www.asix.com.pl

AsixConnect

Manual for operators

Doc. No ENP7065
Version: 2014-06-30

ASKOM

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

Table of Contents

1 Introduction

The AsixConnect is a package of servers that extends scope of applications of the 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 4 has to be activated.

1.2 Licensing

Licensing of AsixConnect package is discussed in the commercial information brochure.

1.3 Requirements of the 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 the Package

Version 7.0

- Adding support for long variable names.
- Adding support for combined variable definition databases. As the value of the parameter *ItemsDatabase* now you can specify multiple names of variable definition databases, separated by a comma.

Version 6.0

- Adding support for aggregated values calculated by using the Aggregator module (see more: Asix.chm/pdf, *7.10 Aggregation of Archival Data*).
- Adding the functions: SelectAlarm and SelectAlarms to the server .NET of a variable definition database.

Version 5.0

- Adding support for variable definitions database.
- New statuses of archival data.

Version 4.0

- Extension of the package with .NET servers of current, archive and alarm data as well as Variable Definitions Database.
- Extension of the 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 the version 3.

2 Installation as a Part of the Asix Package

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

In order to install the AsixConnect package, the Asix package has to be installed (AsixConnect components are an integral part of Asix).

3 Connection Configuration

3.1 Channels

The channel is a set of configuration options of AsixConnect package servers, options of connections with the Asix system server and options of Variable Definition Database.

The channel named ’*’ is the basic channel of servers of the 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 is used, the options of this channel will be 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 the AsixConnect package in the following way.

Table: Methods of Specyfying the Channel Name in Individual Servers of the AsixConnect Package.

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

PRZYKŁAD:

```
<?xml version=”1.0” encoding=”utf-8” ?>
<configuration>
  <appSettings>
    <add key=”DefaultChannelName” value=”AsEmis” />
  </appSettings>
&ldots;
```

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 the Asix 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 from the menu *Start* > *Programs* > *Asix* > *AsixConnect* > *AsixConnect - Configurator*. When started, this program displays the main window - See: *Fig.* below.

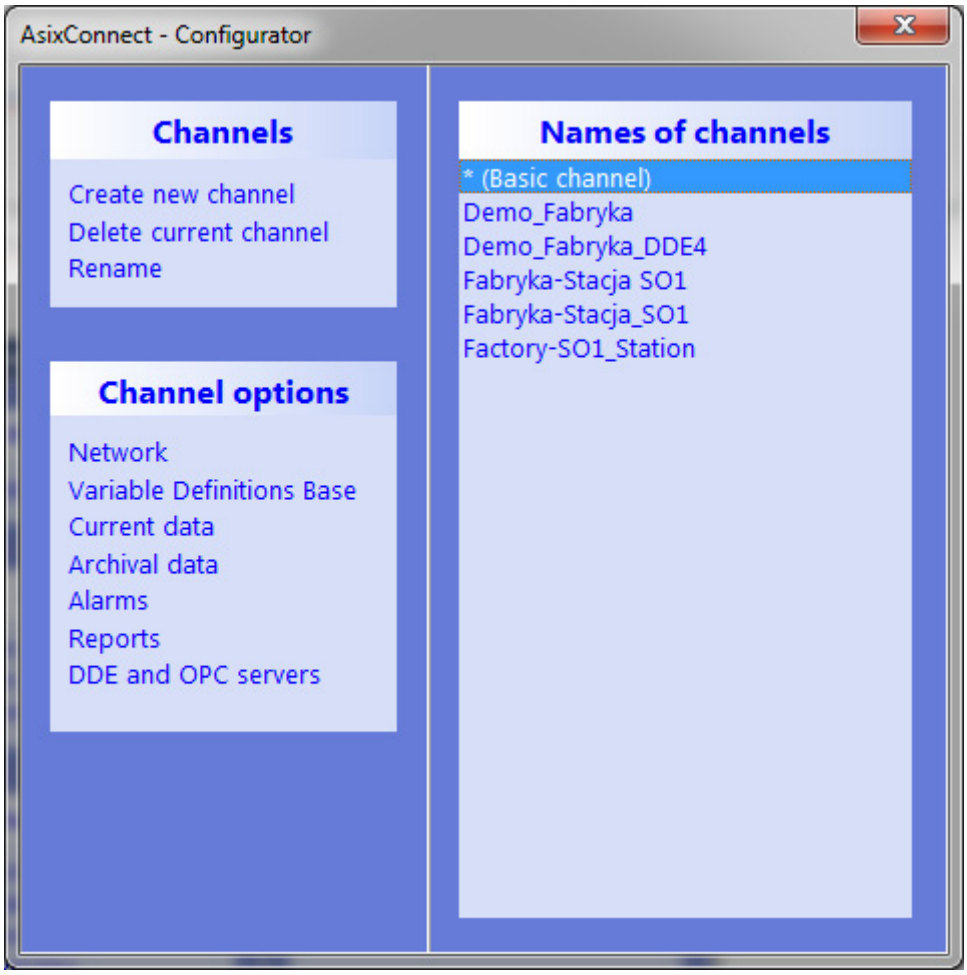


Fig. The Configurator Window.

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 ’r;*’ exists all the time.

3.4.2 Channel Management

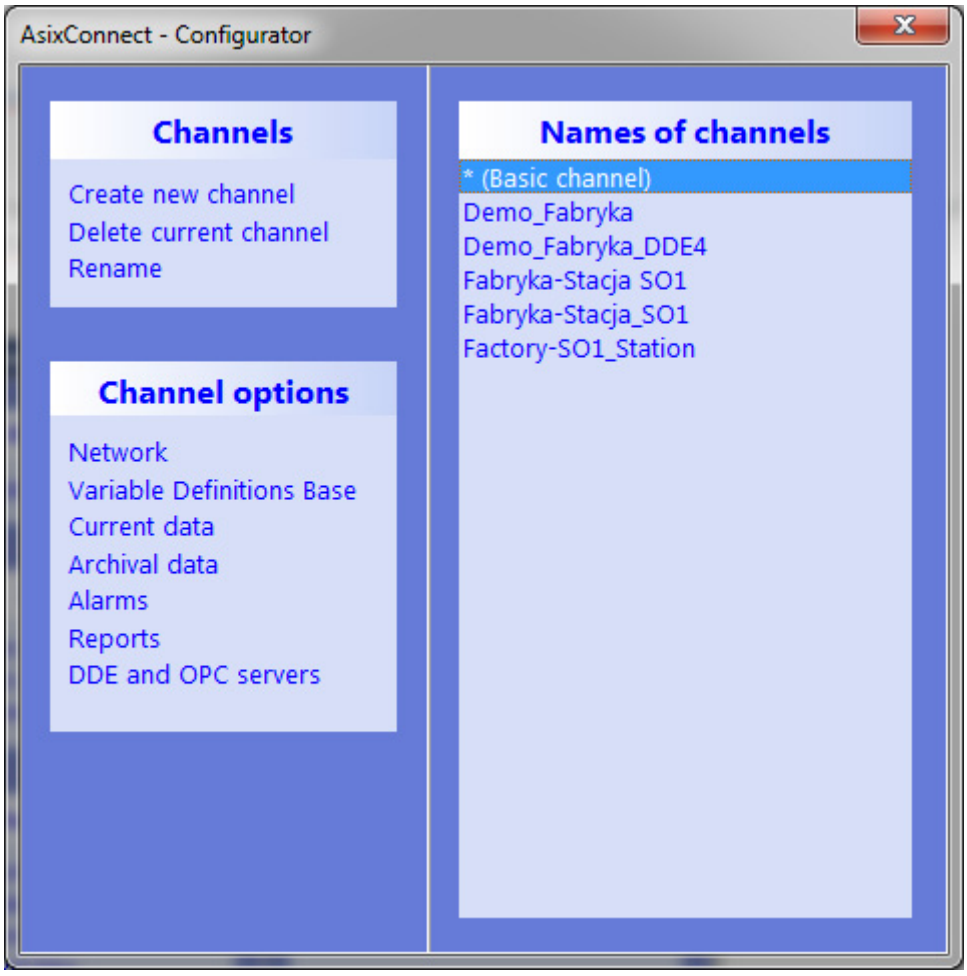


Fig. The Configurator Window.

