

AtouchX Parametrization

Abstract

The Application note deals with a demonstration of application of AtouchX communication control in the environment of Microsoft Visual C# Express and in Microsoft Excel.

Author: Michal Kupčák, Zbyněk Říha
File: ap0013_en_04.pdf

Attachments

File content: ap0013_en_03.zip

ap0013_p01_en_02.zip	Projects for control systems
ap0013_p02_en_02.xls	Excel – reading and writing simple variables
ap0013_p03_en_02.xls	Excel – reading and writing matrices
ap0013_p04_en_02.xls	Excel – editing time in the control system
ap0013_p05_en_02.xls	Excel – reading the archive
ap0013_p06_en_02.xls	Excel – reading the operations journal
ap0013_p07_en_02.xls	Excel – communication with two identical control systems in a network
ap0013_p08_en_02.xls	Excel – active communication between a control system and PC
ap0013_p09_en_02.zip	C# – reading and writing a simple variable
ap0013_p10_en_02.zip	C# – reading and writing matrices
ap0013_p11_en_02.zip	C# – editing time in the control system
ap0013_p12_en_02.zip	C# – reading the archive
ap0013_p13_en_02.zip	C# – reading the operation log
ap0013_p14_en_02.zip	C# – communication with two identical control systems in a network
ap0013_p15_en_02.zip	C# – active communication between the control system and PC
ap0013_p16_en_01.zip	Delphi – a sample application, other useful information

Contents

	Contents.....	2
	Revision history	3
	Related documentation	3
1	Definitions of terms.....	4
2	AtouchX	5
2.1	AtouchX installation.....	5
3	AtouchX parametrization	6
3.1	Exporting parametrization files from DetStudio.....	6
4	Sample applications in Microsoft Excel	9
4.1	Microsoft Excel 2013 setting.....	9
4.2	Creating an application.....	11
4.2.1	AtouchApp definition	11
4.2.2	Initialization to connect with the DB-Net network (DB-Net/IP)	11
4.2.3	A sample method not causing an event.....	12
4.2.4	A sample method causing an event.....	13
4.2.5	Communication termination.....	14
4.2.6	Working with archives	14
	Archive initialization.....	15
	Enabling archive function	15
	Saving an archive sample	15
	Termination of the AtouchArch object activity	16
5	Sample applications in Microsoft Visual C# Express	17
5.1	Visual C# Express settings.....	17
5.2	Creating an application.....	19
5.2.1	AtouchApp definition	20
5.2.2	Initialization to connect with the DB-Net network (DB-Net/IP)	20
5.2.3	A sample method not causing an event.....	22
5.2.4	A sample method causing an event.....	23
5.2.5	Communication termination.....	26
5.2.6	Working with archives	26
	Archive initialization.....	26
	Enabling archive function	27
	Saving an archive sample	27
	Terminating archive function	28
	Termination of the AtouchArch object activity	28
6	APPENDIX A	29
6.1	Conversion of variable types in C#	29
7	APPENDIX B	30
7.1	AtouchX in Delphi.....	30
8	Technical support	31
9	Warning.....	32

Revision history

Version	Date	Changes by	Changes
001	22. 06. 2009	Říha Zbyněk, Kupčák Michal	New document.
002	02. 01. 2013	Říha Zbyněk	Pictures changed, textBox (ErrText) names unified in chapter 5
003	11. 05. 2017	Kupčák Michal	Images modified, chapter 5.1 amended.
004	07. 02. 2018	Říha Zbyněk	The reference to the object AtouchXArchive cancelled in applications for Excel.

Related documentation

1. Help tab in the PseDet section of the DetStudio development environment
file: Psedet_en.chm
2. Help tab in the AtouchX communication controller
file: AtouchX.chm

1 Definitions of terms

DetStudio

A development environment by the company AMiT serving for control systems parametrization. This environment is freely accessible at amitautomation.com.

Station

Control system or PC in the network DB-Net(/IP).

2 AtouchX

AtouchX is a communication controller that provides connection between control systems and applications on PC. AtouchX is the perfect means for all programmers who develop their own applications on PC and need to provide data transfer from / to AMiT's control systems. It includes several ActiveX objects that are designed for full access to data in control systems. Apart from a basic data transfer between the control systems and applications on PC, it also supports so called "local archives", date and time transfers, and also provides detection of station status and other functions.

All objects in the AtouchX library have functional character (interface). In practice, this means that objects have no properties (or only a few properties of a rather functional nature) and they are handled exclusively by means of methods (functions).

2.1 AtouchX installation

The AtouchX communication controller installation file is available for download free of charge at amitautomation.com. After the installation launches, an installation wizard opens in which we specify the path of locations where sample applications and help section for the communication controller are to be installed. Libraries that AtouchX uses for communication are always installed into the Windows system directory.

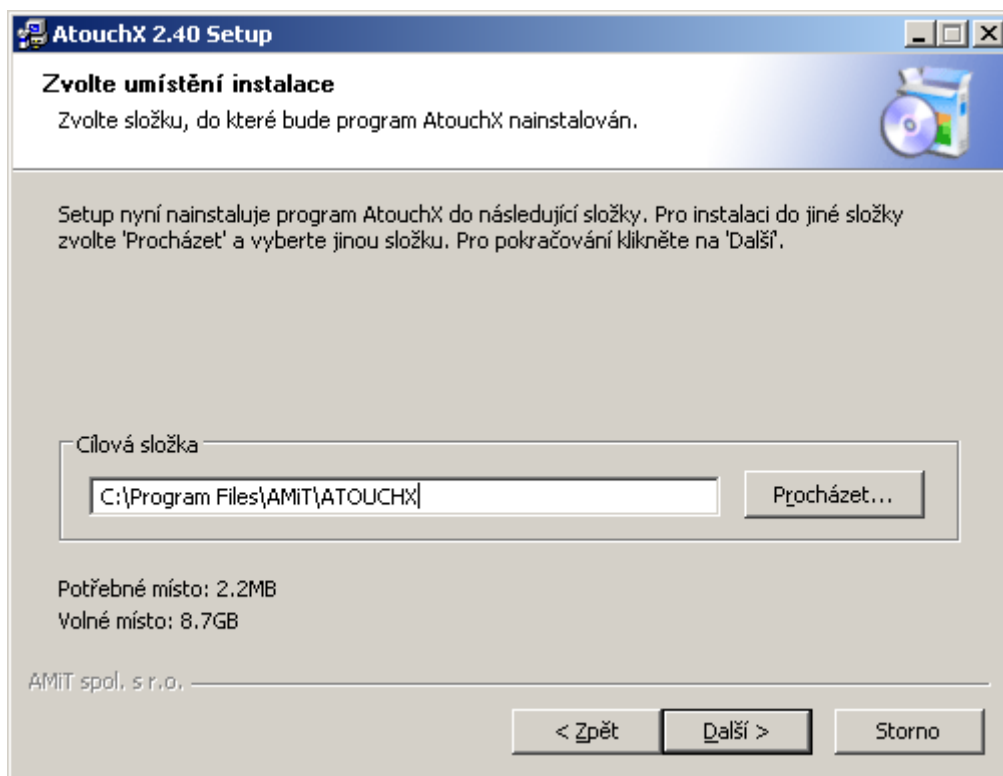


Fig. 1 – AtouchX installation

3 AtouchX parametrization

The communication controller parametrization can be performed by means of three *.ini files (among other methods) which we use to specify a list of control system variables to the controller from / to which we want to read / write, and the interface by means of which we want to communicate with the control system (a serial line, Ethernet, etc.) and the description of archives or the operation log. The structure of these files is described in more detail in the Help section to the AtouchX communication controller.

3.1 Exporting parametrization files from DetStudio

We create parametrization files directly by means of the DetStudio development environment using the menu **Tools / Export / Atouch**.

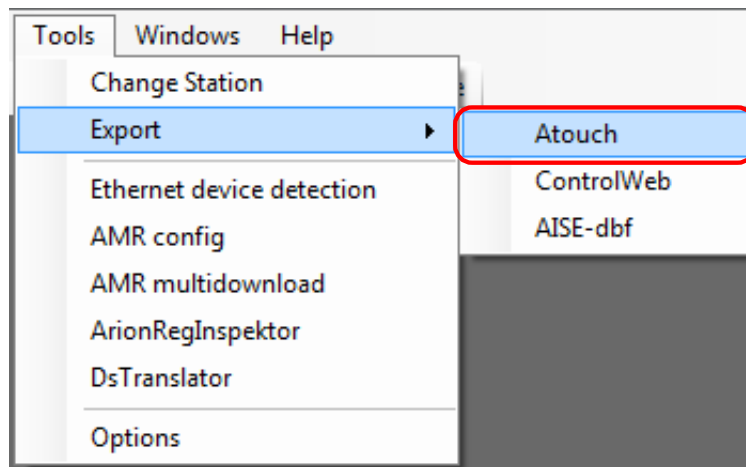


Fig. 2 – Exporting parametrization files

By selecting the item **Atouch**, we open a window in which we specify the directory the parametrization files should be exported into. We select the variables to be read / written and in case work with archives from the control system is required we also define the archives.

