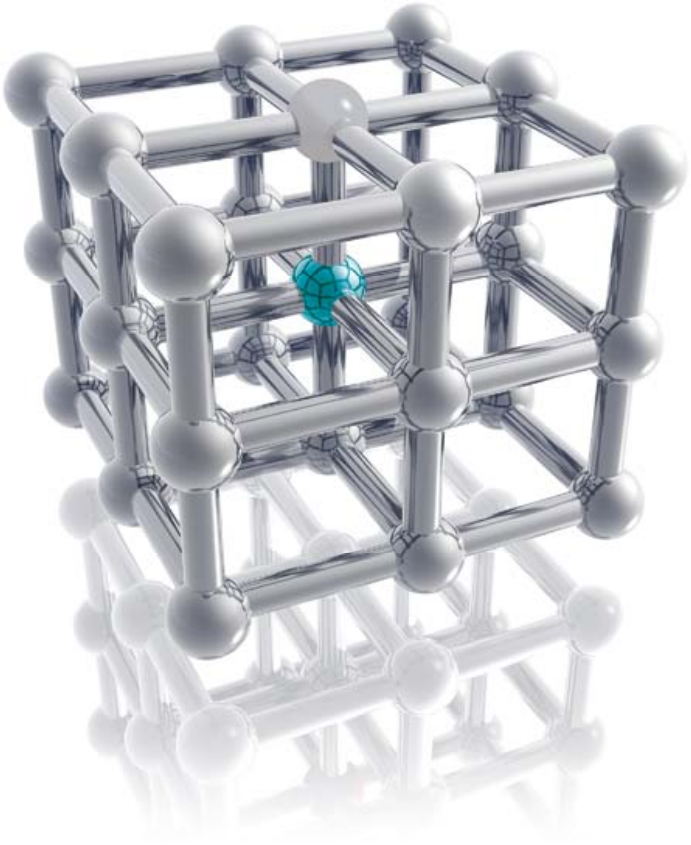
see and get more ...



AsixConnect5 – User's Manual

Doc. No. ENP5065
Version: 28-10-2007

ASKOM Sp. z o. o., ul. Józefa Sowińskiego 13, 44-121 Gliwice,
tel. +48 (0) 32 3018100, fax +48 (0) 32 3018101,
http://www.askom.com.pl, e-mail: office@askom.com.pl

1. Introduction

The AsixConnect is a package of servers that extends scope of applications of **asix** package such as monitoring and computer supervision of industrial processes.

Computer provided with physical link to controller (with use of serial line, industrial network or local area network) is a source of information on process variables for other computers in the local area network. Such a computer is a server of current and archive data. The access to that data is enabled with intermediation of AsixConnect package.

AsixConnect package includes *OPC*, *Automation*, *.NET* and *DDE* servers that enable access to current process variables in the database of **asix** applications as well as *Automation*, *.NET* and *OLE DB* servers enabling access to archive values of process variables and *.NET* server providing data on alarms. In addition, the package includes *Web Service* server providing any types of data in **asix** system applications in the *XML Web Services* standard.

Each program of Windows environment provided with *Automation*, *OPC*, *.NET* or *DDE* mechanism can cooperate with application of **asix** program with intermediation of AsixConnect package servers. Such a program may be both the client of data from industrial process and the source of data for upper level control or configuration. In other words, with that method in Windows environment, the current values of process variables are accessible on-line as well as their historical values, i.e. registered trends. Example applications provided with Automation and DDE mechanisms of data exchange are components of Microsoft Office package – Excel and Access. The applications developed using these products and AsixConnect package may effectively enrich the computer supervisory systems. These applications may be used for data analyses and presentations, model tests, specialist reporting or creation of process databases.

AsixConnect is an integral part of **asix** package but it is also accessible as a separate product for application on PC stations connected to the local area networks and having access to data servers provided with **asix** packages. In this case AsixConnect makes access in Windows environment to the data imported from remote computer stands provided with links to process controllers.

AsixConnect is the main element of open system strategy of **asix** package in its applications in Windows 2003, XP, 2000 and NT4 environments.

In the following chapters of this manual, the installation of AsixConnect program and methods of access to process variables are described. The user of this manual should have the basic knowledge of *Automation*, *OLE DB*, *OPC*, *DDE, NET* and *XML Web Services* mechanisms.

1.1. Package Components

AsixConnect package is distributed as a component of **asix** system or as an independent package. AsixConnect includes the following servers:

- current data servers: *.NET, Automation, OPC, DDE,*

- archive data servers: *.NET, Automation*, *OLE DB*,
- variable base servers: *.NET, Automation*,
- alarm servers: *.NET*.
- *Web Service* server,
- DDE service (installed as an option).

To run AsixConnect package servers *HASP asix* or *HASP AsixConnect* key is required. In both cases in the key, the flag *Version 5* has to be activated.

1.2. Licensing

Licensing of AsixConnect package is discussed in the trade documentation.

1.3. Requirements of asix System

When AsixConnect package servers operate with **asix** system working on the same computer, all types of **asix** system licences are supported. When **asix** system works on another computer, all versions of asix system licences are supported (including the former versions working under DOS operating system), but the required type of **asix** system licence is *operator server* (symbol *WAxS*).

1.4. Most Important Modifications in Package

Version 4.0

- Extension of package with *.NET* servers of current, archive and alarm data as well as Variable Definitions Database.
- Extension of package with *Web Service* server.
- Making it possible to define channels, i.e. independent sets of parameters for connections with **asix** system servers. Extension of package with external program for configuration of channel parameters.
- Change in the system names of Automation servers, maintenance of operating compatibility with version 3.

Version 5.0

- Support for variable definitions database.
- New statuses of archival data.

2. Installation

2.1. Autonomous Installation of AsixConnect Package

Installation package of AsixConnect includes the following components:

- user manual,
- HASP protection key,
- software CD ROM.

Start *AsixConnect_5.EXE* program from CD ROM and proceed with indications that will appear. After installation, you will find included in target directory (default *c:\asix*):

- files of servers included in AsixConnect package;
- subdirectory ***AC4Examples*** containing examples of using AsixConnect package; the examples are written to spreadsheets of Excel 2000;
- subdirectory *Applications\Acid_Manufacturer* containing the application of **asix** system and Variable Definitions Database; data from that application are used by examples of the application of AsixConnect package;
- subdirectory *WebService* contains the *WebService* server files.

The AsixConnect package installation program does not support installation on a computer on which **asix** in version 3 is installed. Installation enabling simultaneous operation of AsixConnect 5 and **asix** 3 packages can be carried out by the Askom employee only.

2.2. Installation as a Part of asix Package

As a part of **asix** package, the user receives AsixConnect package and can use both these packages simultaneously on the same computer.

In order to install AsixConnect package together with **asix** package, start the program named *ASIX_PL_5.EXE* from the **asix** package installation CD and proceed with indications that are displayed. The default installation directory is c*:\asix*.

3. Connection Configuration

3.1. Channels

The channel is a set of configuration options of AsixConnect package servers, options of **asix** system's connections with the server and options of Variable Definitions Database.

The channel named '*' is the basic channel of servers of AsixConnect package. It is created during the package installation and cannot be removed. The options of this channel are used when the channel name will be:

- *;
- Null text;
- Text starting with *ANY.*

If a channel name defined using the *Configurator* program will be used, the options of this channel are used.

If a channel name starting with *NEW* is used*, no* configuration options will be sent to the servers and the client should send his own set of options using the mechanisms that are appropriate to this server.

The use of undefined channel name other than the one specified above is an error.

3.2. How to Specify the Channel Name

The channel names are specified in individual servers of AsixConnect package in the following way.

Table 1. Methods of Specyfying the Channel Name.

Server Type	Method of Specifying the Channel Name
Automation servers	Servers make *LoadChannel* function available. The channel name is the function parameter.
DDE server	The channel name is *topic* of DDE connection.
OPC server	The channel name is variable *access path.*
OLE DB server	The channel name should be transferred in *Data Source* parameter.
.NET serves	The channel name is a parameter of konstruktorów *.NET* object designers.
.NET servers designed in ASP.NET environment by *SessionServer* function	The channel name is received from *Web.Config* file – see below.
Web Service server	The channel name is received from *Web.Config*.file.

For storage of default channel name, the ASP.NET applications use the configuration file named *Web.Config*. This file is located in the application directory. In order to define the default channel name, the user should create the *appSettings* element in the *configuration* superior element. Then, one *add* element should be created in the *appSettings* element and two attributes should be defined in it. The first attribute should be named *key* and assigned the value *"DefaultChannelName".* The latter attribute should be named *value* and its value should be the channel name. The channel name should be put in quotation marks.

EXAMPLE

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="DefaultChannelName" value="AsEmis" />
  </appSettings>
…
```

3.3. Configuration File

The information on defined channels is stored in the *ASIXConnect.ini* file, in the directory the AsixConnect package is installed in.

One should remember to make a copy of this file before installation of a new version of **asix** or AsixConnect package, as it will be overwritten during the installation.

3.4. Interactive Configuration

3.4.1. Configurator Program

In order to configure the channel options, the *Configurator* program should be started. When started, this program displays the main window. (*See: Figure 1*).

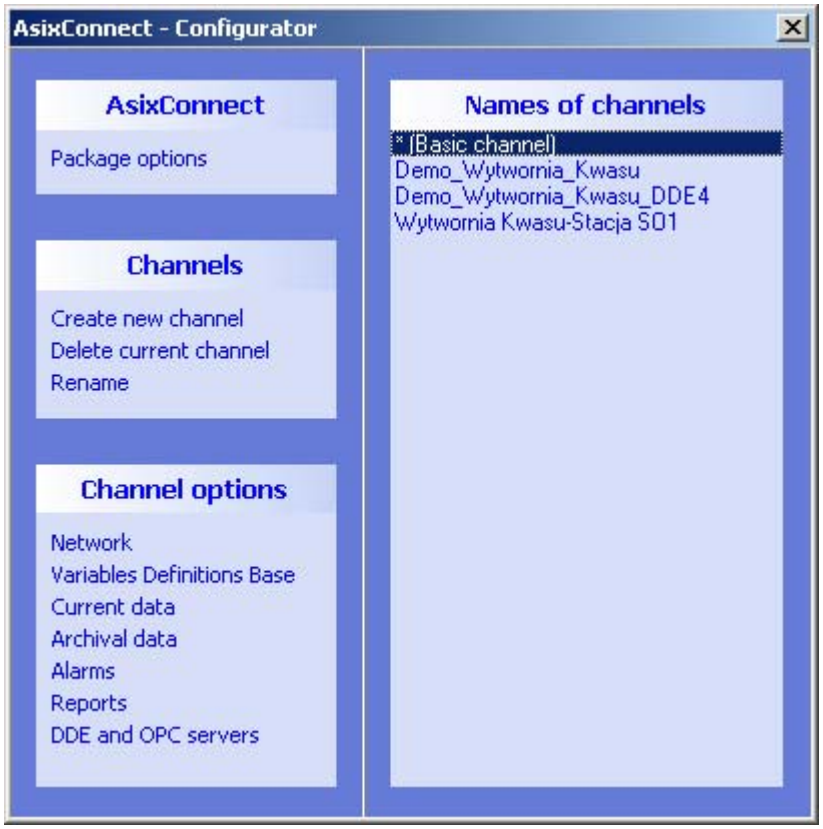


Figure 1. The 'Configurator' Window.

In the left-hand section of the window, the list of available operations is displayed. In the right-hand section of the window, the list of defined channels is displayed. The channel named '*' exists all the time.

3.4.2. Channel Management

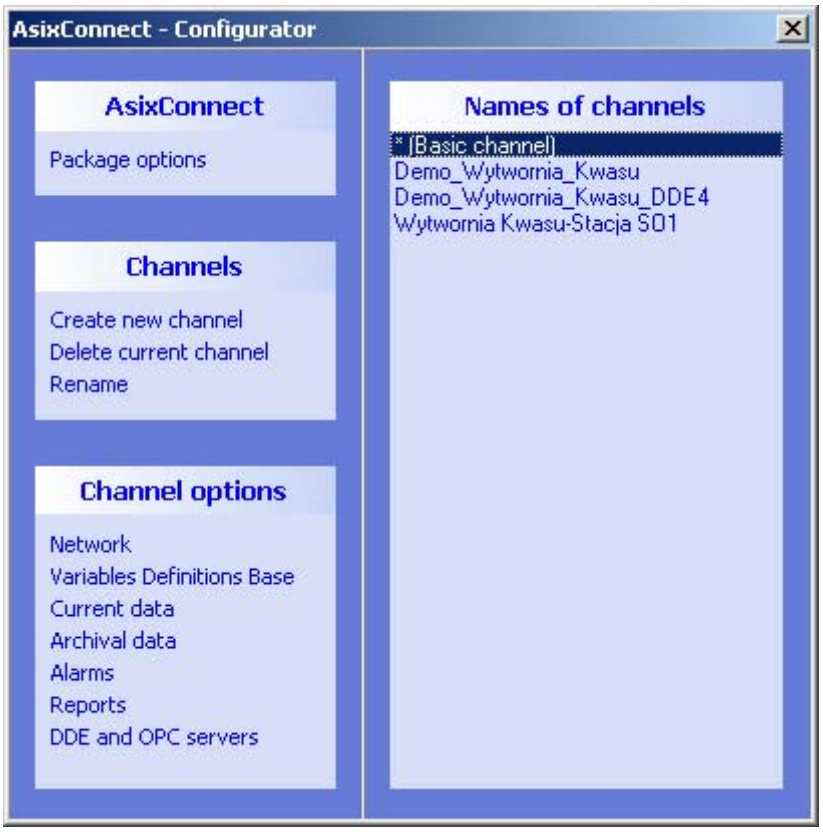


Figure 2. The 'Configurator' Window.

The command *Create new channel* enables creation of a new channel. The command *Delete current channel* enables the currently highlighted channel to be removed. The command *Rename* enables the name of the currently highlighted channel to be changed. The name of the basic channel cannot be removed or changed.

3.4.3. Package Options

The options in the group *Package options* relate to every server of the AsixConnect package.

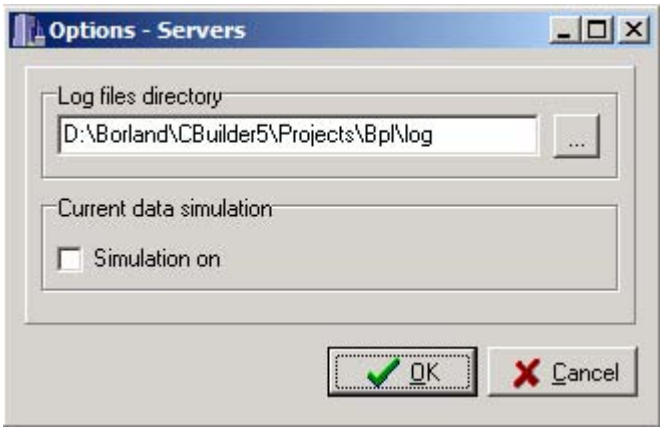


Figure 3. The 'Configurator' Window - Package Options.

The option *Log files directory* defines location of the server log files. By default, this option is not defined and log files are placed in the directory the AsixConnect package is installed in.

The options of this group are modifiable on an interactive basis only.

3.5. Program Configuration

Most servers in the AsixConnect package provide the ability of program setting of their options. For setting the options a relevant function is called and transferred as a text parameter in the following form:

```
Option1=Value1;Option2=Value2; ...
```

The number of *Option=Value* components is free. The components are separated with semicolons.

The options are set in individual servers of AsixConnect package in the way in *Table 2*.

Table 2. Method of Setting the Options - in Individual Servers of AsixConnect Package.

Server Type	**Method of Setting the Options**
Automation servers	The *Init* function call.
.NET servers	
DDE server	The *XTYP_POKE* transaction call (write to DDE server). As *item* parameter you should pass *InitString*. As *data* parameter you should pass initialization text. Programmatic option modification during a given connection may have influence on options used in this connection.
OPC server	Servers may used options of default channel only.
OLE DB *server*	
Web Service server	

The required values of options may be text, number or logical value. For logical-type options, the logical value of *True* may be *yes*, *true* or *1,* while the logical value of *False* may be *no*, *false* or *0*.

The sequence of options within the parameter makes any difference for sequence of their interpretation. If an option with incorrect value is encountered, the initialisation process is interrupted and error message is returned. Unknown options are ignored.

Options can be modified by program until the first operation on data is performed, i.e. readout, write or variable activation.

3.6. Channel Options

3.6.1. asix System Network

Interactive Configuration

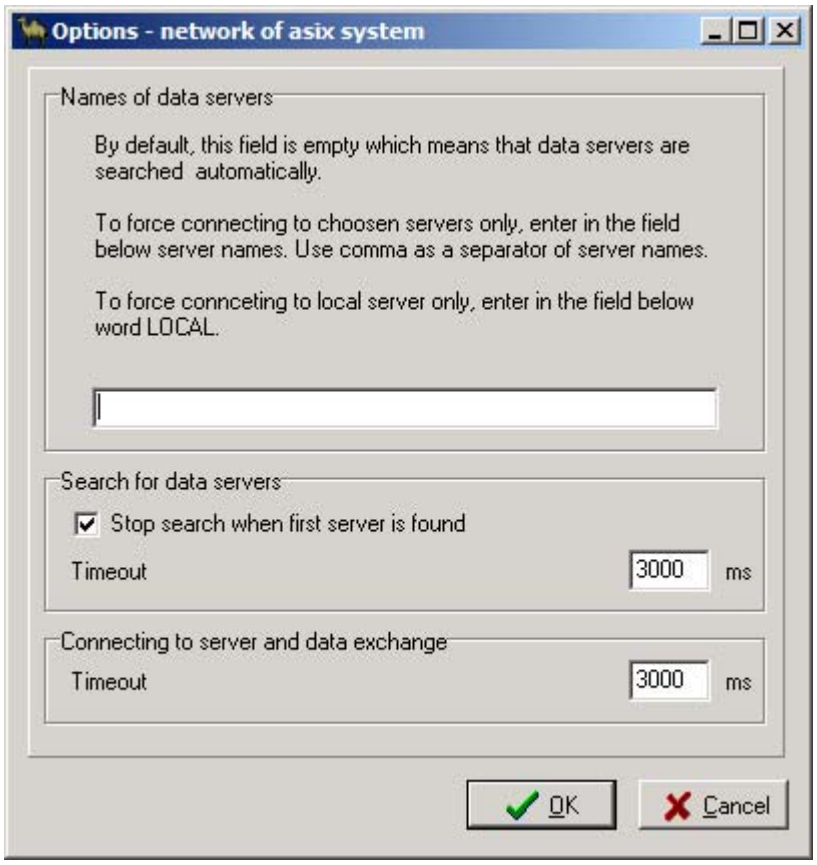


Figure 4 . The 'Configurator' Window - Network.

The option *Name of asix system data server* enables the name or names of **asix** system servers to be defined. Data will only be downloaded from the listed servers. The server names should be separated with commas. By default, this option has no defined value, which means that any sever may be selected from among the found ones. In order to enforce the local server is attached to only, the server name should be defined as *LOCAL*.

The **asix** system, which data are to be taken from via the local area network, must be of *operator server* type (with *WAxS* symbol).

The client's computer name is defined in the way described in *Table 3*.

Table 3. Client Computer Name in asix System.

Description of Client Software Configuration	Client Computer Name in asix System
Only AsixConnect package is used and *aslink.ini* file was not created or *Name* line was not defined in *ASLINK section.*	Name of client computer in WINDOWS plus period (.) at the end, e.g. for computer named „CLI" its name in **asix** system is „CLI."
Only AsixConnect package is used and ASIX.INI file was created, *Name* line was defined in *ASLINK section*.	Name of client computer in **asix** system is provided in ASIX.INI file in ASLINK section, in line titled *Name***.**
AsixConnect and **asix** application run in the same computer.	Name of client computer in **asix** system is provided in INI file of application, in ASLINK section, in line titled *Name***.**

The other options are of advanced-type and usually they do not need modification. Their meaning is described in *3.6.1.1 Searching for data servers of asix system***.**

Program Configuration

Table 4. asix System Network - Program Configuration.

Option	Option Name for Interactive Configuration	Type	Default Value
AsixServerName	*Data server names of asix system*	String	Null (empty string)
StopAfterFirstServerFound	*Stop search when first server is found*	Boolean	Yes
FindServerTimeout	*Search for data servers/Timeout*	Integer	3000 [ms]
NetTimeout	*Connecting to server and data exchange/Timeout*	Integer	3000 [ms]

3.6.1.1. Searching for Data Servers of asix System

All the current, archive and alarm data servers of AsixConnect package use the same algorithm of searching for data servers of **asix** system application.

In the first step, the servers of AsixConnect package send a command to call every **asix** system server, which is attached to the local computer network and makes the specific resource (channel, archive or alarm server) available, to introduce. This query also regards the local server of **asix** system. If computer, which the AsixConnect package is installed

on, is not connected to the computer network, the query regards the local server of **asix** system only. The time of waiting for response is affected by the *StopAfterFirstServerFound* and *FindServerTimeout* options.

In the second step, one server is chosen from among the servers of **asix** system that answered the query. Selection of the server is affected by the *AsixServerName* option.

For details of the searching algorithm variants, see the table below.

Table 5. Algorithm of Searching for asix's System Servers.

Item	Value of *AsixServerName* Option	Server Searching Algorithm
1.	no option (empty string) *****	If an external list of servers was used, then the third variant of server searching algorithm.
		StopAfterFirstServerFound = yes First responding server will be selected. If connection with the server breaks, it is possible that new connection with other server allowing access to the same process data will be established.
		StopAfterFirstServerFound = no After time equal to *FindServerTimeout* one of responding servers, according to following preferences (ordered by importance) is selected: 1. Whether data source is not failed? 2. Whether server is a local one? 3. Whether server allows access to its own data (is not a gateway)? 4. Whether it has the smallest number of clients? If connection with the server breaks, it is possible that new connection with other server allowing access to the same process data will be established.
2.	*server_name*	AsixConnect package will be waiting for responding the server of declared name for the time equal to *FindServerTimeout* maximum. If connection with the server breaks, the new connection may be established with the same server only.
3.	*server_name1, server_name2...* (list of server names, separated with semicolons)	Server of AsixConnect package will be waiting for responding any server from declared list of names for the time equal to *FindServerTimeout* maximum. If connection with the server breaks, the new connection may be established with any server from declared list of names only.
4.	**LOCAL**	Server of AsixConnect package will be waiting for responding the local server for the time equal to *FindServerTimeout* maximum. If connection with the server breaks, the new connection may be established with the local server.

The default values of the options used when searching for data servers of **asix** system are as follows.

Table 6. Default Values of Options for Searching for Data Servers of the asix System.

Option Name	Default Value
StopAfterFirstServerFound	Yes
FindServerTimeout	3000 [ms] (3 seconds)
AsixServerName	no value (empty string)

3.6.1.2. External List of Servers

The external list of servers allows the list of admissible data servers to be specified individually for every resource of **asix** system application. The external list of servers is used only if the name of the server, which the data are to be taken from, is not provided openly in the channel configuration. The list of servers is shared by all the channels.

The external list of servers is stored in the *ASIXConnect.ini* file, in the directory the AsixConnect package is installed in. For every type of server, the section should be defined; every line in the section should have the following form:

