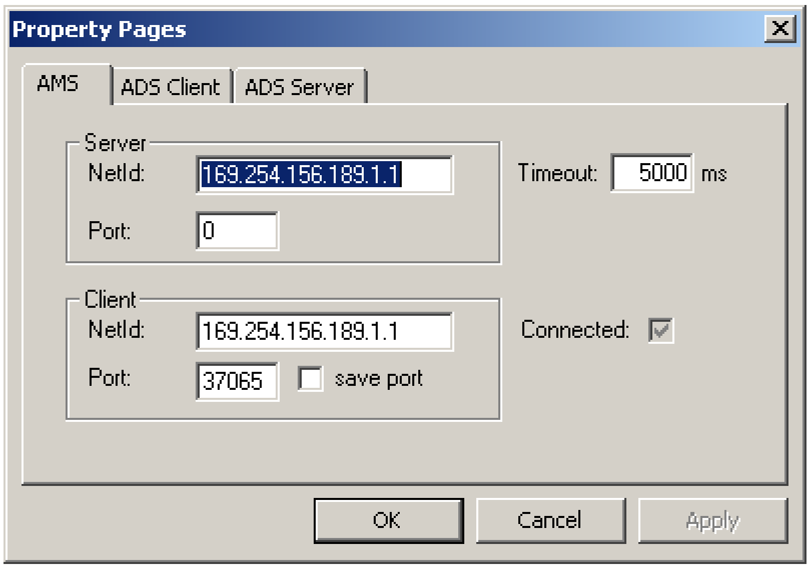
## 2.2. Add ADS-OCX to a Visual Basic 6 form

To create a link between the Visual Basic 6 program and TwinCAT, select the ADS-OCX control.

1. Select the ADSOCX control
2. Draw a box on the form design screen to place the control anywhere on the form. The ADS-OCX control will not be shown on the form at run-time, so it does not matter where it is placed
3. Right click on the new control and select "Properties". This will display the "Property Pages" for this instance of the ADS-OCX control.



4. If the target TwinCAT system is the local system, i.e. the Visual Basic 6 program will be running on the same PC as the target TwinCAT system, the NetID fields should be left blank.
5. The Client Port will be entered automatically when an ADS-OCX control is placed on the Visual Basic 6 form.
6. The Server Port number will depend on which ADS device within TwinCAT is to be accessed by this instance of ADS-OCX. TwinCAT PLC and TwinCAT I/O Server are 2 ADS devices that can be accessed by ADS-OCX.



Server
NetId: 169.254.156.189.1.1
Port: 0

Timeout: 5000 ms

Client
NetId: 169.254.156.189.1.1
Port: 37065 ☐ save port

Connected: ✓

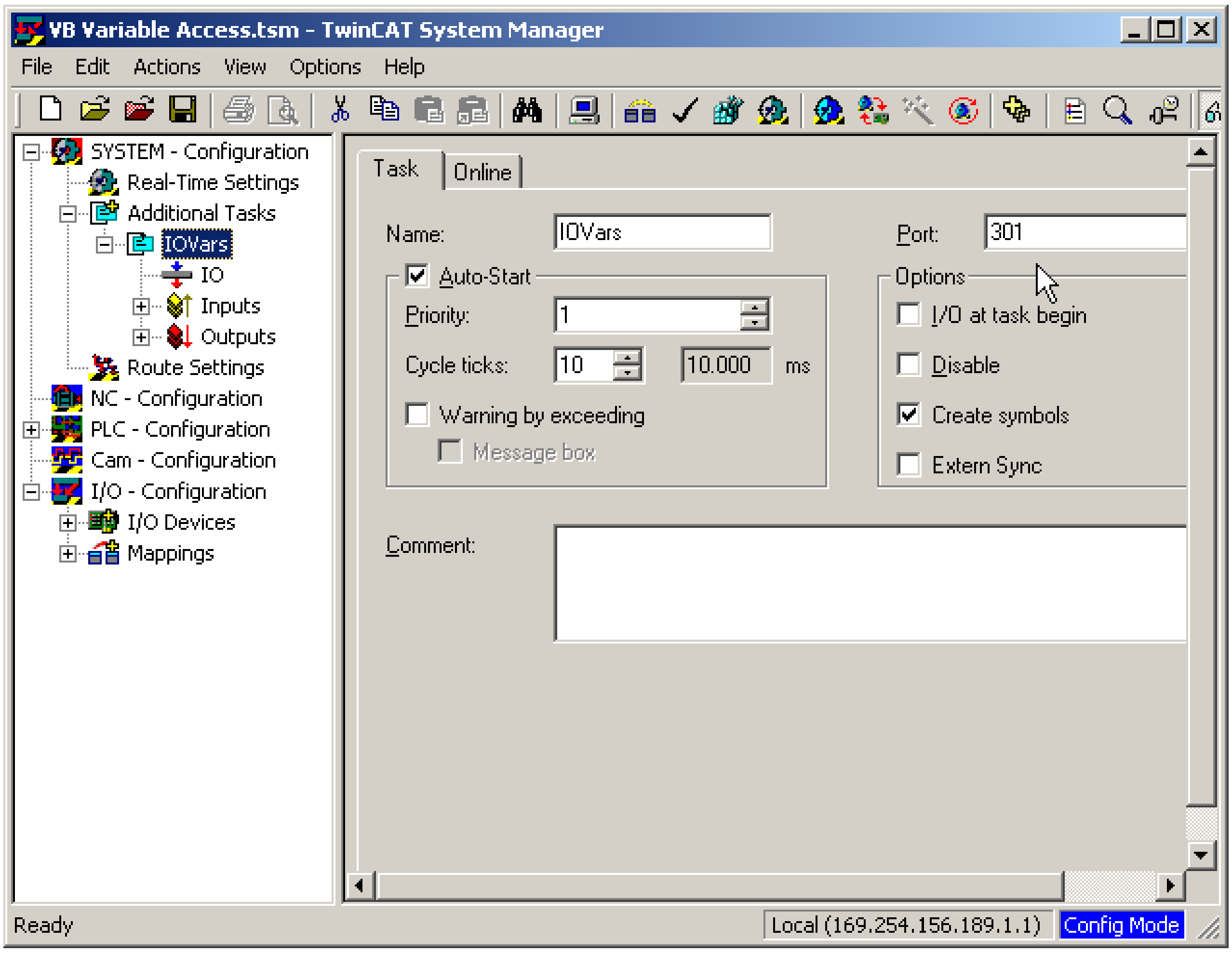
It is worth remembering that the CLIENT requests information from a SERVER, so the Visual Basic 6 program with the ADS-OCX control is the ADS client and TwinCAT PLC is the ADS server.

The ADS Server port number specifies which TwinCAT ADS device (e.g. TwinCAT I/O Server, TwinCAT PLC Run-time 1, TwinCAT NC, etc.) the data will be requested from.