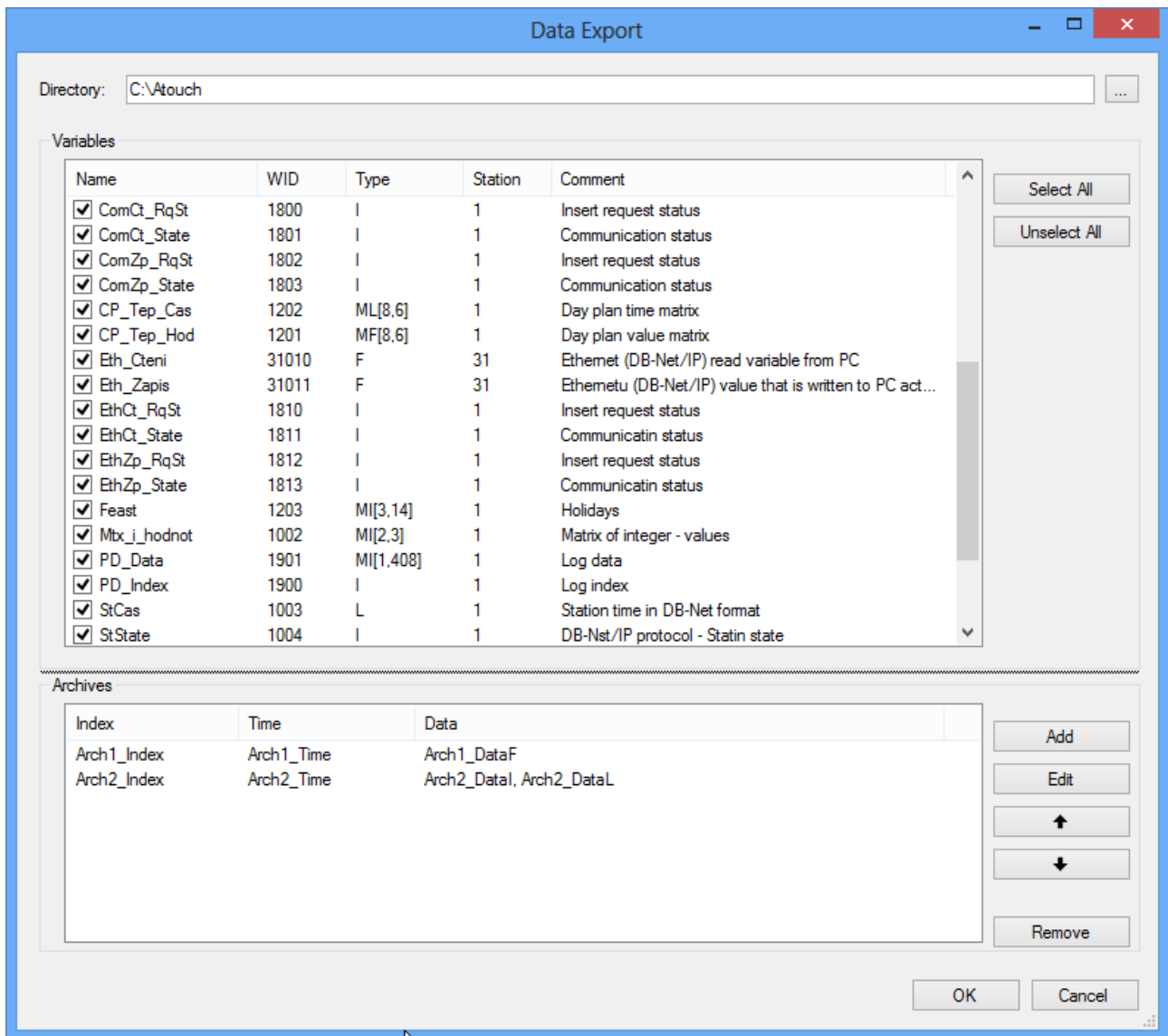


Fig. 3 – Selecting directory to export parametrization files

Clicking the button “**OK**” closes the window “Select the list of variables from the database list” and create two to three parametrization files HW.ini and SW.ini or ARC.ini in the directory of our choice.

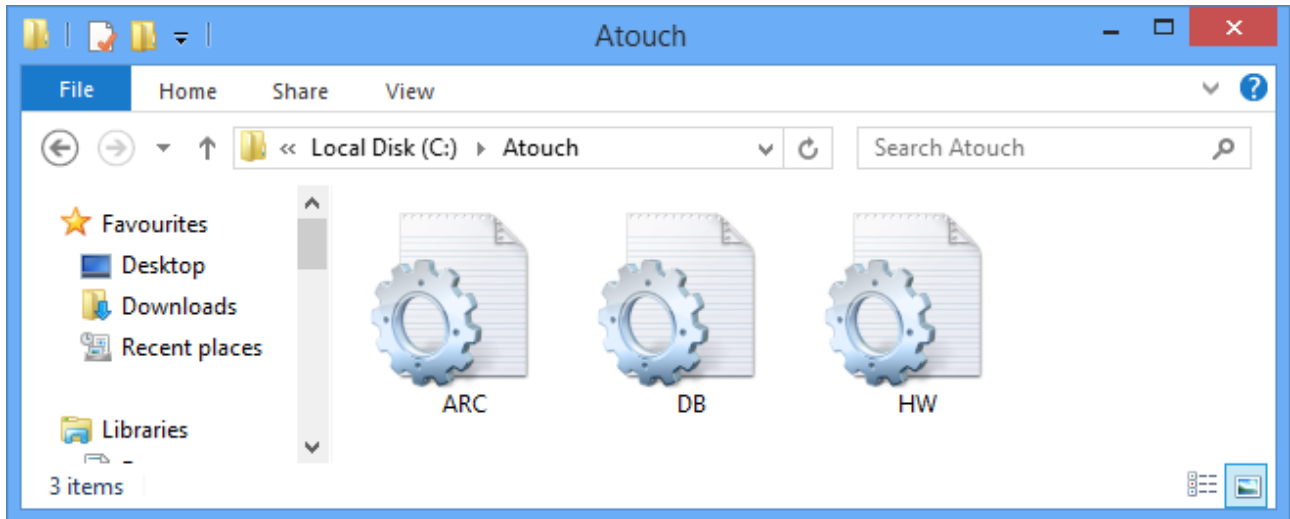


Fig. 4 – Created *.ini files

The file db.ini includes description of all database variables selected during the export. The HW.ini file includes communication parameters for a connection with the control system. The ARC.ini file contains descriptions of selected archives and the application operation log. Communication parameters are generated according to communication set in the DetStudio development environment.

4 Sample applications in Microsoft Excel

The Appendix to this Application note includes sample projects created in Microsoft Excel 2013 chart editor.

4.1 Microsoft Excel 2013 setting

In order to work with the AtouchX controller, it is necessary to enable working with macros in Microsoft Excel and running the macros.

The AtouchX library in Excel can only be used in connection with the programming language Visual Basic. We enter the Visual Basic editor by means of the **Visual Basic** button in the menu **Developer**.

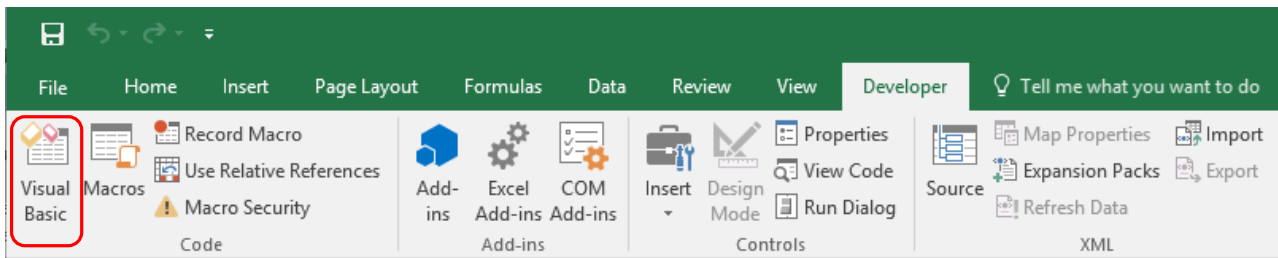


Fig. 5 – Transition to the Visual Basic language editor

In the Visual Basic language editor, we need to tell Excel that we are going to work with the AtouchX library, which we do by means of the menu **Tools / References**.

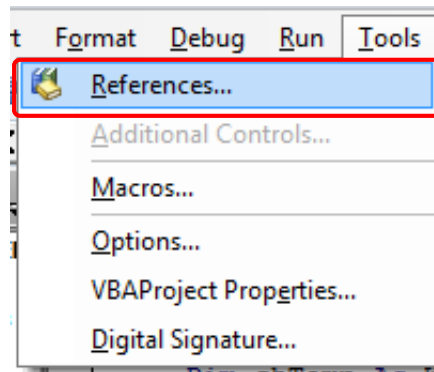


Fig. 6 – Opening the References window

We open the window “References – VBA project” where we search the appropriate library and select it.