```
resource_name = server1_name, server2_name...
```

For names of the *ASIXConnect.ini* file sections corresponding to individual types of servers and meaning of the *resource_name* element, see the following table.

Table 7. Names of the ASIXConnect.ini File Sections for Individual Types of Servers.

Server Type	Section Name	Meaning of *resource_name* Element
Current data server	*AsixCTServers*	Channel name
Archive data server	*AsixHTServers*	Archive name
Alarm server	*AsixALServers*	Network name of alarm server

The server names in the list are separated with a comma.

3.6.2. Variable Definitions Database

Interactive Configuration

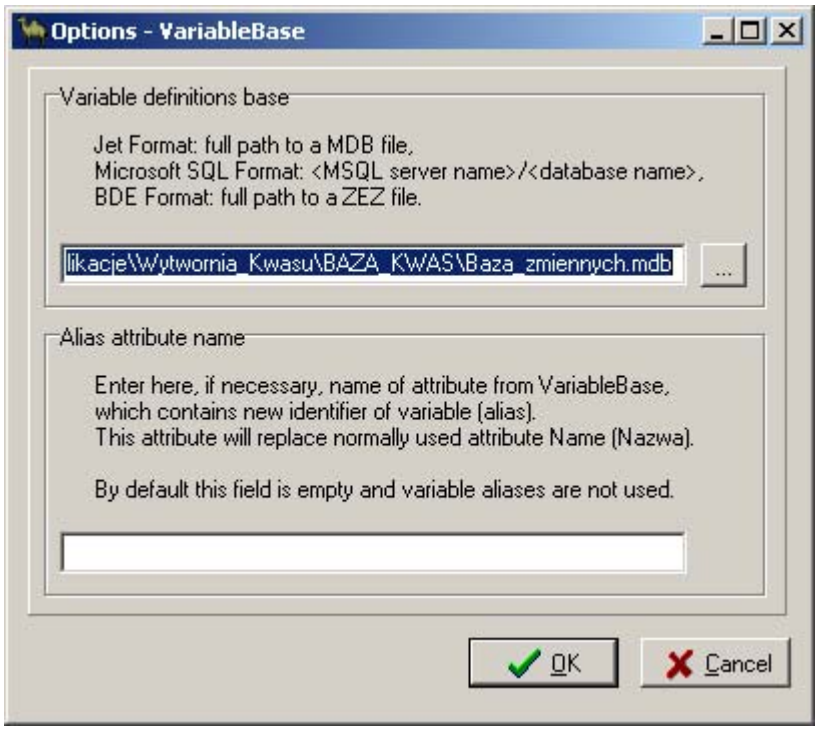


Figure 5. The 'Configurator' Window - Variable Definitions Database.

The option *Variable definitions base* is used for entering information on location of the Variable Definitions Database containing definitions of variables of the **asix** system application.

The option *Alias attribute name* allows alternative names of variables to be used in clients of the servers of AsixConnect package. The alternative names of variables must be stored in the Variable Definitions Database as values of certain variable attribute; the name of this attribute must be defined as the option value. If the option is used, the variables the alias attribute of which is not defined, will be unavailable.

The alternative names of variables are also used by OPC server when reviewing its Variable Definitions Database using OPC mechanisms. However, one should remember that for review of the Variable Definitions Database, the base defined in the basic channel '*' is used only.

The alternative variable names are not supported by .NET servers.

Program Configuration

Table 8. Variable Definitions Database - Program Configuration.

Option	Option Name for Interactive Configuration	Type	Default Value
ItemsDatabase	*File name of set of variables*	String	null
AliasAttributeName	*Alias attribute name*	String	null

3.6.3. Current Data

Interactive Configuration

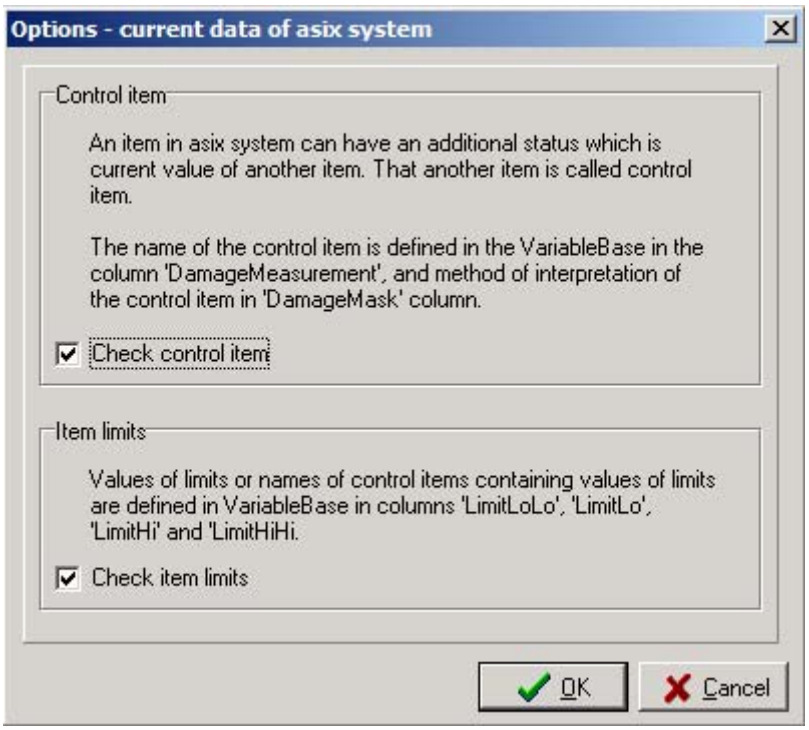


Figure 6. The 'Configurator' Window – Current Data.

The options *Check control item* and *Check item limits* allow verification of control variables and verification of the limits of variables, similarly to verification carried out by the *NUMBER* and *BAR* objects in **asix** system, to be enabled. By default, the options are off. When the channel is used by dynamic HTML pages, these options must be checked.

To enable verification, the relevant check boxes should be checked. In order for verification to be carried out, the Variable Definitions Database must contain the appropriate attributes in which testing algorithms will be defined. It regards the same attributes that are used in configuration of the NUMBER and BAR objects in the Variable

Definitions Database. For information about the attributes, see **asix** system documentation (asix.hlp).

Program Configuration

Table 9. Current Data - Program Configuration.

Option	Meaning	Type	Default Value
CheckControlVariables	*Check control item*	Boolean	No
CheckLimits	*Check item limits*	Boolean	No

3.6.4. Archive data

Interactive Configuration

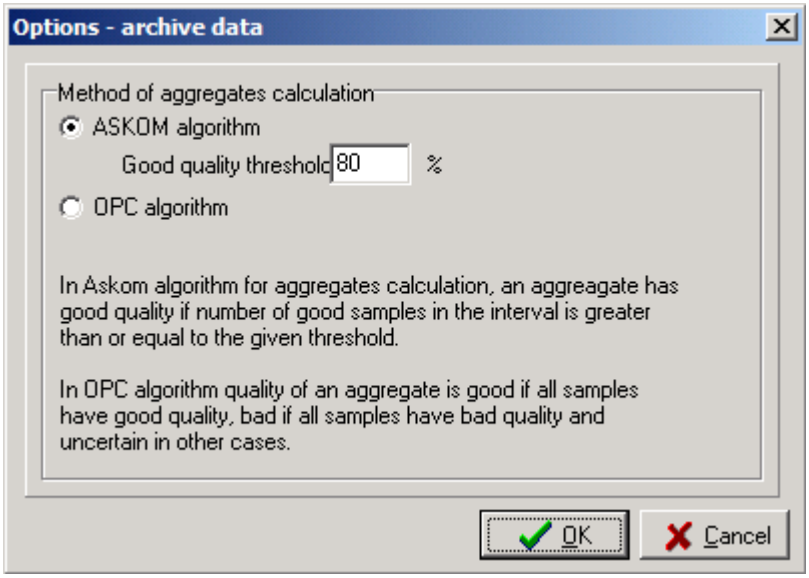


Figure 7. The 'Configurator' Window - Archival Data.

The option *Method of aggregates calculation* allows the required type of algorithm to be defined. Algorithms are described in *7.3.1 Description of Aggregates*.

Program Configuration

Table 10. Archive Data - Program Configuration.

Option	Meaning	Type	Default Value
AggregateMode	*Method of aggregates calculation*	Integer	0
QualityGoodThreshold	*OPC algotithm*	Integer	80

3.6.5. Alarms

Interactive Configuration

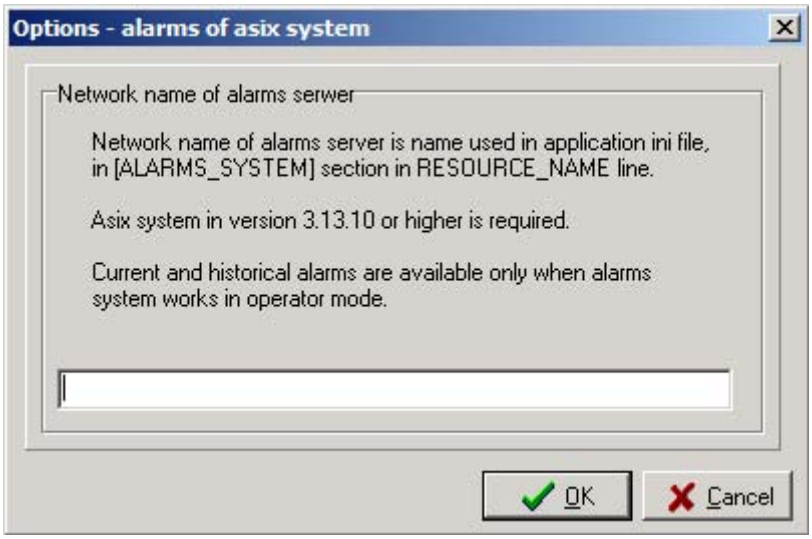


Figure 8. The 'Configurator' Window - Alarms.

If the client uses the alarm server of the AsixConnect package, it is necessary to specify the value of the *Network name of alarms server* option. The network name of the alarm server is defined in the *ini* file of the application, in the *ALARM_SYSTEM* section, in *RESOURCE_NETWORK_NAME* line. The alarms are available only when the network name is defined in the application and the alarm system works in the operator mode.

Program Configuration

Table 11. Alarms - Program Configuration.

Option	Meaning	Type	Default Value
AlarmsSystemNetworkName	*Network name of alarms server*	String	null (empty string)

3.6.6. Reports

Interactive Configuration

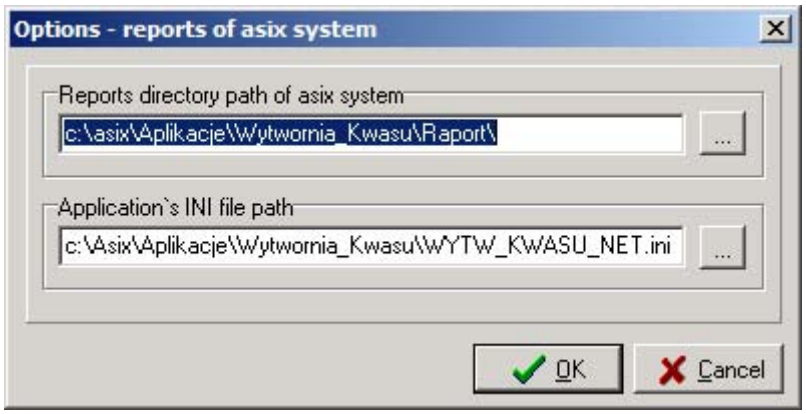


Figure 9. The 'Configurator' Window - Reports.

The *Reports direcotory path of asix system* option allows to enter the information on directory the report definition files and **asix**'s application reports are stored in.

In the *application's INI file path* field the full path to an **asix**'s application INI file should be passed.

Program Configuration

Table 12. Reports - Program Configuration.

Option	Meaning	Type	Default Value
ReportsDirectory	Path to the directory of **asix** system reports	Text	Null
IniFilePath	Path to the INI file of an **asix** system application	Text	Null

3.6.7. DDE and OPC Servers

Interactive Configuration

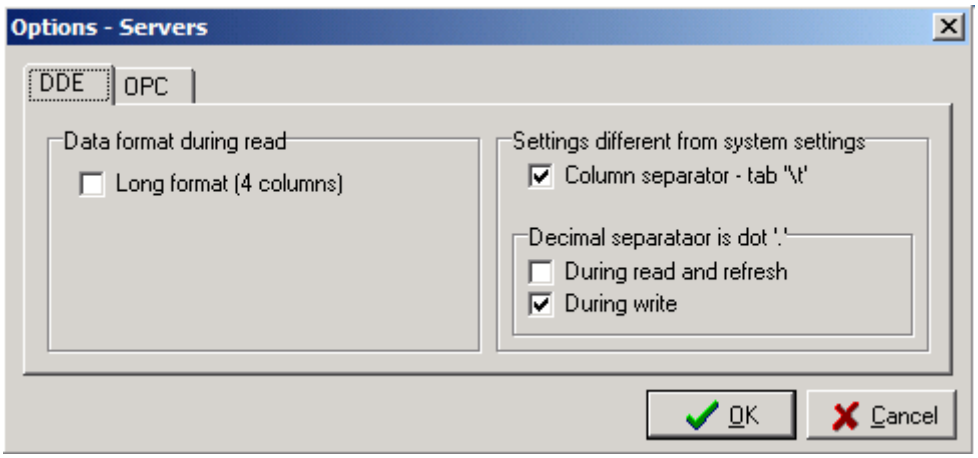


Figure 10. The 'Configurator' Window - Reports.

By default, DDE server sends either the variable value or the error message in the form of a text-type error description. The option *Long format (4 columns)* allows sending of detailed information on the current variables by DDE server to be enabled: the readout status, value, quality and time stamp are defined individually.

If the option *Column separator – tab '\t'* is disabled, the way of column distribution in data tables is changed. Instead of tabulation mark, the list separator defined in the settings of the operating system is used. By default, for Polish language it is semicolon ';'.

The options of the *Decimal separator is dott '.'* group control the method of separating the decimal parts in real numbers sent from and to DDE server. The option *During read and refresh* is by default disabled, which means that when the current and archive data are being read out and the current data are being updated, the decimal symbol defined in the operating system settings is used. By default, for Polish language it is comma ','. The option *During write* is by default enabled, which means that when the current data are being saved, the decimal part is separated with a point. The reason for adoption of this default value is the fact that Excel always uses the point as a decimal symbol when saving data into DDE server.

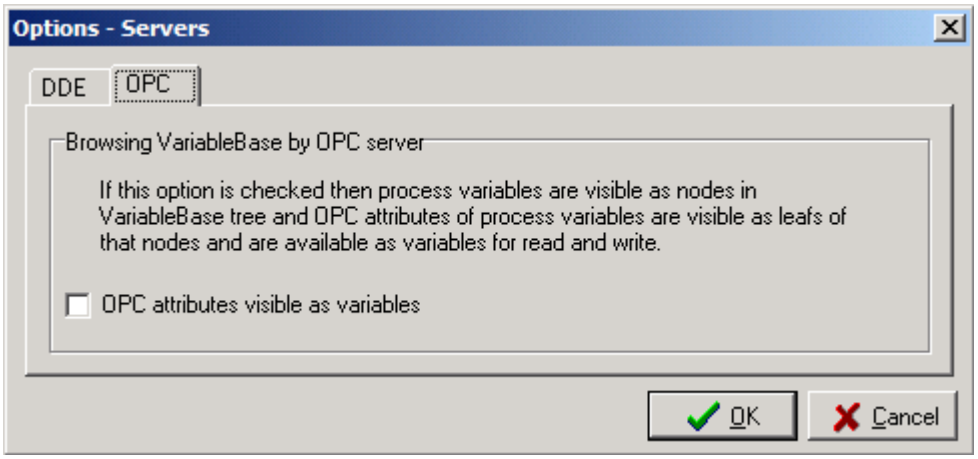


Figure 11. The 'Configurator' Window - OPC Servers.

The option *OPC attributes visible as variables* change the way of reviewing the Variable Definitions Database using OPC mechanisms, which are made available by OPC server. By default, when this option is disabled, the branches of the tree that represent the Variable Definitions Database are the groups of variables, while the leaves are individual variables. If this option is enabled, the internal branches of the tree are the groups of variables, while the external branches are individual variables. The leaves are the attributes of variables, such as value, description, unit, upper range and bottom range.

Program Configuration

Table 13 . DDE and OPC Servers - Program Configurator.

Option	Meaning	Type	Default Value
DataFormatIsLong	*Long format (4 columns)*	Boolean	No
ColumnSeparatorIsTab	*Column separator – tab '\t'*	Boolean	Yes
DecimalSymbolWhenReadIsDot	*Decimal separator is dot '.' During read and refresh*	Boolean	No
DecimalSymbolWhenWriteIsDot	*Decimal separator is dot '.' During write*	Boolean	Yes

4. Variable Definitions Database

In **asix** system the variable definitions database is the space for storing of all information on process variables. Each variable identified by unique *Name* is assigned to one record in the variable definitions database; the fields of this record contain the values of so-called variable attributes. The attribute list includes mandatory, optional and user-defined items. The mandatory attributes are e.g. variable description unit, channel number for communication with the object, archiving parameters. The optional attributes are e.g. limits, format, ranges. Furthermore, the user may also create their own text attributes, individual for each project, e.g. KKS or assembly data.

VarDef system supports Microsoft SQL Server 2000/2005 databases and MDB (Jet/Microsoft Access) databases. In order to migrate the application from earlier **asix** versions, the variable definitions may be converted from Paradox database or text files to the MSQL or Jet database. Architect module provides a full support of VarDef database in Jet or MS SQL format. In this regard, it combines the functionality of two separate programs applied previously for creating and editing the variable definitions databases: Variable Database Manager and Variable Database Editor (used in the older **asix** versions). Those applications are still included in **asix** suite to enable the use of the older database version.

See more: Architect user's manual, chapter *2. Handling of the asix application variable definitions databases*.

4.1. Automation Server

Automation server of Variable Definitions Database enables access to Variable Definitions Database of **asix** system application with use of Automation mechanism. Automation server is in-process server implemented in form of DDL dynamic library and executed in client memory space. It is registered in Windows operating system as an object named *XConnect.ServerVB*. Detailed description of the functions, properties and constants accessed by this object is given later in this chapter. The server also registers in the operating system its own library of types named *AsixConnect 4 Type Library*.

Automation server of the current data complies with Automation mechanism and may be used in programming languages handling the Automation mechanism. These languages are *Visual Basic*, *Visual Basic for Applications* (e.g. from *Microsoft Office* package) or *Visual Basic Script*.

When converting Visual Basic application that uses the AsixConnect package in version 3, the name of the server object *ServerBZ.App* should be changed into *XConnect.ServerVB* and the name of the currently used library of types should be changed into *AsixConnect 4 Type Library*.

4.1.1. Application of Server

When you are going to perform operations on the Variable Definitions Database with use of Automation server, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** application from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Develop a program operating on *XConnect.ServerVB* server. In this program, you need to:
  - create an object of *XConnect.ServerVB* type,
  - call the *LoadChannel* function, giving as parameter the name of the previously configured channel,
  - using the procedures *SelectVar* and *ReadAttribute*, execute the variable selection and attribute readout.

All parameters of the server function are of VARIANT type.

Rather than using channels, the Variable Definitions Database may also be loaded using the *Init* function.

4.1.2. LoadChannel Function

Function calling syntax:

```
LoadChannel ChannelName
```

This function is used for initialisation of *XConnect.ServerVB* object by loading the channel. As *ChannelName* parameter the channel name should be given *(see: 3. Connection Configuration)*.

4.1.3. Init Function

Function calling syntax:

```
Init InitString
```

This function is designed for setting server parameters. For description, *see 3.4.1.Configurator Program*, and for available options, see *3.6.2. Variable Definitions Database.*

4.1.4. ReadAttribute Function

Function calling syntax:

```
ReadAttribute (VarNamej, AttributeName)
```

ReadAttribute function returns the value of variable attribute.

As *VarNamej* parameter the variable name should be passed.

As *AttributeName* parameter the attribute name or one of constants defined by Automation server should be passed.

Table 14. Constants Defined by Automation Server for the 'AttributeName' Parameter for the 'ReadAttribute' Function.

Name of Constant	Meaning
atrDescription	Variable description
atrEU	Measuring unit
atrRangeLo	Lower measuring range
atrRangeHi	Upper measuring range
atrSampleRate	Sampling rate
atrArchiveRate	Archiving rate

4.1.5. SelectAttribute Function

Function calling syntax:

```
SelectAttribute (SelectedAttributeName)
```

The *SelectAttribute* function is designed for interactive selection of attribute name by the user. When called, the dialog box is displayed and it contains a list of variable attribute names defined in the Variable Definitions Database. The user can choose one of them and accept the selection by clicking on *OK* button or cancel the selection by clicking on *Cancel* button.

SelectAttribute function returns value *true,* if the user has accepted the selection by clicking on *OK* button or *false*, if the user has cancelled the selection by clicking on *Cancel* button.

SelectedAttributeName is an input-output parameter. If on entry it contains the name of attribute, then this attribute will be highlighted after the attribute selection window has been displayed. If on output the *SelectAttribute* function has returned *true* value, the parameter contains the name of selected attribute.

4.1.6. SelectVar Function

Function calling syntax:

```
SelectVar (SelectedVarName)
```

SelectVar function is designed for interactive selection of a variable name from Variable Definitions Database by the user. When called, the dialog box is displayed and it contains a list of variable names defined in the Variable Definitions Database. The user can choose one of them and accept the selection by clicking on *OK* button or cancel the selection by clicking on *Cancel* button.

The *SelectVar* function returns value *true*, if the user has accepted the selection by clicking on *OK* button or *false*, if the user has cancelled the selection by clicking on *Cancel* button.

SelectedVarName is an input-output parameter. If on entry it contains the name of variable, then this variable will be highlighted after the variable selection window has been displayed. If on output the *SelectVarName* function has returned *true* value, the parameter contains the name of selected variable.

4.1.7. SelectVars Function

Function calling syntax:

```
SelectVars (SelectedVarsNames)
```

SelectVars function is designed for interactive selection of variable names from Variable Definitions Database by the user. When called, the dialog box is displayed and it contains a list of variable names defined in the Variable Definitions Database. The user can choose one or more of them and accept the selection by clicking on *OK* button or cancel the selection by clicking on *Cancel* button.

The *SelectVars* function returns value *true*, if the user has accepted the selection by clicking on *OK* button or *false*, if the user has cancelled the selection by clicking on *Cancel* button.

SelectedVarNames is an output parameter. If on output the *SelectVars* function has returned *true* value, the parameter contains the table of the names of selected variables.

4.2. NET Server

4.2.1. ServerVB Class

4.2.1.1. Application of Server

The *ServerVB* class enables access to the functional part of **asix** system to read out the data from the Variable Definitions Database. When you are going to perform operations on the Variable Definitions Database with use of *ServerVB*, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** system from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Generate the project in Visual Studio package and then:
  - highlight the *References* directory in the project tree, select the *Add Reference* command from the *Project* menu, click on *Browse* button and select the *XConnectNet.dll* file in the *c:\asix* subdirectory, click on *OK* and close the *Add Reference* window. In every file with C# source code, the following line should be added in the *using* declaration area:

    ```
    using XConnectNet;
    ```

- Develop a program operating on object of *ServerVB* class. In this program, you need to:
  - create object of *ServerVB* type, giving as parameter the name of the previously configured channel,
  - using the component functions of the *Read** server, activate data readout;
  - during data exchange, you should remember about handling of exceptions as they may be reported by the server;
  - after the use of *ServerVB* object has been finished, call its component function *Dispose*;
  - during data exchange, you should remember about handling of exceptions as they may be reported by the server.

4.2.1.2. ServerVB Designer

```
 [C#]
public ServerVB(
  string channelName);
```

This function is used to create and initiate the object of *ServerVB* class.

As *channelName* parameter, the channel name should be given (*see: 3. Connection Configuration*).

4.2.1.3. Dispose Function

```
 [C#]
public void Dispose();
```

This function is used for releasing the resources used by *ServerVB* object. This function must be called after use of *ServerVB* object has been finished. Calling should take place from the same thread the object was created in.

4.2.1.4. Init Function

```
 [C#]
public void Init(
    string initString);
```

This function is designed for setting server parameters. For description of the function, see *3.5. Program Configuration*, and for available options, see *3.6.2. Variable Definitions Database.*

4.2.1.5. ReadAttributes Function

```
 [C#]
public string[] ReadAttributes(
    string varName,
    string[] attributeNames);
```

The *ReadAttributes* function is used for reading out the values of variable attributes.

As *varName* parameter the variable name should be passed.

As *VarNames* parameter the table of attribute names should be passed.

As a result, the function returns the table of the texts containing the attribute values.

4.2.1.6. ReadAttributesN Function

```
 [C#]
public string[,] ReadAttributesN(
    string[] varNames,
    string[] attributeNames);
```

The *ReadAttributesN* function is used for reading out the values of attributes of a few variables at once.

As *varNames* parameter the table of variable names should be passed.

As *VarNames* parameter the table of attribute names should be passed.

As a result, the function returns the two-dimensional table of the texts containing the attribute values. In the first row, there are attributes of the first variable, in the second – attributes of the second variable, etc.

4.2.1.7. Operation in ASP.NET Environment

In case of operation in ASP.NET environment, the object should be created using the *ServerVB.ServerPool.Get()* expression. The *ServerPool* object is a static field of *ServerVB* class and it implements the pool of current data servers. Using the *Get* function, every time before generation of the page the *ServerVB* object should be retrieved. Declaration of the *Get* function is as follows:

```
 [C#]
public ServerVB Get ();
```

After the generation has been completed, the server must be returned to the pool with use of *ServerVB.ServerPool.Release()* expression. As the *Release* function parameter the server returned to the pool should be passed. Declaration of the *Release* function is as follows:

```
 [C#]
public Release (ServerVB server);
```

The server may also be taken from the pool in the beginning of each function and returned in the end of the function. To ensure that each server is returned to the pool, the main code of the function must be included in the *try* block and the server should be returned to the pool in the *finally* block.

EXAMPLE

```
private void Page_Load(object sender, System.EventArgs e)
{
    ServerVB server = null;

    try
    {
        server = ServerVB.ServerPool.Get();

        // function code
    }
    catch(Exception e)
    {
        // handling of exceptions reported when getting the server from the pool
and
        // during the function operation
    }
    finally
    {
        if (server != null)
            ServerVB.ServerPool.Release(server);
    }
}
```

Pool of servers:

- creates several objects of *ServerVB* class for the application (the channel name is retrieved from *Web.Config* file, see: *3.2. How to Specify the Channel Name*),
- stores the objects in cache memory of ASP.NET application,
- makes the objects available for successive calls under ASP.NET application,
- reports *PoolApplicationException* exception when the pool of servers has reached its maximum size and there is no free server.

4.2.2. ServerVBUI Class

4.2.2.1. Application of Server

The *ServerVBUI* class enables access to the functional part of **asix** system to select variables from Variable Definitions Database and select attributes from Variable Definitions Database. When you are going to perform operations on the Variable Definitions Database with use of *ServerVBUI*, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** system from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Generate the project in Visual Studio package and then:
  - highlight the *References* directory in the *Project* tree, select the *Add Reference* command from the Project menu, click on *Browse* button and select the *XConnectNet* file in the *c:\asix* subdirectory, click on *OK* and close the *Add Reference* window. In every file with C# source code, the following line should be added in the *using* declaration area:

    ```
    using XConnectNet;
    ```

- Develop a program operating on object of *ServerVBUI* class. In this program, you need to:
  - create object of *ServerVBUI* type, giving as parameter the name of the previously configured channel,
  - using the component functions of the *Select** server, activate data exchange;
  - during data exchange, you should remember about handling of exceptions as they may be reported by the server;
  - after the use of *ServerVBUI* object has been finished, call its component function *Dispose*.

4.2.2.2. ServerVBUI Designer

```
 [C#]
public ServerVBUI(
  string channelName);
```

This function is used to create and initiate the object of *ServerVBUI* class.

As *channelName* parameter, the channel name should be given *(see: 3. Connection Configuration).*

4.2.2.3. Dispose Function

```
 [C#]
public void Dispose();
```

This function is used for releasing the resources used by *ServerVBUI* object. This function must be called after use of *ServerVBUI* object has been finished. Calling should take place from the same thread the object was created in.

4.2.2.4. Init Function

```
 [C#]
public void Init(
    string initString);
```

This function is designed for setting server parameters. For description, see *3.5. Program Configuration*, and for available options, see 3.6.2. *Variable Definitions Database*.

4.2.2.5. SelectVar Function

```
 [C#]
public bool SelectVar (
    IntPtr parentWindowHandle,
    ref string selectedVarName);
```

SelectVar function is designed for interactive selection of a variable name from Variable Definitions Database by the user. When called, the dialog box is displayed and it contains a list of variables defined in the Variable Definitions Database. The user can choose one of them and accept the selection by clicking on *OK* button or cancel the selection by clicking on *Cancel* button.

The *SelectVar* function returns value *true* if the user has accepted the selection by clicking on *OK* button or *false*, if the user has cancelled the selection by clicking on *Cancel* button.

The *parentWindowHandle* is an input parameter. As the value of this parameter the handle of the window, which the function has been called from, should be passed. For objects of *Form* class, the window handle returns the value *Handle*.

The *selectedVarName* is an input-output parameter. If on entry it contains the name of variable, then this variable will be highlighted after the variable selection window has been displayed. If on output the *SelectedVarName* function has returned *true* value, the parameter contains the name of selected variable.

4.2.2.6. SelectVars Function

```
 [C#]
public bool SelectVars(
    IntPtr parentWindowHandle,
    ref string[] selectedVarNames);
```

The *SelectVars* function is designed for interactive selection of variable names from Variable Definitions Database by the user. When called, the dialog box is displayed and it contains a list of variable names defined in the Variable Definitions Database. The user can choose one or more of them and accept the selection by clicking on *OK* button or cancel the selection by clicking on *Cancel* button.

The *SelectVars* function returns value *true* if the user has accepted the selection by clicking on *OK* button or *false*, if the user has cancelled the selection by clicking on *Cancel* button.

The *parentWindowHandle* is an input parameter. As the value of this parameter the handle of the window, which the function has been called from, should be passed. For objects of *Form* class, the window handle is stored in the property of *Handle*.

The *selectedVarName* is an output parameter. If on output the *selectedVarName* function has returned *true* value, the parameter contains the table of the names of selected variables.

4.2.2.7. SelectAttribute Function

```
 [C#]
public bool SelectAttribute(
    IntPtr parentWindowHandle,
    ref string selectedAttributeName);
```

The *SelectAttribute* function is designed for interactive selection of attribute name by the user. When called, the dialog box is displayed and it contains a list of variable attribute names defined in the Variable Definitions Database. The user can choose one of them and accept the selection by clicking on *OK* button or cancel the selection by clicking on *Cancel* button.

The *SelectAttribute* function returns value *true* if the user has accepted the selection by clicking on *OK* button or *false*, if the user has cancelled the selection by clicking on *Cancel* button.

The *parentWindowHandle* is an input parameter. As the value of this parameter the handle of the window, which the function has been called from, should be passed. For objects of *Form* class, the window handle is stored in the property of *Handle*.

The *SelectedAttributeName* is an input-output parameter. If on entry, it contains the name of attribute, then this attribute will be highlighted after the attribute selection window has been displayed. If on output the *SelectAttribute* function has returned *true* value, the parameter contains the name of selected attribute.

5. Measurement Status Description

The measurement status is described by the following threesome: time stamp, quality and value. Depending on the server, the threesome may be presented as a structure, three parameters or a text in which the elements are separated with a separator mark.

The time stamp is expressed in the natural format for the specific server. The time is always provided as local time (except for OPC server where UTC time is used).

The quality is provided as a 16-bit number (in case of the current data servers) or 32-bit number (in case of the archive data servers). In this number, the meaning of bits complies with OPC 2.05 specification. Description of bits is provided in the chapters below.

The measurement value may take one of the following types:

- 16-bit number with or without a sign,
- 32-bit number with or without a sign,
- single- or double-precision real number,
- text,
- table of numbers.

5.1. Measurement Quality

The quality element represents the quality of the measurement value status.

The lower 8 bits (bits 0-7) of the quality flag are defined as three-bit fields: *Quality*, *Substatus* and *Limit;* the bits are arranged as follows:

```
QQSSSSLL
```

Bits 8-15 of the quality flag are remained to be used by the software originators. These bits are defined as a *Vendor* bit field.

5.2. Quality Bit Field

Table 15. Quality Bit Field.

QQ	Value of Bits	Definition	Description
0	00SSSSLL (0x00)	*Bad*	Value is not available; details in *Substatus* field.
1	01SSSSLL (0x40)	*Uncertain*	Quality of variable is uncertain; details in *Substatus* field. In **asix** system application uncertain value is treated as „almost" bad. The way of displaying the variable with uncertain value is the same as for value with bad value and value of *Substatus* field equal to *Last Known Value*.
2	10SSSSLL	*N/A*	I not used by OPC.
3	11SSSSLL (0xC0)	*Good*	Value quality is good.

It is recommended that the client should verify at least the contents of the *Quality* bit field. Verification usually takes place by carrying out the bit product operation for the *Quality* field and mask of the *Quality* bit field with value of *0xC0*. The result is compared to *Bad*, *Uncertain* and *Good* constants like in the following example (C/C++/C# language):

```
int q = Quality & 0xC0;
if (q == 0x00)
{
    // bad quality
}
else
if (q == 0x40)
{
    // uncertain quality
}
else
{
   // good quality.
}
```

5.3. Substatus Bit Field for Bad Quality

Table 16. Substatus Bit Field for Bad Quality.

SSSS	Value of Bits	Definition	Description
0	000000LL (0x00)	*Non-specific*	Value is bad, but the reason is unknown.
1	000001LL (0x04)	*Configuration Error*	There is the problem with server configuration.
2	000010LL (0x08)	*Not Connected*	It is demanded the entry to be logically joined – but it is not. The data source has not provided the value.
3	000011LL (0x0C)	*Device Failure*	Device failure is detected.
4	000100LL (0x10)	*Sensor Failure*	Sensor failure is detected. Quality is signalized by the driver of device or by check variable (in the second possibility – only when *Check control items* is set).
5	000101LL (0x14)	*Last Known Value*	Communication error. The last variable is available. „Age" of value is defined by its time stamp.
6	000110LL (0x18)	*Comm Failure*	Communication error. The last variable is available.
7	000111LL (0x1C)	*Out of Service*	Block is not deleted or is blocked. It is also used when inactive variable is read.
8-15		*N/A*	Is not used by OPC..

5.4. Substatus Bit Field for UNCERTAIN Quality

Table 17. Substatus Bit Field for UNCERTAIN Quality.

SSSS	Value of Bits	Definition	Description
0	010000LL (0x40)	*Non-specific*	Value is uncertain but the reason of this situation is unknown
1	010001LL (0x44)	*Last Usable Value*	Device stopped reading the variable values. The value should be treated as stale.
2-3		*N/A*	It is not ised by OPC.
4	010100LL (0x50)	*Sensor Not Accurate*	Either the value exceeded some of the z sensor limits nor the sensor is out of calibration.
5	010101LL (0x54)	*Engineering Units Exceeded*	Value exceeded limits defined for the measurement.
6	010110LL (0x58)	*Sub-Normal*	Value is defined on the basis of several sources anld the number of available sources is less than demanded number.
7-15		*N/A*	It is not used by OPC.

5.5. Substatus Bbit Field for GOOD Quality

Table 18. Substatus Bit Field for GOOD Quality.

SSSS	Value of Bits	Definition	Descriotion
0	110000LL(0xC0)	*Non-specific*	Value is good.
1-5		*N/A*	It is not used by OPC.
6	110110LL(0xD8)	*Local Override*	Value was overwritten. That usually means that entry was disconnected and a new value was written manually.
7-15		*N/A*	It is not used by OPC.

5.6. Limit Bit Field

The *Limit* field is valid regardless of the contents of *Quality* and *Substatus* fields.

Table 19. Limit Bit Field.

LL	Value of Bits	Definition	Description
0	QQSSSS00 (0x00)	*Not Limited*	Value may increase and decrease.
1	QQSSSS01 (0x01)	*Low Limited*	Value exceeded the lower limit.
2	QQSSSS10 (0x02)	*High Limited*	Value exceeded the upper limit.
3	QQSSSS11 (0x03)	*Constant*	Value is constant and can not change.

When reading the values of the current variable from **asix** system, the *Limit* bit field usually takes *Not Limited* value. If the current data server's option *Verify limits of variables* is enabled, then the values *Low Limited* or *High Limited* will appear after the warning limits are exceeded. When the alarm limits are exceeded, then additionally the *Second Limit* flag in the *Vendor* bit field is activated.

When reading the values of the variable attributes from the Variable Definitions Database, the *Limit* bit field always takes *Constant* value.

5.7. Vendor Bit Field

The servers of AsixConnect package use one flag from this field. It name is *Second Limit*.

Table 20. Vendor Bit Field.

Hexadecimal Value	Definition	Description
0x0800	*Second Limit*	Value exceeded the second limit – alarm limit. The bit field *Limit* determined whether upper or lower limit is exceeded.

The *Second Limit* flag may appear only if the current data server's option *Verify limits of variables* is enabled.

5.8. Archive Data Bit Fields

Flags specific to the archive data are within the range of bit numbers 16-31.

These flags are used by .NET server of archive data and *Web Service* server.

Table 21. Archive Data Bit Field.

Hexadecimal Value	**Definition**	**Description**
0x00100000	*Quality Bad – No Bound*	The limitary value can not be passed because the archive is not available for the time moment responded this limitary time moment.
0x00200000	*Quality Bad – No Data*	Process variable value is not available because the archive with data from the precise time period is not available.
0x01000000	*Asix Archive End*	Process variable value is not available because it found the end of archive when reading the data (reading the data from the period that does not occur yet or in the case of the lack of timer synchronization between a client computer and the **asix** system station).
0x0040000	*Quality Good – Raw Value*	Process variable value received from the archive.
0x00080000	*Quality Good – Calculated Value*	Value calculated on the basis of archival data.

6. Current Data

6.1. Identifiers

In the servers of current data, the variable name itself is sufficient for unique identification of a process variable, i.e. the name used in the application of **asix** system. All other information on the variable needed for communication with **asix** system is retrieved from the Variable Definitions Database. In order to obtain the Variable Definitions Database, you should address to the administrator of application of **asix** system from which the data are to be retrieved.

In the channel options, full path to the file containing the collection of variables from the Variable Definitions Database should be determined on an interactive basis or by program. For details, see the chapter on server configuration *3.6.2.Variable Definitions Database.*

In the simplest case, the identifier is the variable name itself. However, identifiers may be much more complex. Identifiers may be divided into simple identifiers and intermediate identifiers.

Simple Identifiers

For list of simple identifiers, see the table below. The notation <*Variable*> means the name of variable in the **asix** system application. The notation <*Attribute*> means the name of attribute in the Variable Definitions Database of **asix** system application.

Table 22. List of Simple Identifiers for Unique Identification of a Process Variable in Servers of Current Data.

Identifier	Description	Possible Value Type
<Variable> *<Variable>.CV*	Current value of variable.	R4, I2, I4, UI2, UI4
<Variable>.DATA_TYPE	canonical type of Current value of variable.	I2
<Variable>.EU_UNIT	Variable unit.	STRING
<Variable>.DESC	Variable description.	STRING
<Variable>.HI_EU	Max value of current value of variable.	R8
<Variable>.LO_EU	Min value of current value of variable.	R8
<Variable>.<Attribute>	Variable attribute value.	STRING, R8
<Variable>FormatedCV	Current value of variable converted in accordance with *Format* attribute value of this variable.	STRING
<Variable>.PercentageCV	Percentage current value of variable.	R8
<Variable>.BarCVs	Percentage current value of variable complying with bar basis and current percentage value of bar basis (2-element table).	[R8, R8]
<Variable>.<Attribute>.CV	Variable attribute value in database, if the value is a number.	R8
<Variable>.<Attribute>.CV. FormatedCV	Variable attribute value formated in accordance with *Format* attribute value of this variable. The row value of variable is a number.	STRING
<Variable>.<Attribute>. CV.PercentageCV	Percentage value of variable attribute. The row value of attribute is a number.	R8

Intermediate Identifiers

The intermediate identifier always starts with the phrase *<Variable>.<Attribute>.CV*. The attribute *<Attribute>* of variable *<Variable>* must contain the name of another variable – such a variable is named an intermediate variable.

The example of an intermediate variable is a variable that makes available the values of the warning or alarm limit of another variable.

For the list of possible intermediate identifiers, see the table below:

Table 23. List of Possible Intermediate Identifiers for Unique Identification of a Process Variable in Servers of Current Data.

Identifier	Description	Type
<Variable>.<Attribute>.CV	Current value of mediate variable.	R4, I2, I4, UI2, UI4
<Variable>.<Attribute>.CV. FormatedCV	Current value of mediate variable converted in accordance with *Format* attribute value of this variable.	STRING
<Variable>.<Attribute>. CV.PercentageCV	Percentage current value of mediate variable.	R8
<Variable>.<Attribute>.CV.BarCV	Percentage current value of mediate variable complying with bar basis and current percentage value of bar basis (2-element table).	[R8, R8]
<Variable>.<Attribute>.CV .<Attribute2>	Attribute value *<Attribute2>* of mediate variable.	STRING , R8

6.2. Operation without Variable Definitions Database

In general using the Variable Definitions Database is recommended. Using the information contained in this chapter should be exceptional.

AsixConnect server of the current and archive variables accepts also descriptive identifiers of variables and where you use these identifiers only, loading the Variable Definitions Database is not needed.

During verification of descriptive identifier no communication with **asix** system to perform this verification is needed. Any errors concerning communication with **asix** system, which may occur later, are signalled in field/parameter *quality* during the read operation and as a result of read/write operation of variable value.

Descriptive identifier takes the following form:

```
asmen.<channel name>.<variable name>.<variable type>
```

where:

- *channel name* – a group name of the current variables registered in Asmen module of application of **asix** system,
- *variable name* – a name under which process variable is known in application of **asix** system,
- *variable type* – two or three characters defining the type of variable according to table below:

Table 24. Types of Variables (Operation without Variable Definitions Database).

Variable Type	Description of Variable Type
R4	Real variable
I2	16-bit signed integer
UI2	16-bit unsigned integer (word)
I4	32-bit signed integer
UI4	32-bit unsigned integer (long word)
UI1	8-bit unsigned integer (byte)

The type of variable in **asix** system can be defined on the basis of its converting function.

EXAMPLE

Example of variable identifier:

asmen.Camac.06C_A111AF00.R4

This identifier denotes a process variable named *06C_A111AF00*, accessible in the *Camac* channel. This is value of real type.

NOTE *In **asix** system uppercase and lowercase letters in the names of channels and variables are distinguished. This may sometimes cause problems, because some DDE clients automatically change all letters to the lowercase letters or to the uppercase letters.*

6.3. Defining Write Rights

6.3.1. Write Function

Write is an operation to assign new value to process variable. Time stamp (i.e. current time) is assigned by **asix** system; quality is assumed to be good.

Write operation will be successful if:

- variable is provided with write property (this depends on controller's driver and on controller itself; the most frequently variable is provided with write property);
- the client is assigned authority to set the new values in the channel, which variable belongs to.

Client is always authorized to set the value of variable in any channel in application of **asix** system running on the same computer as the client. In order to assign the client running in other computer, authority to set the values of variables in the defined channel, you need to place the following string to ASMEN section in **asix** system application file:

```
WRITE_PERMIT = <name of channel>, <name of client computer>
```

<*name of channel*> denotes name of channel of **asix** system to which we want to write; <*name of client computer*> is name of client computer in **asix** system, which will be

assigned authority to write to the defined channel. Name of the computer is defined as follows.

Table 25. Client Computer Name in asix System.

Description of Client Software Configuration	Client Computer Name in asix System
Only AsixConnect package is used and *aslink.ini* file was not created or *Name* line was not defined in *ASLINK section.*	Name of client computer in WINDOWS plus period (.) at the end, e.g. for computer named „CLI" its name in **asix** system is „CLI."
Only AsixConnect package is used and ASIX.INI file was created, *Name* line was defined in *ASLINK section*.	Name of client computer in **asix** system is provided in ASIX.INI file in ASLINK section, in line titled *Name***.**
AsixConnect and **asix** application run in the same computer.	Name of client computer in **asix** system is provided in INI file of application, in ASLINK section, in line titled *Name***.**

EXAMPLE

WRITE_PERMIT = SINEC1, KE2
This string denotes that servers from AsixConnect package, running in computer of name KE2 in **asix** network have authority to write into the variables in channel SINEC1.

6.3.2. Extended Write Function

Extended write is an operation to assign new value, quality and time stamp to process variable by client.

Extended write operation will be successful if:

- variable is provided with write property (this depends on controller's driver and on controller itself; the most frequently variable is provided with write property);
- the client is assigned authority to set the new value, new status and new time stamp in the channel, which variable belongs to.

In order for the client to have this authority:

- for the channel relevant write permit, which the variable belongs to as described in *Extended write function*, should be configured;
- the channel must be supported by driver of name *NONE*;
- in *ini* file of the application of **asix** system, there must be section of the same name as the channel name. This section must contain the entry *WRITE_TIME_AND_STATUS=YES.*

EXAMPLE

Example of the fragment of *ini* file:

```
 [ASMEN]
CHANNEL1=NONE
WRITE_PERMIT = CHANNEL1, KE2
[CHANNEL1]
WRITE_TIME_AND_STATUS = YES
```

This fragment of file denotes that servers from AsixConnect package, running in computer of name *KE2* in **asix** network have authority to write values, status and time stamp into the variables in channel *CHANNEL1*

NOTE The *WriteEx* function is supported by **asix** system in versions supplied after 2000/04/20 (*Netsrv* module in version 2.2.0 or later and *Asmen* module in version 3.2.8 or later).

6.4. Automation Server

Automation mechanism developed by Microsoft and available in the family of Windows operating systems enables the applications an access to its functionality as programming objects. Objects include the functions, properties and events. Automation server of current data allows access to part of the functionality of **asix** system in the scope of current data with use of Automation mechanism. Automation server is in-process server implemented in form of DDL dynamic library and executed in client memory space. The server is registered in Windows operating system as object named *XConnect.ServerCT*. Detailed description of the functions, properties, events and constants of this object is given later in this chapter. The server also registers in the operating system its own library of types named *AsixConnect 4 Type Library*.

Automation server of the current data complies with Automation mechanism and may be used in programming languages handling the Automation mechanism. These languages are *Visual Basic*, *Visual Basic for Applications* (e.g. from *Microsoft Office* package) or *Visual Basic Script*

When converting Visual Basic application that uses the AsixConnect package in version 3, the name of the server object *ServerCT.App* should be changed into *XConnect.ServerCT* and the name of the currently used library of types should be changed into *AsixConnect 4 Type Library*. In addition, attention should be paid to the fact that server returns full quality of variable, not the *Good*, *Bad* and *Uncertain* flags only.

6.4.1. Application of Server

When you are going to perform operations on the current variables with use of Automation server, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** application from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.

- Develop a program operating on *XConnect.ServerCT*. In this program, you need to:
  - create an object of *XConnect.ServerCT*;
  - call the *LoadChannel* function, giving as parameter the name of the previously configured channel;
  - using the *Read* and *Write* procedures, execute the data exchange and/or
  - activate automatic data transfer from the server with use of events:
    - pass *DataChange* events to the handler object in order to receive through them information on current values of variables,
    - using *SetItemActive* procedure, activate supplying by the server current values of variables,
    - assign *True* value to the property named *Active*, in order to start generating *DataChange* events by server.

All procedure parameters, properties and event handler parameters are of VARIANT type.

Rather than using channels, the server parameters may be set (and Variable Definitions Database may be loaded) using the *Init* function.

6.4.2. LoadChannel Function

Function calling syntax:

```
LoadChannel ChannelName
```

This function is designed for initialisation of *XConnect.ServerCT* object by loading the channel. As *ChannelName* parameter, the channel name should be given *(see: 3. Connection Configuration).*

6.4.3. Init Function

Function calling syntax:

```
Init InitString
```

This function is designed for setting server parameters. For description of the function, see *3.5. Program Configuration*, and for available options, see the following sections.

6.4.4. Read Function

Function calling syntax:

```
Read DataSource, ItemID, Value, Quality, TimeStamp
```

The *Read* function is designed for reading the current values of process variables.

The *DataSource* is an input parameter and defines data source. It may take one of the following three values: *dsCache*, *dsDevice*, *dsDrive*. Their meaning is given in the table below.

Table 26. Values of the 'DataSource' Parameter for the 'Read' Function for the Automation Server (Current Data).

Data source	Description of Data Source	Reading Time	Delay of Variable	Application	Number Value	Notes
dsCache	Cache memory of Automation server	Shortest	Highest: **asix** system sampling rate + time from latest update of server's cache memory	Variables to be read many times and variables active at the moment	1	Read variable must be active (See: *SetItemActive* function)
dsDevice	Cache memory of **asix** system	Average	Average: **asix** system sampling rate + network transfer time	Variables to be read small number of times	2	
dsDriver	Industrial controller	Longest	Smallest: reading time from device + network transfer time	Apply only if value of process variable from this current time is needed.	3	

NOTE Reading from the cache memory of **asix** system (*dsDevice*) is supported by **asix** system in versions supplied after 2000/06/01 (Netsrv module in v. 3.1.0 or later). In earlier versions, reading from **asix** cache memory is automatically replaced by reading from controller.

ItemID is an input parameter and should contain the variable identifier.

After read operation, *Value* parameter contains the value of variable; *Quality* parameter contains variable quality and *TimeStamp* parameter contains the variable time stamp.

Quality (variable quality) parameter contains the quality flag described in section *5.2. Quality Bit Field.*

TimeStamp parameter contains value of type VARIANT/DATE representing the time stamp of current value of process variable; local time is used. *TimeStamp* parameter is optional.

6.4.5. SetItemActive Function

Function calling syntax:

```
SetItemActive ItemID, ActiveState
```

SetItemActive function is designed for activating and deactivating the process variable, i.e. it adds or removes variable from list of variables refreshed in the *cache* memory

ItemID is an input parameter and should contain the variable identifier.

ActiveState parameter should contain value *True* if variable is to be activated and value *False* if variable is to be deactivated.

If variable is active then:

- **asix** system sends to Automation server the current value of process variable; the sending time is equal to variable sampling time in **asix** system;
- variable value may be read from cache memory of Automation server with use of *Read* function, passing *ds.Cache* constant as *DataSource* parameter;
- value of variable may be sent to client automatically through *DataChange* event.

6.4.6. Write Function

Function calling syntax:

```
Write ItemID, Value
```

Write function is designed to set a new value to the process variable.

ItemID is an input parameter and should contain the variable identifier, which value is to be set.

In *Value* parameter you should pass the new value of process variable as integer or real number. Together with the value, good quality (*qualityGood*) and time stamp equal to current time of **asix** system are set.

The write configuration method is described in chapter *6.3.1. Write Function*.

6.4.7. WriteEx Function

Function calling syntax:

```
WriteEx ItemID, Value, Quality, TimeStamp
```

WriteEx function is designed to set a new value, new status and time stamp to a process variable.

ItemID is an input parameter and should contain the variable identifier, which value is to be set.

In *Value* parameter you should pass the new value of process variable as integer or real number.

In *Quality* parameter you should pass the new status of process variable. You can use one of the following constants: *qualityGood*, *qualityUncertain* or *qualityBad*.

In *TimeStamp* parameter you should pass the new time stamp of process variable. This value must be that of *VARIANT / DATE* type (i.e. date and time).

The write configuration method is described in chapter *6.3.2. Extended Write Function*.

6.4.8. Active Property

Active is a read/write property and is designed to control transfer of current values of the active variables from server to client. Variable is active when function *SetItemActive* with *itemID* parameter containing the name of the variable and parameter *ActiveState* equal to *True* was called. In order the current values would be transferred, *Active* property has to take value *True* Default value of *Active* property is *False*

6.4.9. ServerState Property

ServerState is a read only property. It returns current state of the server. It takes one of the following values.

Table 27. Values of the 'ServerState' Property for the Automation Server (Current Data).

Value	Meaning	Number Value
ssRunning	Server is running correctly. Variable collection has been loaded.	1
ssFailed	Error during starting the server.	2
ssNoConfig	Server is running correctly but variable collection was not loaded.	3
ssSuspended	Server is suspended (at present this value is not used by Automation server of AsixConnect package).	4
ssTest	Server is running in test mode (at present this value is not used by Automation server of AsixConnect package	5

6.4.10. StartTime Property

StartTime is a read only property. It contains server start time.

6.4.11. DataChange Event

Syntax of event:

```
DataChange ItemID, Value, Quality, TimeStamp
```

DataChange event is triggered after every change of process variable value. To trigger an event, the process variable must be active (see section *6.4.5. SetItemActive Function*) and *Active* server property has to have value *True*.

DataChange event parameters have the same meaning as those of *Read* function.

6.4.12. Error Handling

If an operation performed by the server fails then the client receives error code. The error codes are described in the server specification. To retrieve text description of the error, a client can use the standard Automation mechanism, i.e. *GetErrorInfo* function and *IErrorInfo* interface.

If the client is a program developed in any version of *Visual Basic* language, then in case of error, program goes to execute the line declared with use of *ON ERROR GOTO* instruction. Information on error will then be available via standard object of *Visual Basic* named *Err*. This object allows access to the error code and its text description.

EXAMPLE

```
Sub test ()
    On Error GoTo blad
    ...
    ' code of program using the Automation server
    ...
    Exit Sub
error:
    MsgBox Err.Description
End Sub
```

6.5. DDE Server

DDE mechanism developed by Microsoft Company and available in Windows operating system family enables applications access to its functionality with use of several transactions defined in this mechanism. DDE server of current data enables access to part of the functionality of **asix** system in the scope of current data with use of DDE mechanism. DDE server is implemented in the form of program of EXE type. After starting, it registers in Windows operating system as server under the name: *ServerCTDDE* and *CTDDE*.

DDE server of process variables must be operating before a client program attempts to establish connection with it. You can start server from *Start* menu of Windows or with Windows Explorer by double-clicking on *ServerCTDDE.exe* file contained in directory where AsixConnect package was installed.

6.5.1. Application of Server

When you are going to perform operations on current data with use of DDE server, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** system from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Start DDE server.

Using C++ programming language, you should in the program:

- establish connection with *DDE* server, passing as *service* parameter the text *ServerCTDDE* or *CTDDE*; as topic parameter you should pass the name of previously configured channel;
- using the *XTYP_REQUEST* and *XTYP_POKE* transactions, execute the data exchange.

Using Visual Basic programming language, you should in the program:

- establish connection with DDE server, passing as the first parameter of the *DDEInitiate* function the text *CTDDE*; as the second parameter of the function you should pass the name of previously configured channel;
- using the *DDERequest* and *DDEPoke* functions, execute the data exchange.

6.5.2. DDE Operations Supported by Server

Functions described in this section are those of Windows system, designed for communication with DDE servers and available from C/C++ language level. The functions are made available by standard DDEML library, facilitating the use of DDE mechanism. Equivalents of those functions are also available in Visual Basic language and described in *6.5.5. Using the DDE Server in Excel*.

DDEConnect
DDEConnect function is designed to establish connection with DDE server.

If you like to connect to server of **asix** system pass *ServerCTDDE* or *CTDDE* string as *service* parameter. As *topic* parameter the channel name should be passed. If the user has configured the basic channel, then null string or one * character is passed as *service* parameter.

DDEDisconnect
DDEDisconnect function is designed to disconnect the client from DDE server.

DDEClientTransaction
DdeClientTransaction function enables execution of several operations referred to as transactions in DDE terminology. Type of the transaction depends on value of *wType* parameter of this function. Constants that may be used as value of this parameter are discussed below. In Visual Basic language a separate function to perform every type of transaction is provided.

XTYP_REQUEST
XTYP_REQUEST transaction is designed to read the current value of variable. The identifier of variable to be read is passed in *item* transaction parameter. Data are retrieved from the cache memory of **asix** system. If retrieved value is a real number, then default decimal separator is the same as declared in Windows system configuration.

XTYP_POKE
XTYP_POKE transaction is designed to set a new value of the variable. The transaction parameter includes variable identifier and the new value. If retrieved value is a real number, then default decimal separator is point. Decimal separator may be changed in DDE options to that as declared in Windows system configuration.

The write configuration method is described in *6.3.1. Write Function*.

For description of setting the server parameters, see *3.4.1. Configurator Program*, and for available options, see the following sections.

XTYP_ADVSTART
XTYP_ADVSTART transaction activates the transfer of current variable value from server to client. The transaction parameter includes the variable identifier of the variable which value is to be received.

It is possible to transfer the current values of group of variables in one transaction. In this case as *item* parameter you should pass the list of identifiers separated with semicolons. Maximum size of the list of variable identifiers is 255 characters. This limit is specified by DDE mechanism.

XTYP_ADVSTOP
XTYP_ADVSTOP transaction stops passing the current value of process variable from server to the client. As a transaction parameter you should pass, either the variable identifier for single variables, or the list of identifiers for the group of variables.

XTYP_EXECUTE
This transaction is not used by DDE server of AsixConnect package.

6.5.3. Format of Transferred Data

The DDE may transfer data in single-column or in four-column format. As default, the single-column format is specified but it may be changed in the program options.

A single-column format is used to transfer either a variable value or error information. The error information contains error text description.

In four-column format, the first column contains: variable status, the second one: variable value, the third one: variable quality and the fourth one: time stamp.

A status less than zero denote that read operation failed; this status specifies also a reading error code. The status equal to 0 or higher than 0 denotes that read operation was successful and other columns contain correct values.

The quality may be sent as one of the following values: *qualityGood*, *qualityUncertain* or *qualityBad*. Their meaning is as follows.

Table 28. Values of the Quality (for the DDE Server /Current Data).

Quality	Meaning	Number Value
qualityGood	The second column contains current value of process variable.	0xC0(192)
qualityUncertain	The second column contains current value of process variable but its value is uncertain. e.g. exceeds maximal range defined for this measurement.	0x40 (64)
qualityBad	Current value of process variable is not accessible, e.g. because of connection failure with controller.	0

During reading and updating the variable, data are transferred in the form of one row containing one or four columns. During updating the group of variables, data are transferred in the form of table in which each row contains information concerning one variable.

The DDE server applies the *CF_TEXT* format for data transfer, i.e. text format; numbers are sent in this format in text representation. If the row includes four columns, then their separator depends on program configuration. The columns may be separated with a separator such as configured in Control Panel of Windows system or with Tab character. If a table is transferred, the individual rows are separated with a new line character. Data in the *CF_TEXT* format are terminated with a character of 0 code.

6.5.4. Transfer of Error Information

If an operation fails, you can get error code by reading the value of *_LastError_* variable of special name. Reading the variable named *_LastErrorMessage_* you can get text description of the error. Don't forget an underscore character at the beginning and at the end of the name.

It is very important to retrieve information on an error code in programs developed in Visual Basic language, because language itself doesn't signal such error.

6.5.5. Using the DDE Server in Excel

We assume that DDE server transfers data in default single-column format.

A cell in the spreadsheet of Excel program may contain formula referencing to remote data accessible with use of the DDE protocol. The formula takes the following form:

```
= service | topic ! item
```

The name of AsixConnect DDE server, i.e. *ServerCTDDE* or *CTDDE*, should be passed as *service* parameter. As *topic*, the channel name (character * if the basic channel is to be used) and as *item*, the variable identifier should be passed. If any item of the formula

contains spaces or non-alphanumeric characters they must be closed in apostrophes. After introducing the formula, Excel connects to the DDE server and attempts refreshing the values of variables. The result may be threefold:

- OK – in cell appears the current value of variable and then it is refreshed,
- error in attempt of access to the variable encountered by the DDE server – Excel receives text error description,
- Error on start of refreshing cycle because the connection to the DDE server could not be set up – Excel displays N/A (*not available*) error. Check whether DDE server is started or its name is correct.

The names of the DDE functions available in Visual Basic are as follow.

Table 29. The Names of the DDE Functions Available in Visual Basic.

Name of DDE Functions/Operations	Name of Function in Visual Basic
DDEConnect	DDEInitiate
DDEDisconnect	DDETerminate
DdeClientTransaction, XTYP_REQUEST	DDERequest
DdeClientTransaction, XTYP_POKE	DDEPoke

The *XTYP_ADVSTART* i *XTYP_ADVSTOP* transactions have no equivalents in the Visual Basic functions.

Groups of Variables

In order to begin updating the group of variables, enter so called 'table formula' to the cells of spreadsheet. The table formula may be entered to the range of cells in the form of a rectangle. Select the range and go to edition by pressing the F2 key. In this formula *item* contains the names of variables separated with semicolons. Maximum size of the *item* parameter is equal to 255 characters. This limit is specified by DDE mechanism.

Application of the variable groups in place of single variables has large effect on updating efficiency of variables. The DDE mechanism enables updating about 200 variables per second (with use of Pentium 166 MHz). If, in place of single variables, you will use the variable groups, then efficiency increases almost proportionally to the size of the group, e.g. for groups including 25 variables it is possible to transfer even 3000 variables per second.

The selected range of cells should be one or four column wide, depending on transfer mode during updating, and have so many rows as number of variables in the group. After editing click on and hold down *CTRL+SHIFT* and then click on *ENTER* key. In this way, an introduced formula will be placed to all cells of the range.

Attempt to cancel any cell will cause an error message. The table formula can be cancelled after selecting the complete range only. In order to select the complete range occupied by the formula quickly, activate any cell of the range and click on *CTRL-/* keyboard shortcut.

6.5.6. DDE Server Utility

The AsixConnect includes the DDE server module, which is not operating as a normal application but as utility of Windows operating system. This module may be executed in Windows 2003/XP/2000/NT4 environment only. The DDE utility is added to the list of utilities registered in the operating system during the installation of AsixConnect. Its name is *DDE server of current data of asix system* and it is configured as "utility started manually".

In order to start this utility automatically during start of the operating system, the user should change the starting mode from *manual* to *automatic* with use of *Utility* applet in *Control Panel/Administrating Tools*. In Windows NT4 this applet is available in *Control Panel*.

The *DDE server* utility makes use of the same configuration file as every server of the AsixConnect package. In order to change the options of *DDE Server* utility, change the options using *Configurator* program and restart the utility.

NOTE The *DDE Server* utility of AsixConnect requires configuration of the system access rights to the package components. The configuration procedure consists of the following steps.

STEP 1
Run *dcomcnfg.exe*.

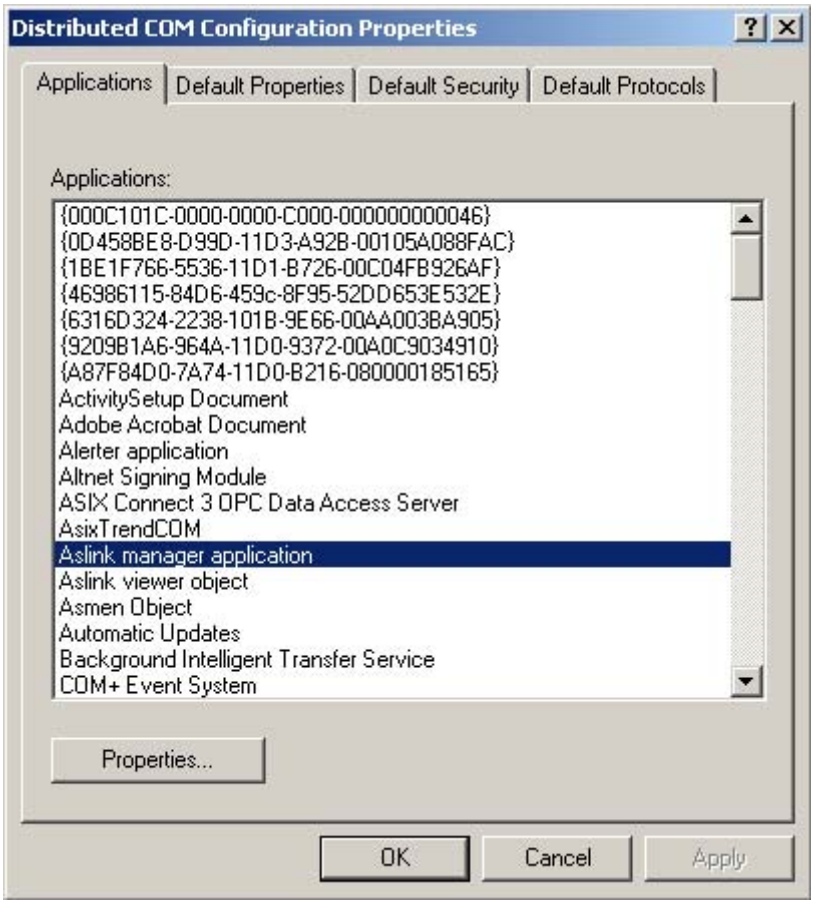


Figure 12. 'Distributed COM Configuration Properties' Window.

STEP 2

In the main window of the program, look for Configuration *DCOM/Aslink manager application* and select *Properties (see the window below)*.

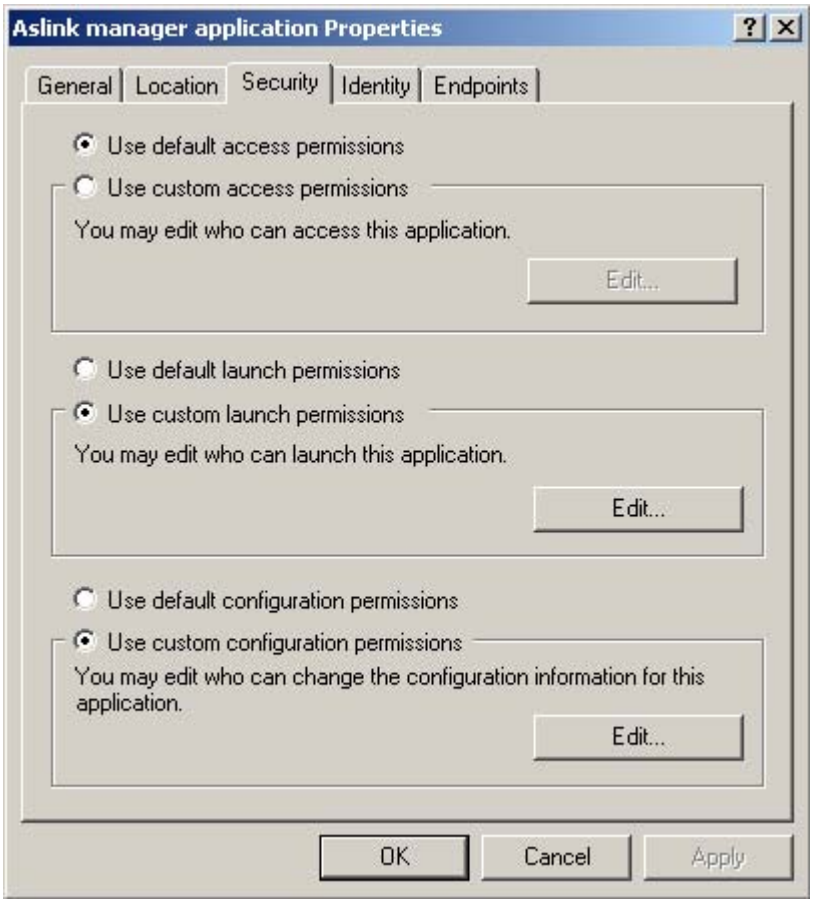


Figure 13. 'Aslink Manager Application Properties' Window.

STEP 3

In the *Security* tab, after the *Use custom access permissions* option is selected and *Edit* button is pressed, the window will be opened where the ASPNET user should be assigned the *Access rights*. The operation should be accepted by clicking on *OK*.

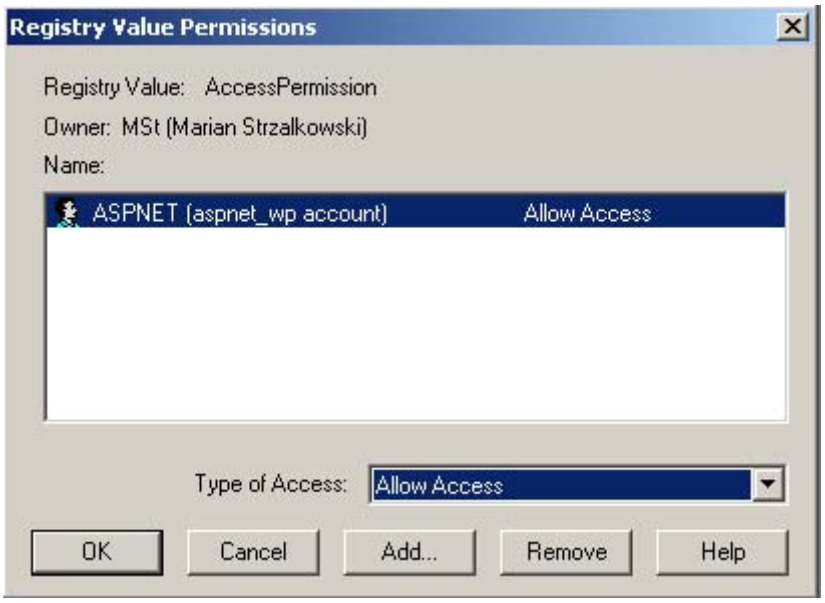


Figure 14. 'Registry Value Permissions' Window.

NOTE To save the modifications, they should be accepted by clicking on *Apply* (window '*Properties: Aslink manager application*').

In the window '*Aslink manager application Properties'*, click on (**!**) the button *Apply*.

STEP 4
In the next step, identity of the user who will be authorised to start the Aslink module should be defined. It is done in the *Identity* tab in the '*Aslink manager application Properties'* window. The username will be *Administrator*. The relevant password should be entered. If **asix** system is not operating on the given computer, the option *Starting user* may also be selected. The modifications are accepted by clicking on the button *Apply*.

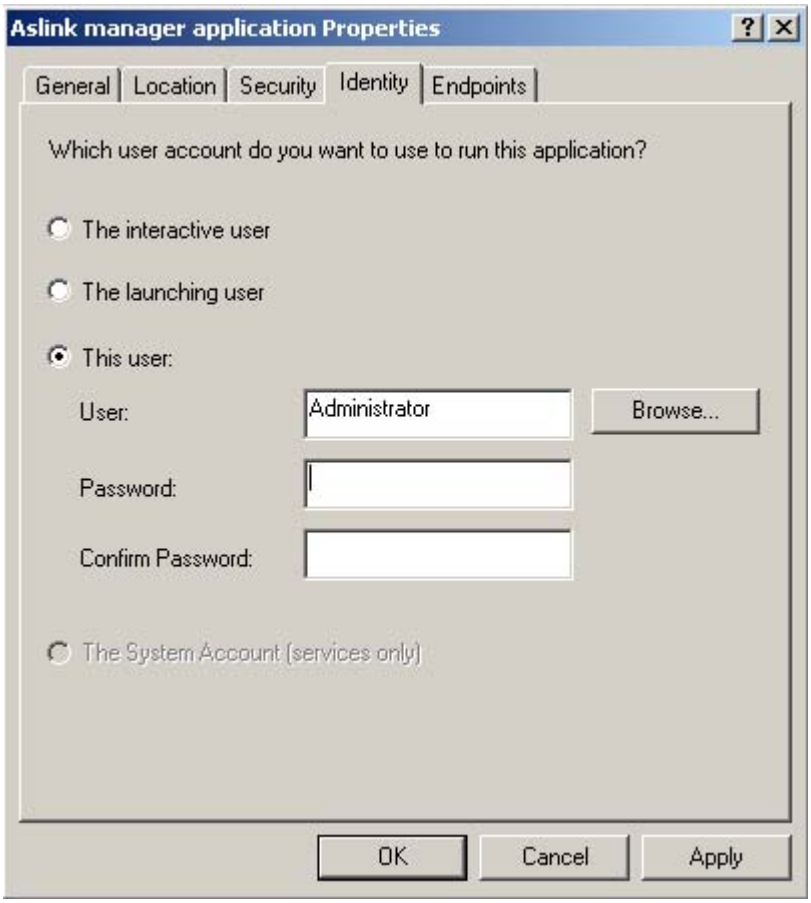


Figure 15. 'Aslink Manager Application Properties' Window - Identity.

6.6. OPC Server

6.6.1. Technical Specification

OPC server of **asix** system comply with the specification *Data Access Custom Interface Standard Version 2.04* available on website *http://www.opcfoundation.org*. This specification is also loaded to subdirectory *Documentation/OPC* during the installation of AsixConnect package.

The *OPC* specification, (*OLE for Process Control*), was developed by OPC Consortium, established by several leading manufacturers developing software for industrial process control. This specification enables development of servers that allow access to the current data retrieved from different systems via one common interface.

OPC server, supplying current data from **asix** system, registers in Windows system under the name *ServerCTOPC.App* and under this name is available for all tools that can apply services of OPC servers.

6.6.2. Details of Implementation

6.6.2.1. Introduction

OPC fully complies with the OPC Specification but this specification includes the optional features that may be implemented by the different methods. Therefore, this chapter is dedicated to description of the implementation methods of these features.

6.6.2.2. OPC Server Object

OPC server (below referred to as server) enabling access to the current data of **asix** system, registers in Windows system under the name *ServerCTOPC.App*. Server registers also in the system registry of objects category as the one complying with specification *OPC Data Access 2.0*. Furthermore, a key *HKEY_CLASSESS_ROOT\ServerCTOPC.App\OPC* is added which enables old versions of OPC clients to identify the server as complying with the OPC specification.

The server is implemented as a local COM out-of-process server, i.e. in the form of EXE file. OPC calls are handled in the main thread of server process. Additional thread operating in background is designed to update the cash memory with the values and status of variables; updating is performed with data transferred to the server by network communication module of **asix** system.

Server is of multiple-use type, i.e. one server process may handle many clients; each client "receives" its own COM object that represents the server. The server implements *APARTMENT_THREADED* thread handling model. In this model, objects are created and handled by main thread of process and all callings to objects are handled in this thread.

Server accepts two languages: Polish (system identifier *langid* 0x0409) and English US (*langid* 0x0415). If the user tries to choose as obligatory the dialect of English other then US (e.g. UK, AUS) then this operation is accepted but is treated as selection of US English. Default language of the server is Polish, if Polish is the language of Windows, or English US in other case.

Server returns OPC_STATUS_NOCONFIG status constants if either no Variable Definitions Database was defined in server options or you failed when trying to load the defined Variable Definitions Database. Then descriptive variable identifiers are handled only.

6.6.2.3. Browsing of Variable Definitions Database

OPC server ensures browsing of variable identifiers available in the server. The server is provided with a hierarchical namespace defined by the selected collection of variables.

The option *OPC attributes displayed as variables* change the way of browsing the Variable Definitions Database. By default, when this option is disabled, the branches of the tree that represent the Variable Definitions Database are the groups of variables, while the leaves are individual variables. If this option is enabled, the internal branches of the tree are the groups of variables, while the external branches are individual variables. The leaves are the attributes of variables, such as value, description, unit, upper range and bottom range.

All types of identifier filtering are implemented i.e.:

- retrieving identifiers that have sub-identifiers;
- retrieving identifiers that have no sub-identifiers;
- retrieving all identifiers from current and lower level.

Handling the filtering criteria, i.e. the pattern with which returned identifiers must be compliant is also implemented. To check compliance with the pattern, function *MatchPattern* working as the *LIKE* function in Visual Basic language is recommended by OPC specification. The table presents special characters allowed in the pattern and possibility of their matching.

Table 30. Special Characters Allowed in the Pattern for the 'MatchPattern' Function.

Character in Pattern	May Be Matched to
?	Any single letter.
*	Zero or more letters.
#	Any single digit (0–9).
[lista_znaków]	Any single character from list of characters.
[!lista_znaków]	Any single character off the list of characters.

Filtering according to variable type was not implemented because server may pass practically every variable in any format if appropriate conversion may be performed by *VariantChangeType* function of Windows. Likewise, filtering according to access rights was also not implemented because access to the information on write permission to the variables was not implemented as yet in **asix** system

According to OPC specification, function *GetItemID* returns for given identifier "fully qualified" identifier in the hierarchical space of identifiers i.e. identifier with its full path. Server does not generate such names and returns only the identifier. Such implementation was chosen because in **asix** system identifiers are unique and providing them with path is not required.

Function *BrowseAccessPath* always returns the information that access paths cannot be found.

6.6.2.4. Browsing Variable Properties

OPC server ensures inspection of properties that are available for the selected variable. For all variables the following properties are available: *Cardinal Type*, *Value*, *Status*, *Time Stamp*, *Access Rights*, *Server Scanning Period*, *Description*. For some variables the following properties may also be available: *Unit*, *Upper Range*, *Lower Range*.

Each property is assigned its code, short text description and property type. Server takes into account the current language during publishing descriptions of properties, i.e. descriptions are in Polish if the current language is Polish, otherwise in English

6.6.2.5. Variable Access Path

An *access path to variable* variable parameter is interpreted as name of **asix** computer, which the variable is to be retrieved from. For description of channel definition, see *3.1. Channels* and further sections.

6.6.2.6. Process Variables

As minimal sampling time was assumed 1 second. If client requires shorter time then sampling time will be increased to the minimal one.

Any type of variable required by client is accepted. If this type is different than that in **asix** system, conversion is performed with use of *VariantChangeType* function of Windows system. If conversion is not possible, the client will receive *OPC_E_BADTYPE* error code.

If added variable need to be active, the server attempts to perform the initialisation action for updating the variable from **asix** system. Result of this action has no effect on result of *AddItem* function.

If an error occurs during initialisation of updating, the server will cyclically try to begin updating until this action will be successful. The current status of variable is returned by functions reading the variable value from server cash memory.

6.6.2.7. Synchronous Operations

The *Read* function is designed to read the values, quality and time stamps of variables that belong to the given group. Server is provided with internal buffer for variable values, from which values are retrieved when they are read from cash memory. Data are read from the device via the network of **asix** system. Time of read operation from the device depends on the time necessary to find the data server of **asix** system. This time is set in the server configuration dialog window and its standard value is equal to 3 seconds. Searching is

performed only once during the first read operation from given resource of **asix** and only if there are no active variables having source of data in this resource.

6.6.2.8. Asynchronous Operations

The asynchronous operations are put in queue and due to this asynchronous operations end their action immediately and client may continue its operation.

Server may put in queue only one operation of the same type at a time, what complies with OPC specification. Server returns error *CONNECT_E_ADVISELIMIT* during an attempt to put in queue more transactions.

In general, server uses the main thread for handling the requests of OPC clients and to send and receive data to/from **asix** system. The operating thread receives and handles messages about changing values of variables. This means that for asynchronous operations, the main thread at first puts into queue operation to be performed and after informing client that operation is accepted, asynchronous operation begins. During operation the main thread is busy and requests to perform other operations for the client wait until the end of the asynchronous operation.

6.6.2.9. Assigning New Value, Quality and Time Stamp

The *Write* function of *ISyncIO* interface was extended, with regard to OPC specification, by the possibility to assign the new value, quality and time stamp at the same time. In order to use this feature, pass the value of *VARIANT* type containing one-dimensional array of 3 items as respective array item of *pItemValues* parameter. The first item of this array contains a new value, the second a quality of variable, and the third item a time stamp. Quality has to be passed in *VARIANT/VT_I4* format or convertible to it. Time stamp have to be passed in *VARIANT/VT_DATE* or convertible to it.

6.7. .NET Server

6.7.1. Application of Server

The *ServerCT* class enables access to the functional part of **asix** system related to the current data. When you are going to perform operations on current data with use of .NET server, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** system from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Generate the project in Visual Studio package and then:
  - highlight the *References* directory in the project tree, select the *Add Reference* command from the *Project* menu, click on *Browse* button and select the

XConnectNet.dll file in the c:\asix subdirectory, click on *OK* and close the *Add Reference* window. In every file with C# source code, the following line should be added in the *using* declaration area:

```
using XConnectNet;
```

- Develop a program operating on *ServerCT*. In this program, you need to:
  - create object of *ServerCT* type, giving as parameter the name of the previously configured channel;
  - using the *Read* and *Write* functions, execute the data exchange;
  - during data exchange, you should remember about handling of exceptions as they may be reported by the server.

6.7.2. ServerCT Designer

```
 [C#]
public ServerCT(
  string channelName);
```

This function is used to create and initiate the object of *ServerCT* class.

As *channelName* parameter, the channel name should be given *(see: 3. Connection Configuration*).

6.7.3. Dispose Function

```
 [C#]
public void Dispose();
```

This function is used for releasing the resources used by *ServerCT* object. This function must be called after use of *ServerCT* object has been finished. Calling should take place from the same thread the object was created in.

6.7.4. Init Function

```
 [C#]
public void Init(
    string initString);
```

This function is designed for setting the server parameters. For description of the function, see *3.5. Program Configuration*, and for available options, see the following sections.

6.7.5. Read Function

```
[C#]
public ItemState Read(
    DataSource dataSource,
    string itemID);
public ItemState[] Read(
    DataSource dataSource,
    string[] itemIDs);
```

```
public DataSet Read(
    DataSource dataSource,
    string[] itemIDs,
    CTDataSetFlags dataSetFlags);
```

Read function is designed for reading the current values of the process variables

DataSource defines the data source. It may take one of the three values described in the table below.

Table 31. Values of the 'DataSource' Parameter for the 'Read' Function (for the .NET Server /Current Data).

Data Source	Description of Data Source	Reading Time	Delay of Variable	Application	Notes
DataSource. Cache	Cache memory of .NET server	Shortest	Highest: asix system sampling rate + time from latest update of server's cache memory	Variables to be read many times and variables active at the moment	Read variable must be active (see: *SetItemActive* function)
DataSource. Device	Cache memory of asix system	Average	Average: asix system sampling rate + network transfer time	Variables to be read small number of times	
DataSource .Driver	Industrial controller	Longest	Smallest: reading time from device + network transfer time	Apply only if value of process variable from this current time is needed.	

In the first version of the *Read* function the second parameter is *itemId*. This input parameter should include identifier of the variable which value is to be read. As a result the function returns the structure of *ItemState* type, which includes the variable status.

In the second version of the *Read* function the second parameter is *itemIDs*. This input parameter should include the table of identifiers of variables, which values are to be read. As a result the function returns the table of structures of *ItemState* type, which include the variable status.

In the third version of the *Read* function, the variable status is returned as an object of *DateSet* class. This object contains one table named *CTData*. The third parameter of the *Read* function is value of *CTDataSetFlags* type.

Table 32. The Value of 'CTDataSetFlags' Type (for the .NET Server / Current Data).

Constant	Description
CTDataSetFlags.Default	The *CTData* table contains columns named: *ItemID*, *ReadResult*, *ErrorString*, *TimeStamp*, *Quality* and *ItemValue*. Each row of the table contains state of one variable. The *ItemValue* column contains values of *Object* type.
CTDataSetFlags.ItemsInColumns	The *CTData* table contains as many columns as variable names are in *itemIDs* parameter. The table contains one row with current values of each variable. The value is of *Object* type.
CTDataSetFlags.ItemValueAsString	Value of variable is turrned back as a text. The constant may be used indyvidually or simultaneously with the constant *CTDataSetFlags.ItemsInColumns*.

NOTE Reading from the cache memory of **asix** system (*dsDevice* parameter equal to *DataSource.Device*) is supported by **asix** system in versions supplied after 2000/06/01 (Netsrv module in v. 3.1.0 or later). In earlier versions, reading from **asix** cache memory is automatically replaced by reading from controller.

6.7.6. Write Function

[C#]
```
public void Write (ItemState[] itemStates);
```

Write function is designed to assign a new value to the process variable.

itemStates parameter is an input-output parameter. Each object in the *itemStates* table should contain in the *name* field the variable name and in the *dataValue* field the value to be assigned to the process variable as a result of execution of the write function. This value may be integer or real number. Together with value of the variable, good quality and time stamp equal to the current time in the operating system (on the server of application of **asix** system) are set.

The write configuration method is described in *6.3.1. Write Function*.

The result of the operation is in the *result* field of the *ItemState* structure. If the *Succeeded()* function of the *ItemState* structure returns value *true,* then the write operation has succeeded.

6.7.7. Write Function – Extended Write Operation

It is possible to assign new value, new status and new time stamp to the process variable at the same time. To do so, in the *ItemState* structure in the *timeStamp* field a new time stamp

of the variable should be set, and in the *quality* field a new value of variable quality should be set.

The write configuration method is described in *6.3.2. Extended Write Function*.

6.7.8. ItemState Structure

Object of *ItemState* type is used for transferring the value of variable. It is applied by *Read* and *Write* functions and *ItemsChange* event.

After execution of the *Read* function has been finished or the *ItemsChange* event is called, the contents of the structure is as follows.

Table 33. The Contents of the 'ItemState' Structure (for the .NET Server / Current Data).

Field	Content
ItemName	Name of variable which state contains an object.
ReadResult	Result of variable read or write. Negative value means error and is simultaneously an aerror code. The 0 or positive value means that read operation has ended with success and fields *TimeStamp*, *Quality* and *ItemValue* are full.
ReadSucceeded()	The function turns back *true* value, if value of *ReadResult* field indicates that read or write operation has ended with success.
GetErrorString()	The function turns back a text description of error code included in *ReadResult* field.
TimeStamp	Time stamp of read variable state. Local time is used.
Quality	Quality of read variable state according to OPC specification. Possible values are described below. The good quality or uncertain means that *ItemValue* field is full.
ItemValue	Variable value. The field is of *object* type and contains value of the type suitable for read variable.
IsQualityGood()	The function turns back *true* value, if the value of *Quality* field indicates that quality of variable state is good and *ItemValue* field is full.

6.7.9. SetItemActive Function

```
 [C#]
public ItemState[] SetItemActive(
    string[] itemIDs,
    bool activeState);
```

SetItemActive function is designed for activating and deactivating the process variable, i.e. adds or removes variable from list of variables refreshed in the cache memory.

ItemIDs is an input parameter and should contain the table of variable identifiers.

ActiveState parameter should contain value *true* if variable is to be activated and value *false* if variable is to be deactivated.

If variable is active then:

- **asix** system sends to object of *ServerCT* class the current value of process variable; the sending time is equal to variable sampling time in **asix** system;
- variable value may be read from cache memory of object of *ServerCT* class with use of *Read* function, passing *DataSource.Cache* constant as *DataSource* parameter;
- variable value is sent automatically to the client with use of *ItemsChange* event if property of object of *ServerCT* class named *Active* is assigned value *true*.

6.7.10. ItemsChange Event

Declaration of event source:

```
 [C#]
public event ItemsChange OnItemsChange;
```

Declaration of delegation, which may be related to source of events:

```
 [C#]
public void ItemsChangeHandler(
    ItemState[] itemsStates);
```

OnItemsChange event is triggered after every change of process variable value. To trigger an event:

- the process variable must be active *(see section 6.7.9. SetItemActive Function)*;
- *Active* server property has to have value *true*.

6.7.11. Active Property

```
 [C#]
public bool Active {get; set;}
```

Active is a read/write property and is designed to control transfer of current values of the active variables from server to client. Variable is active when function *SetItemActive* with *itemIDs* parameter containing the name of the variable and parameter *activeState* equal to *true* was called. In order the current values would be transferred, *Active* property has to take value *true*. Default value of *Active* property is *false*.

6.7.12. Operation in ASP.NET Environment

In case of operation in ASP.NET environment, the object should be created using the *ServerCT.ServerPool.Get()* expression. The *ServerPool* object is a static field of *ServerCT* class and it implements the pool of current data servers. Using the *Get* function, every time before generation of the page the *ServerCT* object should be retrieved. Declaration of the *Get* function is as follows:

```
 [C#]
public ServerCT Get ();
```

After the generation has been completed, the server must be returned to the pool with use of *ServerCT.ServerPool.Release()* expression. As *Release* function parameter the server returned to the pool should be passed. Declaration of the *Release* function is as follows:

```
 [C#]
public Release (ServerCT server);
```

The server may also be taken from the pool in the beginning of each function and returned in the end of the function. To ensure that each server is returned to the pool, the main code of the function must be included in the *try* block and the server should be returned to the pool in the *finally* block.