## 2.3. TwinCAT I/O Server Port

The ADS-OCX Server Port property must be set to connect with the TwinCAT I/O Server port in. To find out the TwinCAT I/O Server Port number for the local system:

1. Go to TwinCAT System Manager
2. Double click on "SYSTEM – Configuration"
3. Double click on "Additional Tasks"
4. Select the task that has been created to provide system variables that can be linked to the I/O terminals. In the example below, the task is called "IOVars".
5. The Port number is shown on the Task tab, and is 301 in the example screen below. This value should be entered in the Server Port box on the ADS-OCX "Property Pages". (See section 2.2 on page 6)

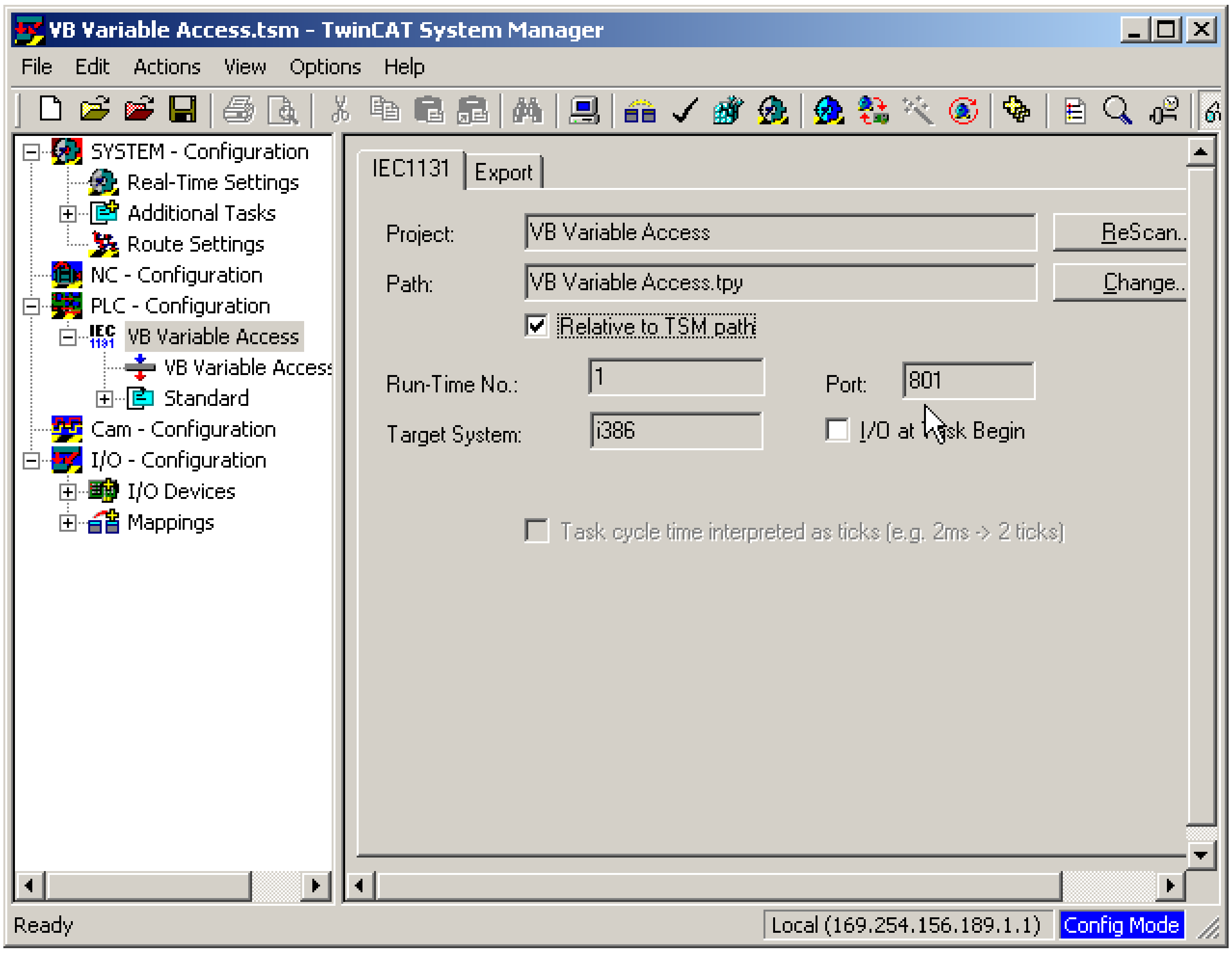


Name: IOVars
Port: 301
[✓] Auto-Start
Priority: 1
Cycle ticks: 10 10.000 ms
[ ] Warning by exceeding
[ ] Message box
Options
[ ] I/O at task begin
[ ] Disable
[✓] Create symbols
[ ] Extern Sync
Comment:

## 2.4. TwinCAT PLC Server Port

The Server Port must be set to the TwinCAT PLC port in the ADS-OCX properties. To check the Server Port number for the local system,

1. Go to TwinCAT System Manager
2. Double click on “PLC – Configuration”
3. Click on the attached PLC project to view the properties pages. (In the example below, the PLC project is called “**`VBExample`**”.)
4. The Port number is shown on the IEC1131 tab, and is 801 in the example screen below. This value should be entered in the Server Port box on the ADS-OCX “Property Pages”. (See section 2.2 on page 6)



Project: VB Variable Access
Path: VB Variable Access.tpy
[✓] Relative to TSM path
Run-Time No.: 1
Port: 801
Target System: i386
[ ] I/O at Task Begin
[ ] Task cycle time interpreted as ticks (e.g. 2ms -> 2 ticks)

## 2.5. Accessing Multiple TwinCAT ADS Devices

Access to 2 or more TwinCAT ADS devices can be achieved by using multiple instances of ADS-OCX in a single Visual Basic 6 application. Each instance of ADS-OCX is assigned to a single ADS device, and the appropriate instance of ADS-OCX is called to access the required TwinCAT ADS device.

For example, 2 instances of ADS-OCX can be added and given the names “**`IO`**” and “**`PLC`**”. By setting the Server Port properties to 301 and 801 respectively, they will provide access to the I/O Server and PLC respectively.

*NOTE: the Server NetID and Port properties should NOT be modified while the system is running. Always use multiple instances of ADS-OCX to access multiple TwinCAT ADS devices.*