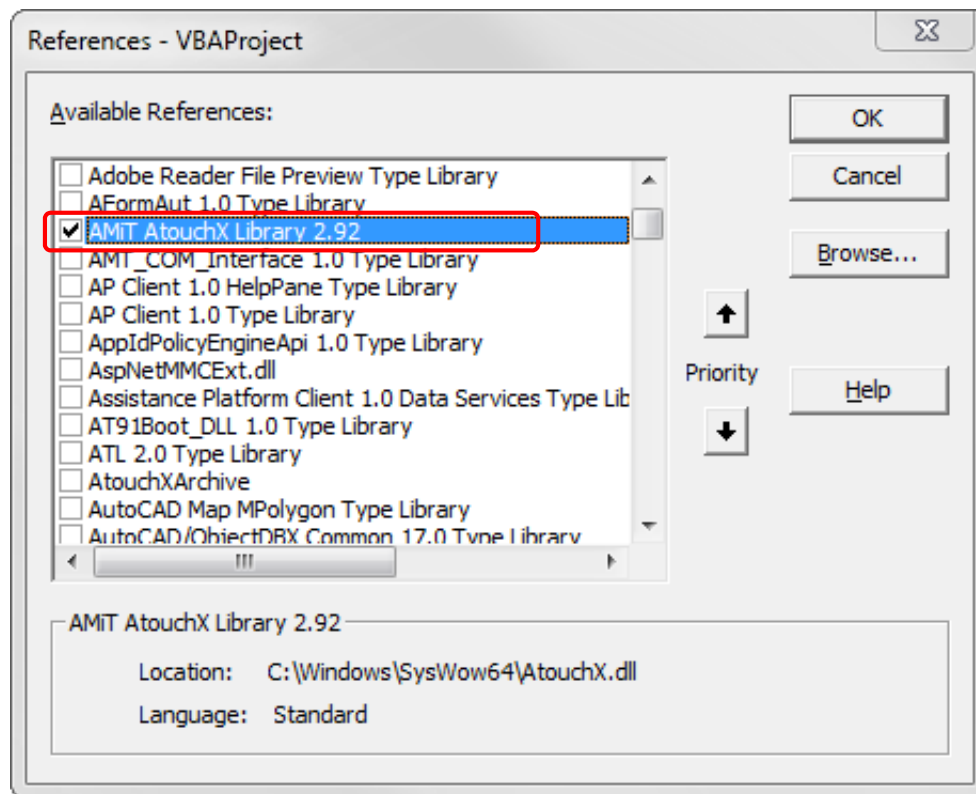


Fig. 7 – Selecting the appropriate libraries

The window “Object Browser” tells us about the AtouchX library options; we open this window from the menu **View / Object Browser**.

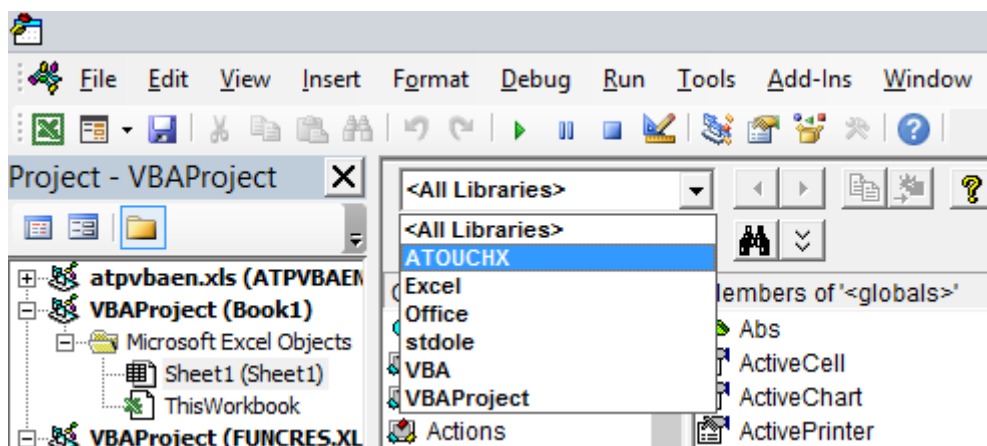


Fig. 8 – Displaying the list of AtouchX library objects

After we select the ATOUGHX library, the window “Object Browser” displays a list of the library’s objects. Descriptions of these objects are available in the Help section that is installed together with the AtouchX communication controller.

4.2 Creating an application

In the project tree, we double-click the left mouse button e.g. on the item "List1 (List1)". An empty form opens where we can enter the programme's code.

4.2.1 AtouchApp definition

We are going to use the "AtouchApp" object for communication in sample applications. First, we need to define the object type "AtouchApp". We do so e.g. with the following code:

```
Public WithEvents ATC As AtouchApp
```

4.2.2 Initialization to connect with the DB-Net network (DB-Net/IP)

In order to initialize it, we create a macro that we name "AtouchInitialization".

```
Public Sub AtouchInitialization()  
EndSub
```

In this macro, we first create a communication controller instance by means of the command "New" (we create an object called "ATC").

```
Set ATC = New AtouchApp
```

In order to actually initialize the connect with the DB-Net network (DB-Net/IP), we use e.g. The method "InitFromFile". This method performs the network connection initialization by means of two external files. In our case, they are called hw.ini and db.ini. The method also returns an error code. We save this code into the global variable "Retrn" of Integer type.

```
Retrn = ATC.InitFromFile(ActiveWorkbook.Path & "\hw.ini", ActiveWorkbook.Path &  
"\db.ini")
```

Subsequently, we find out whether the initialization went on successfully or failed. In case there was an error in initialization, we open the window with the error message and we cancel the object "ATC". If the initialization went on successfully, we write "Initialization OK" into the cell B4.

```
If Retrn <> arrOK Then  
    MsgBox "Connection to DBNet failed. " & vbCrLf & "Error number " & Hex$(Retrn),  
vbOKOnly  
    ATC.Done  
    Exit Sub  
Else  
    Me.Cells(4, 2) = "Initialization OK"  
End If
```

We insert the button into the MS Excel chart, change its text to "Initialization" and match it with the macro we created.

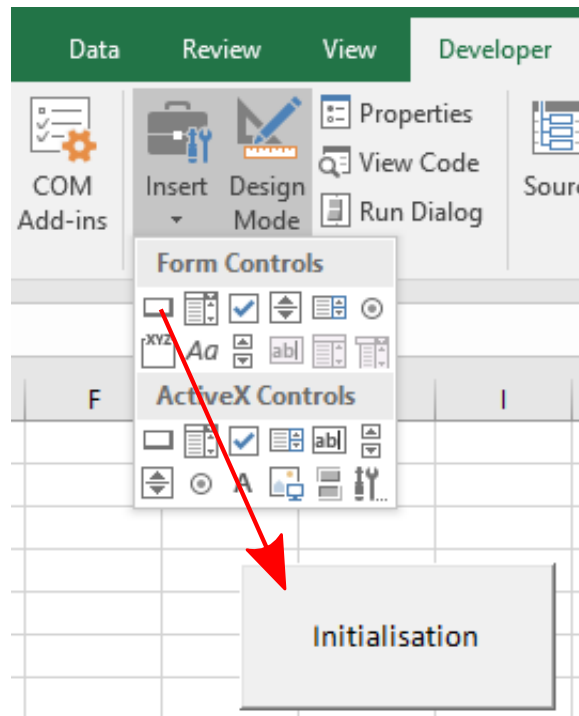


Fig. 9 – Inserting the button

After turning off the design regime, we click the button to initialize network connection to DB-Net (DB-Net/IP).

4.2.3 A sample method not causing an event

We select method “Station Status” (finding out the station’s connection status) as a sample method that does not cause an event. We create a macro again, and name it “StationStatus”.

```
Public Sub StationStatus()  
End Sub
```

The guide says that the syntax for the “StationStatus” method is as follows:

```
object.StationStatus (ByVal Station As Integer, ByRef INFO As Variant) As Integer
```

In our case, we use the object “ATC”. The simplest way to find out what the correct entry is the floating Help that displays automatically when we write commands.

```
ATC.StationStatus |  
StationStatus(Station As Integer, INFO) As Integer
```

Fig. 10 – The floating Help

We define the variable “Station” type Integer that includes the number of the station the status of which we want to find out and a variable “Info” type Variant.

Subsequently, we put the following code into the macro we created:

```
Station = INT(1)           'saving value 1 into Station variable  
Retrn = ATC.StationStatus(Station, Info) 'event call
```

Subsequently, we find out whether the method processing went on successfully or failed. In case the processing failed, we display the window with the error message. In case the processing went on successfully, we save the status and type of HW connection into cells C4 and C5.

```
If Retrtn <> arrOK Then
    MsgBox "Status query failed. " & vbCrLf & "Error number " & Hex$(Retrtn), vbOKOnly
    Exit Sub
Else
    Me.Cells(4, 3) = Info(0)           'saving HW connection type into cell C4
    Me.Cells(5, 3) = Info(1)           'saving connection status into cell C5
EndIf
```

We enter the button into the MS Excel chart again. We change the button text to “Status” and match it with the macro we created. After the network connection to DB-Net initializes successfully, pressing the button will process the code contained in the macro.

4.2.4 A sample method causing an event

We select the method “NetGetData” as a sample method causing an event as it causes the event “EndNetGetData”.

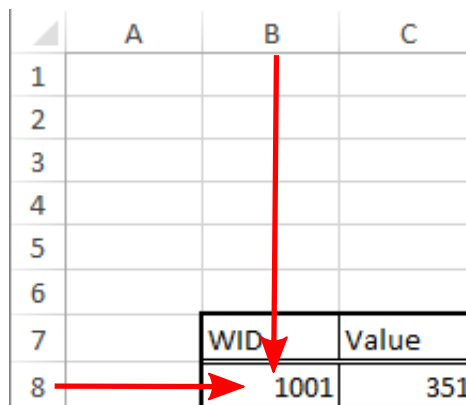
We create a macro using which we execute the command to read the variable value. We name the macro “GetData”.

```
Public Sub GetData()
End Sub
```

In order to make the request for data reading, we use the “NetGetData” method which has the following syntax:

```
object.NetGetData (ByVal WID As Long, ByVal Param As Long) As Integer
```

We enter the WID variable whose value we want to read from the control system into the cell B8 of the MS Excel chart.