Commands relating to the Channels group have the following meanings:

Create New Channel - enables creation of a new channel;
Delete Current Channel - enables removal of the currently highlighted channel;
Rename - enables change of the currently highlighted channel.

The name of the basic channel cannot be removed or changed.

3.4.3 Package Options

Location of the AsixConnect servers log files and directory of configuration files are defined with the use of the options available from Architect program > ***Tools*** menu > ***Log and Configuration Directories*** option:

Directory of Log Files - name of directory of log files created by AsixConnect servers;
Directory of Configuration Files - the directory contains AsixConnect.ini used for configuration of OPC, Automation, DDE, .NET, AsPortal and AsWWW servers.

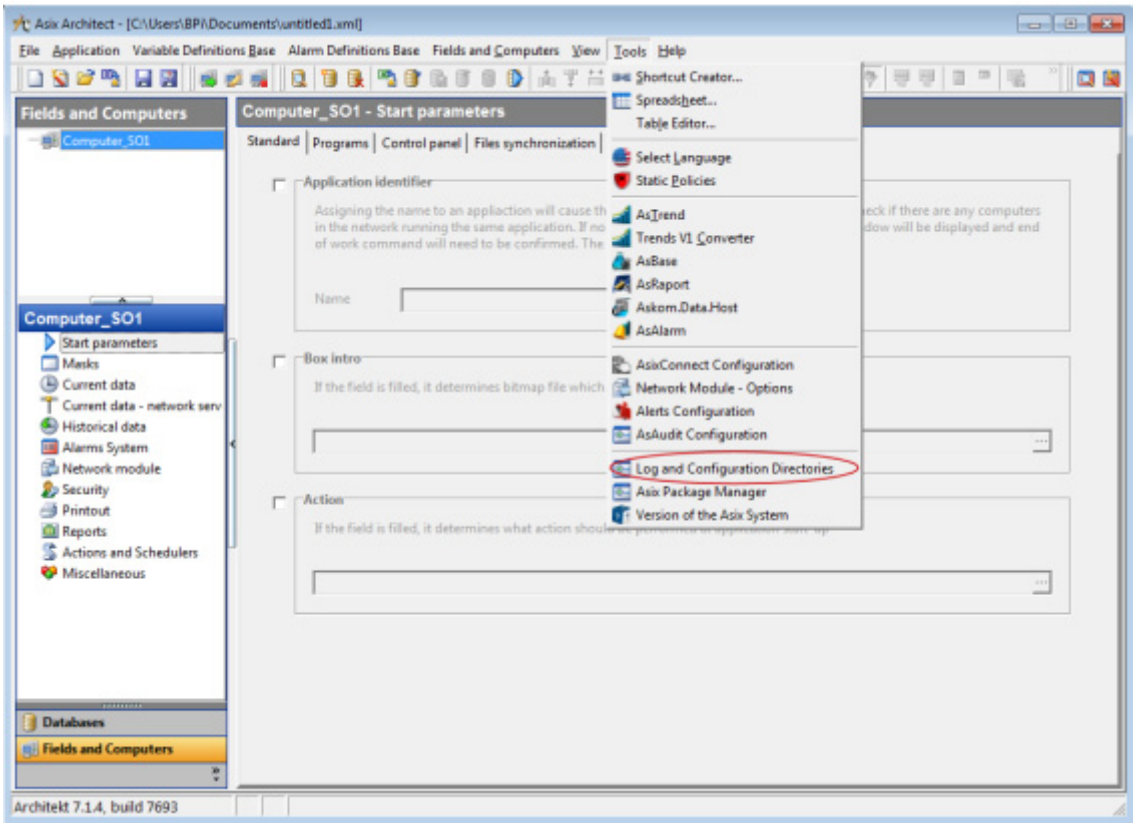


Fig. Log and Configuration Directories of AsixConnect servers.

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 the parameter of this function is transferred as a text parameter:

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* below.

Tabela: Method of Setting the Options - in Individual Servers of the 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

3.6.1.1 Asix System Network Configuration

Interactive Configuration

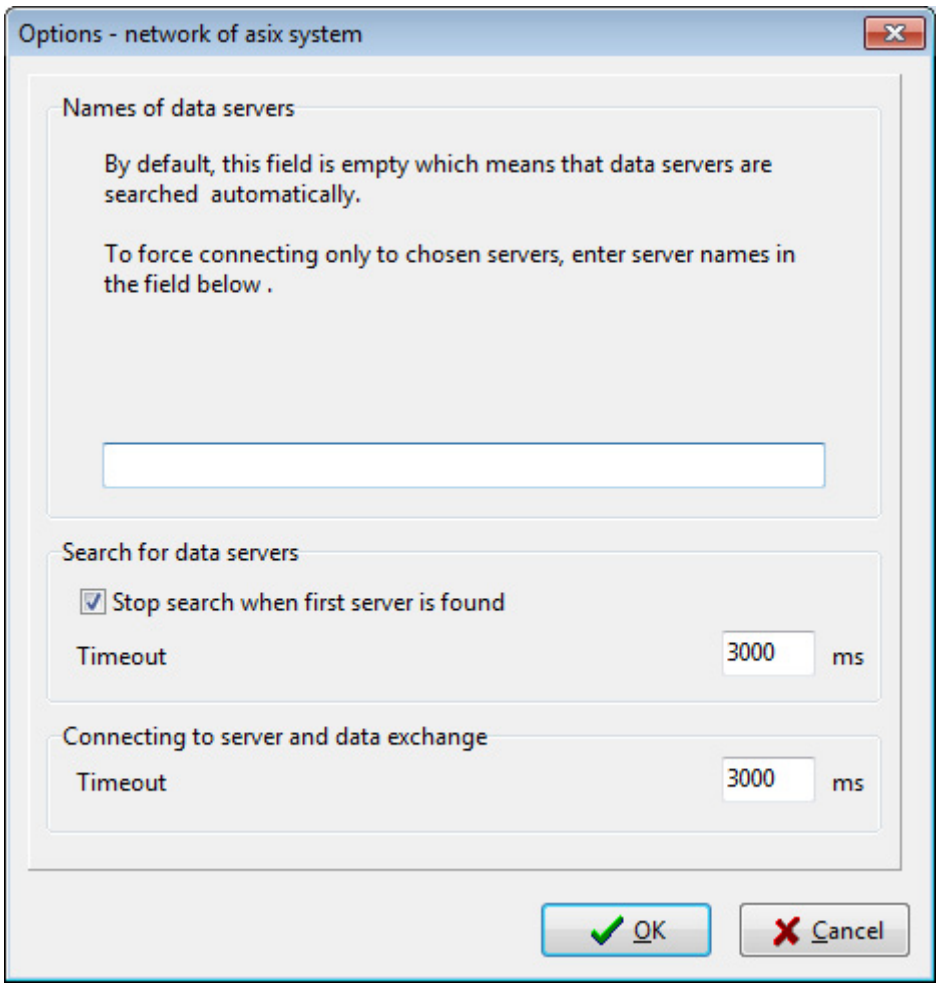


Fig. Log and Configuration Directories of AsixConnect Servers.