# 3. TwinCAT Symbols

## 3.1. TwinCAT PLC Symbols

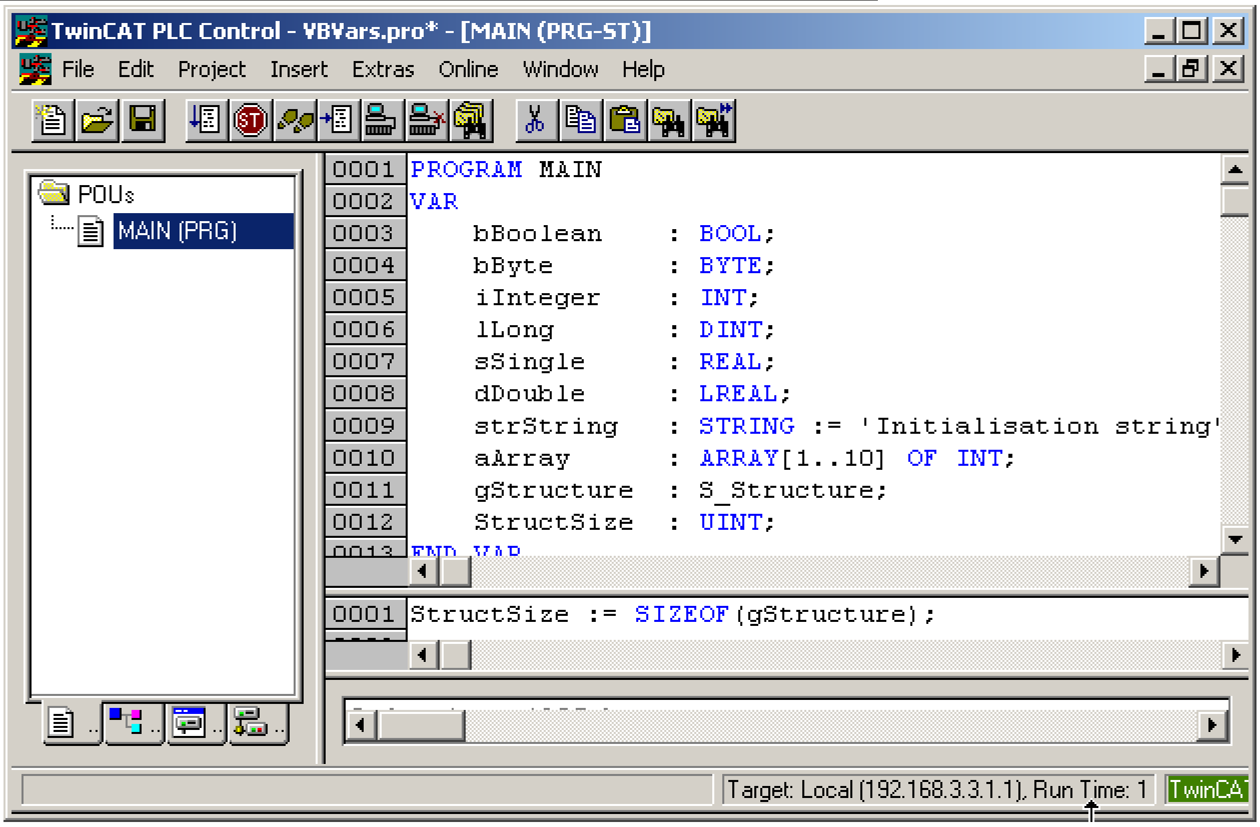
TwinCAT PLC variables are declared as local variables in a POU, or as global variables. In the example below, a PLC program task called "**MAIN**" has been created with several variables declared. Symbols are created when the TwinCAT PLC program is compiled, and are formatted as:

***<POU>.<VarName>***

where:

| | |
|---|---|
| ***<POU>*** | the PLC program task name (POU) where the variable has been declared. For global variables, < ***POU*** > will be left blank |
| ***<VarName>*** | the variable name declared in the ***<POU>*** |

| Variable | Symbol |
|---|---|
| **bBoolean** | **MAIN.bBoolean** |
| **bBByte** | **MAIN.bByte** |
| **iInteger** | **MAIN.iInteger** |
| **lLong** | **MAIN.lLong** |
| **sSingle** | **MAIN.sSingle** |
| **dDouble** | **MAIN.dDouble** |
| **sString** | **MAIN.sString** |
| **aArray(n)** | **MAIN.aArray(n)** |
| **gStructure** | **MAIN.gStructure** |


TwinCAT PLC Control - VBVars.pro* - [MAIN (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
MAIN (PRG)
Target: Local (192.168.3.3.1.1), Run Time: 1
TwinCA

```
PROGRAM MAIN
VAR
    bBoolean     : BOOL;
    bByte        : BYTE;
    iInteger     : INT;
    lLong        : DINT;
    sSingle      : REAL;
    dDouble      : LREAL;
    strString    : STRING := 'Initialisation string'
    aArray       : ARRAY[1..10] OF INT;
    gStructure   : S_Structure;
    StructSize   : UINT;
END_VAR
```

```
StructSize := SIZEOF(gStructure);
```

## 3.2. TwinCAT I/O Server Symbol Names

Symbols for TwinCAT I/O Server variables are formatted as:

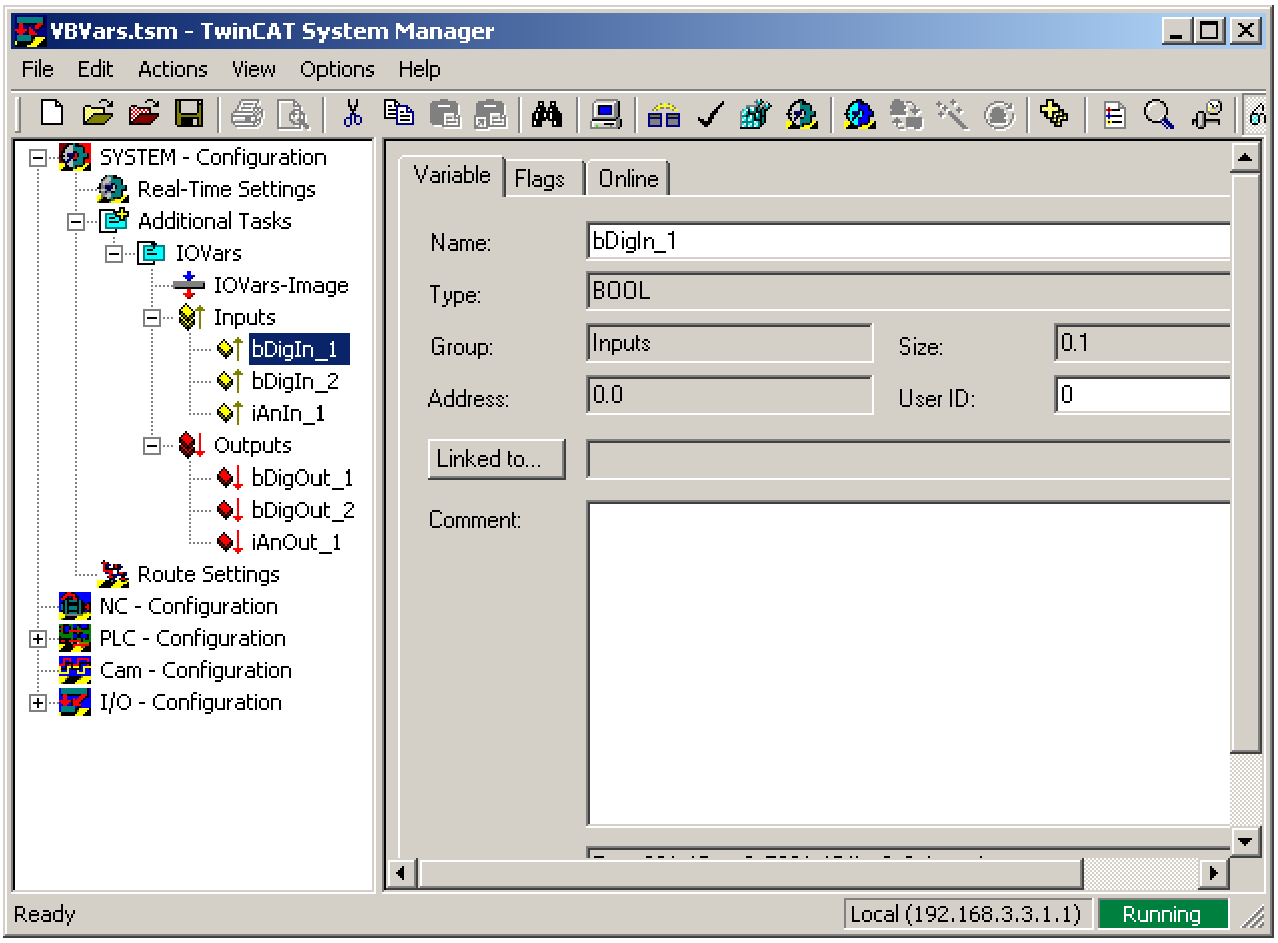
**`<Task>.Inputs.<VarName>`**

**`<Task>.Outputs.<VarName>`**

where:

| | |
|---|---|
| ***<Task>*** | I/O Server task name. In the screen shot below, ***<Task>*** is "IOVars" |
| ***<VarName>*** | the variable name declared under "Inputs" or "Output" for ***<Task>*** |

| Variable | Symbol |
|---|---|
| **bDigIn_1** | **IOVars.Inputs.bDigIn_1** |
| **bDigIn_2** | **IOVars.Inputs.bDigIn_2** |
| **bDigOut_1** | **IOVars.Outputs.bDigOut_1** |
| **bDigOut_2** | **IOVars.Outputs.bDigOut_2** |
| **iAnIn_1** | **IOVars.Inputs.iAnIn_1** |
| **iAnOut_1** | **IOVars.Outputs.iAnOut_1** |



Name: bDigIn_1
Type: BOOL
Group: Inputs
Size: 0.1
Address: 0.0
User ID: 0
Comment:

# 4. Connection Read Access to TwinCAT Variables

"Connection" read access to TwinCAT PLC variables is driven by the ADS device itself. A series of fixed connections are created between the ADS device and Visual Basic variables, and the ADS device updates the ADS-OCX control when a data value changes. The ADS-OCX control will generate an event in the Visual Basic program for each change of data value to allow the updated variable to be identified

## 4.1. Create Fixed Read Connection (.AdsReadVarConnectEx)

Fixed connections for reading TwinCAT variables must be created in the **`Form_Load()`** event. The **`.AdsReadVarConnectEx`** property will search the TwinCAT PLC port for the specified symbol name, and create a connection to the target variable. If the symbol name is not found, an error code is returned to indicate why the function failed.

```
    lErrCode = <ADSOCX>.AdsReadVarConnectEx("<TcSymbol>",
                ADSTRANS_SERVERONCHA, <CycleTime>, <hConnect>, <hUser>)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **`PLC`** |
| ***<TcSymbol>*** | TwinCAT symbol (see Chapter 3, "TwinCAT Symbols" on page 10) |
| ***<CycleTime>*** | update cycle time in ms, rounded up to the nearest multiple of 55ms |
| ***<hConnect>*** | unique handle for the connection (NOT the ADS handle!!) |
| ***<hUser>*** | user-defined value that is returned when the **`.AdsReadConnectUpdate`** event. This allows the Visual Basic program to identify which variable value has changed |

```
Dim hConnect(0 to 7) As Long

Private Sub Form_Load()

   ' create fixed connections to PLC variables
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.bBoolean",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(0), 0)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.bByte",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(1), 1)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.iInteger",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(2), 2)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.lLong",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(3), 3)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.sSingle",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(4), 4)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.dDouble",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(5), 5)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.strString",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(6), 6)
   lErrCode = PLC.AdsReadVarConnectEx ("MAIN.aArray",
                                 ADSTRANS_SERVERONCHA, 100, hConnect(7), 7)

End Sub
```

## 4.2. Update Read Fixed Connection (.AdsReadConnectUpdateEx)

The **.AdsReadConnectupdateEx** event is called when one of the value of one of the fixed-connection variables changes. Time stamp information is also returned to indicate when the change of value occurred, and the user-defined value is also returned to allow the updated variable to be identified.

```
Private Sub <ADSOCX>_AdsReadConnectUpdateEx(ByVal dateTime As Date,
            ByVal nMs As Long,          ByVal hConnect As Long,
            ByVal data As Variant,      Optional ByVal hUser As Variant)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **PLC** |
| **dateTime** | time stamp when new data was received |
| **nMs** | milliseconds for time stamp |
| **hConnect** | handle for the connection |
| **hUser** | user-defined value for the variable that generated the event |

```
Private Sub PLC_AdsReadConnectUpdateEx(ByVal dateTime As Date, _
                                       ByVal nMs As Long, _
                                       ByVal hConnect As Long, _
                                       ByVal data As Variant, _
                                       Optional ByVal hUser As Variant)

    ‘ use hUser to identify which variable that has changed
    Select Case hUser
        Case 0:
            bBoolean = data         ‘ bBoolean changed
        Case 1:
            bByte = data            ‘ bByte changed
        Case 2:
            iInteger = data         ‘ iInteger changed
        Case 3:
            lLong = data            ‘ lLong changed
        Case 4:
            sSingle = data          ‘ sSingle changed
        Case 5:
            dDouble = data          ‘ dDouble changed
        Case 6:
            strString = data        ‘ strString changed
        Case 7:                     ‘ 2 or more elements of aArray changed
            For iIndex = 0 To 4
                aArray(iIndex) = data(iIndex)
            Next
    End Select
End Sub
```

*NOTE: it is not possible to pass data structures using the Connection method.*

## 4.3. Close Read Fixed Connection (.AdsDisconnectEx)

The **.AdsDisconnectEx** event is called during the **Form_Unload()** event to close a fixed connection between the ADS-OCX and the ADS device,

```
    Call <ADSOCX>.AdsDisconnectEx(hConnect)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **PLC** |
| **hConnect** | handle for the connection |

```
Private Sub Form_Unload()

    ' delete fixed connections
    For iIndex = 0 To 7
        Call PLC.AdsDisconnectEx(hConnect(iIndex)
    Next
End Sub
```

# 5. Synchronous Access to TwinCAT Variables

"Synchronous" read/write access to TwinCAT PLC variables is driven by "events" in the Visual Basic program. When the Read or Write command is issued, the Visual Basic program will wait for a response from the function before continuing.

## 5.1. Create TwinCAT Variable Handle (.AdsCreateVarHandle)

"Handles" must be created to the required TwinCAT PLC variables in the **`Form_Load()`** event. The **`.AdsCreateVarHandle`** property will search the ADS port for the specified symbol name, and create a "handle" to allow fast direct access to the variable. If the symbol name is not found, an error code is returned to indicate why the function failed.