	A	B	C
1			
2			
3			
4			
5			
6			
7		WID	Value
8		1001	351

Fig. 11 – The cell for entering the WID value

We define the variable “WID” type Integer and the variable “Param” type Long. Subsequently, we put the following code into the macro we created:

```
WID = Me.Cells(8, 2)           'reading the value of WID from cell B8 into
                                variable WID
Retrtn = ATC.NetGetData(WID, Param) 'causing event
Me.Cells(2, 3) = Retrtn         ' saving the result of the event call into cell C2
```

The method reads the content of the variable the WID of which we entered into cell B8 (reading is asynchronous) and after the reading is finished, the event “EndNetGetData” is caused. The event syntax is as follows:

```
Private Sub object_EndNetGetData ([index As Integer], ByVal WID As Long, ByVal Result As Long, ByVal Param As Long, ByVal DATA As Variant)
```

The method “EndNetGetData” announces the result of the communication and the value of the variable with WID required.

```
Private Sub ATC_EndNetGetData(ByVal WID As Long, ByVal Result As Long, ByVal Param As Long, ByVal DATA As Variant)
```

First, we find out the communication result. If the communication resulted in an error, we display the window with the corresponding error message. If the communication succeeded, we save the value of the variable read into cell C8.

```
If ((Result And atfMaskstate) <> atfOk) Then
    MsgBox "Communication failed. " & vbCrLf & "Error number " & Hex$(Result And atfMaskstate), vbOKOnly
Else
    Me.Cells(8, 3) = DATA
End If
End Sub
```

We enter the button into the MS Excel chart again. We change its text to “Read Data” and match it with macro “GetData”. After successful initialization of connection to the DB-Net network, pressing the button will read the value of the scalar variable with WID 1001 into the cell C8.

4.2.5 Communication termination

Same as we initialized the connection to the DB-Net network (DB-Net/IP), we also have to end this connection properly. In order to end the connection to the DB-Net network (DB-Net/IP), we use the method “Done” with the following syntax:

```
object.Done () As Integer
```

We create a macro and name it “End”.

```
Public Sub End()
End Sub
```

The we perform a check in the macro to see whether the object “ATC” exists and in case it does we cancel it and free up memory. The resulting code we enter into the macro will be as follows:

```
If Not (ATC Is Nothing) Then ATC.Done 'if the object exists, we cancel it
Set ATC = Nothing 'we free up memory
```

We enter the button into the MS Excel chart. We change its text to “End” and match it with macro “End”.

4.2.6 Working with archives

In order to work with archives, we need to create an object type “AtouchArch”. We create it in the same way we created the object “AtouchApp” in chapter 4.2.1 “AtouchApp definition”.

```
Public WithEvents ATCA As AtouchArch
```

Caution!

In order to work properly, the object requires that an object providing connection to DB-Net network (DB-Net/IP) exists and works along with it.

Archive initialization

In order to initialize it, we create a macro that we name "ArchiveInitialization".

```
Public Sub ArchiveInitialization()
End Sub
```

In this macro, we first create an archive instance using the command "New".

```
Set ATCA = New AtouchArch
```

For the actual archive initialization, we use e.g. the method "InitFromFile". This method performs the archive initialization by means of an external file. In our case, we name it pd_arch.ini (the structure is described in the Help section of the AtouchX communication control). The method also returns an error code. We save this code into the global variable "Retrn" of Integer type.

```
Retrn = ATCA.InitFromFile(ActiveWorkbook.Path & "\pd_arch.ini")
```

We display the value of the "Retrn" variable in the MS Excel chart cell.

```
Me.Cells(2, 3) = Retrn
```

We enter the button into the MS Excel chart. We change the button text to "Arch Init" and match it with the macro we created. After successful initialization of the connection to the DB-Net network, the press of the button initializes the archive and the initialization result will be entered into cell C2.

Enabling archive function

After we enable the archive function, we create a macro that we call "ArcFun".

```
Public Sub ArcFun()
End Sub
```

In our application, we will be using an automatic archive (for more information see Help section on the AtouchX communication controller). We will use the "Control" method to enable the archive function. The method syntax is as follows:

```
object.Control (ByVal AID As Integer, ByVal Run As Boolean) As Integer
```

We create a variable "AID" type Integer in the macro and a variable "RUN" type Boolean. The code we enter into the macro will be as follows:

```
AID = 0 'archive number (see Help section on AtouchX)
Run = True 'enabling archive function
RetrnArc = ATCA.Control(AID, Run) 'Method for enabling archive function
```

We enter the button into the MS Excel chart. We change the button text to "Arch Start" and match it with the macro we created. After the archive initializes successfully, pressing this button will enable its function.

Saving an archive sample

We use the event "Sample" to save archive samples. We cause the event if one sample of an automatic archive is available. The syntax is as follows:

```
Private Sub object_Sample ([index As Integer], ByVal AID As Integer, ByVal DATA As Variant)
```

Then we define the "Sample" event including the sample reception process. The code will then look like this:

```
Private Sub ATCA_Sample(ByVal AID As Long, ByVal DATA As Variant)
Dim SampleOK As Boolean 'Definition of a variable type Boolean
Dim AcceptOK As Integer 'Definition of a variable type Integer

Me.Cells(Radek, 1) = DATA(0) 'Saving the sample time into a cell
```

```
Me.Cells(Radek, 2) = DATA(1)      'Saving the archive value into a cell
Radek = Radek + 1                  'Moving to the next row
SampleOK = True                    'We always accept the sample
AcceptOK = ATCA.Accept(AID, SampleOK) 'Sample acceptance verified
End Sub
```

In the event “Sample”, we need to use the method “Accept” in order to accept or reject a sample. This method is described in the Help section on the AtouchX communication controller.

Termination of the AtouchArch object activity

Same as we initialized the archive, we also have to terminate it properly. In order to do so, we use the method “Done” with the following syntax:

```
object.Done () As Integer
```

We enter the AtouchArch object termination e.g. Into the same macro as the termination of the connection to the DB-Net network (DB-Net/IP). Then we perform a check in the macro to see whether the object “ATCA” exists and in case it does we cancel it and free up memory. The resulting code we enter into the macro will be as follows:

```
If Not (ATCA Is Nothing) Then ATCA.Done  'if the object existed, we cancel it
Set ATCA = Nothing                       'we free up memory
```


5 Sample applications in Microsoft Visual C# Express

The Appendix to this Application note includes sample projects created in the Microsoft Visual C# Express environment.

5.1 Visual C# Express settings

In order to launch the programme, we select the item **File / New project** from the main menu. A window “NewProject” opens in which we select the item **Windows Form Application** and change the name of the project created.

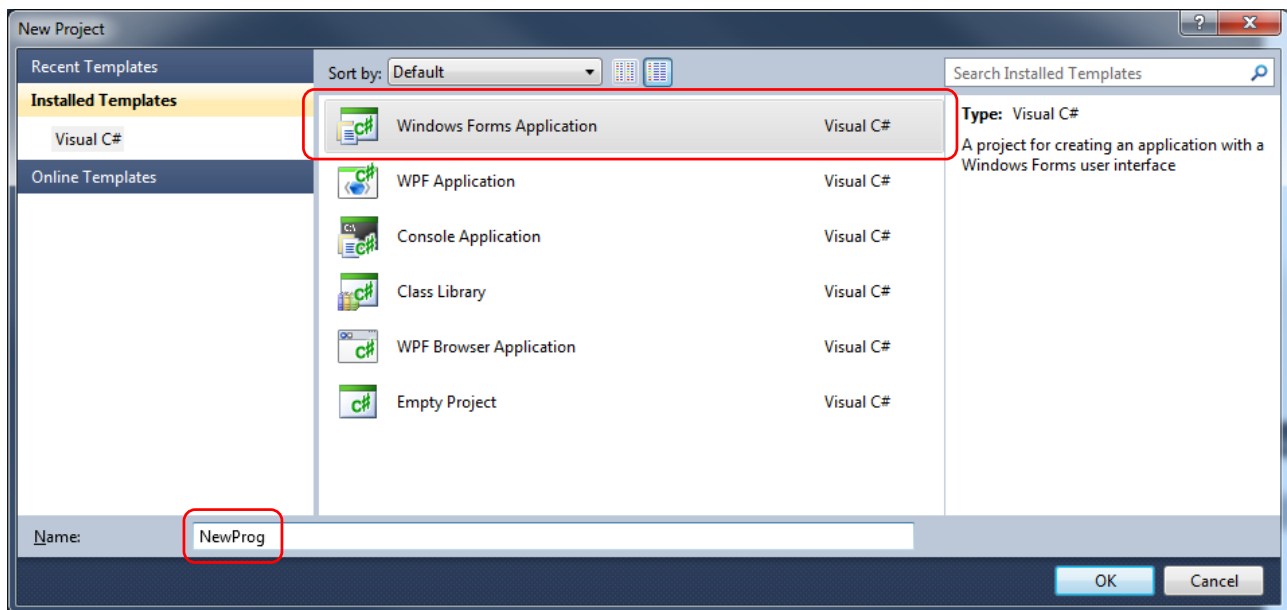


Fig. 12 – Creating a project in Visual C# Express

In order to be able to use an AtouchX library in the Visual C# Express environment, we first need to add a reference to the corresponding ActiveX controls in the window “Solution Explorer”. We achieve this by clicking the right mouse button on the **Reference** item and subsequently selecting the item **Add reference**.

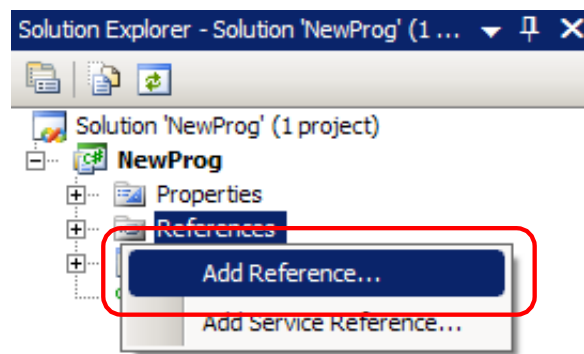


Fig. 13 – Opening the References window

The window “Add Reference” opens in which we search and select the corresponding library in the “COM” tab according to the following picture.