The option Names of data servers 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 below.

Table. Client Computer Name in the Asix System.

Description of Client Software Configuration	Client Computer Name in the Asix System
Only AsixConnect package is used and the file *aslink.ini* was not created or the *Name* line was not defined in the *ASLINK* section. By default aslink.ini is in c:\AsixApp\cfg.	Name of a client computer in WINDOWS plus period (.) at the end, e.g. for a computer named ”r;r;CLI” its name in the Asix system is ”r;r;CLI.”
Only the AsixConnect package is used and *ASIX.INI* file was created, the *Name* line was defined in the ASLINK section.	Name of a client computer in the Asix system is provided inthe *ASIX.INI* file in the *ASLINK* section, in the line titled *Name*.
AsixConnect and an Asix application run in the same computer.	Name ofa client computer in the Asix system is provided in the XML file of an Asix application, in the *Network Module* group, on the tab *Computer name*.

The other options are of advanced-type and usually they do not need modification. Their meaning is described in *3.6.1.2 Searching for data servers of Asix system*.

Program Configuration

Table. Asix System Network - Program Configuration.

Option	Option Name for Interactive Configuration	Type	Default Value
AsixServerName	Data server names of the 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]

3.6.1.2 Searching for Data Servers of the Asix System

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

In the first step, the servers of the 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 the 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 only. The time of waiting for response is affected by the StopAfterFirstServerFound and FindServerTimeout options.

In the second step, one server is chosen from Asix servers 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. Algorithm of Searching for Asix 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 are as follows.

Table. 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)

3.6.1.3 Searching for Data Servers of the Asix System in Other Subnetworks

Standard searching data of Asix servers includes searching within one subnetwork. Devices within one (common) subnetwork have the same initial part of a binary record of an IP address. The length of this part determines the value of the subnetwork mask.
You have to configure the so-called "Connection via TCP/IP" to ensure searching on the data server located in other network.

Table. Configuration of Connection via TCP/IP.

Description of Client Software Configuration	Description of Configuration "Connection via TCP/IP"
Only AsixConnect is used.	Run the Architect program and select the command *Network Module - Options* from the *Tools* menu. There is the editor *Connections Using TCP/IP Protocol* on the *TCP/IP Communication* tab.

AsixConnect and Asix are used on the same computer.	Run the Architect program, open the file of Asix system application and select the option group *Network Module*. There is the editor *Connections Using TCP/IP Protocol* on the *TCP/IP Communication* tab.

3.6.1.4 External List of Servers

The external list of servers allows the list of admissible data servers to be specified individually for every resource of the 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 explicitly 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.

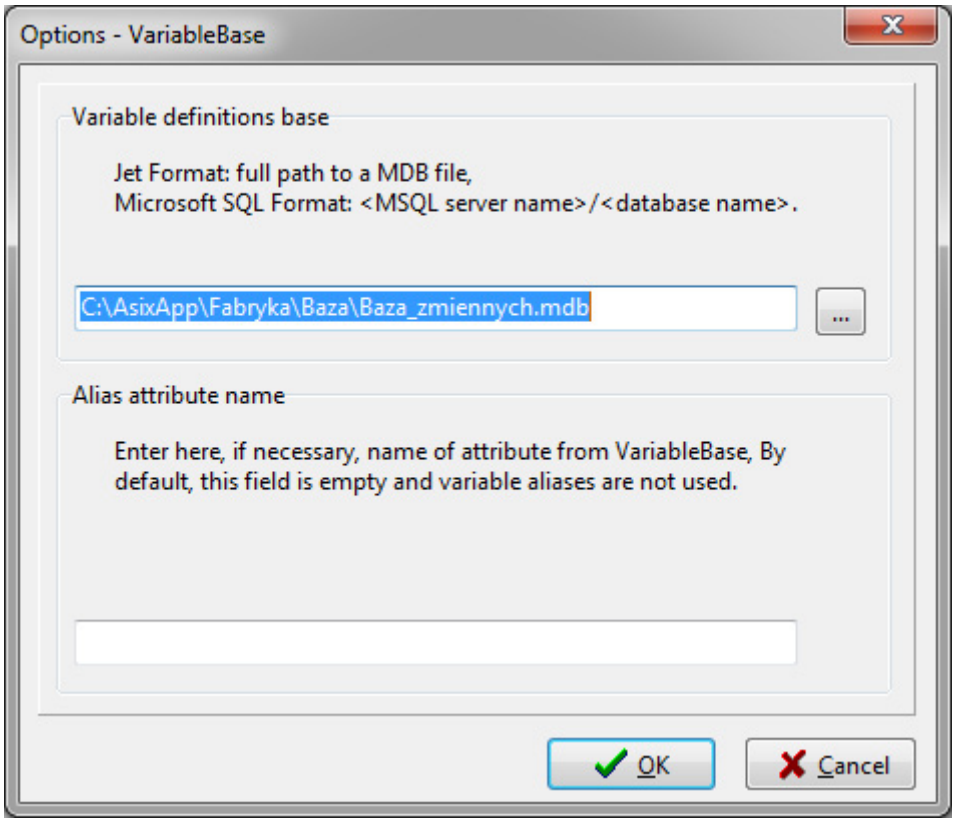
Table. Names of the ASIXConnect.ini File Sections for Individual Types of Servers.

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 Definition Database

Interactive Configuration



F*ig. AsixConnect Configurator - Channel Options - Variable Definition Base.*

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. As the option value the path to the .mdb file or database location on SQL server should be specified in the following notation:

<MSSQL server>/<database name>

It is possible to use combined databases of variables specifying the path to more than one database. The paths should be separated by a comma.

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 Definition Database using OPC mechanisms. However, one should remember that for review of the Variable Definition Database, the base defined in the basic channel '*' is used only.

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

Program Configuration

Table. Variable Definition Database - Program Configuration.