```
lErrCode = <ADSOCX>.AdsCreateVarHandle("<TcSymbol>",<Handle>)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **`PLC`** |
| ***<TcSymbol>*** | TwinCAT symbol (see Chapter 3, "TwinCAT Symbols" on page 10) |
| ***<Handle>*** | handle used for subsequent accesses to the variable |

```
Dim hbBoolean As Long
Dim hbByte As Long
Dim hiInteger As Long
Dim hlLong As Long
Dim hsSingle As Long
Dim hdDouble As Long
Dim hsString As Long
Dim haArray As Long
Dim hgStructure As Long

Private Sub Form_Load()

   ' create handles to PLC variables
   lErrCode = PLC.AdsCreateVarHandle("MAIN.bBoolean", hbBoolean)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.bByte", hbByte)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.iInteger", hiInteger)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.lLong", hlLong)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.sSingle", hsSingle)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.dDouble", hdDouble)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.hsString", hsString)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.haArray", haArray)
   lErrCode = PLC.AdsCreateVarHandle("MAIN.hgStructure", hgStructure)
End Sub
```

## 5.2. Reading a TwinCAT Variable (.AdsSyncReadxxxxVarReq)

Visual Basic 6 uses the **`.AdsSyncReadxxxxVarReq`** properties of ADS-OCX to read a variable from the TwinCAT PLC. The actual command used depends on the data type of the target variable.

| Function | TwinCAT data type | Length (bytes) |
|---|---|---|
| `.AdsSyncReadBoolVarReq` | BOOL | 1 |
| `.AdsSyncReadIntegerVarReq` | SINT, USINT, BYTE | 1 |
| `.AdsSyncReadIntegerVarReq` | INT, UINT, WORD | 2 |
| `.AdsSyncReadLongVarReq` | DINT, UDINT, DWORD | 4 |
| `.AdsSyncReadSingleVarReq` | REAL | 4 |
| `.AdsSyncReadDoubleVarReq` | LREAL | 8 |
| `.AdsSyncReadStringVarReq` | STRING | String Characters * 2 |

```
    lErrCode = <ADSOCX>.AdsSyncReadxxxxVarReq(<Handle>, <DLen>, <VBVar>)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **`PLC`** |
| ***<Handle>*** | handle for target variable |
| ***<DLen>*** | length of the target variable or array in bytes |
| ***<VBVar>*** | VB6 variable containing the data that will be written to TwinCAT. This variable must be the same length as the target TwinCAT variable |

```
Dim bBoolean As Boolean
Dim bByte As Byte
Dim iInteger As Integer
Dim lLong As Long
Dim sSingle As Single
Dim dDouble As Double
Dim sString As String

Private Sub cmdRead_Click()

    ' read variables from PLC
    Call PLC.AdsSyncReadBoolVarReq(hbBoolean, 1, bBoolean)
    Call PLC.AdsSyncReadIntegerVarReq(hbByte, 1, bByte)
    Call PLC.AdsSyncReadIntegerVarReq(hiInteger, 2, iInteger)
    Call PLC.AdsSyncReadLongVarReq(hlLong, 4, lLong)
    Call PLC.AdsSyncReadSingleVarReq(hsSingle, 4, sSingle)
    Call PLC.AdsSyncReadDoubleVarReq(hdDouble, 8, dDouble)
    Call PLC.AdsSyncReadStringVarReq(hsString, 510, sString)
End Sub
```

*NOTE: if **<DLen>** is less than TwinCAT PLC string size in bytes, the returned string will be truncated. If **<DLen>** is larger than the TwinCAT PLC string size, TwinCAT will simply return the number of bytes required for the whole string.*
*As TwinCAT PLC strings are limited to 255 characters (510 bytes), we can ensure that the string will never be truncated by setting **<DLen>** to 510.*

## 5.3. Writing a TwinCAT Variable (.AdsSyncWritexxxxVarReq)

Visual Basic 6 uses the **`.AdsSyncWritexxxxVarReq`** properties of ADS-OCX to write a value to a variable in the TwinCAT PLC. The actual command used depends on the data type of the variable.

| Function | TwinCAT data type | Length (bytes) |
|---|---|---|
| **`.AdsSyncWriteBoolVarReq`** | BOOL | 1 |
| **`.AdsSyncWriteIntegerVarReq`** | SINT, USINT, BYTE | 1 |
| **`.AdsSyncWriteIntegerVarReq`** | INT, UINT, WORD | 2 |
| **`.AdsSyncWriteLongVarReq`** | DINT, UDINT, DWORD | 4 |
| **`.AdsSyncWriteSingleVarReq`** | REAL | 4 |
| **`.AdsSyncWriteDoubleVarReq`** | LREAL | 8 |
| **`.AdsSyncWriteStringVarReq`** | STRING | String Characters * 2 |

```
    lErrCode = <ADSOCX>.AdsSyncWritexxxxVarReq(<Handle>,<DLen>,<VBVar>)
```

where:

***<ADSOCX>*** name of the instance of ADS-OCX that is to be called, e.g. **PLC**

| | |
|---|---|
| ***<Handle>*** | handle for target variable |
| ***<DLen>*** | length of the target variable in bytes. For a STRING variable, ***<DLen>*** must not exceed 510 bytes (255 characters) |
| ***<VBVar>*** | VB6 variable containing the data that will be written to TwinCAT. This variable must be the same length as the target TwinCAT variable. |

```
Dim bBoolean As Boolean
Dim bByte As Integer
Dim iInteger As Integer
Dim lLong As Long
Dim sSingle As Single
Dim dDouble As Double
Dim sString As String

Private Sub cmdWrite_Click()
    bBoolean = TRUE
    bByte = 123
    iInteger = 12345
    lLong = -12345678
    sSingle = 1.234
    dDouble = -9.87654321
    sString = “This an example string”

    ‘ write to variables in PLC
    Call PLC.AdsSyncWriteBoolVarReq(hbBoolean, 1, bBoolean)
    Call PLC.AdsSyncWriteIntegerVarReq(hbByte, 1, bByte)
    Call PLC.AdsSyncWriteIntegerVarReq(hiInteger, 2, iInteger)
    Call PLC.AdsSyncWriteLongVarReq(hlLong, 4, lLong)
    Call PLC.AdsSyncWriteSingleVarReq(hsSingle, 4, sSingle)
    Call PLC.AdsSyncWriteDoubleVarReq(hdDouble, 8, dDouble)
    Call PLC.AdsSyncWriteStringVarReq(hsString, LenB(sString), sString)
End Sub
```

## 5.4. Reading a TwinCAT Array (.AdsSyncReadxxxxVarReq)

Visual Basic 6 uses the **`.AdsSyncReadxxxxVarReq`** properties of ADS-OCX to read an array from the TwinCAT PLC. The actual command used depends on the data type of the array.

| Function | TwinCAT array data type | Length (bytes) |
|---|---|---|
| **`.AdsSyncReadBoolVarReq`** | BOOL | 1 per array element |
| **`.AdsSyncReadIntegerVarReq`** | SINT, USINT, BYTE | 1 per array element |
| **`.AdsSyncReadIntegerVarReq`** | INT, UINT, WORD | 2 per array element |
| **`.AdsSyncReadLongVarReq`** | DINT, UDINT, DWORD | 4 per array element |
| **`.AdsSyncReadSingleVarReq`** | REAL | 4 per array element |
| **`.AdsSyncReadDoubleVarReq`** | LREAL | 8 per array element |
| **`.AdsSyncReadStringVarReq`** | STRING | String characters * 2 * no of array elements |

```
    lErrCode = <ADSOCX>.AdsSyncReadxxxxVarReq(<Handle>, <DLen>, <VBA>)
```

where:

***<ADSOCX>*** name of the instance of ADS-OCX that is to be called, e.g. **PLC**

| | |
|---|---|
| ***<Handle>*** | handle for target array |
| ***<DLen>*** | number of bytes to be read from the target array |
| ***<VBVar>*** | VB6 array to receive the data from the TwinCAT array. The VB6 array should be at least the same length as the target TwinCAT array |