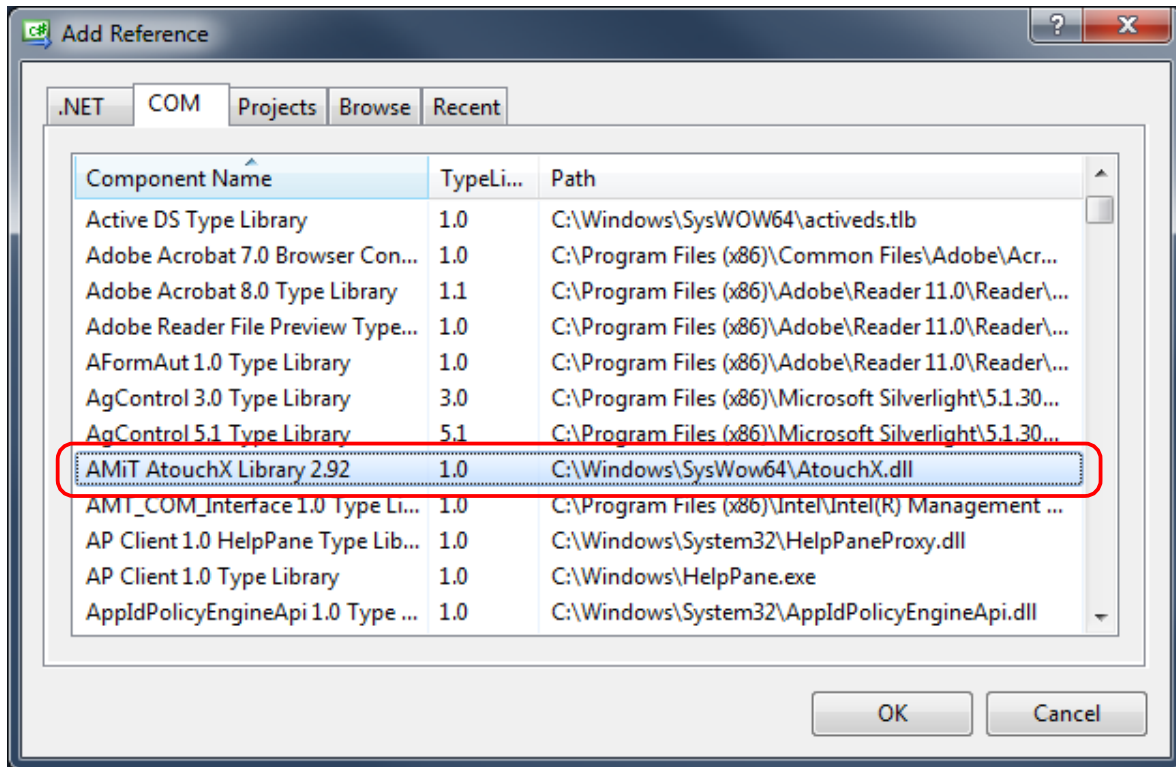


Fig. 14 – Selecting the appropriate libraries

We verify the correct reference addition by unfolding the item **References** in the window “Solution Explorer”. Here we find the item AtouchX.

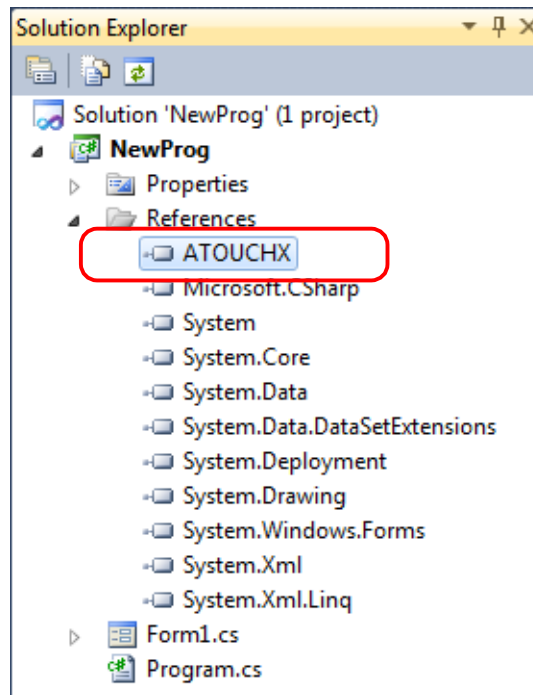


Fig. 15 – The reference to AtouchX has been added correctly

In order for all function of the AtouchX library to work properly in the user application, it is necessary to set the property **Embed Interop Types** to the value **False** in this library.

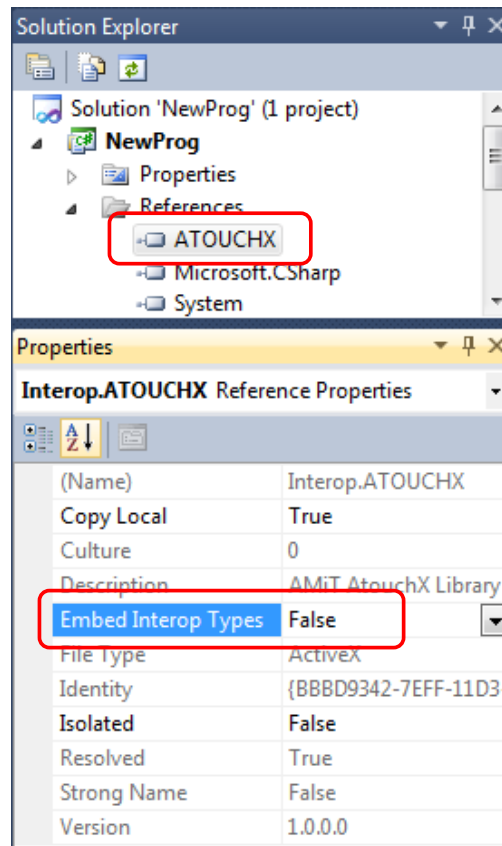


Fig. 16 – Changing the value of the property Embed Interop Types after marking the AtouchX library

5.2 Creating an application

We double-click the empty form. Doing so will get us into the window where we will write the programme code.

First of all, we place the mouse cursor to the start of the code under the word “using”. Here, we write:

```
using ATOUCHX;
```

The resulting code will look as in the following picture.

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;  
using ATOUCHX;
```

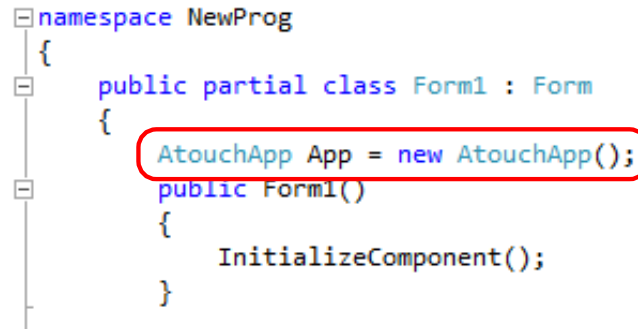
Fig. 17 – Application of the word “using”

5.2.1 AtouchApp definition

In order to work with the object “AtouchApp”, we need to create its instance by means of the command “New” already in the code part “public partial class Form1 : Form”. This part that opens after we double-click on the created form also serves for writing declarations of other global variables and objects.

The code for creating an instance of the “AtouchApp” object will therefore look as follows:

```
AtouchApp App = new AtouchApp();
```



```
namespace NewProg
{
    public partial class Form1 : Form
    {
        AtouchApp App = new AtouchApp();
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Fig. 18 – Creating an instance of the “AtouchApp” object

5.2.2 Initialization to connect with the DB-Net network (DB-Net/IP)

We perform the initialization in the code part “Form1_Load”. In order to actually initialize the connect with the DB-Net network (DB-Net/IP), we use e.g. The method “InitFromFile”. This method performs the network connection initialization by means of two external files. In our case, they are called hw.ini and db.ini. The method also returns an error code. We save this code into the global variable “Err” type Integer and write its value in the form.

First, we switch back to the form design draft and select the item “TextBox” in the window “Toolbox”.