Option	Option Name for Interactive Configuration	Type	Default Value
ItemsDatabase	Variable definition database	String	null
AliasAttributeName	Alias attribute name	String	null

3.6.3 Current Data

Interactive Configuration

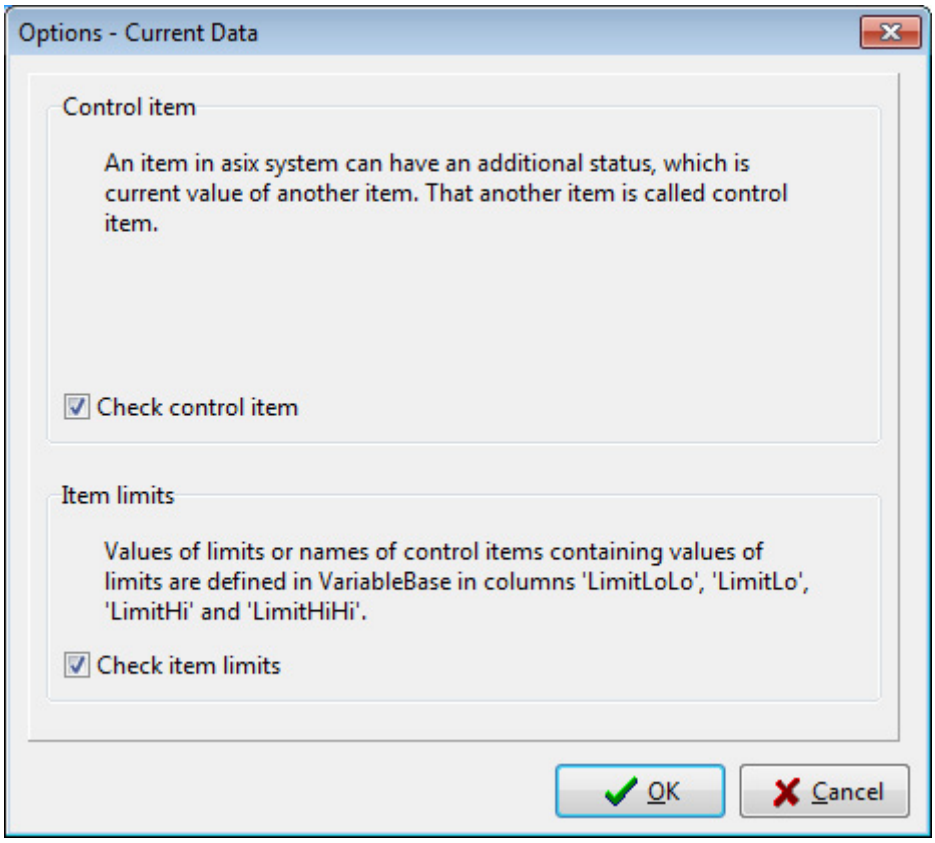


Fig. AsixConnect Configurator - Current Data Options.

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 the 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 Definition 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 Definition Database.

For information about the attributes, see the Asix system documentation (Architect.chm/pdf).

Program Configuration

Table. Program Configuration of Current Data.

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

3.6.4 Archival Data

Interactive Configuration

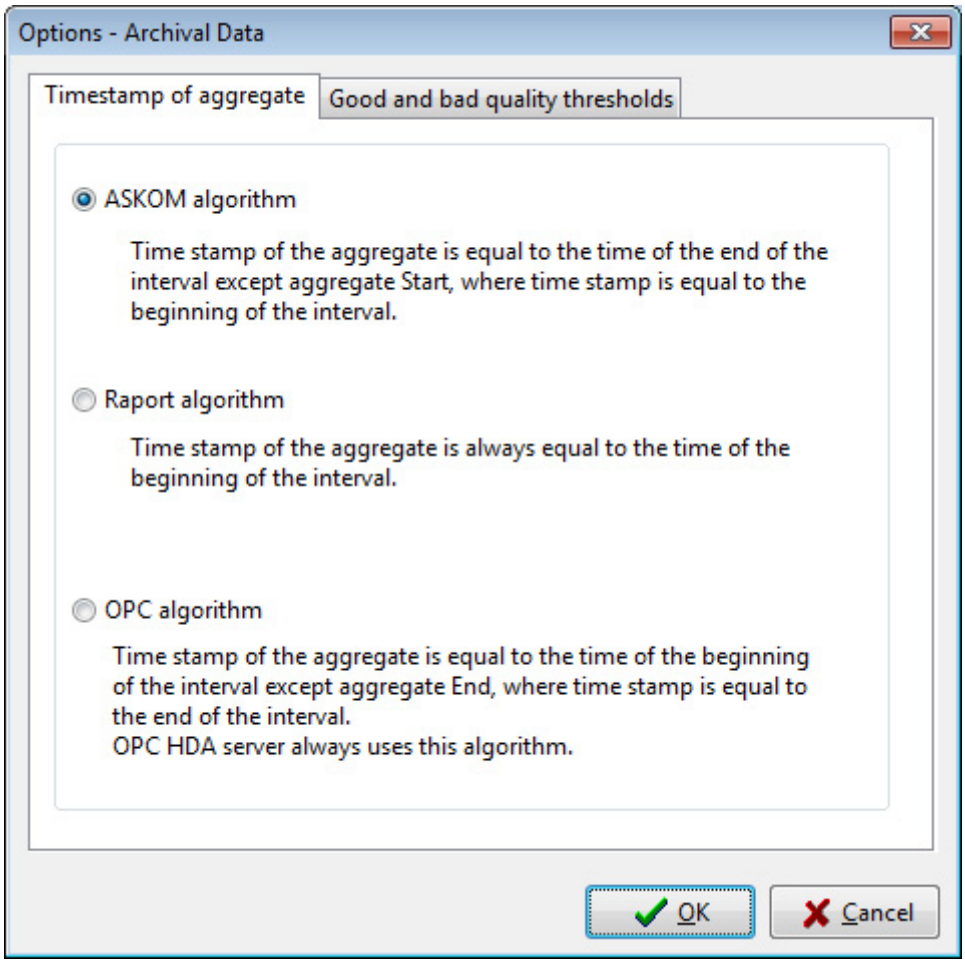


Fig. AsixConnect Configurator - Archival Data Options (1).