```
Dim aArray(1 To 10) As Integer

Private Sub cmdRead_Click()

    ' read array from the PLC and put into 1st location in aArray
    Call PLC.AdsSyncReadIntegerVarReq(haArray, LenB(aArray), aArray(1))
End Sub
```

*NOTE: if **<DLen>** is less than TwinCAT PLC array size in bytes, the returned array will be truncated. If **<DLen>** is larger than the TwinCAT PLC array size, TwinCAT will simply return the number of bytes required for the whole array.*

Partial arrays can be read from TwinCAT by reducing the number of bytes requested in ***<DLen>***. Any VB6 array element can be specified in ***<VBVar>*** to receive the first array value, provided that rest of the VB6 array is large enough to handle the total number of bytes requested in ***<DLen>***.

## 5.5. Writing a TwinCAT Array (.AdsSyncWritexxxxVarReq)

Visual Basic 6 uses the **`.AdsSyncWritexxxxVarReq`** properties of ADS-OCX to write a value to an array in the TwinCAT PLC. The actual command used depends on the data type of the target array.

| Function | TwinCAT data type | Length (bytes) |
|---|---|---|
| `.AdsSyncWriteBoolVarReq` | BOOL | 1 per array element |
| `.AdsSyncWriteIntegerVarReq` | SINT, USINT, BYTE | 1 per array element |
| `.AdsSyncWriteIntegerVarReq` | INT, UINT, WORD | 2 per array element |
| `.AdsSyncWriteLongVarReq` | DINT, UDINT, DWORD | 4 per array element |
| `.AdsSyncWriteSingleVarReq` | REAL | 4 per array element |
| `.AdsSyncWriteDoubleVarReq` | LREAL | 8 per array element |
| `.AdsSyncWriteStringVarReq` | STRING | String characters * 2 * no of array elements |

```
    lErrCode = <ADSOCX>.AdsSyncWritexxxxVarReq(<Handle>,<DLen>,<VBVar>)
```

where:

***<ADSOCX>*** name of the instance of ADS-OCX that is to be called, e.g. **PLC**

| | |
|---|---|
| ***<Handle>*** | handle for target array |
| ***<DLen>*** | number of bytes to be written to the target array |
| ***<VBVar>*** | VB6 array containing the data that will be written to TwinCAT. This array must not be larger than the target TwinCAT array |

```
Dim aArray(1 To 10) As Integer

Private Sub cmdWrite_Click()

    aArray(1) = 12
    aArray(2) = 34
    aArray(3) = 56
    aArray(4) = 78
    aArray(5) = 90

    ' write to array in PLC
    Call PLC.AdsSyncWriteIntegerVarReq(haArray, LenB(aArray), aArray(1))
End Sub
```

Partial arrays can be written to TwinCAT by reducing the number of bytes specified in ***<DLen>***. Any VB6 array element can be specified in ***<VBVar>*** as the first array value to be sent, provided that rest of the VB6 array is large enough to supply the total number of bytes requested in ***<DLen>*** and the TwinCAT PLC is large enough to receive the number of bytes being sent.

## 5.6. Delete a TwinCAT Variable Handle (.AdsDeleteVarHandle)

Links to TwinCAT PLC variables should be deleted using the **`.AdsDeleteVarHandle`** property in the **`Form_Unload()`** event when the form is closed.

```
    lErrCode = <ADSOCX>.AdsDeleteVarHandle(<Handle>)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **PLC** |
| ***<Handle>*** | handle that is to be deleted |

```
Private Sub Form_Unload()

    ' delete handles for PLC variables
    lErrCode = PLC.AdsDeleteVarHandle(hbBoolean)
    lErrCode = PLC.AdsDeleteVarHandle(hbByte)
    lErrCode = PLC.AdsDeleteVarHandle(hiInteger)
    lErrCode = PLC.AdsDeleteVarHandle(hlLong)
    lErrCode = PLC.AdsDeleteVarHandle(hsSingle)
    lErrCode = PLC.AdsDeleteVarHandle(hdDouble)
    lErrCode = PLC.AdsDeleteVarHandle(hsString)
    lErrCode = PLC.AdsDeleteVarHandle(haArray)
    lErrCode = PLC.AdsDeleteVarHandle(hgStructure)
End Sub
```

# 6. Accessing Data Structures in TwinCAT

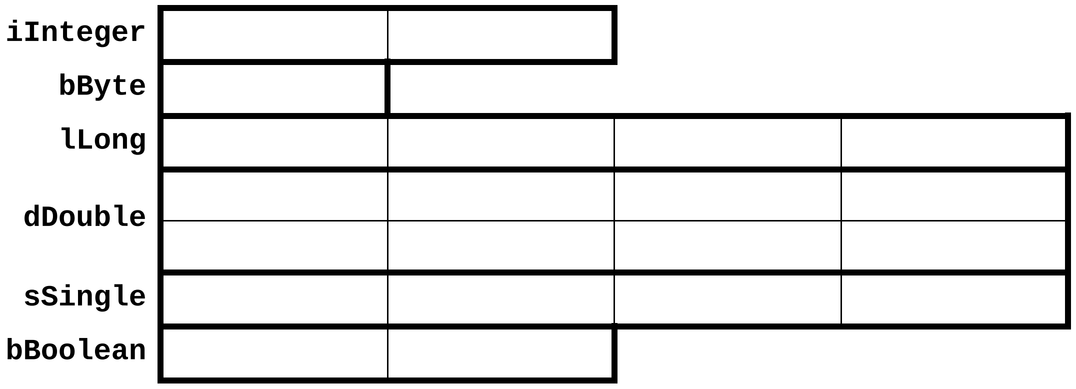
## 6.1. Variable Alignment

In order for PC running under Windows NT/2000/XP to be able to access the variables more quickly, Visual Basic (and other programming languages) arranges them in the main memory according to certain rules using a process called "alignment". VB6 and IEC1131-3 have different alignment rules, and this can lead to "memory holes" appearing in data structures passed between VB6 and TwinCAT. The result is that VB6 and TwinCAT may interpret the data differently, and incorrect values will be seen by each application.

The solution is to carefully consider the specification of the data structure, and ensure that all "memory holes" are filled with dummy variables. An example data structure definition for TwinCAT PLC is shown below, along with the number of bytes used by each type of variable.

```
TYPE S_Structure
STRUCT
    iInteger      : INT;      (* 2 bytes *)
    bByte         : BYTE;     (* 1 byte *)
    lLong         : DINT;     (* 4 bytes *)
    sSingle       : REAL;     (* 4 bytes *)
    dDouble       : LREAL;    (* 8 bytes *)
    bBoolean      : BOOL;     (* 1 byte *)
END_STRUCT
END_TYPE
```

The actual data structure arrangement in TwinCAT VB6 memory is shown below, where each box represents a data byte. The total data length of the data structure is 21 bytes.