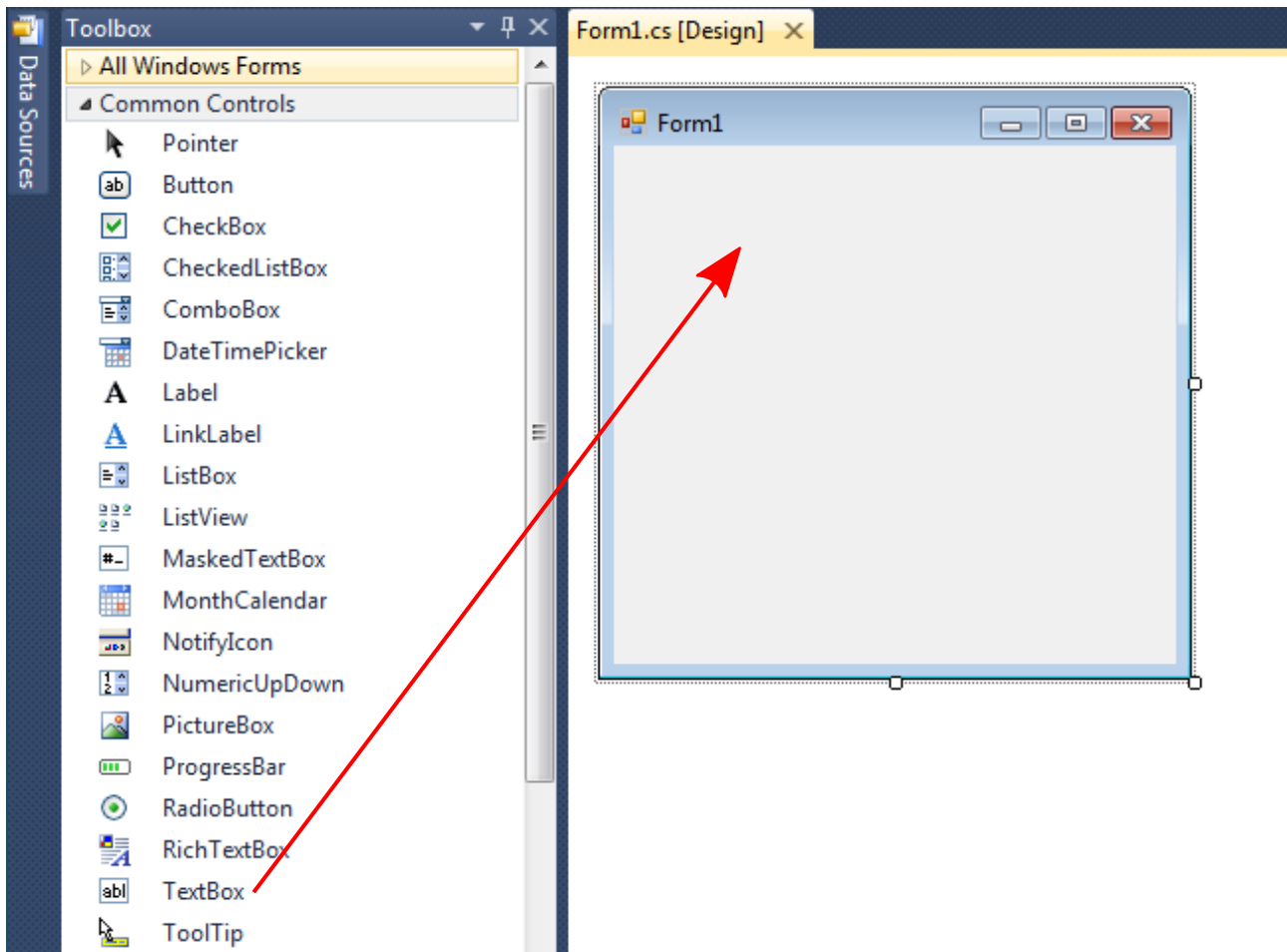


Fig. 19 – Placing the TextBox control into the form

We place it into the form and name it e.g. “ErrText” in the item “Name” in the window “Properties”.

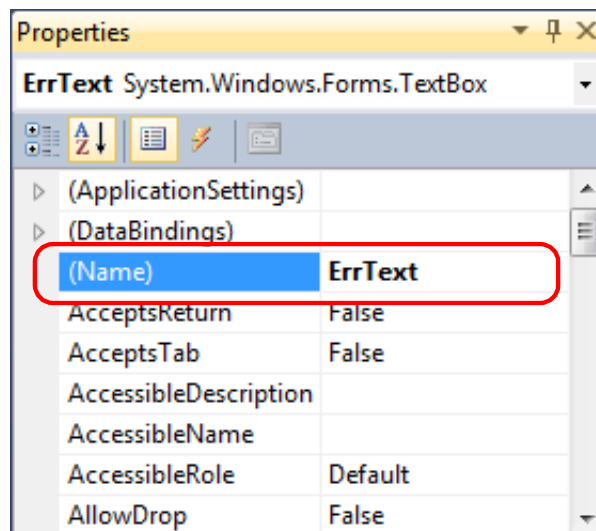



Fig. 20 – Renaming the TextBox control

The method “InitFromFile” returns the error code as a number. However, the “TextBox” control displays a text in the form (type String). That is why we first retype the variable “Err” by means of the method “ToString()”. The resulting code will then look like this:



```
private void Form1_Load(object sender, EventArgs e)
{
    //Initialization method for DB-net network communication.
    App.VariantOnly = false;
    Err = App.InitFromFile("hw.ini", "db.ini");
    ErrText.Text = Err.ToString();
}
```

Fig. 21 – The initialization of the AtouchApp and the list of initialization results to the form

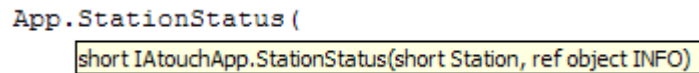
5.2.3 A sample method not causing an event

We select method “Station Status” (finding out the station’s connection status) as a sample method that does not cause an event.

The guide says that the syntax for the “StationStatus” method is as follows:

```
object.StationStatus (ByVal Station As Integer, ByRef INFO As Variant) As Integer
```

In our case, we use the object “App”. The simplest way to find out what the correct entry is the floating Help that displays automatically when we write commands.



```
App.StationStatus (
    short IAtouchApp.StationStatus(short Station, ref object INFO)
```

Fig. 22 – The floating Help

The floating Help shows that the VBA type Integer and Variant became type Short and Object in C#. That is why we define two variables (whether global or local) type Short and Object that we name e.g. “ShoStation” and “ObjInfo”.

```
short ShoStation = 1; //we are determining the status of control system number 1
object ObjInfo = null;
```

We place the button to the form from the window “Toolbox”. We name it e.g. “ShowStat”. We will initiate the method to determine the station status by pressing this button.

The method “StationStatus” also returns error code. We will enter this code to the “TextBox” control that is already created and that we named “ErrText”.

The code entered into the event of pressing the button will then look like this:

```
Err = App.StationStatus(ShoStation, ref ObjInfo);
```

After executing this order, the variable type and content change to “ObjInfo”. The guide to the AtouchX communication controller states that it turns into a one-dimensional matrix consisting of two cells.

Therefore, the following step is to verify whether this happened. In order to do that, we use a condition testing the variable type:

```
if (ObjInfo is ushort[])
```

We test the ushort[] type since we know that values we may obtain range from 0 to 65535. We write the remaining part of the code into the condition body.

First, we need to create a new variable type `ushort[]`, e.g. “UshInfo”, into which we save the variable “ObjInfo” that we need to retype using the expression `(ushort[])`.

```
ushort[] UshInfo = (ushort[])ObjInfo;
```

In order to display the content of both matrix controls in the form, we place two more controls type “TextBox”. We name them “Text1” and “Text2”. By means of their property “Text”, we will write into these two controls the matrix values converted to type String using the method “ToString”.

```
Text1.Text = UshInfo[0].ToString();  
Text2.Text = UshInfo[1].ToString();
```

The resulting code will look as in the following picture.

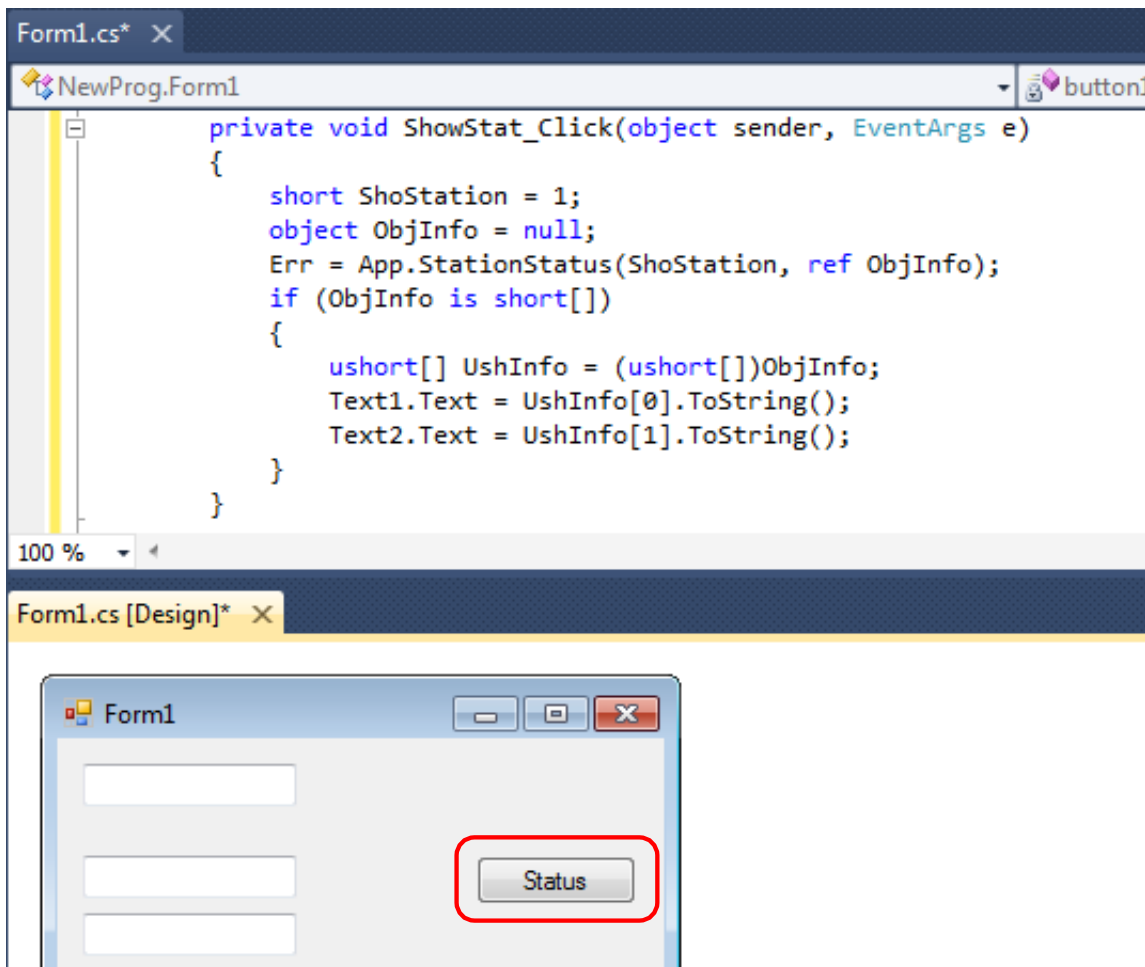


Fig. 23 – The code called in the event of pressing the button

5.2.4 A sample method causing an event

We select the method for variable reading “NetGetData” as a sample method causing an event as it causes the event “EndNetGetData”.