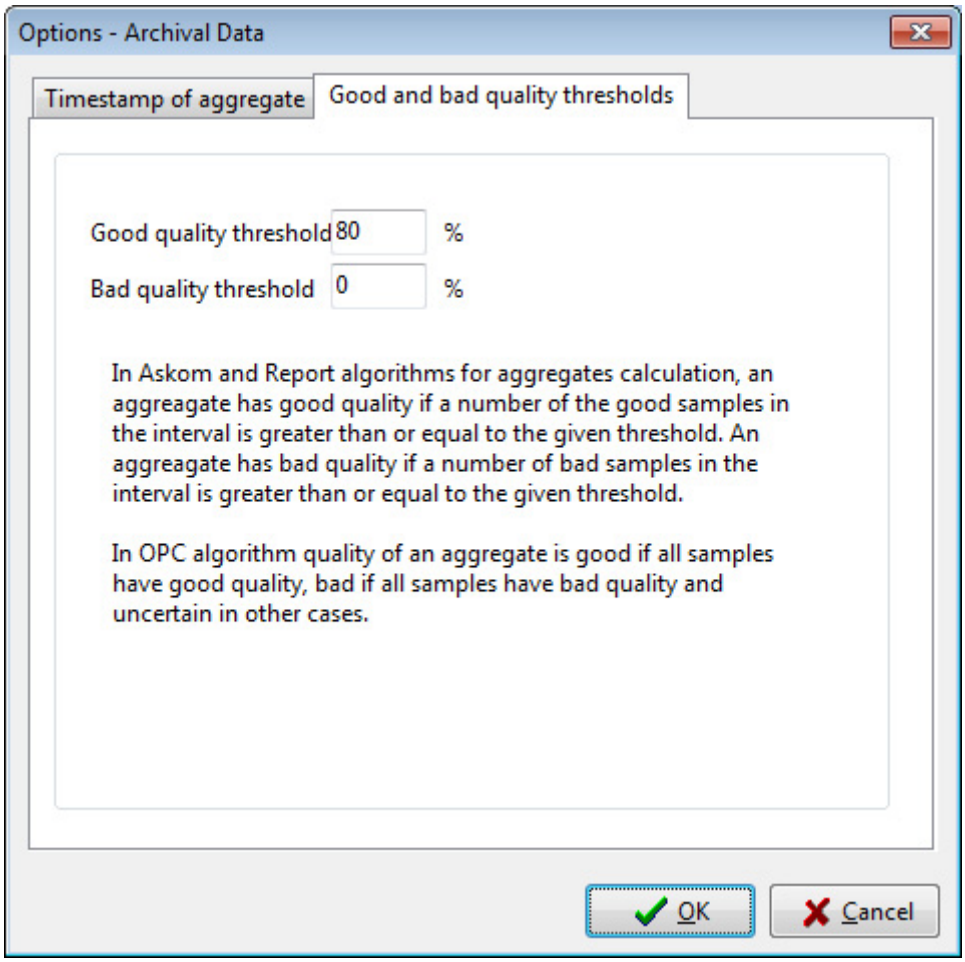


Fig. AsixConnect Configurator - Archival Data Options (2).

The options 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. Program Configuration of Archival Data - Names of Options.

Option	**Meaning**	**Type**	**Default Value**
AggregateMode	Select algorithm for aggregate calculation; 0 - ASKOM algorithm, 1 - OPC algorithm, 2 - Report algorithm	Integer	0

QualityGoodThreshold	Treshold of good quality	Integer	80

3.6.5 Alarms

Interactive Configuration

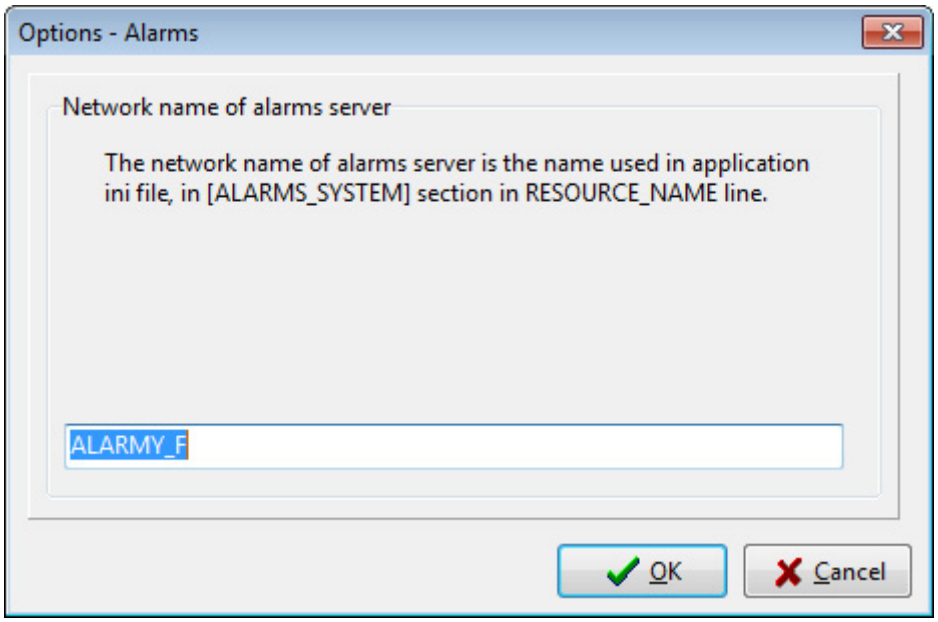


Fig. AsixConnect Configurator - Alarm Options.

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 Asix application configuration file (for older versions of Asix application parameterized in .ini files: in the section ALARM_SYSTEM, in the line RESOURCE_NETWORK_NAME; for applications parameterized with the use of Asix ver. 5 (and upper) in .xml files, using the Architect program: in the parameter *Network name of alarms set* in the *Alarms System* parameter group of Architect). 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

Tab. Program Configuration of Alarm Data.

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

3.6.6 Reports

Interactive Configuration

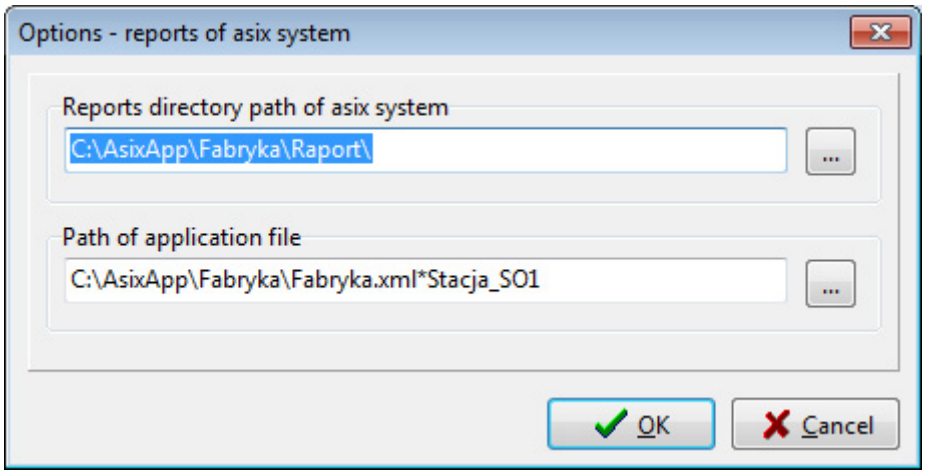


Fig. AsixConnect Configurator - Asix Report Options.

The *Reports Direcotory Path of Asix System* option allows to enter the information on directory the report definition files and Asix application reports are stored in.

In the *Path of Application File* field the full path to an Asix application configuration file should be passed.

Program Configuration

Table. Program Configuration of Report Data - Names of Options.

Option	Meaning	Type	Default Value
ReportsDirectory	Path to the directory of Asix system reports	Text	Null
IniFilePath	Path to the configuration file of an Asix system application	Text	Null