```
private void Page_Load(object sender, System.EventArgs e)
{
    ServerCT server = null;

    try
    {
        server = ServerCT.ServerPool.Get();

        // function code
  }
  catch(Exception e)
  {
    // handling of exceptions reported when getting the server from the pool and
    // during the function operation
  }
  finally
  {
    if (server != null)
      ServerCT.ServerPool.Release(server);
    }
}
```

Pool of servers*:*

- creates several objects of *ServerCT* class for the application (the channel name is retrieved from *Web.Config* file, see *3.2. How to Specify the Channel Name*);
- stores the objects in cache memory of ASP.NET application;
- makes the objects available for successive calls under ASP.NET application;
- reports *PoolApplicationException* exception when the pool of servers has reached its maximum size and there is no free server.

7. Archive Data

7.1. Identifiers

In the servers of archive data, the variable name itself is sufficient for unique identification of a process variable, i.e. the name used in the application of **asix** system. All other information on the variable needed for communication with **asix** system is retrieved from the Variable Definitions Database. In order to obtain the Variable Definitions Database, you should address to the administrator of application of **asix** system from which the data are to be retrieved.

In the channel options of AsixConnect package, full path to the file containing the collection of variables from the Variable Definitions Database should be determined on an interactive basis or by program. For details, see the chapter on configuration *3.6.2. Introduction.*

In the simplest case, the identifier is the variable name itself. However, identifiers may be more complex. For list of identifiers, see the table below.

Table 34. The List of Identifiers for Unique Identification of a Process Variable in Servers of Archive Data.

Identifier	Description	Possible Variable Type
<*Variable*>	Value of current variable.	R8, STRING
<*Variable*>.*FormatedCV*	Current variable vale formated in accordance with value of *Format* attribute of this variable.	R8, STRING
<*Variable*>.*PercentageCV*	Percent value of current variable.	R8, STRING

STRING type is available in .NET server only.

7.2. Operation without Variable Definitions Database

In general using the Variable Definitions Database is recommended. Using the information contained in this chapter should be exceptional.

AsixConnect server of the archive variables accepts also descriptive identifiers of variables and where you use these identifiers only, loading the Variable Definitions Database is not needed.

During verification of descriptive identifier no communication with **asix** system to perform this verification is needed. Any errors concerning communication with **asix** system, which may occur later, are signalled as a result of read/write operation of variable sample values.

Descriptive identifier takes the following form:

```
aspad.<archive name>.<archive type>.<variable name>
```

where:

- *archive name* – a group name of the archive variables registered in Aspad module of application of **asix** system,
- *archive type* – one letter – type of archive, which variable values are to be read from,
- *variable name* – a name under which process variable is known in application of **asix** system.

asix system in version 2.68, 3.x and 4.x makes the following types of archive available: D, M, Y, H and B.

EXAMPLE

Example of variable identifier:

```
aspad.Camac.D.06C_A111AF00
```

This identifier denotes a process variable named *06C_A111AF00*, accessible in the *Camac* archive. The type of archive is *D*.

NOTE *In **asix** system uppercase and lowercase letters in the names of archives and variables are distinguished.*

7.3. Aggregates

7.3.1. Description of Aggregates

Below is the list of supported aggregates.

Table 35. The List of Supported Aggregates.

English Name	Polish Name	Procedure of Calculation
Start	*Początek*	Value on the beginning of interval.
End	*Koniec*	Value on the end of interval.
Delta	*Przyrost*	Difference of values from end and beginning of interval.
Min	*Min*	Minimum value in interval.
Max	*Max*	Maximum value in interval.
Range	*Zakres*	Difference between maximum end minimum values in interval.

Total	*Suma*	Sum of time weighted values in interval (time integral).
Average	*Średnia*	Average from time weighted values in interval.
Average0	*Średnia0*	Average from time weighted values in interval. For periods when value of variables is not accessible, 0 value is used.
Quality_Good	*Jakość_Dobra*	Percentage of samples with good quality in interval.
Quality_Uncertain	*Jakość_Niepewna*	Percentage of samples with uncertain quality in interval.
Quality_Bad	*Jakość_Zła*	Percentage of samples with bad quality in interval.

Quality and time stamp of aggregate is set depending on value of the *Aggregate calculation algorithm* option. In AsixConnect 4 package, the default algorithm is Askom algorithm; in the previous version the only available one was OPC algorithm.

7.3.2. Askom Algorithm

In *Askom* algorithm, on the basis of parameters of the archive data read function, i.e. *periodStart*, *periodLen* and *resampleInterval,* time stamps and number of aggregates to be read are set. Time stamp of the first aggregate is equal to *periodStart*, of the second – *periodStart* + *resampleInterval* etc.

Then for each aggregate the beginning and duration of data to be read in order to calculate the aggregate is set. To this end, for the first aggregate data are read from the period beginning of which is equal to *periodStart* - *resampleInterval*, for the second aggregate – from the period beginning of which is equal to *periodStart* etc. The exception is the *Start* aggregate where for the first aggregate data are read from the period beginning of which is equal to *periodStart*, for the second aggregate – from the period beginning of which is equal to *periodStart* + *resampleInterval* etc.

Sample quality is assigned value *qualityGood* if per cent of all *qualityGood* samples used to calculate the aggregate is equal to or exceeds the value of the *Good quality threshold* option. By default this threshold is 80%.

7.3.3. OPC Algorithm

In OPC algorithm, for calculation of aggregates the period from which data are retrieved is divided into equal intervals. The aggregate is calculated for every interval using the data archived during this interval. Every interval is assumed to be closed on the left and opened on the right.

Time stamp for each aggregate is the beginning time of interval, with exception of *End* aggregate where time stamp is the end time of interval.

Sample quality will take value *qualityGood* if qualities of all samples used for aggregate calculation have value *qualityGood*. If one or more samples used for calculation of aggregate have qualities that differ from *qualityGood,* then quality of aggregate will take value *qualityUncertainSubNormal*. If status of all samples used in calculation of aggregate have values *qualityBad* then status of aggregate will take value *qualityBad.*

7.4. OPC Time Format

OPC syntax of relative time is as follow:

```
keyword +/- offset +/- offset …
```

Possible values of *keyword* and *offset* parameters are given in table below. Spaces and Tab characters are ignored. Every *offset* parameter has to be preceded with integer number that specifies multiplication factor and direction.

Table 36. Values of 'keyword' for the OPC Time Format.

Keyword	Description
NOW	Current time of archive data server.
SECOND	Beginning of current second.
MINUTE	Beginning of current minute.
HOUR	Beginning of current hour.
DAY	Beginning of current day.
WEEK	Beginning of current week.
MONTH	Beginning of current month.
YEAR	Beginning of current year.

Table 37. Values of 'offset' for the OPC Time Format.

Offset	Description
S	Time offset in seconds.
M	Time offset in minutes.
H	Time offset in hours.
D	Time offset in days.
W	Time offset in weeks.
MO	Time offset in months.
Y	Time offset in years.

For example, the text *DAY -1D+7H30M* might represent beginning time of data for day report generated in current day (*DAY* = first today's time stamp, *-1D* first yesterday's time

stamp, *+7H* means 7:00 hours yesterday, *+30M* means 7:30 hours yesterday; character + in last offset is transferred from previous offset).

Similarly, *MONTH-1D+5h* denotes 5:00 hours of last day of previous month; *NOW-1H15M* denotes an hour and 15 minutes ago and *YEAR+3MO* denotes 1st April of the current year.

In this format, time length may also be given. In this situation, the first member *keyword* in the described format should be omitted.

7.5. Automation Server

Automation server of archive data is in-process server implemented in form of DDL dynamic library and executed in client memory space. It is registered in Windows operating system as an object named *XConnect.ServerHT*. The server also registers in the operating system its own library of types named *AsixConnect 4 Type Library*.

When converting Visual Basic application that uses the AsixConnect package in version 3, the name of the server object *ServerHT.App* should be changed into *XConnect.ServerHT* and the name of the currently used library of types should be changed into *AsixConnect 4 Type Library*.

7.5.1. Application of Server

When you are going to perform operations on the archive data with use of Automation server, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** application from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Develop a program operating on *XConnect.ServerHT* server. In this program, you need to:
  - create an object of *XConnect.ServerHT* type,
  - call the *LoadChannel* function, giving as parameter the name of the previously configured channel,
  - using the *ReadRaw* and *ReadProcessed* functions, execute the data exchange.

All procedure parameters are of VARIANT type.

Rather than using channels, the server parameters may be set (and Variable Definitions Database may be loaded) using the *Init* function.

7.5.2. LoadChannel Function

Function calling syntax:

```
LoadChannel ChannelName
```

This function is used for initialisation of *XConnect.ServerHT* object by loading the channel. As *ChannelName* parameter, the channel name should be given *(see: 3. Connection Configuration).*

7.5.3. Init Function

Function calling syntax:

```
Init InitString
```

This function is designed for setting the server parameters. For description of the function, see *3.5. Program Configuration*, and for available options, see the following sections.

7.5.4. ReadRaw Function

Function syntax:

```
ReadRaw ItemId, DateTimeFrom, DateTimeTo, Data
```

ReadRaw function reads values, statuses and time stamps of variable from archive of specified period. It is designed for clients that want to read real values saved in the archive. Limit values are also supplied to interpolate, when required, values of the variables at the beginning and end of period for which data are to be displayed.

ItemID is an entry parameter; it should contain the variable identifier for which samples are to be read.

DateTimeFrom and *DateTimeTo* are entry parameters, which should contain, respectively, beginning and end of period from which data are to be read. These parameters should contain value of *DATE* or *STRING* type. Value of *DATE* type directly contains time stamp; local time should be used. Value of *STRING* type is considered as relative time. Two formats of relative time are recognised: *OPC* and *asix-raporter*. OPC syntax is described in *7.4. OPC Time Format*. Format *asix-raporter* is described in **asix** system documentation in chapter on *Raporter (asix.hlp)* module.

After the read operation has been finished the *Data* parameter contains array of samples that have been read. Array contains as many rows as number of samples that have been read. Each row contains in first item the time stamp of sample, in second item the quality of sample and in third item the value of sample.

Array returns as well limit values for given period of time. If in archive the sample, corresponding exactly to the time point defined by parameter *DateTimeFrom***,** was not registered, the latest sample before that time is returned. If in archive the sample, corresponding exactly to the time point defined by parameter *DateTimeTo***,** was not registered, the latest sample after that time is returned. If it is not possible to retrieve limit values from the archive, the sample for which quality parameter takes value *qualityBadNoBound* is returned.

7.5.5. ReadProcessed Function

Function syntax:

```
ReadProcessed ItemId, DateTimeFrom, DateTimeTo, AggregateName, ResampleInterval,
Data
```

ReadProcessed function calculates, in defined period of time and for given process value, the aggregates from data included in the archive of **asix** system.

Meaning of *ItemId***,** *DateTimeFrom***,** *DateTimeTo* and *Data* parameters is the same as those of *ReadRaw* function (*see: ReadRow function*).

For how to calculate aggregates, see *7.3. Aggregates*.

AggregateName parameter should contain the name of aggregate.

ResampleInterval parameter should contain the length of interval. The length may be given in the format described in *7.4. OPC Time Format*. The member *keyword* should be omitted.

7.5.6. ShowProgresWindow Property

ShowProgresWindow property is a read/write property. If this property takes *True* value, then during data transmission from archive, the window showing progress of the transmission is displayed. Default value of *ShowProgresWindow* property is *False*.

7.6. OLE DB Server

OLE DB server enables access to data from one or more **asix** systems for clients which can communicate directly with server with use of OLE DB protocol, or the most frequently, indirectly using ADO (ActiveX Data Objects). OLE DB server of **asix** system enables retrieving raw data or those aggregated with use of one of nine aggregating functions. Reading the data is only possible. Modifying this way, either of the existing data or setting the new data is not possible.

Server complies with OLE DB specification of Microsoft but implements the basic functionality of this specification only. Available objects of OLE DB are *data source, session, command* and *rowset.* The query language of the server is *asix.SQL*, which syntax is based on SQL Query Language.

7.6.1. Identification and Configuration

OLE DB of **asix** system registers in operating system under shorted name *ServerHTOleDB.App* and under full name *Askom OLE DB Provider for ASIX*.

The dialog windows given below are displayed by clients of OLE DB with use of the system selection and configuration mechanisms of the server. Some applications may use in this scope their own mechanisms.

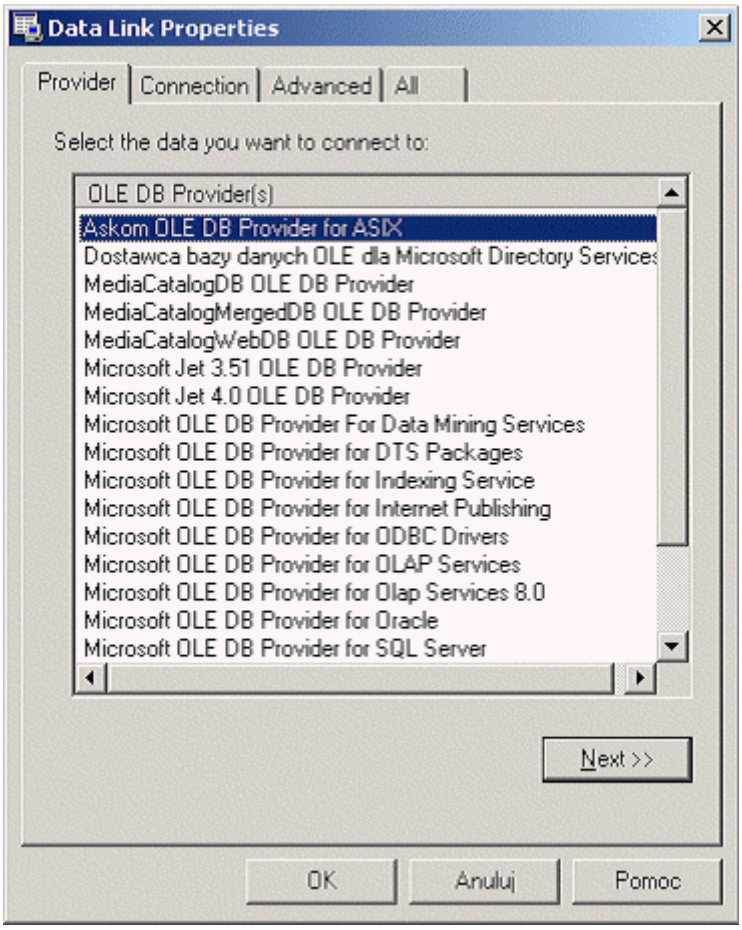


Figure 16. 'Data Link Properties' Window for Parameterization of OLE DB Server.

Server handles the standard parameter named *Data Source*. As *Data Source* parameter full directory path to the Variable Definitions Database should be entered.

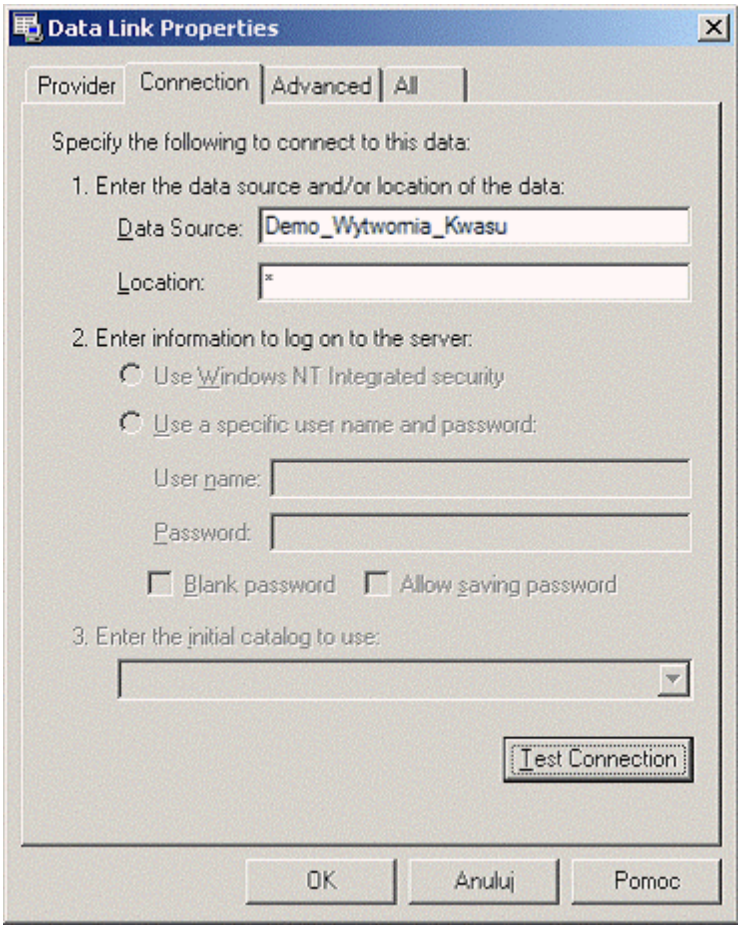


Figure 17. 'Data Link Properties' Window for Parameterization of OLE DB Server.

7.6.2. Tables

Server enables access to one table named *VariablesList*. The table includes two columns named *Name* and *Description*, containing name and description of the variable. The table contains information on all variables in the Variable Definitions Database specified in *Data Source* parameter.

7.6.3. asix.SQL Queries

To describe syntax of *asix.SQL* language, the meta-language applied by Microsoft in documentation of Microsoft SQL 2000 server is used.

Table 38. Characters of Meta Language for asix.SQL Queries.

Characters of Meta Language	Meaning
UPPERCASE LETTERS	Keywords of Asix.SQL language.
Italics	Parameters supplied by user
| (vertical line)	Separates items in parenthesis. Only one item may be selected.
[] (brackets)	Optional item.
{} (curly brackets)	Obligatory item.
[,...n]	Former item may be repeated many times. Repeats should be separated with commas.
[...n]	Former item may be repeated many times. Repeats should be separated with spaces.
bold characters	Text that has to be entered as declared.

asix.SQL query language includes only one SELECT command of the following syntax:

SELECT select_list
FROM var_name
WHERE search_condition
[AGGREGATE aggregate_name [, resample_interva]]
*select_list ::= { * | { column_name [AS column_name_alias]}[,...n] }*
column_name ::= { **TimeStamp** | **Status** | **Value** }
search_condition ::= { time_predicate [AND (value_status_search_condition)] }
time_predicate ::= { **TimeStamp** BETWEEN *date* AND *date* }
value_status_search_condition ::=
{ [NOT] *predicate* | (*value_status_search_condition*) }
[{ AND | OR }
[NOT] { *predicate* | (*value_status_search_condition*) }] [*...n*] }
predicate ::=
{ **Status** { = | < > | ! = } s*tatus_value*
| **Value** { = | < > | ! = | > | > = | ! > | < | < = | ! < } *number*
| **Value** IS [NOT] NULL }
s*tatus_value* ::= { **qualityGood | qualityUncertain | qualityUncertainSubNormal |**
qualityBad | qualityBadNoData | qualityBadNoBound }
aggregate_name ::= { **Start** | **End** | **Delta** | **Min** | **Max** |
Range | **Total** | **Average** | **Average0 }**

In *SELECT* clause value * means that all three columns in order *TimeStamp, Quality, Value* are defined. As option, *column_name_alias* may be defined as alternative name, replacing in query the column name. Alternative name must not be used in clause WHERE. If alternative name includes spaces or non-alphanumeric characters it should be closed in brackets.

In the result of query:

- column *TimeStamp* will contain time stamp of the sample;
- column *Quality* will contain time quality of the sample;

- column *Value* will contain value of the sample provided that quality of the sample is good. If quality of the sample takes value *qualityBad*, *qualityBadNoData* or *qualityBadNoBound*, value of the sample will be equal to NULL.

In FROM clause as *var_name* parameter pass the name of one variable of **asix** system. It is possible to use descriptive identifier (see: *7.2. Operation without*). The descriptive identifier should be put in quotation marks.

In WHERE clause as *date* parameter pass the string closed in quotation marks containing:
- absolute date in international format, declared in operating system;
- absolute date in ODBC *yyyymmddhhnnss* format;
- relative date in format described in section *7.4. OPC Time Format***.**

As *number* pass integer or real number in decimal format. Decimal separator is point.

In AGGREGATE clause as *aggregate name* parameter pass the name of one of the available aggregating method described in section *7.3. Aggregates*. As *sample rate* pass the period of sampling of the variable. The sampling period should be given in the format described in section *7.4. OPC Time Format* with member *keyword* omitted. When you omit the *sample rate* parameter, sampling period will be that as defined in **asix** system. If clause AGGREGATE is omitted, the raw values will be read.

If clause AGGREGATE is used, the format of read data is the same as given in description of *ReadProcessed* function *(7.5.5. Introduction)* of Automation server of archive data. If clause AGGREGATE is omitted, format of read data is the same as given in description of *ReadRaw* function (*7.5.4.ReadRaw Function).*

7.6.4. Examples of Queries

Read raw values of variable K8_11U14 from date 2002-06-16 and time from 8:00 to 8:10 hours. Time is given in Polish format.

```
select Timestamp, Value from K8_11U14
where Timestamp between ’2002-06-16 8:00:00’
                and ’2002-06-16 8:10:00’
```

Read raw values of variable K8_11U14 from date 2002-06-16 and time from 8:00 to 8:10 hours. Time is given in ODBC format. Time stamp, quality and value of the sample are returned.

```
select * from K8_11U14
where Timestamp between ’2002-06-16 8:00:00’
                and ’2002-06-16 8:10:00’
```

Read raw values of variable K8_11U14 from the previous hour. Variable name in example is closed in brackets. Brackets should be used when variable name includes characters other than letters, digits and underscore.

```
select Timestamp, Value from [K8_11U14]
where Timestamp between ’hour-2h’ and ’hour-1h’
```

Read 5-minute average values of variable from the current hour.

```
select * from K8_11U14
where Timestamp between ’hour-1h’ and ’hour’
aggregate Average, ’5m’
```

Read minimum values of variable for one-hour periods from the previous day.

```
select * from K8_1 U14
where Timestamp between 'day-1d' and 'day'
aggregate Min, '1h'
```

Read minimum values of variable for one-hour periods from the previous day.

```
select * from K8_11U14
where Timestamp between 'day-1d' and 'day'
aggregate Min, '1h'
```

Read 5-minute average values of variable from the previous day. Select sample values less than 66 only.

```
select * from K8_11U14
where Timestamp between 'day-1d' and 'day' and value < 66
aggregate Average, '5m'
```

Read 5-minute average values of variable from the current hour. Select samples of good quality only.

```
select * from K8_11U14
where Timestamp between 'hour' and 'hour+1h'
      and quality = qualityGood
aggregate Average, '5m'
```

Read 5-minute average values of variable from the previous and current hour. Select samples which value is accessible (quality is not bad).

```
select * from K8_11U14
where Timestamp between 'hour-1h' and 'hour+1h'
      and value is not null
aggregate Average, '5m'
```

Read raw values for K8_11U14 from 2002-06-16 for period from 8:00 to 16:00. Select samples with values less than 63 or higher than 73.