The method “NetGetData” syntax is as follows:

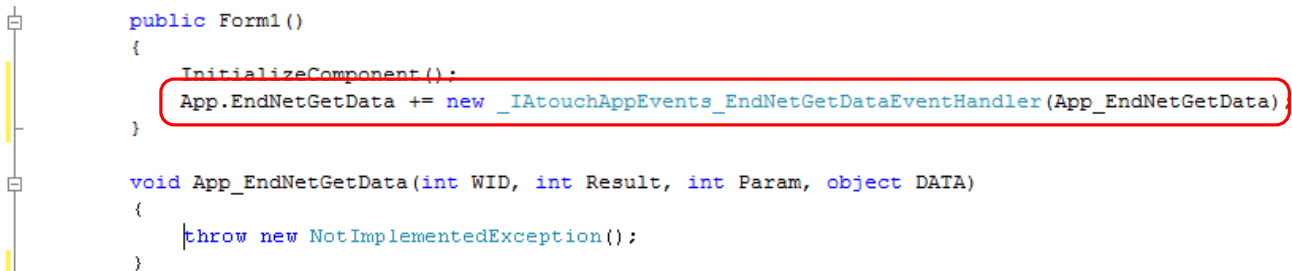
```
object.NetGetData (ByVal WID As Long, ByVal Param As Long) As Integer
```

We create the button in the form in the way we learned previously and we put the following code into this button:

```
Err = App.NetGetData(1001, 1);
ErrText.Text = Err.ToString();
```

This is how we programmed the value of the variable with WID 1001 to be read. The result of this method's call is stored in the "Err" variable, which is displayed again in the "TextBox" control called "ErrText".

It is necessary to "turn on" the event "EndNetGetData" called forth by the method "NetGetData". We achieve this by entering the event name and sign += in the part of the programme headed "public Form1()" and then push the TAB button twice. That will fill in the text for event initialization automatically and create the body of the function to be launched upon the event call.



```
public Form1()
{
    InitializeComponent();
    App.EndNetGetData += new _IAtouchAppEvents_EndNetGetDataEventHandler(App_EndNetGetData);
}

void App_EndNetGetData(int WID, int Result, int Param, object DATA)
{
    throw new NotImplementedException();
}
```

Fig. 24 – Creating an event

We delete the row beginning with "throw". In the future, we replace this row with code to be executed in case the event is called. Since we are working with an event that returns variable value, it is expected we want to keep working with this value in a certain way. If this entailed numerical or any other processing not requiring the value read to be written into the form, we write this processing directly into the prepared operational body of the event. However, if the value read has to be displayed directly on the form, simply writing

```
TextBoxX.Text = DATA.ToString();
ReadData.Text = DATA.ToString();
```

does not lead to a result. An error would always occur after running this part of code; it would be caused by the application being divided into multiple threads. One thread provides the work with the form and another thread operates the events. The aforementioned code makes the threads collide.

In order to write the value read in the form, we need to take the following steps:

1. We create a delegate definition in the place of global variables. The delegate's arguments list all variables you will continue to work with. In our case, it will be WID of the variable read, communication result, one supplementary parameter and data retrieved. Syntax for writing is as follows:

```
private delegate void (return value; there are no returns in our case)
DelEndGetData (a name of our choice) (int WID, int Result, int Param, object DATA)
(variables transferred)
```

The delegate definition in our case will look like this:

```
private delegate void DelEndGetData(int WID, int Result, int Param, object DATA);
```

2. We use the method "this.Invoke" that uses the delegate from the previous row as arguments and subsequently also a list of variables transferred.

```
this.Invoke(new DelEndGetData(App_EndNetGetData), new Object[] { WID, Result,
Param, DATA });
```


The event code is running in a different thread than the one providing the rendering window; that is why it is necessary to use the command “invoke” to run the code for work in the window into the rendering thread.

- Based on the communication result, we write the value read or the error code on the form. In case the communication went well, we find out (e.g. using the parameter “Param”) what variable we communicated and depending on the “Param” value, we write the value read into the appropriate text field.

The resulting code will look as in the following picture.

```
public partial class Form1 : Form
{
    //Delegate declaration for communication between threads.
    1. private delegate void DelEndGetData(int WID, int Result, int Param, object DATA);

    //Necessary objects and variables definition.
    AtouchApp App = new AtouchApp();
    int Err;
    Object Data = new Object();

    public Form1()
    {
        InitializeComponent();
        //After-communication event handler initialization. When App.xxx += is written and then
        //the Tab key is double pressed, the event code block is created automatically.
        App.EndNetGetData += new _IAtouchAppEvents_EndNetGetDataEventHandler(App_EndNetGetData);
    }

    void App_EndNetGetData(int WID, int Result, int Param, object DATA)
    {
        2. if (this.InvokeRequired)
        {
            //The code is running in a different thread than in the UI thread. Therefore it is necessary
            //to use the Invoke method for thread-safe call on form controls.
            this.Invoke(new DelEndGetData(App_EndNetGetData), new Object[] { WID, Result, Param, DATA });
            return;
        }
        //The code is running in the UI thread here and it is possible to call on form controls.
        //Call on a form control after a read request execution.
        3. if ((Result & (Int16)AtouchX_Communication_State.atfOk) != 0)
        {
            //Communication ended OK.
            ComText.Text = "OK";
            switch (Param)
            {
                case 1:
                    ReadInt.Text = DATA.ToString();
                    break;
                case 2:
                    ReadLon.Text = DATA.ToString();
                    break;
                case 3:
                    ReadFlo.Text = DATA.ToString();
                    break;
            }
        }
        else
        {
            //Communication ended with error.
            ComText.Text = Result.ToString();
        }
    }
}
```

Fig. 25 – Three steps to write the value into the form

5.2.5 Communication termination

Same as we initialized the connection to the DB-Net network (DB-Net/IP), we also have to end this connection properly. In order to end the connection to the DB-Net network (DB-Net/IP), we use the method “Done” with the following syntax:

```
object.Done () As Integer
```

We create a button and name it “End”. We insert the following code into the event caused by pressing this button:

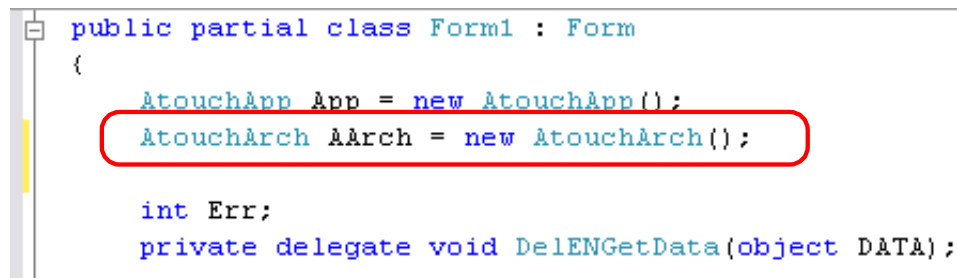
```
App.Done() ;  
this.Close() ;
```

This terminates the communication.

5.2.6 Working with archives

In order to work with archives, we need to create an instance of the object “AtouchArch”. We create it in the same way we created the instance of the object “AtouchApp” in chapter 5.2.1 “AtouchApp definition”.

```
AtouchArch AArch = new AtouchArch() ;
```



```
public partial class Form1 : Form  
{  
    AtouchApp App = new AtouchApp();  
    AtouchArch AArch = new AtouchArch();  
  
    int Err;  
    private delegate void DelENGetData(object DATA);
```

Fig. 26 – Creating an instance of the AtouchArch object

Caution!

In order to work properly, the object requires that an object providing connection to DB-Net network (DB-Net/IP) exists and works along with it.