3.6.7 DDE and OPC Servers

Interactive Configuration

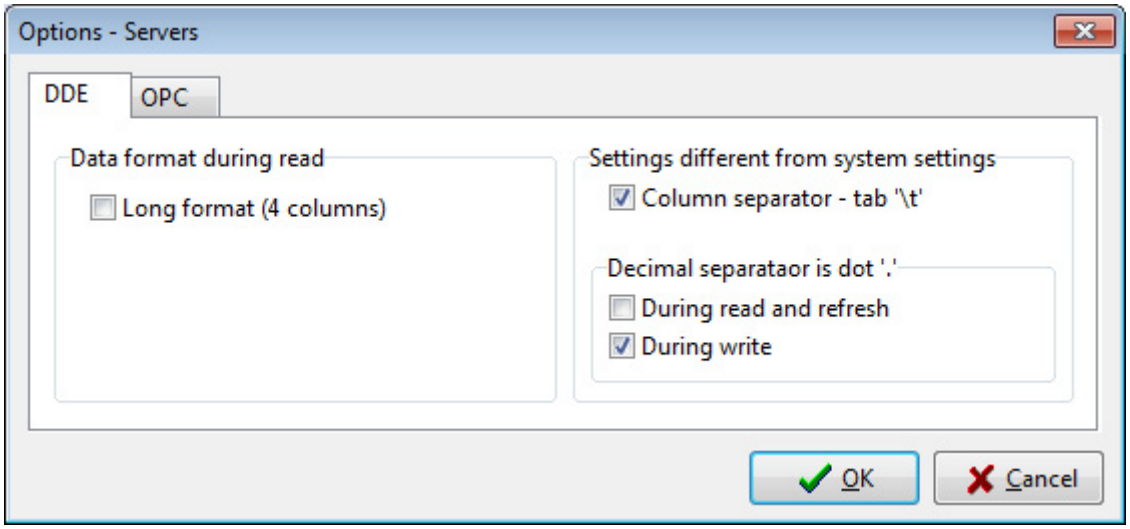


Fig. AsixConnect Configuration - DDE Server Options.

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.

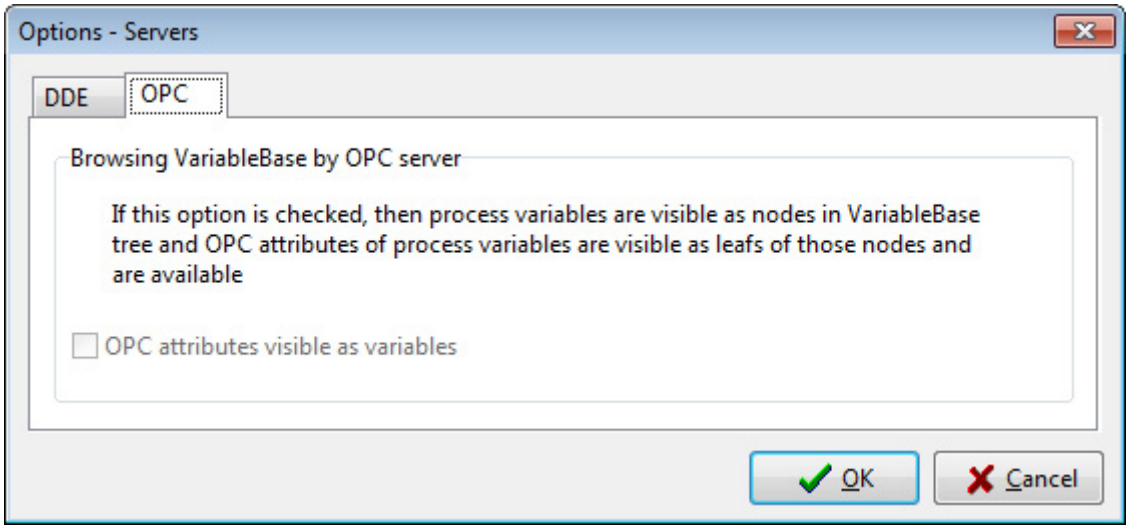


Fig. *AsixConnect Configuration - OPC Server Options.*

The option *OPC attributes visible as variables* change the way of reviewing the Variable Definition 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 Definition 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

Tab. Program Configuration of DDE and OPC Server Data - Names of Options.

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 Definition Database

4.1 Variable Definition Database

In the Asix system the variable definition 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 definition 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.

Variable Definition Database (VarDef) supports Microsoft SQL Server 2000 (and upper) 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. The Architect module provides a full support of VarDef database in Jet or MS SQL format.

Programs included in the Asix system receive from variable definition database (VarDef) all the attributes required to display technological diagrams. However, for the operator or designer, who wants to review a variable definition database, viewing the entire database - all variables at the same time is uncomfortable. The way of displaying VarDef parts including interrelated variables is needed. Therefore, the possibility of grouping variables, naming the groups and arranging the groups in the hierarchical structure has been implemented in VarDef. Grouping of variables is based on assigning variables to any number of groups identified by a grouping attribute added to Scheme Editor. (See: Architect - User's Manual, *Variable Grouping*).

4.2 Automation Server

4.2.1 Automation Server

Automation server of Variable Definition Database enables access to Variable Definition Database of the 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 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*. When converting

Visual Basic application that uses the AsixConnect package in version 6, the name of the server object *XConnect11.ServerVB* should be changed into *XConnect.ServerVB*. When converting Visual Basic application that uses the AsixConnect package of any version, the name of the currently used library of types should be changed into *AsixConnect Type Library*.

4.2.2 Application of Server

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

- Install AsixConnect package.
- Obtain the Variable Definition Database of the 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 Definition Database may also be loaded using the *Init* function.

4.2.3 LoadChannel Function

Function calling syntax:

```
LoadChannel ChannelName
```

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

4.2.4 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*.

4.2.5 ReadAttribute Function

Function calling syntax:

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

The *ReadAttribute* function returns the value of a variable attribute.

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

As the *AttributeName* parameter the attribute name or one of constants defined by the Automation server should be passed (constants listed in the table below).

Table. Constants Defined by Automation Server.

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

4.2.6 SelectAttribute Function

Function calling syntax:

```
SelectAttribute (SelectedAttributeName)
```

The *SelectAttribute* function is designed for interactive selection of an 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 Definition Database. The user can choose one of them and accept the selection by clicking on the *OK* button or cancel the selection by clicking on the *Cancel* button.

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

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

4.2.7 SelectVar Function

Function calling syntax:

```
SelectVar (SelectedVarName)
```

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

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

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

4.2.8 SelectVars Function

Function calling syntax:

```
SelectVars (SelectedVarsNames)
```

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 Definition Database. The user can choose one or more of them and accept the selection by clicking on the *OK* button or cancel the selection by clicking on the *Cancel* button.

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

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

4.3 Server .NET

4.3.1 ServerVB Class

4.3.1.1 Application of Server

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

- Install the AsixConnect package.
- Obtain the Variable Definition Database of the 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 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 object of *ServerVB* class. In this program, you need to:
  - create object of the *ServerVB* type, giving as the 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.

4.3.1.2 ServerVB Designer

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

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

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

4.3.1.3 Dispose Function

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

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

4.3.1.4 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 Definition Database.*

4.3.1.5 ReadAttributes Function

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

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

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

As the *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.

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

As the *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.

4.3.1.7 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 the *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 the *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 the *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 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.

4.3.2 ServerVBUI Class

4.3.2.1 Application of Server

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

- Install the AsixConnect package.
- Obtain the Variable Definition Database of the Asix system from which the current data are to be retrieved or generate such a base.
- 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:

    o 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* file in the c:\asix subdirectory, click on the *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 an object of the *ServerVBUI* class. In this program, you need to:

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

4.3.2.2 ServerVBUI Designer

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

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

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

4.3.2.3 Dispose Function

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

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

4.3.2.4 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*.

4.3.2.5 SelectVar Function

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

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

The *SelectVar* function returns the value *true* if the user has accepted the selection by clicking on the *OK* button or the value *false* if the user has cancelled the selection by clicking on the *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 the *Form* class, the window handle returns the value *Handle*.