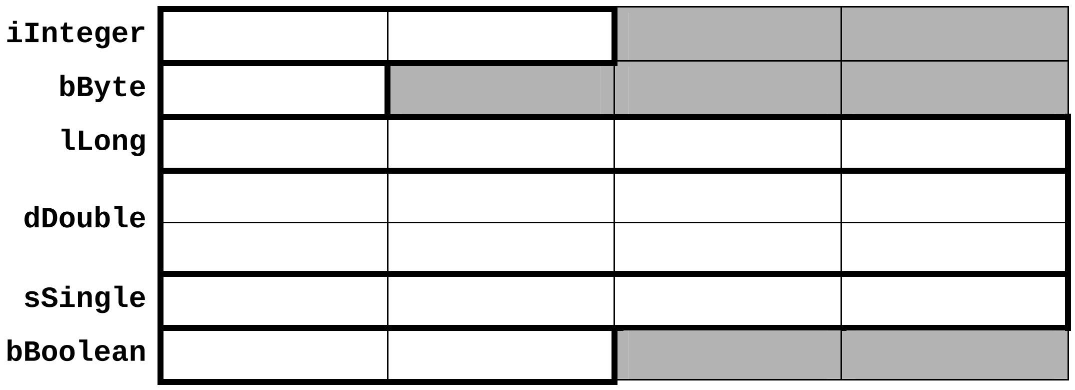


*NOTE: strings cannot be included in a data structure that must be transferred between VB6 and TwinCAT PLC.*

The equivalent data structure definition in VB6 is shown below. The first thing to notice is that VB6 uses 2 bytes for a Boolean variable, as opposed to TwinCAT PLC which only uses 1 byte.

```
Dim gStructure  As S_Structure

Private Type S_Structure
    iInteger      As Integer  ‘ 2 bytes
    bByte         As Byte     ‘ 1 byte
    lLong         As Long     ‘ 4 bytes
    dDouble       As Double   ‘ 8 bytes
    sSingle       As Single   ‘ 4 bytes
    bBoolean      As Boolean  ‘ 2 bytes
End Type
```

In addition, VB6 aligns variables every 4 bytes, so the actual data structure arrangement in VB6 memory is shown below. Every grey shaded cell represents a “memory hole”, i.e. a data byte in the VB6 memory area that is not used due to the alignment, but will be included as part of the data structure. Hence, the actual data structure size in VB6 will be 28 bytes.



VB6 only aligns data at the next block of 4 bytes when it finds a change in variable type. Therefore, if a new integer variable (**iSpareInt1**) is added after **iInteger** in the VB6 data structure, it will be aligned in the 2 spare bytes that follow **iInteger**.

Similarly, adding 3 new variables (**bSpareByte1** to **bSpareByte3**) of type BYTE after **bByte** will also use the 3 spare bytes between **bByte** and **dDouble**.

The solution is to ensure that the data structures in both VB6 and TwinCAT PLC have the same alignment and data size. By declaring additional variables in the data structure in the positions where “memory holes” would occur, we can ensure that the VB6 and TwinCAT data structures are the same size, and data can be transferred between the without problem.

*NOTE: it is advisable to include a variable in the data structure that contains the structure length (in bytes) in the VB6 or TwinCAT application. If these values are not equal, the data structure may have been changed in one of the applications, and can be used to ensure that corrupted data is not used in the machine.*

To calculate the size of the data structure (in bytes) in each program:

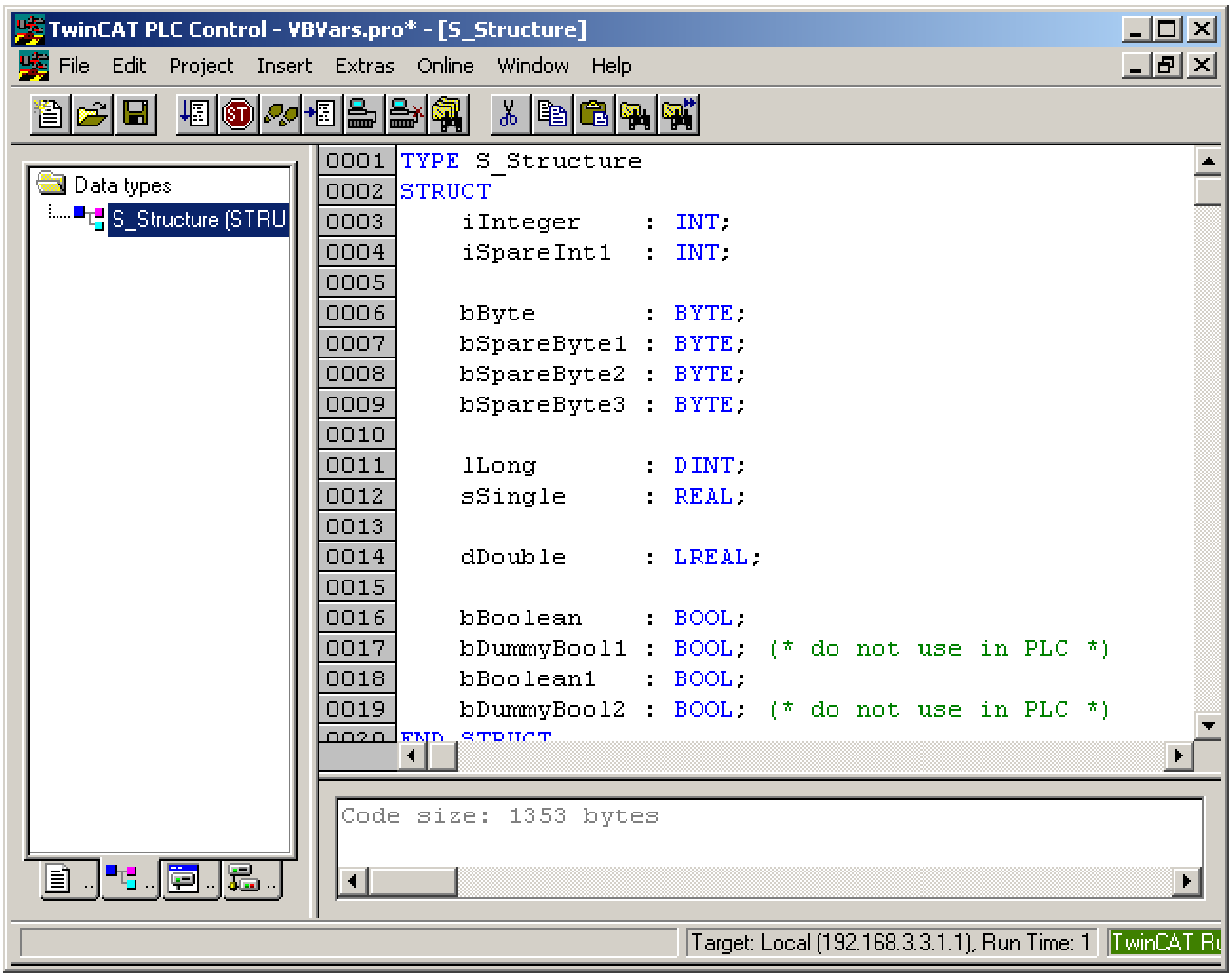
TwinCAT PLC: `StructSize := SIZEOF(gStructure);`

VB6: `StructSize := LenB(gStructure);`

## 6.2. Data Structure Configuration

### 6.2.1. TwinCAT PLC Data Structure

The screen shot below shows a data structure defined in TwinCAT PLC with “spare” or “dummy” variables included to ensure correct data alignment with the data structure in the VB6 application. Spare variables (except dummy BOOL variables) can be renamed and used within the TwinCAT PLC without affecting the existing data structure.