```
select Timestamp, Value from K8_11U14
where Timestamp between '2002-06-16 8:00:00'
                and '2002-06-16 9:00:00'
      and (value < 63 or value > 73)
```

7.7. .NET Server

7.7.1. Application of Server

The *ServerHT* class enables access to the functional part of **asix** system related to the archive data. When you are going to perform operations on archive data with use of *ServerHT*, you should carry out the following steps.

- Install AsixConnect package.
- Obtain the Variable Definitions Database of **asix** system from which the current data are to be retrieved or generate such a base.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Generate the project in Visual Studio package and then:
  - highlight the *References* directory in the project tree, select the *Add Reference* command from the *Project* menu, click on *Browse* button and select the *XConnectNet.dll* file in the *c:\asix* subdirectory, click on *OK* and close the *Add*

Reference window. In every file with C# source code, the following line should be added in the *using* declaration area:

```
using XConnectNet;
```

- Develop a program operating on *ServerHT*. In this program, you need to:
  - create object of *ServerHT* type, giving as parameter the name of the previously configured channel;
  - using the *ReadRaw* and *ReadProcessed* functions, execute the data exchange;
  - during data exchange, you should remember about handling of exceptions as they may be reported by the server.

7.7.2. ServerHT Designer

```
 [C#]
public ServerHT(
  string channelName);
```

This function is used to create and initiate the object of *ServerHT* class.

As *channelName* parameter, the channel name should be given (*see: 3. Connection Configuration*).

7.7.3. Dispose Function

```
 [C#]
public void Dispose();
```

This function is used for releasing the resources used by *ServerHT* object. This function must be called after use of *ServerHT* object has been finished. Calling must take place from the same thread the object was created in.

7.7.4. Init Function

```
 [C#]
public void Init(
    string initString);
```

This function is designed for setting the server parameters. For description of the function, see *3.5. Program Configuration*, and for available options, see the following sections.

7.7.5. ReadRaw Functions

```
 [C#]
public ReadRawResult ReadRaw (
    string itemID,
    DateTime periodStart,
    TimeSpan periodLen);
```

ReadRaw function is used to reads archive, raw values of process variables from specified period. It is designed for clients that want to read real values saved in the archive. The function supplies limit values to interpolate, when required, values of the variables at the beginning and end of period for which data are to be displayed.

itemID is an entry parameter; it should contain the variable identifier for which samples are to be read.

periodStart and periodLen are entry parameters, which should contain, respectively, beginning and length of period from which data are to be read.

After the read operation has been finished, the result is returned in the form of object of *ReadRawResult* class.

The function returns as well limit values for given period of time. If in archive the sample, corresponding exactly to the time point defined by parameter *dateTimeFrom*, was not registered, the latest sample before that time is returned. If in archive the sample, corresponding exactly to the time point defined by parameter *dateTimeTo*, was not registered, the latest sample after that time is returned. If it is not possible to retrieve limit values from the archive, the sample for which quality parameter takes value *Quality Bad – No Bound* is returned.

7.7.6. ReadProcessed Functions

```
[C#]
public ReadProcessedResult ReadProcessed (
    string itemID,
    Aggregate itemAggregate,
    DateTime periodStart,
    TimeSpan periodLen,
    TimeSpan resampleInterval);

public ReadProcessedResult[] ReadProcessed (
    string[] itemIDs,
    Aggregate[]itemAggregates,
    DateTime periodStart,
    TimeSpan periodLen,
    TimeSpan resampleInterval);

public void ReadProcessed(
    string[] itemIDs,
    Aggregate[]itemAggregates,
    DateTime periodStart,
    TimeSpan periodLen,
    TimeSpan resampleInterval,
    HTDataSetFlags dataSetFlags,
    out bool allOk,
    out DataSet dataSet);
```

ReadProcessed function calculates, in defined period of time and for given process value, the aggregates (such as beginning or average value) from data included in the archive of **asix** system.

itemID is an entry parameter; it should contain the variable identifier for which samples are to be read. In the second and third version of the function, *itemIDs* parameter is used and its value should be table of variable identifiers.

The *itemAggregate* parameter should contain type of aggregate to be used for processing of raw archive values. In the second and third version of the function, *itemAggregates*

parameter is used and its value should be table of aggregates. For how to calculate aggregates, see *7.3. Aggregates*. List of supported aggregates is given in the table below.

Table 39. The List of Handled Aggregates Calculated by the 'ReadProcessed' Function (Using .NET Server / Archive Data).

Identifier	Method of Aggregate Calculation
Aggregate.Start	Value on the beginning of interval.
Aggregate.End	Value on the end of interval.
Aggregate.Delta	Difference of values from end and beginning of interval.
Aggregate.Min	Minimum value in interval.
Aggregate.Max	Maximum value in interval.
Aggregate.Range	Difference between maximum end minimum values in interval.
Aggregate.Total	Sum of time weighted values in interval (time integral).
Aggregate.Average	Average from time weighted values in interval.
Aggregate.Average0	Average from time weighted values in interval. For periods when value of variables is not accessible, 0 value is used.

periodStart and *periodLen* are entry parameters, which should contain, respectively, beginning and length of period from which data are to be read.

resampleInterval is output parameter and should contain length of interval for which the aggregates are calculated.

The third version of *ReadProcessed* function contains the *dataSetFlags* parameter. As its value constant of *HTDataSetFlags* type should be passed.

Table 40. Constants of 'HTDataSetFlags' Type for the 'ReadProcessed' Function (for the .NET Server / Archive Data).

Constant	Description
HTDataSetFlags.Default	There is one column, named as variable name, created in *HTData* table for each value in *itemIDs* parameter. If variable name recurs, then suffixes '-1', '-2', etc are added to successive repetitions. Values of sample variables will be inserted into this column. If sample quality is good, the DBNull value is put into the column. The HTData table also includes one column named TimeStamp with time stamp of samples.
HTDataSetFlags.ShowQuality	For all variable (without value column) will be inserted the column with sample quality into *HTDataSet* table. This column has a name composed of variabe name with '-Quality' text added.
HTDataSetFlags. ShowErrorStringWhenReadError	There is an error description inserted into the column, if read error occurs. If the flag is not used, null value is inserted.

After the read operation has been finished, the result is returned in the form of object of *ReadProcessedResult* class for the first version of the function or in the form of array of objects of *ReadProcessedResult* class for the second version of the function. The third version of the function contains two output parameters: *allOk* and *dataSet*. The *allOk* parameter determines whether the read operations for all variables were successful. The *dataSet* parameter returns the object of *DataSet* class containing the read archive data.

7.7.7. ReadProcessedAsString Function

```
[C#]
public ReadProcessedAsStringResult ReadProcessedAsString (
    string itemID,
    Aggregate itemAggregate,
    DateTime periodStart,
    TimeSpan periodLen,
    TimeSpan resampleInterval);

public ReadProcessedAsStringResult[] ReadProcessedAsString (
    string[] itemIDs,
    Aggregate[]itemAggregates,
    DateTime periodStart,
    TimeSpan periodLen,
    TimeSpan resampleInterval);

public void ReadProcessedAsString (
    string[] itemIDs,
    Aggregate[]itemAggregates,
    DateTime periodStart,
    TimeSpan periodLen,
    TimeSpan resampleInterval,
    HTDataSetFlags dataSetFlags
    out bool allOk,
    out DataSet dataSet);
```

ReadProcessedAsString functions are mainly used for reading formatted values of variable samples.

Operation of the *ReadProcessedAsString* function as well as its parameters are the same as in the *ReadProcessed* function. The only difference is that the result is returned in the form of *ReadProcessedAsStringResult* structures for the first version of the function or in the form of array of *ReadProcessedAsStringResult* structures for the second version of the function.

The third version of the function contains two output parameters: *allOk* and *dataSet*. The *allOk* parameter determines whether the read operations for all variables were successful. The *dataSet* parameter returns the object of *DataSet* class containing the read archive data.

7.7.8. RelativeDateTime Function

```
 [C#]
public DateTime RelativeDateTime (
    string time,
    DateTime now);
```

The *RelativeDateTime* function converts the time given in relative format into absolute time.

7.7.9. RelativeTimeSpanm Function

```
 [C#]
public TimeSpan RelativeTimeSpan (
    string time,
    DateTime now);
```

The *RelativeTimeSpan* function converts the period length given in relative format into absolute length.

7.7.10. ReadRawResult Classt

The object of *ReadRawResult* class is designed for transferring the result of execution of the *ReadRaw* function.
Declaration of the *ReadRawResult* class is as follows:

```
public class ReadRawResult
{
    public bool ReadSucceeded();
    public Int32 ReadResult;
    public string ErrorString;

    public string ItemID;
    public DateTime PeriodStart;
    public TimeSpan PeriodLen;

    public ItemSample []Samples;
};
```

After execution of the archive data reading function has been finished, the contents of the structure is as follows.

Table 41. The Contents of the Structure for the 'ReadRawResult' Class Declaration (for the .NET Server / Archive Data).

Field	Content
ReadSucceeded()	The function turns back *true* value, if value of *ReadResult* field indicates that read or write operation has ended with success.
ReadResult	Code of read operation end. The value less then 0 denotes an error.
ErrorString	Textual description code of read operation end.
ItemID	Name of the variable which the read operation refers to.
PeriodStart	Beginnig of the period from which archived data were read.
PeriodLen	Span of the perod from which archived data were read.
Samples	Read archival data.

7.7.11. ReadProcessedResult Class

The object of *ReadProcessedResult* class is designed for transferring the result of execution of the *ReadProcessed* function.

Declaration of the *ReadProcessedResult* class is as follows:

```
public class ReadProcessedResult
{
    bool ReadSucceeded();
    Int32 ReadResult;
    string ErrorString;

    string ItemID;
    Aggregate ItemAggregate;

    DateTime PeriodStart;
    TimeSpan PeriodLen;
    TimeSpan ResampleInterval;

    ItemSample []Samples;
};
```

After execution of the archive data reading function has been finished, the contents of the structure is as follows.

Table 42. The Contents of the Structure for the 'ReadProcessedResult' Class Declaration (for the .NET Server / Archive Data).

Field	Content
ReadSucceeded()	The function turns back *true* value, if value of *ReadResult* field indicates that read or write operation has ended with success.
ReadResult	Code of read operation end. The value less then 0 denotes an error.
ErrorString	Textual description code of read operation end.
ItemID	Name of the variable which read operation refers to.
ItemAggregate	Name of variable aggregate.
PeriodStart	Beginnig of the period from which archived data were read..
PeriodLen	Span of the perod from which archived data were read.
ResampleInterval	Interval span, for which aggregates are calculated.
Samples	Read archival data.

7.7.12. ReadProcessedAsStringResult Class

The object of *ReadProcessedAsStringResult* class is designed for transferring the result of execution of the *ReadProcessedAsString* function.

Declaration of the *ReadProcessedAsStringResult* class is as follows:

```
public class ReadProcessedAsStringResult
{
    public bool ReadSucceeded();
    public Int32 ReadResult;
```

```
    public string ErrorString;

    public string ItemID;
    Aggregate ItemAggregate;

    public DateTime PeriodStart;
    public TimeSpan PeriodLen;
    public TimeSpan ResampleInterval;

    public ItemStringSample []Samples;
};
```

The meaning of the fields is the same as in the *ReadProcessedResult* class. The only difference is type of the *Samples* field.

7.7.13. ItemSample Structure

The objects of the *ItemSample* structure are designed for transferring the values of archive variables. They are used by the *ReadRawResult* and *ReadProcessedResult* classes.

Declaration of the *ItemSample* structure is as follows:

```
public struct ItemSample
{
    public DateTime TimeStamp;
    public Int32    Quality;
    public double   ItemValue;

    public bool     IsQualityGood();
};
```

After execution of the archive data reading function has been finished, the contents of the structure is as follows.

Table 43. The Contents of the 'ItemSample' Structure (for the .NET Server / Archive Data).

Field	Content
TimeStamp	Time stamp of variable sample; local time is used.
Quality	Quality of variable sample. Good or uncertain quality means that *ItemValue* field is filled.
ItemValue	Variable values; the field is of *double* type.
IsQualityGood()	Function turns back *true* value, if value of Quality field indicates variable state quality to be good

The sample quality may take one of the values described in section *5.1. Measurement Quality*. In addition, in case of bad quality the flags specific to archive data may be set. These flags are within the range of bit numbers 16-31.

Table 44. The Flags Specific to Archive Data in Case of Bad Quality for the 'ItemSample' Structure (for the .NET Server / Archive Data).

Bit Value	Definition	Description
0x00100000	*Quality Bad – No Bound*	The limitary value can not be passed because the archive is not available for the time moment responded this limitary time moment.
0x00200000	*Quality Bad – No Data*	Process variable value is not available because the archive with data from the precise time period is not available.
0x01000000	*Asix Archive End*	Process variable value is not available because it found the end of archive when reading the data (reading the data from the period that does not occur yet or in the case of the lack of timer synchronization between a client computer and the **asix** system station)..

7.7.14. ItemStringSample Structure

The object of *ItemStringSample* type is designed for transferring the values of archive variables as texts. The *ItemStringSample* structure is used by the *ReadProcessedAsStringResult* class.

Declaration of the *ItemStringSample* structure is as follows:

```
public struct ItemStringSample
{
    public DateTime TimeStamp;
    public Int32    Quality;
    public string   ItemValue;

    public bool     IsQualityGood();
};
```

After execution of the archive data reading function has been finished, the contents of the structure is as follows:

Table 45. The Contents of the 'ItemStringSample' Structure (for the .NET Server / Archive Data).

Field	Content
TimeStamp	Time stamp of variable sample; local time is used.
Quality	Quality of variable sample. Good or uncertain quality means that *ItemValue* field is filled.
ItemValue	Variable values; the field is of *string* type.
IsQualityGood()	Function turns back *true* value, if value of *Quality* field indicates variable state quality to be good

7.7.15. DataSet Object

The object of *DataSet* class is designed for returning archive data by the *ReadProcessed* and *ReadProcessedAsString* functions. The *DataSet* objects contain two tables: *HTData* and *ReadResult*.

The *HTData* table contains archive data. The first column of the table is named *TimeStamp* and contains time stamp of the samples. For every value in the *itemIDs* parameter of the *ReadProcessed* function one column of values with the name identical to the variable name is created in the *HTData* table. If the variable name repeats, the repeated names are suffixed – '-1', '-2', etc. In this column, the values of variable samples are placed. If sample quality is not good, the value *DBNull* is placed in the column. In the table, for every variable the column of sample quality may be inserted too. The name of this column consists of the variable name with suffix '-*Quality*'.

In case of the *ReadProcessed* function the *HTData* table contains data in the floating-point format; in case of the *ReadProcessedAsString* function the *HTData* table contains text-type data.

The *ReadResult* table contains information on readout operation for individual variables. The table contains the following columns described below.

Table 46. Columns of the 'ReadResult' Table for the 'DataSet' Object (for the .NET Server / Archive Data).

Field	Content
ColumnID	Name of *HTData* table column referred to given variable
ItemID	Name of the variable which read operation refers to.
ItemAggregate	Name of variable aggregate.
PeriodStart	Beginnig of the period from which archived data were read.
PeriodLen	Span of the perod from which archived data were read.
ResampleInterval	Interval span, for which aggregates are calculated.
ReadResult	Code of read operation end. The value less then 0 denotes an error.
ErrorString	Textual description code of read operation end.

7.7.16. Operation in ASP.NET Eenvironment

In case of operation in ASP.NET environment, the object should be created using the *ServerHT.ServerPool.Get()* expression. The *ServerPool* object is a static field of *ServerHT* class and implements the pool of current data servers. Using the *Get* function, every time before generation of the page the *ServerHT* object should be retrieved. Declaration of the *Get* function is as follows:

```
 [C#]
public ServerHT Get ();
```

After the generation has been completed, the server must be returned to the pool with use of the *ServerHT.ServerPool.Release()* expression. As *Release* function parameter the server returned to the pool should be passed. Declaration of the *Release* function is as follows:

```
 [C#]
public Release (ServerHT server);
```

The server may also be taken from the pool in the beginning of each function and returned in the end of the function. To ensure that each server is returned to the pool, the main code of the function must be included in the *try* block and the server should be returned to the pool in the *finally* block.

```
private void Page_Load(object sender, System.EventArgs e)
{
    ServerHT server = null;

    try
    {
        server = ServerHT.ServerPool.Get();

        // function code, e.g.:
        ChartXMLData.InsertChartHTML ("chart.tee", Chart1);
        ChartXMLData chartXMLData = new ChartXMLData();

        chartXMLData.Add (serverHT, "KW_A000", Aggregate.Start, "KW_A000",
                          HTChartFlags.Default, "1M");
        chartXMLData.Add (serverHT, "KW_A008", Aggregate.Average, "KW_A000",
                          HTChartFlags.Default, "1M");

        chartXMLData.SetChartPeriod (serverHT, "DAY-24H", "24H");
        chartXMLData.ReadData (serverHT, XMLIsland1);
    }
    catch(Exception e)
    {
        // handling of exceptions reported when getting the server from the pool
and
        // during the function operation
    }
    finally
    {
        if (server != null)
            ServerHT.ServerPool.Release(server);
    }
}
```

Pool of servers*:*

- creates several objects of *ServerHT* class for the application (the channel name is retrieved from *Web.Config*file, see *3.2. How to Specify the Channel Name*);
- stores the objects in cache memory of ASP.NET application;
- makes the objects available for successive calls under ASP.NET application;
- reports *PoolApplicationException* exception when the pool of servers has reached its maximum size and there is no free server.

8. Alarms

8.1. .NET Server

8.1.1. Application of Server

The *ServerAL* class enables access to the functional part of **asix** system related to alarms. When you are going to operate on alarms with use of *ServerAL*, you should carry out the following steps.

- Install AsixConnect package.
- Using *Configurator* program, configure the basic channel or establish and configure your own channel.
- Generate the project in Visual Studio package and then:
  - highlight the *References* directory in the project tree, select the *Add Reference* command from the Project menu, click on *Browse* button and select the *XConnectNet.dll* file in the *c:\asix* subdirectory, click on *OK* and close the *Add Reference* window. In every file with C# source code, the following line should be added in the *using* declaration area:

    ```
    using XConnectNet;
    ```

- Develop a program operating on *ServerAL*. In this program, you need to:
  - create object of *ServerAL* type, giving as parameter the name of the previously configured channel,
  - using the *ReadActive* and *ReadHistorical* functions, program the data exchange;
  - during data exchange, you should remember about handling of exceptions as they may be reported by the server.

8.1.2. ServerAL Designer

```
 [C#]
public ServerAL(
  string channelName);
```

This function is used to create and initialise the object of *ServerAL* class.

As *channelName* parameter, the channel name should be given (*see: 3. Connection Configuration*).

8.1.3. Dispose Function

```
 [C#]
public void Dispose();
```

This function is used for releasing the resources used by *ServerAL* object. This function must be called after use of *ServerAL* object has been finished. Calling must take place from the same thread the object was created in.

8.1.4. Init Function

```
 [C#]
public void Init(
    string initString);
```

This function is designed for setting server parameters. For description of the function, see *3.5. Program Configuration*, and for available options, see the following sections.

8.1.5. ReadActive Functions

```
 [C#]
public Alarm[] ReadActive();
public Alarm[] ReadActive(
    AlarmType alarmType,
    AlarmStatus alarmStatus,
    string textMask,
    int[] groups);
```

The *ReadActive* function is designed for retrieving information on active alarms in **asix** system application.

The first version of the *ReadActive* function returns all active alarms. The second version of the *ReadActive* function enables filtration of retrieved active alarms by giving the type of alarms, status of alarms, mask of alarm text and tables of alarm group numbers.

The *alarmType* parameter defines the required type of alarm. As its value one of the constants described in the table below or the bit sum of these constants should be given.

Table 47. Constants of the 'alarmType' Parameter for the 'ReadActive' Function for Retrieving the Information on Alarms.

Constant	Description
AlarmType.NoFiltering	Filtering by alarm type is switched off
AlarmType.System	System alarms
AlarmType.Message	Messages
AlarmType.Warning	Warnings
AlarmType.Alarm	Alarms
AlarmType.ImportantAlarmr	Important alarms
AlarmType.All	Bit sum of all types of alarms

The *alarmStatus* parameter defines the required status of alarm. As its value one of the constants described in the table below or the bit sum of these constants should be given.

Table 48. Constants of the 'alarmStatus' Parameter for the 'ReadActive' Function for Retrieving the Information on Alarms.

Constant	Description
AlarmStatus.NoFiltering	Filtering by alarm status is switched off
AlarmStatus.Started	Alarm started
AlarmStatus.Fished	Alarm finished
AlarmStatus.Acknowledged	Alarm acknowledged
AlarmStatus.NotAcknowledged	Alarm not acknowledged
AlarmStatus.All	Bit sum of all types of alarm statuses

If the *alarmsStatus* parameter is to include one of the *Started*, *Finished*, *Acknowledged* or *NotAcknowledged* flags, you should remember to **simultaneously** use at least one of the *Started* or *Finished* flags and at least one of the *Acknowledged* or *NotAcknowledged* flags.

As *textMask* parameter you should give the mask of alarm text, i.e. a fragment of the text that must appear in the alarm description so that the *ReadActive* function will return the alarm. The pattern is case-sensitive. The pattern may include special characters '*' and '?'. The '*' character means that any number of characters may appear in its place in the alarm description. The '?' character means that any one character may appear in its place in the alarm description. The most typical patterns are:

text – corresponds to alarms which description starts with *text*,
**text* – corresponds to alarms which description includes *text* in any place.

As *groups* parameter the table of the alarm group numbers should be given. The table may include up to 10 elements. If empty table or *null* value is given as *groups* parameter, then filtration by groups will not be applied.

8.1.6. ReadHistorical Functions

```
[C#]
public Alarm[] ReadHistorical (
    DateTime periodStart,
    TimeSpan periodLen,
    int maxNumberOfAlarms);
public Alarm[] ReadHistorical (
    DateTime periodStart,
    TimeSpan periodLen,
    int maxNumberOfAlarms,
    AlarmType alarmType,
    AlarmStatus alarmStatus,
    string textMask,
    string idRange,
    int[] groups);
```

The *ReadHistorical* function is designed for retrieving information on historical alarms registered in the alarm archive of **asix** system application. A part of parameters of these

functions is the same as parameters of the *ReadActive* function described in the previous section.

The *periodStart* and *periodLen* parameters to define the period from which alarms are to be retrieved.

The *maxNumberOfAlarms* parameter allows the maximum number of retrieved alarms to be limited.

The *idRange* parameter allows the range of retrieved alarms to be limited only to those defined precisely by numbers. You may declare the list of numbers separated with comma (e.g. 2,34,789), the range of alarm numbers (e.g. 3-128,300-572) or combine both the methods for specification of alarms to be displayed (e.g. 2,4,5-89).

8.1.7. Alarms2DataSet Function

```
 [C#]
public static DataSet Alarms2DataSet(
    Alarm[] alarms,
    bool ascending);
```

The *Alarms2DataSet* function allows the object of *DataSet* class to be created on the basis of the data stored in the table of structures of *Alarm* type.

The table of structures of *Alarm* type is usually the result of execution of the *ReadActive* or *ReadHistorical* function. This table should be passed as the first parameter of the *Alarms2DataSet* function.

The second parameter of the *Alarms2DataSet* function named *ascending* defines whether the data about alarms will be entered into *DataSet* object as sorted out in the ascending or descending order. Sorting out takes place by alarm time.

The resulting object of *DataSet* type contains one table named *ALData*.

8.1.8. Alarm Structure

The objects of *Alarm* structure are designed for transferring the values of alarms. They are used by the *ReadActive* and *ReadHistorical* functions.

Declaration of the *Alarm* structure is as follows:

```
[C#]
public struct Alarm
{
  public int Id;
  public String Text;
  public DateTime TimeStamp;
  public AlarmType AlarmType;
  public AlarmStatus AlarmStatus;
};
```

After execution of the *ReadActive* or *ReadHistorical* functions has been finished, the contents of the *Alarm* structure is as follows:

Table 49. The Contents of the 'Alarm' Structure.

Field	Content
Id	Alarm number
Text	Alarm text
TimeStamp	Time stamp; local time is used
AlarmType	Alarm type
AlarmStatus	Alarm status

8.1.9. Operation in ASP.NET Environment

In case of operation in ASP.NET environment, the object should be created using the *ServerAL.ServerPool.Get()* expression. The *ServerPool* object is a static field of *ServerAL* class and implements the pool of current data servers. Using the *Get* function, every time before generation of the page the *ServerAL* object should be retrieved. Declaration of the *Get* function is as follows:

```
 [C#]
public ServerAL Get ();
```

After the generation has been completed, the server must be returned to the pool with use of the *ServerAL.ServerPool.Release()* expression. As *Release* function parameter the server returned to the pool should be passed. Declaration of the *Release* function is as follows:

```
 [C#]
public Release (ServerAL server);
```

The server may also be taken from the pool in the beginning of each function and returned in the end of the function. To ensure that each server is returned to the pool, the main code of the function must be included in the *try* block and the server should be returned to the pool in the *finally* block.

```
private void Page_Load(object sender, System.EventArgs e)
{
    ServerAL server = null;

    try
    {
        server = ServerAL.ServerPool.Get();

        // function code
  }
  catch(Exception e)
  {
    // handling of exceptions reported when getting the server from the pool and
    // during the function operation
  }
  finally
  {
    if (server != null)
      ServerAL.ServerPool.Release(server);
    }
}
```

Pool of servers:

- creates several objects of *ServerAL* class for the application (the channel name is retrieved from *Web.Config* file, *see: 3.2. How to Specify the Channel Name*);
- stores the objects in cache memory of ASP.NET application;

- makes the objects available for successive calls under ASP.NET application;
- reports the *PoolApplicationException* exception when the pool of servers has reached its maximum size and there is no free server.

9. Reports

9.1. .NET Server

9.1.1. Application of Server

The *ServerRP* class enables access to the reports generated in the **asix** system. When you are going to operate on the .NET server, you should carry out the following steps.

- Install the AsixConnect package.
- Using the *Configurator* program, configure the basic channel or establish and configure your own channel.
- Generate the project in the Visual Studio package and then:
  - highlight the *References* directory in the project tree, select the *Add Reference* command from the *Project* menu, click on the *Browse* button and select the *XConnectNet.dll* file in the *c:\asix* subdirectory, click on *OK* and close the *'Add Reference'* window. In every file with the C# source code, the following line should be added in the *using* declaration area:

    ```
    using XConnectNet;
    ```

- Develop a program operating on ServerRP. In this program, you need to:
  - create object of *ServerRP* type, giving as parameter the name of the previously configured channel;
  - using the server function retrieve the information on reports and reports themselves;
  - during retrieving the information, you should remember about handling of exceptions as they may be reported by the server.

9.1.2. Configuration of Report Definition Files

For proper reporter operation you should configure the files of report definition. It consists in adding the row of information including a name of report group and duration of report generation into these files. This information is used by the functions retrieving informations on reports and definition files (described below).

The full syntax of the information row is as follows:

comment_character REPORT_INFO=GROUP:Name; PERIOD:Length

where:
comment_character - depending on the definition file format, it is necessary to use a proper comment character in accordance with the following table;

Table 50. Comment Characters Used for Configuration of Report Definition Files.

File Type (Extension)	Comment Character
File of report definition (*.r)	/
VBScript (*.vbs)	‘ or //
JavaScript (*.js)	//

Name - name of a report group (it corresponds to tree nodes);
Length – duration of report generation (it corresponds to „leaves” of a tree). The following values are acceptable:

- D - day
- W - week
- M - month
- Q - quarter
- Y - yer
- O - other

EXAMPLE

/ REPORT_INFO = GROUP:Zduny; PERIOD:D

9.1.3. ServerRP Designer

```
[C#]
public ServerRP(
  string channelName);
```

This function is used to create and initialise the object of the *ServerRP* class.

As *channelName* the channel name should be passed (*see: 3. Connection Configuration*).

9.1.4. Dipsose Function

```
[C#]
public void Dispose();
```

This function is used for releasing the resources used by the *ServerRP* object. The function must be called after the use of a *ServerRP* object has been finished. Calling must take place from the same thread the object was created in.

9.1.5. Init Function

```
[C#]
public void Init(
    string initString);
```

The function is designed for setting server parameters. For description of the function, see: *3.5. Program Configuration*, and for available options, see the following sections.

9.1.6. GetReportsInfo Function

```
[C#]
public ReportInfoNet[] GetReportsInfo(
    string group,
    int period,
    bool lastReportsOnly);
```

The *GetReportsInfo* function is designed for retrieving the information on reports generated in an **asix** system application. The result of function operation is returned in the *ReportInfoNet* structure.

The *group* parameter determines the group the reports will be read from. Passing the „*" value will cause readout of all reports (apart from the *period* parameter value).

The *period* parameter defines the report period. The following values of a report period are possible.

Table 51. Values of a Report Period.

Value	Description
-1	Any reports
0	Twenty-four hours reports
1	Weekly reports
2	Monthly reports
3	Quarterly reports
4	Annual reports
5	Reports with indefinite period

The *lastReportsOnly* parameter determines whether all reports or only one (the latest) have to be returned in case when there are a few reports applying to the same period.

9.1.7. ReadReportsInfo Function

```
[C#]
public DataSet ReadReportsInfo(
    string group,
    int period,
    bool lastReportsOnly);
```

The *ReadReportsInfo* function operates in the same way as the *GetReportsInfo* function. The difference is that informations on reports are returned in the *DataSet* class object.

The result object of *DataSet* type includes one table named *ReportsInfo*.

9.1.8. GetDefFilesInfo Function

```
[C#]
public DefFileInfoNet[] GetDefFileInfo();
```

The *GetDefFileInfo* function is designed for retrieving the information on all files of report definitions. The result of the function operation is returned in the *DefFileInfoNet* structure.

9.1.9. ReadDefFilesInfo Function

```
[C#]
public DefFileInfoNet[] GetDefFileInfo();
```

The operation of the *ReadDefFilesInfo* function is the same as the *GetDefFilesInfo* one. The difference is that informations on reports are returned in the *DataSet* class object.

The result object of *DataSet* type includes one table named *DefFilesInfo*.

9.1.10. GetReportsDirectoryPath Function

```
[C#]
public string GetReportsDirectoryPath();
```

The function returnes the path to the application report directory.

9.1.11. ReportInfoNet Structure

The *ReportInfoNet* structure is designed for storing the information on reports.

The structure declaration is as follows:

```
public struct ReportInfoNet
{
    String    FileName;
    String    ReportName;
    DateTime  ReportDate;
    DateTime  FileDate;
    long      FileSize;
    int       ReportFormat;
    int       ReportType;
    int       ReportPeriod;
    String    ReportGroup;
};
```

After the operation of the function for reading the information on reports has been finished, the structure has the following content:

Table 52. Contents of the 'ReportInfoNet' Structure.

Field	Content
FileName	Name of a report file
ReportName	Report name
ReportDate	Report date
FileDate	Date of report file creation
FileSize	File size
ReportFormat	Format of report file (TXT/HTML)
ReportType	Report type (script/disk)
ReportPeriod	Report period (for more information see: *GetReportsInfo Function*)
ReportGroup	Report group

9.1.12. DefFileInfoNet Structure

The *DefFileInfoNet* structure is designed for storing the information on files of report definitions. The structure declaration is as follows:

```
public struct DefFileInfoNet
{
    String    FileName;
    int       ReportPeriod;
    String    ReportGroup;
};
```

The structure stores the information on name, period and group of report definition file. The available values of *ReportPeriod* you can find in *9.1.6. GetReportsInfo Function*.

9.1.13. Operation in ASP.NET Environment

In case of operation in ASP.NET environment, the object should be created using the *ServerRP.ServerPool.Get()* expression. The *ServerPool* object is a static field of the *ServerAL* class and implements the pool of current data servers. Using the *Get* function, every time before generation of the page the *ServerRP* object should be retrieved. Declaration of the *Get* function is as follows:

```
[C#]
public ServerRP Get ();
```

After the generation has been completed, the server must be returned to the pool with use of the *ServerRP.ServerPool.Release()* expression. As the *Release* function parameter the server returned to the pool should be passed. Declaration of the *Release* function is as follows:

```
[C#]
public Release (ServerRP server);
```

The server may also be taken from the pool in the beginning of each function and returned in the end of the function. To ensure that each server is returned to the pool, the main code of the function must be included in the *try* block and the server should be returned to the pool in the *finally* block.