Archive initialization

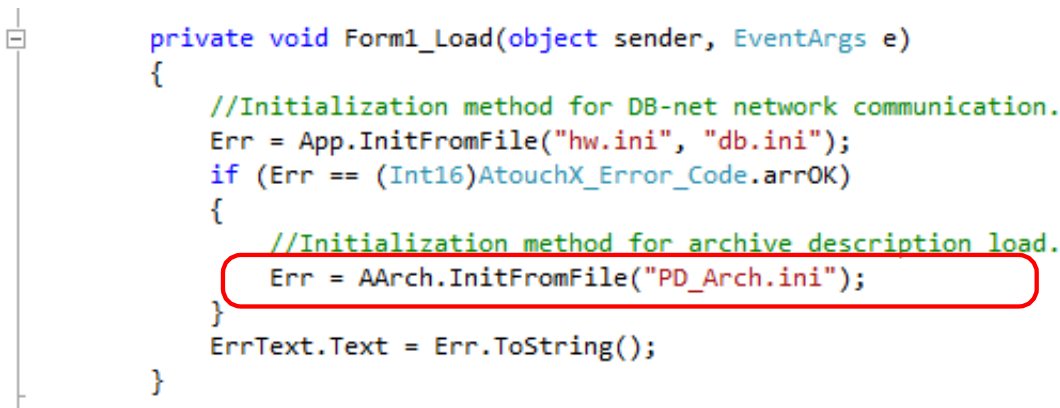
For the actual archive initialization, we use e.g. the method “InitFromFile”. This method performs the archive initialization by means of an external file. In our case, we name it PD_Arch.ini (the structure is described in the Help section of the AtouchX communication control). The method also returns an error code. We save this code into the global variable “Err” type Integer.

```
Err = AArch.InitFromFile("pd_arch.ini");
```

We display the value of variable “Err” in the “TextBox” control named “ArcErr” that we placed on the form.

```
ArcErr.Text = Err.ToString();
```

The resulting archive initialization code will then look like this:



```
private void Form1_Load(object sender, EventArgs e)
{
    //Initialization method for DB-net network communication.
    Err = App.InitFromFile("hw.ini", "db.ini");
    if (Err == (Int16)AtouchX_Error_Code.arOK)
    {
        //Initialization method for archive description load.
        Err = AArch.InitFromFile("PD_Arch.ini");
    }
    ErrText.Text = Err.ToString();
}
```

Fig. 27 – Archive initialization

This object needs to have events “Completed” and “Sample” defined in order to work to the full extent. Therefore, we write the code “ObjectName.EventName +=” into the part of the programme “public Form1()” and press the TAB button twice.



```
public Form1()
{
    InitializeComponent();
    AArch.Sample += new _IAtouchArchEvents_SampleEventHandler(AArch_Sample);
    AArch.Completed += new _IAtouchArchEvents_CompletedEventHandler(AArch_Completed);
}
```

Fig. 28 – Defining events for AtouchArch

We later put the method “Accept” into the “Sample” event code so that the sample gets verified/rejected after accepting the value and another sample could be accepted. We later put the method “Control” into the “Completed” event code in order to determine the archive termination.

Enabling archive function

In our application, we will be using an automatic archive (for more information see Help section on the AtouchX communication controller).

We will use the “Control” method to enable the archive function. The method syntax is as follows:

```
object.Control (ByVal AID As Integer, ByVal Run As Boolean) As Integer
```

We insert the button on the form and we put the following code into the event of pressing this button:

```
Err = AArch.Control(0, true); //Enabling Archive Function
ArcErr.Text = Err.ToString(); //Displaying the code that the method returns
```


By pressing the button, we enable the archive function with AID 0 (see Help section for the AtouchX communication controller).

Saving an archive sample

We use the event “Sample” to save archive samples. We cause the event if one sample of an automatic archive is available. We have already defined this event in subchapter “Archive initialization” and it is necessary to also use the method “Accept” in it in order to verify or reject the sample. This method is described in the Help section on the AtouchX communication controller.

```
AArch.Accept(0, true); //Accepting a sample from the archive AID = 0 (See AtouchX Help)
```

The resulting code of the entire event will look as in the following picture.

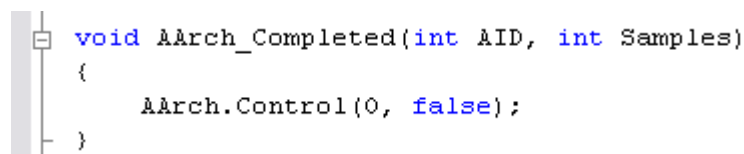


```
void AArch_Sample(int AID, object DATA)
{
    if (DATA is object[])
    {
        //New data processing
    }
    AArch.Accept(0, true);
}
```

Fig. 29 – Confirming the acceptance of a new sample

Terminating archive function

We can perform the archive termination in the event “Completed” that is called in case the archive does not contain any new samples to be read. We have already defined this event in subchapter “Archive initialization”. After terminating the archive function, we use the aforementioned method “Control”. The resulting code of the entire event will look as in the following picture.



```
void AArch_Completed(int AID, int Samples)
{
    AArch.Control(0, false);
}
```

Fig. 30 – Terminating the archive run

Termination of the AtouchArch object activity

Same as we initialized the archive, we also have to terminate it properly. In order to do so, we use the method “Done” with the following syntax:

```
object.Done () As Integer
```

We enter the AtouchArch object termination e.g. into the event of pressing the same button as in case of termination of the connection to the DB-Net network (DB-Net/IP). The resulting code we enter into the event will be as follows:

```
AArch.Done(); // terminating AtouchArch activity
App.Done();   // terminating AtouchApp activity
this.Close(); // closing the form
```

6 APPENDIX A

6.1 Conversion of variable types in C#

This issue may occur e.g. When using methods “AtouchApp” that write values (of variables, time) from PC to control systems. The guide states that some methods (e.g. “NetPutData” and “NetPutTime”) send values according to VBA in variables type Variant. In C# the equivalent type is Object.

In order for a control system to be able to process the input data and communication transmission did not end up in an error, the value in this object variable has to be suitably converted. This is best done with the “Convert” method.

Type Integer in the control system is of 16 bits, therefore we use conversion as follows:

```
Convert.ToInt16(source_variable) .
```

Similarly, we convert the type Long by means of:

```
Convert.ToInt32(source_variable) .
```

Type Float in the control system has an equivalent in C# – type Single:

```
Convert.ToSingle(source_variable) .
```

Example

We have the “TextBox1” control on the form and a variable “Data” type Object in the programme. Our goal is to read the number from the “TextBox1” control and write this value in the control system into the variable type Integer with WID 1001. Therefore, we use the following entry:

```
Data = Convert.ToInt16(TextBox1.Text) ;  
Err = App.NetPutData(1001, 1, Data) ;
```

Another type requiring conversion is the time format. It is type DateTime both in VBA and in C#. If we want to read time from a control on the form where it is displayed in String type, we use the “Convert” method again, this time with function “ToDateTime(source_variable)”.

Example

We want to synchronize time in the control system and in the PC. The time synchronization between the control system and the PC takes place if we write the time “1.1.1980 0:00:00” into the control system.

Solution:

```
DateTime DTCas = Convert.ToDateTime(TextBox1.Text) ;  
Err = App.NetPutTime(1, 1, DTCas) ;
```

Where the “TextBox1” control will host the string 1.1.1980 0:00:00.

7 APPENDIX B

7.1 AtouchX in Delphi

The procedure of running AtouchX in Delphi Standard version 5.00 (Build 5.62).

First, we need to “teach” Delphi to recognize the object “AtouchApp” specifically. We achieve that using the command **Project / ImportTypeLibrary**. A window pops up that includes a list of all ActiveX libraries registered in its upper half. We may also add unregistered libraries by means of the button **Add**. If we installed the AtouchX library properly, this list will feature the item “AMiT AtouchX Library 1.0” and we select it. In the lower part of the window, we leave the selected item “Generate Component Wrapper” and close the window by clicking the button **CreateUnit**. Doing leads Delphi to create the file ATOUCHX_TLB.PAS in the “Imports” subdirectory and the file “encapsulates” all ActiveX objects into Delphi objects.

However, a small problem occurs during generation. AtouchX library objects use parameters with names “Type”, “Set” and “Result”. “Type” and “Set” are Delphi keywords and Delphi supplies them with an underscore on the front during conversion. However, “Result” is not a keyword, so Delphi leaves it unchanged. But the name “Result” is used to mark function return values and the names conflict. It is necessary to remove the problem manually. The modified file ATOUCHX_TLB.PAS is a part of the file ap0013_p16_en_xx.zip. Copying the file into the “Imports” subdirectory should teach Delphi to recognize objects from AtouchX library. Beware – it is a generated file that generates again and again with repeated imports or renewals, which removes the changes made manually (“Result” => “_Result”)

After the import, the object type “TAtouchApp” is available. If we create its instance, we can call its services but we are not able to catch its events yet, despite the fact that the object has everything ready to use them. The object contains private variables with names type FOnXXX (e.g. “FOnEndReqIdentify”) that are supposed to contain indicators to functions called upon the event arrival. According to definitions, the variables are type “object functions” (see for example the declaration type “TAtouchEndReqIdentify”) so we need to perform the following to be able to continue:

We create a descendant object “TAtouchApp”. In this object, we define the function with a prototype corresponding to the event function and we provide that the function is matched into the variable FOnXXX during the object creation. If we write our code into the given function, it will get called upon the event.

To illustrate this situation, the file ap0013_p16_en_xx.zip also features the source code with the connection verified (see file UNIT1.PAS).

We define the object “TMyAtouchApp” as the descendant of “TAtouchApp”. We re-define the “Create” constructor and define a new function “ENI” (as in “EndNetIdentify”). The “Create” constructor calls the ancestor’s constructor and then only matches the “event function” into the “event variable”. The “ENI” function then processes the incoming event – apart from a short beep, it also checks whether the event announces a successful identification reading – and if it does, the data retrieved is displayed. When creating the main form, we create the object “ATC” type “TMyAtouchApp” and initialize it (we use complete paths that we have to rewrite according to our location) by means of the testing NULL connection. When cancelling the form, we cancel the object “ATC”.

A sample application for Delphi including other useful information is included in the file ap0013_p16_en_xx.zip that is included in Appendices to this Application note.

8 Technical support

All information on using the AtouchX communication controller will be provided by the technical support department of the company AMiT. Do not hesitate to contact the technical support via e-mail using the following address: **support@amit.cz**.

9 Warning

AMiT spol. s r. o. does not provide any warranty concerning the contents of this publication and reserves the right to change the documentation without any obligation to inform about it.

This document can be copied and redistributed under the following conditions:

1. The whole text (all pages) must be copied without making any modifications.
2. All redistributed copies must retain the AMiT, spol. s r. o. copyright notice and any other notices contained in the documentation.
3. This document must not be distributed for profit.

The names of products and companies used herein may be trademarks or registered trademarks of their respective owners.