TwinCAT PLC Control - VBVars.pro* - [S_Structure]
File Edit Project Insert Extras Online Window Help
Data types
S_Structure (STRU
Target: Local (192.168.3.3.1.1), Run Time: 1
TwinCAT R

```
TYPE S_Structure
STRUCT
    iInteger     : INT;
    iSpareInt1   : INT;

    bByte        : BYTE;
    bSpareByte1  : BYTE;
    bSpareByte2  : BYTE;
    bSpareByte3  : BYTE;

    lLong        : DINT;
    sSingle      : REAL;

    dDouble      : LREAL;

    bBoolean     : BOOL;
    bDummyBool1  : BOOL; (* do not use in PLC *)
    bBoolean1    : BOOL;
    bDummyBool2  : BOOL; (* do not use in PLC *)
END_STRUCT
```

```
Code size: 1353 bytes
```

INT, UINT and DWORD variables in a data structure must be declared groups of 2, which SINT, USINT and BYTE variables must be declared in groups of 4.

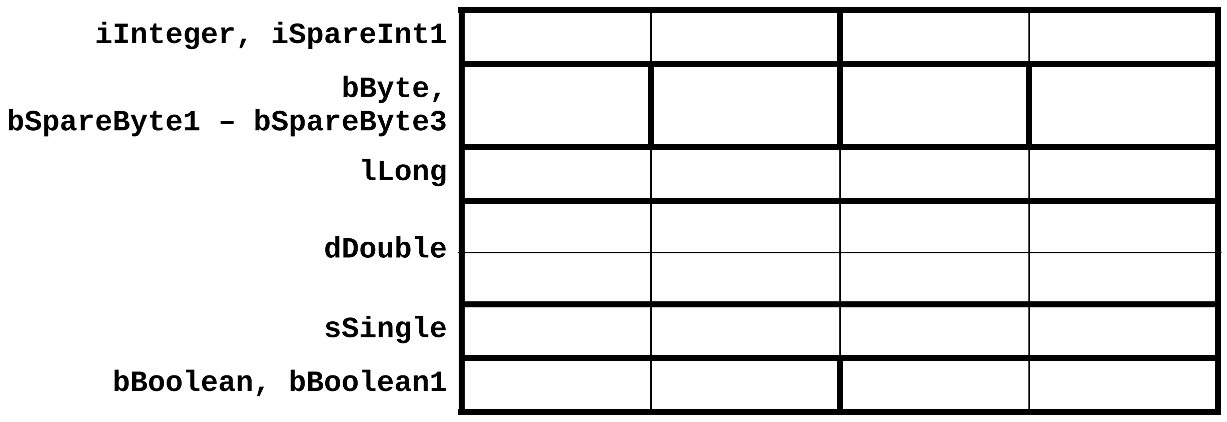
Boolean variables are a special case, as they use 1 byte each in TwinCAT PLC, but 2 bytes in VB6. Therefore, each BOOL variable must be accompanied by a “dummy” BOOL variable that cannot be used in the PLC program, and BOOL variables (plus the dummy variables) must be declared in groups of 2.

The modified TwinCAT PLC data structure (shown above) now has a total length of 28 bytes, which is the same as the VB6 data structure.

### 6.2.2. Visual Basic 6 Data Structure

The VB6 program code to define the data structure is shown below. With the exception of the dummy Boolean variables, the data structure should be the same as the TwinCAT PLC data structure.

```
Dim gStructure  As S_Structure

Private Type S_Structure
    iInteger     As Integer      ‘ 2 bytes
    iSpareInt1   As Integer      ‘ 2 bytes

    bByte        As Byte         ‘ 1 byte
    bSpareByte1  As Byte         ‘ 1 byte
    bSpareByte2  As Byte         ‘ 1 byte
    bSpareByte3  As Byte         ‘ 1 byte

    lLong        As Long         ‘ 4 bytes

    dDouble      As Double       ‘ 8 bytes

    sSingle      As Single       ‘ 4 bytes

    bBoolean     As Boolean      ‘ 2 bytes
    bBoolean1    As Boolean      ‘ 2 bytes
End Type
```



## 6.3. Reading a Data Structure from TwinCAT

The **`.AdsSyncReadxxxxVarReq`** property of ADS-OCX is used to read a TwinCAT data structure into a VB6 program, where **`xxxx`** is the data type of the first defined element of the data structure.

```
    lErrCode = <ADSOCX>.AdsSyncReadxxxxVarReq(<Handle>,<DLen>,<VBVar>)
```

where:

| | |
|---|---|
| ***<ADSOCX>*** | name of the instance of ADS-OCX that is to be called, e.g. **`PLC`** |
| ***<Handle>*** | handle for target structure |
| ***<DLen>*** | total length of the structure in bytes |
| ***<VBVar>*** | 1st element of the VB6 data structure receiving data from the TwinCAT data structure |

```
Private Sub cmdReadStructure_Click()

    'read structure from PLC
    Call PLC.AdsSyncReadIntegerVarReq(hgStructure, LenB(gStructure),
                                               gStructure.iInteger)

    ' display values from PLC in labels
    lblIntegerRead.Caption = gStructure.iInteger
    lblByteRead.Caption = gStructure.bByte
    lblLongRead.Caption = gStructure.lLong
    lblSingleRead.Caption = gStructure.sSingle
    lblDoubleRead.Caption = gStructure.dDouble
    lblBooleanRead.Caption = gStructure.bBoolean
End Sub
```

## 6.4. Writing a Data Structure to TwinCAT

The **.AdsSyncWritexxxxVarReq** property of ADS-OCX is used to write to a TwinCAT data structure from a VB6 program, where **xxxx** is the data type of the first defined element of the data structure.

```
    lErrCode = <ADSOCX>.AdsSyncWritexxxxVarReq(<Handle>,<DLen>,<VBVar>)
```

where:

***<ADSOCX>*** name of the instance of ADS-OCX that is to be called, e.g. **PLC**

| | |
|---|---|
| ***<Handle>*** | handle for target structure |
| ***<DLen>*** | total length of the structure in bytes |
| ***<VBVar>*** | 1st element of the VB6 data structure being written to the TwinCAT data structure. |

```
Private Sub cmdWriteStructure_Click()

    ' load data values to be written
    gStructure.iInteger = CInt(txtIntegerWrite.Text)
    gStructure.bByte = CByte(txtByteWrite.Text)
    gStructure.lLong = CLng(txtLongWrite.Text)
    gStructure.sSingle = CSng(txtSingleWrite.Text)
    gStructure.dDouble = CDbl(txtDoubleWrite.Text)
    gStructure.bBoolean = CBool(chkBooleanWrite.Value)

    ' write data structure to PLC
    Call PLC.AdsSyncWriteIntegerVarReq(hgStructure, LenB(gStructure),
                                              gStructure.iInteger)
End Sub
```