```
private void Page_Load(object sender, System.EventArgs e)
{
    ServerRP server = null;

    try
    {
        server = ServerRP.ServerPool.Get();

        // function code
    }
    catch(Exception e)
    {
        // handling of exceptions reported when getting the server from the pool
and
        // during the function operation
    }
    finally
    {
        if (server != null)
            ServerRP.ServerPool.Release(server);
    }
}
```

Pool of servers:

- creates several objects of the *ServerRP* class for the application (the channel name is retrieved from the *Web.Config* file, see: *3.2. How to Specify the Channel Name*);
- stores the objects in cache memory of an ASP.NET application;
- makes the objects available for successive calls under an ASP.NET application;
- reports the *PoolApplicationException* exception when the pool of servers has reached its maximum size and there is no free server.

10. Web Service Server

The *Web Service* server of AsixConnect package provides access to full functionality of **asix** system application via *XML Web Services* protocol.

10.1. Installation

The *WebService* server is located in *c:\asix\WebService* directory. In order to make it available within the network, run the *Web information services* program. This program is available in *Start/Control panel/Administrating tools*. In the program window, highlight the *Default website* item. Select *Virtual directory* from the *Action/New* menu. Wizard is started. As *Alias* you should enter the text *WebService* and as *Directory* – *c:\asix\WebService*. The other options are to be left unchanged. By default, on attempt of getting access to the server, Windows authentication is used. In order to enable the anonymous access, highlight the newly created virtual directory, select *Properties* from the *Action* menu, open the *Protections of directories* tab, click on *Edit* in the *Anonymous access* field, and enable the *Anonymous access* option.

Since then the Web Service server is available from:
http://localhost/WebService/XConnectWebService.asmx.

See http:// localhost /WebService/XConnectWebService.asmx?WSDL for description of the server services in WSDL language.

The Web Service server provides data on the local level only. In order to be able to provide data via the local area network or Internet, you need to purchase the *asix4Internet* licence.

10.2. Web.Config Configuration File

For storage of default channel name the Web Service server uses the configuration file named *Web.Config*. This file located in c*:\asix\WebService* directory. Method of defining the channels is described in section *3.1. Channels*.

In order to define the default channel name, you should create the *appSettings* element in *Web.Config* file in the superior element. Then, one *add* element should be created in the *appSettings* element and two attributes should be defined in it. The first attribute should be named *key* and assigned the value *"DefaultChannelName"*. The latter attribute should be named *value* and assigned the channel name as the value. The channel name should be put in quotation marks.

EXAMPLE

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="DefaultChannelName" value="AsEmis" />
  </appSettings>
...
```

If the *WebService* server supply data for dynamic HTML pages, than the *Control variable* and *Limits of variables* options should be enabled in the channel.

10.3. Clients

10.3.1. Internet Explorer

The simplest way to test the Web Service server for correct operation is to use the Internet Explorer browser as the server client. After the server is configured, run the browser and open the site:
http://localhost/WebService/XConnectWebService.asmx.

The loaded page contains the list of all functions made available by the server. After a function is selected, information on this function is displayed. In addition, for some of the functions the fields where you may enter the function parameters and the *Call* button are displayed.

Using the Internet Explorer browser, you may call only these functions that use simple types in parameters and results. After a function is selected and the *Call* button is pressed, a new page containing the result of execution of the function is opened. The result is given in XML format. The functions that may be called using a browser are: *Read, ReadRaw*, *ReadProcessed*, *ReadActiveAlarms*, and *ReadHistoricalAlarms*. For parameters that require the date, you should use the *yyyy/mm/ddThh:mm:ss* format (for example, *2004/07/13T01:00:00)*.

10.3.2. WebForm Application

First of all, the WebForm Visual Studio 2003 application, which is to use the *WebService* server, must contain the reference to this server. To this end, after the project is generated, you need to:

- highlight the *Reference* directory in the project tree;
- select Add Web Reference from the *Project* menu;
- write URL address from which the *WebService* server is available, i.e. *http://localhost/WebService/XConnectWebService.asmx* and click on *Go*;
- pass *Asix* as the reference name and click on *Add Reference*.

The *WebService* server requires that *cookie* mechanism should be enabled. To enable this mechanism, after the server object of *Asix.XConnectWebService* type is created in the application, you should assign the object of *System.Net.CookieContainer* type to its *CookieContainer* properties.

```
Asix.XConnectWebService server;
server = new Asix.XConnectWebService();
server.CookieContainer = new System.Net.CookieContainer();
```

The object of *Asix.XConnectWebService* class contains all functions of the *WebService* server. The following example presents the operation of reading the current value of KW_A110 variable from the server.

```
Asix.ItemStateWS itemStateWS = server.Read("KW_A110");
if (itemStateWS.ReadResult >= 0)
{
    // readout succeeded
    // value
    textBox1.Text = itemStateWS.ItemValues[0].ToString ();
    // quality
    textBox2.Text = "0x" + System.Convert.ToString (itemStateWS.Quality,
16).PadLeft (4,'0');
    // time stamp
    textBox3.Text = itemStateWS.TimeStamp.ToString();
}
else
{
    // readout ended with error
    textBox1.Text = itemStateWS.ErrorString;
    textBox2.Text = textBox3.Text = "";
}
```

10.4. Variable Definitions Database

```
 [C#]
string[] ReadAttributes(
    string varName,
    string[] attributeNames);
string[][] ReadAttributesN(
    string[] varNames,
    string[] attributeNames);
```

The *ReadAttributes* function is used for reading out the values of variable attributes. Its operation is the same as of the *ReadAttributes* function of *ServerHT* class (*see: 4.2.1.5. ReadAttributes Function*).

The *ReadAttributesN* function is used for reading the values of attributes of many variables. Its operation is the same as of the *ReadAttributesN* function of *ServerHT* class (*see: 4.2.1.6. ReadAttributesN Function*).

10.5. Current Data

```
 [C#]
ItemStateWS Read(
    string   itemID);
ItemStateWS[] ReadN(
    string[] itemIDs);
```

The *Read* and *ReadN* functions are designed for reading current values of process variables. As parameter of the *Read* function the variable name is assigned and the function returns the value of this variable in the form of *ItemStateWS* structure. As parameter of the *ReadN* function the table of variable names is assigned and the function returns the value of these variables in the form of the table of *ItemStateWS* structures.

ItemStateWS Structure

Objects of *ItemStateWS* type are used for transferring values of variables. The *ItemStateWS* structure is used by the *Read* and *ReadN* functions.

Declaration of the *ItemStateWS* structure is as follows:

```
[C#]
public struct ItemStateWS
{
  public String ItemID;
  public Int32 ReadResult;
  public bool ReadSucceeded;
  public String ErrorString;

  public DateTime TimeStamp;
  public Int32 Quality;
  public Object ItemValues;
};
```

After execution of the *Read* function has been finished, the contents of the structure is as follows.

Table 53. Contents of the 'ItemStateWS' Structure (Web Service Server).

Field	Content
ItemID	Variable name
ReadResult	Result of variable read. Negative value means error and is simultaneously an aerror code. The 0 or positive value means that read operation has ended with success and fields *TimeStamp*, *Quality* and *ItemValue* are filled.
ReadSucceeded	The function turns back *true* value, if value of *ReadResult* field indicates that read operation has ended with success
ErrorString	Textual description of error code included in *ReadResult* field.
TimeStamp	Time stamp of measurement. Local time is used.
Quality	Measurement quality
ItemValues	Values of measurement. The field is of *object type* and includes value table of type that corresponds to read variable. Most often it is 1-element table. Only in case of tables or when reading the variable including *BarCVs* phraze in its identifier, the field includes multi-element table.

10.6. Raw Archive Data

```
 [C#]
ReadRawResult ReadRaw(
    string itemID,
    DateTime periodStart,
    int periodLenS);
```

The *ReadRaw* function is designed for reading raw archive values of process variables from a specified period. Its operation is the same as of the *ReadRaw* function of *ServerHT* object described in section *7.7.5. ReadRaw Functions*.

10.7. Aggregated Raw Data

```
[C#]
ReadProcessedResult ReadProcessed(
    string itemID,
    Aggregate itemAggregate,
    DateTime periodStart,
    int periodLenS,
    int resampleIntervalS);
ReadProcessedResult [] ReadProcessedN(
    string[] itemIDs,
    Aggregate[] itemAggregates,
    DateTime periodStart,
    int periodLenS,
    int resampleIntervalS);
```

The *ReadProcessed* function calculates aggregates for given process variable in defined period. Its operation is the same as of the *ReadProcessed* object of *ServerHT* class (*see: 7.5.5. ReadProcessed Function)*. It only differs in method of giving the length of period and the length of interval – these values should be given in seconds.

The *ReadProcessedN* function operates in the same way as the *ReadProcessed* function, but it allows data concerning many variables and their aggregates to be retrieved at one call. For example, in order to retrieve minimum, maximum and average values of a variable at one call, as the first parameter you should pass the table containing threefold the same variable name, and as the second parameter – the table containing identifiers of relevant aggregates.

In order to achieve the maximum efficiency of reading two or more aggregates of the same variable, you should group the values of the *itemIDs* parameter according to variable names.

10.8. Active Alarms

```
[C#]
Alarm[] ReadActiveAlarms();
Alarm[] ReadActiveAlarmsEx(
    Severity severity,
    Status status,
    string textMask,
    int[] groups);
```

The *ReadActiveAlarms* and *ReadActiveAlarmsEx* functions are designed for retrieving information on active alarms in **asix** system application. The operation of the *ReadActiveAlarms* function is the same as of the parameterless *ReadActive* function of *ServerAL* class.

The operation of the *ReadActiveAlarmsEx* function is the same as of the 4-parameter *ReadActive* function of *ServerAL* class.

10.9. Historical Alarms

```
[C#]
Alarm[] ReadHistoricalAlarms(
    DateTime periodStart,
    int periodLenS,
    int maxNumberOfAlarms);
Alarm[] ReadHistoricalAlarmsEx(
    DateTime periodStart,
    int periodLenS,
    int maxNumberOfAlarms,
    Severity severity,
    Status status,
    string textMask,
    string idRange,
    int[] groups);
```

The *ReadHistoricalAlarms* and *ReadHistoricalAlarmsEx* functions are designed for retrieving information on historical alarms in **asix** system application.

The operation of the *ReadHistoricalAlarms* function is the same as of the 3-parameter *ReadHistorical* function of *ServerAL* class. The operation of the *ReadHistoricalAlarmsEx* function is the same as of the 8-parameter *ReadHistorical* function of *ServerAL* class (*see: 8.1.6. ReadHistorical Functions*). It only differs in method of giving the length of period – this value should be given in seconds.

11. Diagnostics of Server Operation

11.1. Logs

During operation, the servers write to their log files information on the beginning and the end of their operation and on the serious errors encountered during their operation. Filename of this log is:

```
<identifier>.<current date>.log
```

As *identifier* the following values may passed.

Table 54. Values of 'identifier' Used for Log Names (Diagnostics of Server Operation).

Server Type	Identifier
All Automation servers	*XConnect*
DDE of current data	*ServerCTDDE*
DDE of current data – service	*ServiceCTDDE*
OPC of current data	*ServerCTOPC*
All .NET servers, WebService server	*XConnectNet*
OLEDB of archived data	*ServerHTOLEDB*

As <*current data*> date of log creation in *YYYYMMDD* format is used. If the log file becomes bigger than 10 MB, it will be closed at midnight of the current day and a new log file with new name will be created. In case of lack of disk memory, the old log files are automatically removed.

Example name of the log created the 28 August 2001 by Automation server of current data:

```
ServerCTOPC.20010828.log
```

By default, log files are saved in *Log* subdirectory of the same directory where AsixConnect package was installed; the most frequently in c*:\asix\Log*. Directory in which log files are saved may be changed using *Configurator* program. See *3.4.3. Package Options*.

11.2. Error Codes

Every function that is made available by OPC server of AsixConnect package returns 0 if operation was performed successfully. If operation was performed successfully, but there is additional information on function execution, a positive value is returned. In case of error a negative value is returned.

Error codes returned by OPC server are given in the table below.

Table 55. Error Codes Returned by OPC Server.

Name of Error Code	Error Code	Description
ASKOM_E_NetInit	0xC0048100L	Error in initiating network of ASIX system
ASKOM_E_NetInitTimeout	0xC0048101L	Timeout in waiting on initiating the network of ASIX.
ASKOM_E_NetInstallClient	0xC0048102L	Error during initiating the client of network of ASIX system
ASKOM_E_NetDeinstallClient	0xC0048103L	Error during closing the client of network of ASIX system
ASKOM_E_NetFindServer	0xC0048104L	Function calling error *Searching of data server*
ASKOM_E_NetFindServer_NotFound	0xC0048105L	Data server in network of ASIX system was not found
ASKOM_E_NetLinkToServer	0xC0048106L	Error when connecting to data server of system ASIX
ASKOM_E_NetLinkToServerTimeout	0xC0048107L	Timeout during connecting to data server of system ASIX
ASKOM_E_NetLinkToServerNegativeAnswer	0xC0048108L	Error during connecting to data server of system ASIX – negative response
ASKOM_E_NetCloseLink	0xC0048109L	Error during closing connection with data server of ASIX system
ASKOM_E_NetSendData	0xC004810aL	Data transmission error in network of ASIX system.
ASKOM_E_NetAnswerTimeout	0xC004810bL	Timeout in waiting on response in network of ASIX system
ASKOM_E_ItemUnknown	0xC0048120L	Error – variable unknown in ASIX system.
ASKOM_E_TheSameItemWrittenMoreThanOnce	0xC0048121L	Errors in parameters during write operation. Variable is written more then once.
ASKOM_E_WriteError	0xC0048128L	ASIX system returned error during attempt of write operation
ASKOM_E_WriteError_NetworkServerNotFound	0xC0048129L	ASIX system returned error during attempt of write operation – server servicing the variable was not found
ASKOM_E_WriteError_IllegalCommand	0xC004812aL	ASIX system returned error during attempt of write operation – device regarded command as non acceptable
ASKOM_E_WriteError_DriverTimeout	0xC004812bL	ASIX system returned error during attempt of write operation - timeout during writing to device
ASKOM_E_WriteError_DriverTransmisionError	0xC004812cL	ASIX system returned error during attempt of write operation – transmission error during communication with device
ASKOM_E_WriteError_DeviceRequestIllegal	0xC004812dL	ASIX system returned error during attempt of write operation – request regarded as non acceptable.
ASKOM_E_GroupItemsCantBeUsed	0xC0048139L	Function doesn't handle identifiers

		of variable groups.
ASKOM_E_ErrorInitializingHaspKey	0xC0048047L	Error during initiating HASP key
ASKOM_E_HaspNotFound	0xC0048048L	HASP key was not found.
ASKOM_E_HaspConstructorExpired	0xC004804AL	Time in designer mode expired
ASKOM_E_VarBaseNotFound	0xC004a001L	No base of variables in given directory
ASKOM_E_ErrorDuringOpeningVarBase	0xC004a006L	Error during opening base of variables
ASKOM_E_ErrorDuringOpeningVarBase _SharedMemLocation	0xC004a008L	Error during opening base of variables – incorrect parameter SharedMemLocation of BDE
ASKOM_S_DataTransmisionInterruptedByUser	0x00048140L	Data transmission broken by user
ASKOM_E_ErrorDuringReceivingArchiveVarInformation	0xC0048141L	Error during retrieving information on archive variable
ASKOM_E_ErrorDuringOpeningArchiveVar	0xC0048142L	Error during opening archive variable
ASKOM_E_ErrorDuringClosingArchiveVar	0xC0048143L	Error during closing archive variable
ASKOM_E_ErrorDuringSeekingArchiveVarData	0xC0048144L	Error during data searching of archive variable
ASKOM_E_ErrorDuringReadingArchiveVarData	0xC004815L	Error during reading archive
ASKOM_E_ErrorDuringConfirmingReadingArchiveVarData	0xC0048146L	Acceptance error during reading archive variable

11.3. DDE Server

If the script developed in *VisualBasic* executes the read operation from DDE server that has been suddenly closed, Excel function returns error 2023.

12. Examples

After installation of program in target directory (default c*:\asix*) in subdirectory AC4Examples you will find examples of application of AsixConnect package.

All examples use *Acid_Manufacturing_Plant* demo application installed in *c:\Asix\Applications\Acid_Manufacturing_Plant* directory. This application may be run by selection the *Acid_Manufacturing_Plant* from *Start/Programs/asix/Demo* menu. Application should be started before loading the examples.

Examples of application of Automation, DDE and OLE DB servers are written in spreadsheet in format of Excel 97/2002. Majority of examples include macros developed in Visual Basic. To see the macros load the spreadsheet, select command *Visual Basic Editor* from *Tools/Macros* menu or click on keyboard shortcut *Alt-F11*. The examples use AsixConnect package channels named *Demo_ Acid_Manufacturing_Plant* and *Demo_ Acid_Manufacturing_PlantDDE4*. These channels are created during installation of **asix** package.

12.1. Automation server – Current Data and Variable Definitions Database

This example demonstrates access method to the current data and Variable Definitions Database of **asix** system application with use of Automation server.

In this spreadsheet connection to library of Automation server is defined. Thanks to this, macros can directly make use of objects of Automation server of AsixConnect package. Apart from the easy programming, maximum communication efficiency was obtained. In order to verify current links, select command *References* from the *Tools* menu of Visual Basic Editor. Name of Automation server library reads *AsixConnect 4 Type Library*.

In *ReadWrite* sheet there is a table containing names of four variables, their values, status and time stamps. In addition, there are *Read*, *Write*, *Select name* i *Show description* buttons.

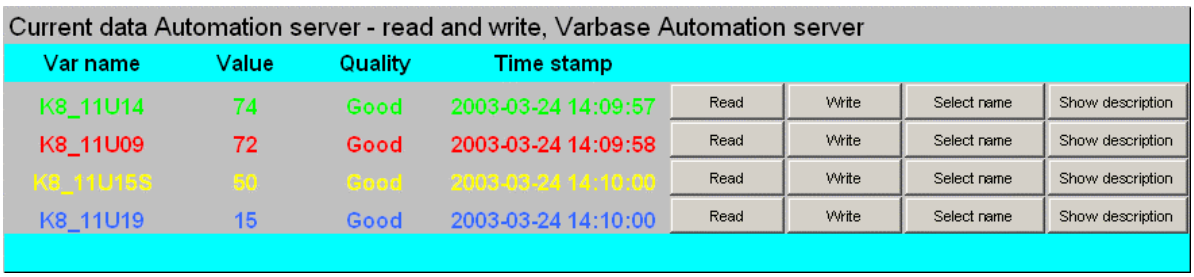


Figure 18. Automation Server – Current Data and Variable Definitions Database / Examplary Application.

Clicking on Read button corresponding to the selected variable a macro is called that retrieves value of the variable using the Automation server of current data and writes it to the table. If you enter a number to *Value* field and click on *Write* button, number will be send to **asix** system as new value of the variable. Note: before writing, check if conditions needed to perform write operation to applications of **asix** system, described in section *6.3.1. Write Function* are observed.

Clicking on *Select name* a macro is called that displays the variable selection window using the Automation server of Variable Definitions Database. If you select the name of variable and close the window by clicking on OK, the selected name appears in the *Variable name* column. Clicking on *Show description* opens the window displaying the variable description retrieved from the Variable Definitions Database.

In *DataChange* tab you may find example of sending the current data to client with use of Automation server. When you click on *Start* button the dialog window is displayed:

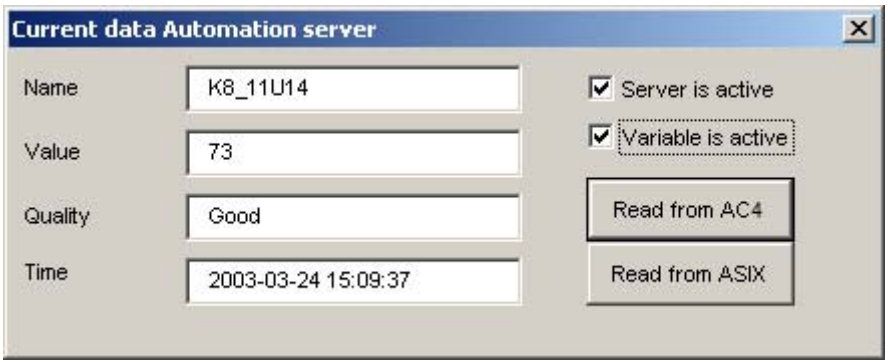


Figure 19. Automation Server – Current Data and Variable Definitions Database – Examplary Application.

If you check *Variable is Active* option box, variable will be sent from application of **asix** system to Automation server. This variable may be read by clicking on *Read from AC4* button. If the option box is not checked, direct readout from **asix** system is active only (*Read from ASIX* button). If in addition option *Server is active* is checked, the variable is also sent from Automation server to Visual Basic application. Triggered event (*asix_DataChange* procedure) writes retrieved variable value to dialog box

12.2. Automation Server – Current Data – Late Binding

This example demonstrates access method to current data of application of **asix** system with use of Automation server. It is functionally identical to previous example *Automation server – current data* but the early binding to Automation system, by established connection to library of Automation server, is not used. To create Automation object, the *CreateObject* function in place of *New* operator, was used. This method of object creating is only one available in *VBScript* language.

12.3. DDE Server – Current Data

This example demonstrates access method to current data of application of **asix** system with use of DDE server. To run the example, start DDE server before.

In *ReadWrite* tab of sheet there are two tables. The first includes names of four variables and their values and *Read* and *Write* buttons. The latter includes, in addition columns for status and time stamps.

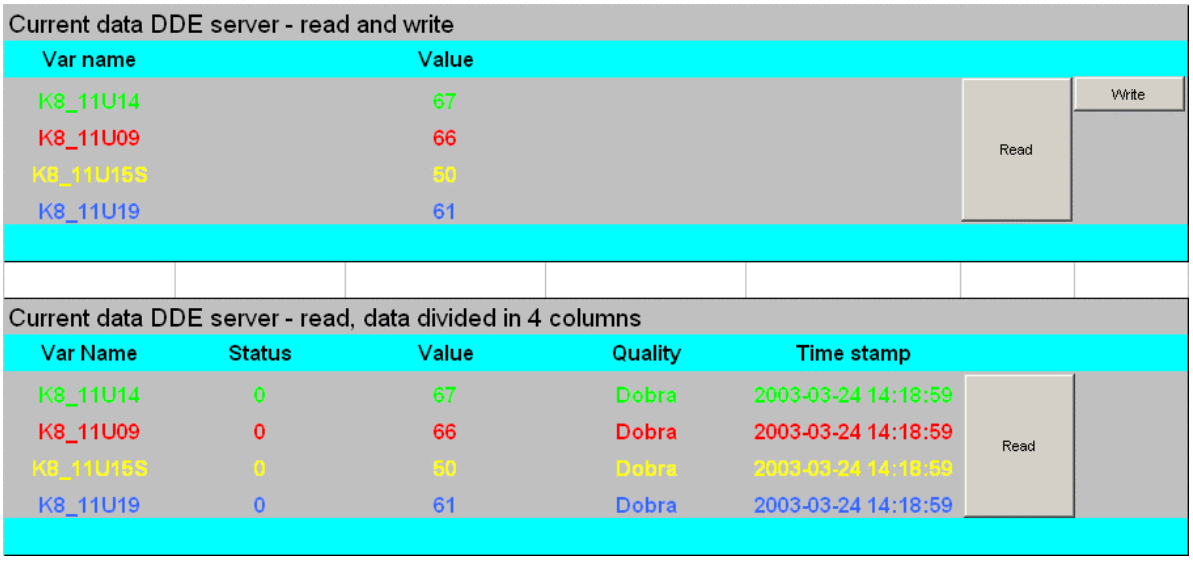


Figure 20. DDE Server – Current Data / Examplary Application.

Clicking on *Read* button in the first table, a macro is called, which retrieves values of all variables with use of Automation server of current data and writes them to the table. Data are retrieved in single-column format. If you write number to *Value* column in row of *KW_A110* variable and click on *Write* button, this number will be sent to **asix** system as a new value of variable. Note: before writing, check if conditions needed to perform write operation to applications of **asix** system, described in section *6.3.1. Write Function*, are observed.

Clicking on *Read* button in the second table, a macro is called, which retrieves values of all variables with use of Automation server and writes them to the table. The data being retrieved in four-column format are split on status, value, quality and time stamp.

12.4. DDE Server – Current Data - Updating

This example demonstrates access method to current data of application of **asix** system with use of DDE server by sending current value of variable to DDE client. In column G of the sheet there are formulas that are references to remote data accessed with use of DDE server. To run the example, start DDE server before.

Formulas in column G are references to single variables with except of formulas contained in rows 46-49, that is a group formula and references to four variables.

12.5. OPC Server – Current Data

The example of OPC server application is not included to the package. You may not use OPC server in Visual Basic macro.
You may get acquainted with features of OPC server by downloading the free *OPC Client* program from *http://www.matrikon.com/drivers/* website. The other steps to see the status of variables are as follows:

- Configure the basic channel with use of AsixConnect package program named *Configurator*,
- Run the *Matrikon OPC Explorer* program,
- Check *ServerCTOPC.App* and select command *Connect* from the *Server* menu.
- Select command *Add Group* from the *Server* menu, give any name of group and click on *OK*,
- Select *Add Items* from the *Group* menu,
- Place any number of variables in the *Tags To Be Added* window and select command *Update and Return to Explorer* from the *File* menu,
- Upon a while information on status of selected variables is displayed in the main window.

12.6. Automation Server – Archive Data

This example demonstrates method of access to archive data of **asix** system applications with use Automation server. Example needs Microsoft Excel 2000 or XP system for proper operation.

In this spreadsheet connection to library of Automation server is defined. Thanks to this, macros can directly make use of objects of Automation server of AsixConnect package. Apart from the easy programming, maximum communication efficiency was obtained. In order to check what are the current links, select command *References* from menu tools of Visual Basic. Name of Automation server library reads *AsixConnect 4 Type Library*.

In *Hour Aggregates* Tab of the sheet, there is an example report containing hourly aggregates.

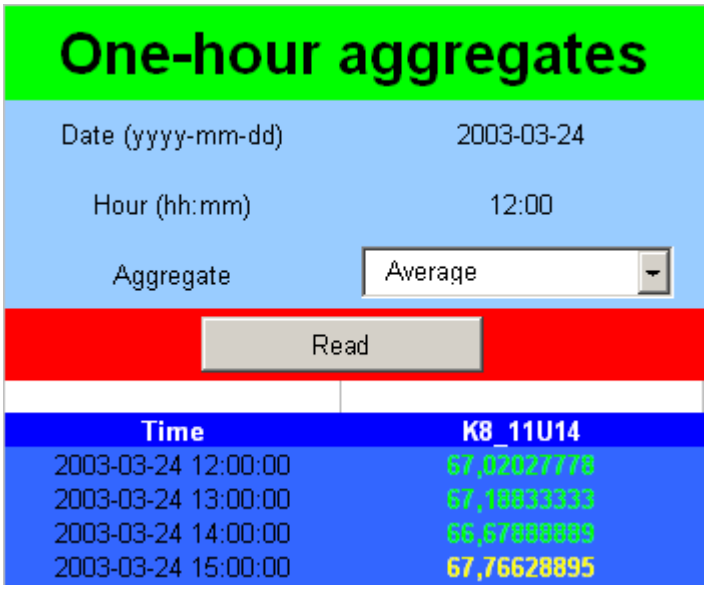


Figure 21. Automation Server – Archive Data / Examplary Application.

In this example, in cell C6 you can find report generation date, in cell C7 the beginning hour of report. C8 cell contains list including available aggregates. As default *Average* is selected. After clicking *Read* button**,** macro is called, which starts Automation server and calls function of retrieving the treated data – aggregates. The results are written to the range B12-C35.

In *Latest Quarter* tab of the sheet, there is an example of report containing 10-second aggregates calculated from data of last 15 minutes. In this example, to specify time period in the function that retrieves treated data, the relative time is used.

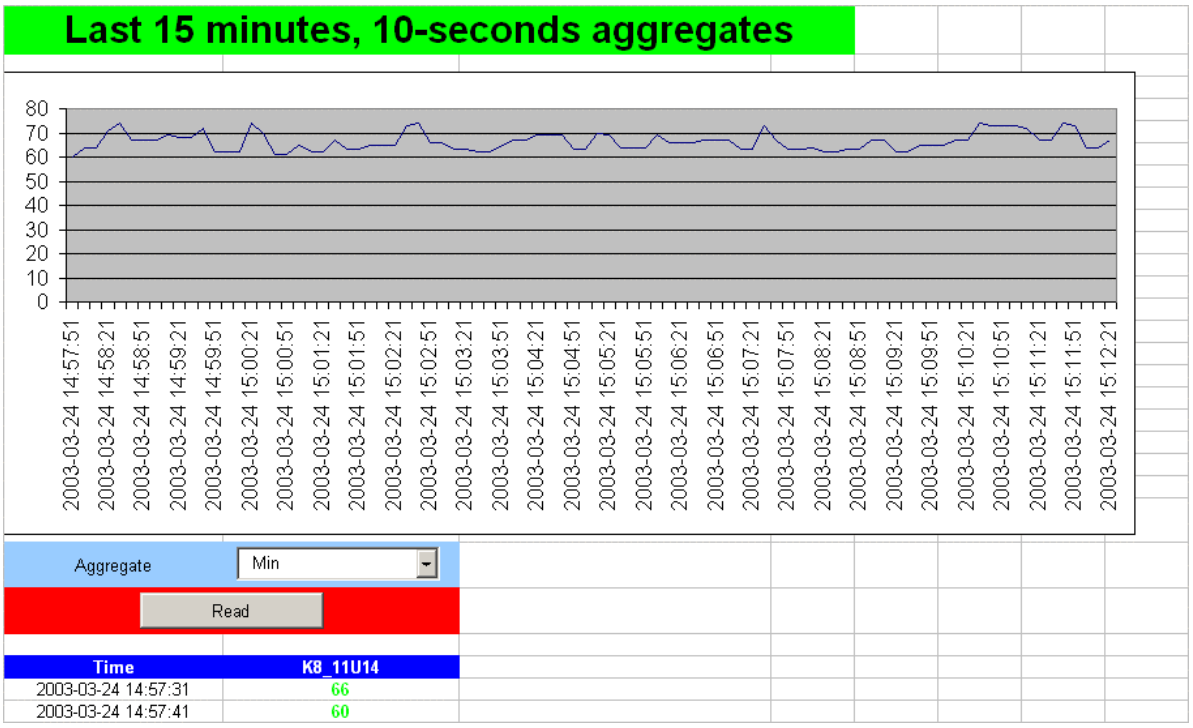


Figure 22. Automation Server – Archive Data / Examplary Application.

12.7. OLE DB Server – Archive Data - Macro

This example demonstrates method of access to archive data of applications of **asix** system with use of OLE DB server. Example needs Microsoft Excel 2000 or 2002 (XP) system for proper operation.

In this sheet link to the ADO library was established. In order to check what are current links, select command *References* from the *Tools* menu of Visual Basic. Library of Automation server of current data is named *Microsoft Active Data Objects 2.5*. Version of ADO may be different but should be higher than v. 2.1.

In *Hour Aggregates* Tab of the sheet, there is an example report containing hourly aggregates.

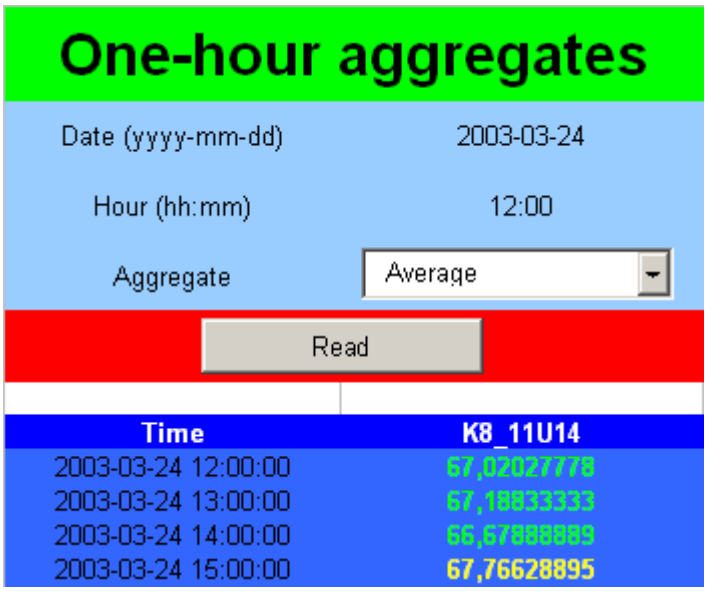


Figure 23. OLE DB Server – Archive Data – Macro / Examplary Application.

In this example, in cell C6 you can find report generation date, in cell C7 the beginning hour of report. C8 cell contains list including available aggregates. As default average is selected. After clicking *Read* button**,** macro is called which starts OLE DB server and retrieves data. The results are written into the range B12-C35.

In *Last Quarter* tab of the sheet there is an example of report containing 10-second aggregates calculated from data of last 15 minutes. In this example, to specify the time period in the function that retrieves treated data, the relative time is used.

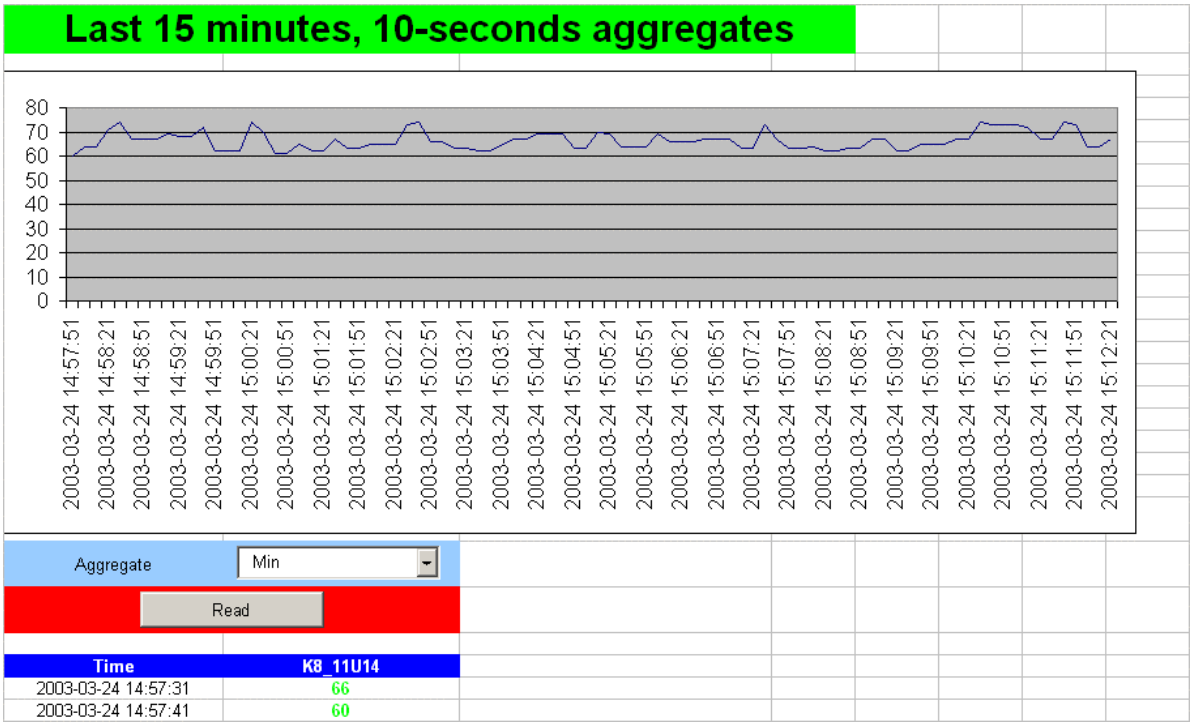


Figure 24. OLE DB Server – Archive Data – Macro / Examplary Application.

12.8. OLE DB Server - Borland C++Builder 6

Example program was developed with use of *Borland C++Builder 6 Enterprise Edition* package. It may be also compiled with use of *Professional* suit completed with *ADO Express* library.

Program retrieves data from example AC3Demo application of **asix** system. Program assumes that **asix** was installed in *c:\asix* directory. Displayed table contains 5-minutes averages of K811_U14 variable for the last hour period.

The main element of this program is *ADOConnection1* object. It is designed to establish connection with OLE DB server. The *ConnectionString* property contains description of this connection. To configure *ConnectionString* property, the *C++Builder 6* package uses the mechanism described in section *7.6.1. Identification and Configuration*. Query is placed in *ADOQuery1* object. After the *Read data* field is selected, the *ADOConnection1* i *ADOQuery1* objects are activated and data appear in the *DBGrid1* table object.

13. List of Figures

Figure 1. The 'Configurator' Window. *9*
Figure 2. The 'Configurator' Window. *10*
Figure 3. The 'Configurator' Window - Package Options. *11*
Figure 4 . The 'Configurator' Window - Network. *13*
Figure 5. The 'Configurator' Window - Variable Definitions Database. *17*
Figure 6. The 'Configurator' Window – Current Data. *18*
Figure 7. The 'Configurator' Window - Archival Data. *19*
Figure 8. The 'Configurator' Window - Alarms. *20*
Figure 9. The 'Configurator' Window - Reports. *21*
Figure 10. The 'Configurator' Window - Reports. *22*
Figure 11. The 'Configurator' Window - OPC Servers. *23*
Figure 12. 'Distributed COM Configuration Properties' Window. *56*
Figure 13. 'Aslink Manager Application Properties' Window. *57*
Figure 14. 'Registry Value Permissions' Window. *58*
Figure 15. 'Aslink Manager Application Properties' Window - Identity. *59*
Figure 16. 'Data Link Properties' Window for Parameterization of OLE DB Server. *78*
Figure 17. 'Data Link Properties' Window for Parameterization of OLE DB Server. *79*
Figure 18. Automation Server – Current Data and Variable Definitions Database / Examplary Application. *115*
Figure 19. Automation Server – Current Data and Variable Definitions Database – Examplary Application. *116*
Figure 20. DDE Server – Current Data / Examplary Application. *117*
Figure 21. Automation Server – Archive Data / Examplary Application. *119*
Figure 22. Automation Server – Archive Data / Examplary Application. *119*
Figure 23. OLE DB Server – Archive Data – Macro / Examplary Application. *120*
Figure 24. OLE DB Server – Archive Data – Macro / Examplary Application. *121*

14. List of Tables

Table 1. Methods of Specyfying the Channel Name. ... 7
Table 2. Method of Setting the Options - in Individual Servers of AsixConnect Package. ... 12
Table 3. Client Computer Name in asix System. ... 14
Table 4. asix System Network - Program Configuration. ... 14
Table 5. Algorithm of Searching for asix's System Servers. ... 15
Table 6. Default Values of Options for Searching for Data Servers of the asix System. ... 16
Table 7. Names of the ASIXConnect.ini File Sections for Individual Types of Servers. ... 16
Table 8. Variable Definitions Database - Program Configuration. ... 18
Table 9. Current Data - Program Configuration. ... 19
Table 10. Archive Data - Program Configuration. ... 20
Table 11. Alarms - Program Configuration. ... 21
Table 12. Reports - Program Configuration. ... 21
Table 13 . DDE and OPC Servers - Program Configurator. ... 23
Table 14. Constants Defined by Automation Server for the 'AttributeName' Parameter for the 'ReadAttribute' Function. ... 27
Table 15. Quality Bit Field. ... 36
Table 16. Substatus Bit Field for Bad Quality. ... 37
Table 17. Substatus Bit Field for UNCERTAIN Quality. ... 38
Table 18. Substatus Bit Field for GOOD Quality. ... 38
Table 19. Limit Bit Field. ... 39
Table 20. Vendor Bit Field. ... 39
Table 21. Archive Data Bit Field. ... 40
Table 22. List of Simple Identifiers for Unique Identification of a Process Variable in Servers of Current Data. ... 42
Table 23. List of Possible Intermediate Identifiers for Unique Identification of a Process Variable in Servers of Current Data. ... 43
Table 24. Types of Variables (Operation without Variable Definitions Database). ... 44
Table 25. Client Computer Name in asix System. ... 45
Table 26. Values of the 'DataSource' Parameter for the 'Read' Function for the Automation Server (Current Data). ... 48
Table 27. Values of the 'ServerState' Property for the Automation Server (Current Data). ... 50
Table 28. Values of the Quality (for the DDE Server /Current Data). ... 54
Table 29. The Names of the DDE Functions Available in Visual Basic. ... 55
Table 30. Special Characters Allowed in the Pattern for the 'MatchPattern' Function. ... 61
Table 31. Values of the 'DataSource' Parameter for the 'Read' Function (for the .NET Server /Current Data). 65
Table 32. The Value of 'CTDataSetFlags' Type (for the .NET Server / Current Data). ... 66
Table 33. The Contents of the 'ItemState' Structure (for the .NET Server / Current Data). ... 67
Table 34. The List of Identifiers for Unique Identification of a Process Variable in Servers of Archive Data. . 71
Table 35. The List of Supported Aggregates. ... 72
Table 36. Values of 'keyword' for the OPC Time Format. ... 74
Table 37. Values of 'offset' for the OPC Time Format. ... 74
Table 38. Characters of Meta Language for asix.SQL Queries. ... 80
Table 39. The List of Handled Aggregates Calculated by the 'ReadProcessed' Function (Using .NET Server / Archive Data). ... 85
Table 40. Constants of 'HTDataSetFlags' Type for the 'ReadProcessed' Function (for the .NET Server / Archive Data). ... 85
Table 41. The Contents of the Structure for the 'ReadRawResult' Class Declaration (for the .NET Server / Archive Data). ... 87
Table 42. The Contents of the Structure for the 'ReadProcessedResult' Class Declaration (for the .NET Server / Archive Data). ... 88
Table 43. The Contents of the 'ItemSample' Structure (for the .NET Server / Archive Data). ... 89
Table 44. The Flags Specific to Archive Data in Case of Bad Quality for the 'ItemSample' Structure (for the .NET Server / Archive Data). ... 90
Table 45. The Contents of the 'ItemStringSample' Structure (for the .NET Server / Archive Data). ... 90
Table 46. Columns of the 'ReadResult' Table for the 'DataSet' Object (for the .NET Server / Archive Data). 91

Table 47. Constants of the 'alarmType' Parameter for the 'ReadActive' Function for Retrieving the Information on Alarms. *94*
Table 48. Constants of the 'alarmStatus' Parameter for the 'ReadActive' Function for Retrieving the Information on Alarms. *95*
Table 49. The Contents of the 'Alarm' Structure. *97*
Table 50. Comment Characters Used for Configuration of Report Definition Files. *100*
Table 51. Values of a Report Period. *101*
Table 52. Contents of the 'ReportInfoNet' Structure. *103*
Table 53. Contents of the 'ItemStateWS' Structure (Web Service Server). *108*
Table 54. Values of 'identifier' Used for Log Names (Diagnostics of Server Operation). *111*
Table 55. Error Codes Returned by OPC Server. *112*

1. INTRODUCTION ... 3

1.1. PACKAGE COMPONENTS ... 3
1.2. LICENSING ... 4
1.3. REQUIREMENTS OF ASIX SYSTEM ... 4
1.4. MOST IMPORTANT MODIFICATIONS IN PACKAGE ... 4

2. INSTALLATION ... 5

2.1. AUTONOMOUS INSTALLATION OF ASIXCONNECT PACKAGE ... 5
2.2. INSTALLATION AS A PART OF ASIX PACKAGE ... 5

3. CONNECTION CONFIGURATION ... 7

3.1. CHANNELS ... 7
3.2. HOW TO SPECIFY THE CHANNEL NAME ... 7
3.3. CONFIGURATION FILE ... 8
3.4. INTERACTIVE CONFIGURATION ... 8
3.4.1. Configurator Program ... 8
... ... 9
3.4.2. Channel Management ... 10
3.4.3. Package Options ... 10
3.5. PROGRAM CONFIGURATION ... 11
SERVER TYPE ... 12
3.6. CHANNEL OPTIONS ... 13
3.6.1. asix System Network ... 13
3.6.1.1. Searching for Data Servers of asix System ... 14
3.6.1.2. External List of Servers ... 16
... ... 16
3.6.2. Variable Definitions Database ... 17
3.6.3. Current Data ... 18
3.6.4. Archive data ... 19
3.6.5. Alarms ... 20
3.6.6. Reports ... 21
3.6.7. DDE and OPC Servers ... 22

4. VARIABLE DEFINITIONS DATABASE ... 25

4.1. AUTOMATION SERVER ... 25
4.1.1. Application of Server ... 25
4.1.2. LoadChannel Function ... 26
4.1.3. Init Function ... 26
4.1.4. ReadAttribute Function ... 27
4.1.5. SelectAttribute Function ... 27
4.1.6. SelectVar Function ... 28
4.1.7. SelectVars Function ... 28
4.2. NET SERVER ... 29
4.2.1. ServerVB Class ... 29
4.2.1.1. Application of Server ... 29
4.2.1.2. ServerVB Designer ... 29
4.2.1.3. Dispose Function ... 30
4.2.1.4. Init Function ... 30
4.2.1.5. ReadAttributes Function ... 30
4.2.1.6. ReadAttributesN Function ... 30
4.2.1.7. Operation in ASP.NET Environment ... 31
4.2.2. ServerVBUI Class ... 32
4.2.2.1. Application of Server ... 32
4.2.2.2. ServerVBUI Designer ... 32
4.2.2.3. Dispose Function ... 32
4.2.2.4. Init Function ... 33
4.2.2.5. SelectVar Function ... 33
4.2.2.6. SelectVars Function ... 33
4.2.2.7. SelectAttribute Function ... 34

5. MEASUREMENT STATUS DESCRIPTION ... 35

5.1. MEASUREMENT QUALITY 35
5.2. QUALITY BIT FIELD 36
5.3. SUBSTATUS BIT FIELD FOR BAD QUALITY 37
5.4. SUBSTATUS BIT FIELD FOR UNCERTAIN QUALITY 38
5.5. SUBSTATUS BBIT FIELD FOR GOOD QUALITY 38
5.6. LIMIT BIT FIELD 39
5.7. VENDOR BIT FIELD 39
5.8. ARCHIVE DATA BIT FIELDS 39

6. CURRENT DATA 41

6.1. IDENTIFIERS 41
6.2. OPERATION WITHOUT VARIABLE DEFINITIONS DATABASE 43
6.3. DEFINING WRITE RIGHTS 44
6.3.1. Write Function 44
6.3.2. Extended Write Function 45
6.4. AUTOMATION SERVER 46
6.4.1. Application of Server 46
6.4.2. LoadChannel Function 47
6.4.3. Init Function 47
6.4.4. Read Function 47
6.4.5. SetItemActive Function 48
6.4.6. Write Function 49
6.4.7. WriteEx Function 49
6.4.8. Active Property 50
6.4.9. ServerState Property 50
6.4.10. StartTime Property 50
6.4.11. DataChange Event 50
6.4.12. Error Handling 51
6.5. DDE SERVER 51
6.5.1. Application of Server 51
6.5.2. DDE Operations Supported by Server 52
6.5.3. Format of Transferred Data 53
6.5.4. Transfer of Error Information 54
6.5.5. Using the DDE Server in Excel 54
6.5.6. DDE Server Utility 56
6.6. OPC SERVER 59
6.6.1. Technical Specification 59
6.6.2. Details of Implementation 60
6.6.2.1. Introduction 60
6.6.2.2. OPC Server Object 60
6.6.2.3. Browsing of Variable Definitions Database 61
6.6.2.4. Browsing Variable Properties 62
6.6.2.5. Variable Access Path 62
6.6.2.6. Process Variables 62
6.6.2.7. Synchronous Operations 62
6.6.2.8. Asynchronous Operations 63
6.6.2.9. Assigning New Value, Quality and Time Stamp 63
6.7. .NET SERVER 63
6.7.1. Application of Server 63
6.7.2. ServerCT Designer 64
6.7.3. Dispose Function 64
6.7.4. Init Function 64
6.7.5. Read Function 64
6.7.6. Write Function 66
6.7.7. Write Function – Extended Write Operation 66
6.7.8. ItemState Structure 67
6.7.9. SetItemActive Function 68
6.7.10. ItemsChange Event 68
6.7.11. Active Property 68
6.7.12. Operation in ASP.NET Environment 69

7. ARCHIVE DATA 71

7.1. IDENTIFIERS 71
7.2. OPERATION WITHOUT VARIABLE DEFINITIONS DATABASE 71
7.3. AGGREGATES 72
7.3.1. Description of Aggregates 72
7.3.2. Askom Algorithm 73
7.3.3. OPC Algorithm 73
7.4. OPC TIME FORMAT 74
7.5. AUTOMATION SERVER 75
7.5.1. Application of Server 75
7.5.2. LoadChannel Function 75
7.5.3. Init Function 76
7.5.4. ReadRaw Function 76
7.5.5. ReadProcessed Function 77
7.5.6. ShowProgresWindow Property 77
7.6. OLE DB SERVER 77
7.6.1. Identification and Configuration 77
7.6.2. Tables 79
7.6.3. asix.SQL Queries 79
7.6.4. Examples of Queries 81
7.7. .NET SERVER 82
7.7.1. Application of Server 82
7.7.2. ServerHT Designer 83
7.7.3. Dispose Function 83
7.7.4. Init Function 83
7.7.5. ReadRaw Functions 83
7.7.6. ReadProcessed Functions 84
7.7.7. ReadProcessedAsString Function 86
7.7.8. RelativeDateTime Function 86
7.7.9. RelativeTimeSpanm Function 87
7.7.10. ReadRawResult Classt 87
7.7.11. ReadProcessedResult Class 88
7.7.12. ReadProcessedAsStringResult Class 88
7.7.13. ItemSample Structure 89
7.7.14. ItemStringSample Structure 90
7.7.15. DataSet Object 91
7.7.16. Operation in ASP.NET Eenvironment 91

8. ALARMS 93

8.1. .NET SERVER 93
8.1.1. Application of Server 93
8.1.2. ServerAL Designer 93
8.1.3. Dispose Function 94
8.1.4. Init Function 94
8.1.5. ReadActive Functions 94
8.1.6. ReadHistorical Functions 95
8.1.7. Alarms2DataSet Function 96
8.1.8. Alarm Structure 96
8.1.9. Operation in ASP.NET Environment 97

9. REPORTS 99

9.1. .NET SERVER 99
9.1.1. Application of Server 99
9.1.2. Configuration of Report Definition Files 99
9.1.3. ServerRP Designer 100
9.1.4. Dipsose Function 100
9.1.5. Init Function 100
9.1.6. GetReportsInfo Function 101
9.1.7. ReadReportsInfo Function 101
9.1.8. GetDefFilesInfo Function 102
9.1.9. ReadDefFilesInfo Function 102
9.1.10. GetReportsDirectoryPath Function 102

9.1.11. ReportInfoNet Structure ... 102
9.1.12. DefFileInfoNet Structure ... 103
9.1.13. Operation in ASP.NET Environment ... 103

10. WEB SERVICE SERVER ... 105

10.1. INSTALLATION ... 105
10.2. WEB.CONFIG CONFIGURATION FILE ... 105
10.3. CLIENTS ... 106
10.3.1. Internet Explorer ... 106
10.3.2. WebForm Application ... 106
10.4. VARIABLE DEFINITIONS DATABASE ... 107
10.5. CURRENT DATA ... 107
10.6. RAW ARCHIVE DATA ... 108
10.7. AGGREGATED RAW DATA ... 109
10.8. ACTIVE ALARMS ... 109
10.9. HISTORICAL ALARMS ... 110

11. DIAGNOSTICS OF SERVER OPERATION ... 111

11.1. LOGS ... 111
11.2. ERROR CODES ... 111
11.3. DDE SERVER ... 113

12. EXAMPLES ... 115

12.1. AUTOMATION SERVER – CURRENT DATA AND VARIABLE DEFINITIONS DATABASE ... 115
12.2. AUTOMATION SERVER – CURRENT DATA – LATE BINDING ... 116
12.3. DDE SERVER – CURRENT DATA ... 117
12.4. DDE SERVER – CURRENT DATA - UPDATING ... 117
12.5. OPC SERVER – CURRENT DATA ... 118
12.6. AUTOMATION SERVER – ARCHIVE DATA ... 118
12.7. OLE DB SERVER – ARCHIVE DATA - MACRO ... 120
12.8. OLE DB SERVER - BORLAND C++BUILDER 6 ... 121

13. LIST OF FIGURES ... 123

14. LIST OF TABLES ... 125