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

4.3.2.6 SelectVars Function

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

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

The *SelectVars* function returns the value *true* if the user has accepted the selection by clicking on the *OK* button or the value *false* if the user has cancelled the selection by clicking on the *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 the *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 the *true* value, the parameter contains the table of the names of selected variables.

4.3.2.7 SelectAttribute Function

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

The *SelectAttribute* function is designed for interactive selection of an 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 the *OK* button or cancel the selection by clicking on the *Cancel* button.

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

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 the *Form* class, the window handle is stored in the property *Handle*.

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 the *true* value, the parameter contains the selected attribute name.

5 Measurement Status Description

5.1 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 a specific server. The time is always provided as the local time (except for OPC Server where the 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 the 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.

5.2 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 the *Vendor* bit field.

5.3 Quality Bit Field

Table. Quality Bit Field.

QQ	Value of Bits	Definition	Description
0	00SSSSLL (0x00)	*Bad*	The value is not available; details in the Substatus field.
1	01SSSSLL (0x40)	*Uncertain*	The quality of a variable is uncertain; details in the Substatus field. An uncertain value is treated as ”almost” bad in an Asix system application. The way of displaying the variable with the uncertain value is the same as for the value with the bad value and the value of the *Substatus* field equal to *Last Known Value*.
2	10SSSSLL	*N/A*	Is not used by OPC.
3	11SSSSLL (0xC0)	*Good*	The 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 the mask of the *Quality* bit field with the 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.
}
```

5.4 Substatus Bit Field for Bad Quality

Table. Substatus Bit Field for Bad Quality.

SSSS	Value of Bits	Definition	Description
0	000000LL (0x00)	*Non-specific*	The 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. The quality is signalized by the driver of the device or by the control 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.

5.5 Substatus Bit Field for UNCERTAIN Quality

Table. 'Substatus' Bit Field for UNCERTAIN Quality.

SSSS	Value of Bits	Definition	Description
0	010000LL (0x40)	*Non-specific*	The value is uncertain but the reason of this situation is unknown.
1	010001LL (0x44)	*Last Usable Value*	The device stopped reading the variable values. The value should be treated as stale.
2-3		*N/A*	It is not used by OPC.

4	010100LL (0x50)	*Sensor Not Accurate*	Either the value exceeded some of the sensor limits nor the sensor is out of calibration.
5	010101LL (0x54)	*Engineering Units Exceeded*	The value exceeded limits defined for the measurement.
6	010110LL (0x58)	*Sub-Normal*	The value is defined on the basis of several sources and the number of available sources is less than demanded number.
7-15		*N/A*	It is not used by OPC.

5.6 Substatus Bit Field for GOOD Quality

Table. 'Substatus' Bit Field for GOOD Quality.

SSSS	Value of Bits	Definition	Descriotion
0	110000LL(0xC0)	*Non-specific*	The value is good.
1-5		*N/A*	It is not used by OPC.
6	110110LL(0xD8)	*Local Override*	The 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.

5.7 Limit Bit Field

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

Table. 'Limit' Bit Field.

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

When reading the values of the current variable from the Asix system, the *Limit* bit field usually takes the value *Not Limited*. If the current data server 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 the value *Constant*.

5.8 Vendor Bit Field

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

Table. 'Second Limit' Flag.

Hexadecimal Value	Definition	Description
0x0800	*Second Limit*	The value exceeded the second limit - an 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 option *Verify limits of variables* is enabled.

5.9 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. 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*	The process variable value is not available because the archive with data from the precise time period is not available.

0x01000000	*Asix Archive End*	The 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 an Asix system station).
0x0040000	*Quality Good - Raw Value*	The process variable value received from the archive.
0x00080000	*Quality Good - Calculated Value*	The 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 Asix system application. All other information on the variable needed for communication with the Asix system is retrieved from the Variable Definition Database. In order to obtain the Variable Definition Database, you should address to the administrator of an Asix application 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 Definition Database should be determined on an interactive basis or by program. For details, see the chapter on server configuration 3.6.2.Variable Definition 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 a variable in the Asix system application. The notation <*Attribute*> means the name of an attribute in the Variable Definition Database of the Asix system application.

Table. Current Data - Simple Identifiers.

Identifier	Description	Possible Value Type
<Variable> *<Variable>.CV*	The current value of a variable.	R4, I2, I4, UI2, UI4
<Variable>.DATA_TYPE	A canonical type of the current value of a variable.	I2
<Variable>.EU_UNIT	Variable unit.	STRING
<Variable>.DESC	Variable description.	STRING
<Variable>.HI_EU	The max value of the current value of a variable.	R8
<Variable>.LO_EU	The min value of the current value of a variable.	R8
<Variable>.<Attribute>	Variable attribute value.	STRING, R8
<Variable>FormatedCV	The current value of a variable converted in accordance with the Format attribute value of this variable.	STRING
<Variable>.PercentageCV	The percentage current value of a variable.	R8
<Variable>.BarCVs	The percentage current value of a variable complying with a bar	[R8, R8]

	basis and the current percentage value of the bar basis (2-element table).	
<Variable>.<Attribute>.CV	The variable attribute value in the database, if the value is a number.	R8
<Variable>.<Attribute>.CV. FormatedCV	The variable attribute value formated in accordance with the Format attribute value of this variable. The row value of the variable is a number.	STRING
<Variable>.<Attribute>. CV.PercentageCV	The percentage value of a variable attribute. The row value of the attribute is a number.	R8

Intermediate Identifiers

The intermediate identifier always starts with the phrase <*Variable*>.<*Attribute*>.CV. The attribute <*Attribute*> of the 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. Current Data - Intermediate Identifiers.

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

6.2 Operation Without Variable Definition Database

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

AsixConnect servers of current data accepts also descriptive identifiers of variables and where you use these identifiers only, loading the Variable Definition Database is not needed.

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

A 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 the Asmen module of an Asix system application,
- *variable name* - a name under which the process variable is known in an Asix system application,
- *variable type* - two or three characters defining the type of a variable according to the table below:

Table. Types of Variables Used for Descriptive Identifiers of Variables.

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 a variable in the Asix system can be defined on the basis of its conversion function.

EXAMPLE

An example of a variable identifier:

```
asmen.Camac.06C_A111AF00.R4
```

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

NOTICE: In the 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 Simple Write Function

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

Write operation will be successful if:

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

The client is always authorized to set the value of a variable in any channel in the application of the 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 enable the option *Remote Write Access* with the use of: Architect > *Current Data* group > <*NONE channel name*> definition > *Advanced* tab > the option *Remote Write Access* for the channel parameters.

<*NONE channel name*> denotes the name of an Asix system channel which the authority to write will be assigned to.

The name of a client computer in the Asix system which will be assigned authority to write to the defined channel - name of the computer is defined as follows:

Table. Client Computer Name in the Asix System.

Description of Client Software Configuration	**Client Computer Name in Asix System**
Only the AsixConnect package is used and the aslink.ini file was not created or the line *Name* was not defined in the *ASLINK* section. By default, aslink.ini is in the directory: c:\AsixApp\cfg.	The name of a client computer in Windows system plus a dot added at the end- e.g. for the computer named "CLI", its name in the Asix system is "CLI." .
Only the AsixConnect package is used and the aslink.ini file was created, the *Name* line was defined in the ASLINK section.	The name of a client computer in the Asix system is provided in the aslink.ini, in the section *ASLINK*, in the line *Name*.
AsixConnect and the Asix application run on the same computer.	The name of a client computer in the Asix system is provided in the Asix application *.xml file with the use of Architect > *Network Module* > *Computer Name* tab.

6.3.2 Extended Write Function

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

Extended write operation will be successful if:

- a variable is provided with a write property (this depends on controller 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 a 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 the driver named NONE;
- in ini file of the application of the 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.

In the application configuration file parameterized by the Architect program (Asix ver. 5 and upper), the time and status write is declared in the channel parameters - if tabs with the channel parameters do not have such a parameter, use the possibility of entering parameters via Architect > *Miscellaneous* > *Directly Entered Options* tab.

EXAMPLE

An example of the fragment of an ini file:

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

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

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

6.4 Automation Server

6.4.1 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 the part of Asix system functionality in the scope of current data with use of Automation mechanism. Automation server is an in-process server implemented in form of DDL dynamic library and executed in the client memory space. The server is registered in Windows operating system as an 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 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*. When using a Visual Basic application that uses the package AsixConnect ver. 6, the name of the object *XConnect11.ServerCT* should be changed into *XConnect.ServerCT*. When converting from any version, change the name of your library types into *AsixConnect Type Library*.

6.4.2 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 the AsixConnect package.
- Obtain the Variable Definition Database of the Asix application from which the current data are to be retrieved or generate such a base.
- Using the *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 a 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 the *DataChange* events to the handler object in order to receive through them the information on current values of variables,
    - using the *SetItemActive* procedure, activate supplying by the server the current values of variables,
    - assign the *True* value to the property named *Active*, in order to start generating *DataChange* events by the 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 Definition Database may be loaded) using the *Init* function.

6.4.3 LoadChannel Function

The function calling syntax:

```
LoadChannel ChannelName
```

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

6.4.4 Init Function

The 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.

6.4.5 Read Function

The function calling syntax:

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

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

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. Values of the 'DataSource' Parameter for the 'Read' Function for 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: an asix system sampling rate + time from the latest update of the server cache memory	Variables to be read many times and variables active at the moment	1	Read variable must be active (See: _SetItem Active f unction)
dsDevice	Cache memory of the Asix system	Average	Average: the Asix system sampling rate + network transfer time	Variables to be read small number of times	2	
dsDriver	Industrial controller	Longest	Smallest: reading time from a device + network transfer time	Apply only if the value of a process variable from this current time is needed.	3	

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

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

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

The *Quality* (variable quality) parameter contains the quality flag described in section *5.3. Quality Bit Field*.

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

6.4.6 SetItemActive Function

Function calling syntax:

```
SetItemActive ItemID, ActiveState
```

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

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

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

If the variable is active then:

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

6.4.7 Write Function

The function calling syntax:

```
Write ItemID, Value
```

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

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

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

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

6.4.8 WriteEx Function

The function calling syntax:

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

The *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 identifier of the variable to which the new value is to be set.

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

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

In the *TimeStamp* parameter you should pass the new time stamp of a 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*.

6.4.9 Active Property

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

6.4.10 ServerState Property

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

Table. The 'ServerState' Property Values.

Value	Meaning	Number Value
ssRunning	The server is running correctly. The variable collection	1

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

6.4.11 StartTime Property

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

6.4.12 DataChange Event

The event syntax:

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

The *DataChange* event is triggered after every change of the process variable value. To trigger an event, the process variable must be active (see section *6.4.6 SetItemActive Function*) and the *Active server* property has to have the value *True*.

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

6.4.13 Error Handling

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

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

EXAMPLE

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

6.5 DDE Server

6.5.1 DDE Server

DDE, developed by Microsoft Company and available in the Windows operating system family, allows applications to share their functionality with operation of several messages defined in the DDE mechanism. The DDE server of current data enables access to the part of Asix system functionality in the scope of current data with use of DDE mechanism. The DDE server is implemented in the form of EXE type program. After starting, it logs on to the Windows operating system as a server under the names: *ServerCTDDE* and *CTDDE*.

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

6.5.2 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 the AsixConnect package.
- Obtain the Variable Definition Database of the 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 the C++ programming language, you should in the program:

- establish connection with the DDE server, passing as the *service* parameter the text *ServerCTDDE* or *CTDDE*; as the *topic* parameter you should pass the previously configured channel name;

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

6.5.3 DDE Operations Supported by the Server

Functions described in this section are those of the Windows system, designed for communication with DDE servers and available from C/C++ language level. The functions are made available by the 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
The *DDEConnect* function is designed to establish a connection with a DDE server.

If you like to connect to server of the Asix system pass the *ServerCTDDE* or *CTDDE* string as the *service* parameter. As the *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 the *service* parameter.

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

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

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

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

The write configuration method is described in 6.3.1. Simple 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
The *XTYP_ADVSTART* transaction activates the transfer of the current variable value from a server to a 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 the variable group in one transaction. In this case as the *item* parameter you should pass the list of identifiers separated with semicolons. The maximum size of the list of variable identifiers is 255 characters. This limit is specified by DDE mechanism.

XTYP_ADVSTOP
The *XTYP_ADVSTOP* transaction stops passing the current value of a process variable from a server to a 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 the AsixConnect package.

6.5.4 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 the 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 denotes that read operation failed; this status specifies also the 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. Values of the Quality (for DDE Server /Current Data).

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

During reading and updating a variable, data are transferred in the form of one row containing one or four columns. During updating the group of variables, data is 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 Windows system Control Panel 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.

6.5.5 Transfer of Error Information

If an operation fails, you can get error code by reading the value of the variable of a special name *_LastError_*. Reading the variable named *_LastErrorMessage_* you can get a text description of the error. Don't forget about 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.

6.5.6 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 the AsixConnect DDE server, i.e. *ServerCTDDE* or *CTDDE*, should be passed as the *service* parameter. As *topic*, the channel name (character * if the basic channel is to be used) should be passed. 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 a cell appears the current value of a variable and then it is refreshed,
- error in attempt of access to the variable encountered by the DDE server - Excel receives the text error description,
- error on start of refreshing cycle because the connection to the DDE server could not be set up - Excel displays the N/A (*not available*) error. Check whether the DDE server is started and its name is correct.

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

Table. 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* and *XTYP_ADVSTOP* transactions have no equivalents in 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. *Item* contains the names of variables separated with semicolons, in this formula. A maximum size of the *item* parameter is equal to 255 characters. This limit is specified by DDE mechanism.

Application of variable groups in place of single variables has a large effect on efficiency of updating variable values. 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 should have so many rows as the number of variables in the group. After editing click on and hold down *CTRL+SHIFT* keys and then click on *ENTER* key. In this way, an introduced formula will be placed to all cells of the range.

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

6.5.7 'DDE Server' Service

AsixConnect includes the DDE server module, which is not operating as a normal application but as a service of the Windows operating system. This module may be executed in the Windows 2003/XP/2000/NT4 environment only. The DDE service is added to the list of services 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 a "service started manually".

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

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

NOTE: The *DDE Server* service 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.

STEP 2

In the main window of the program, look for *DCOM Configuration* > *Aslink Manager Application* and select *Properties.*

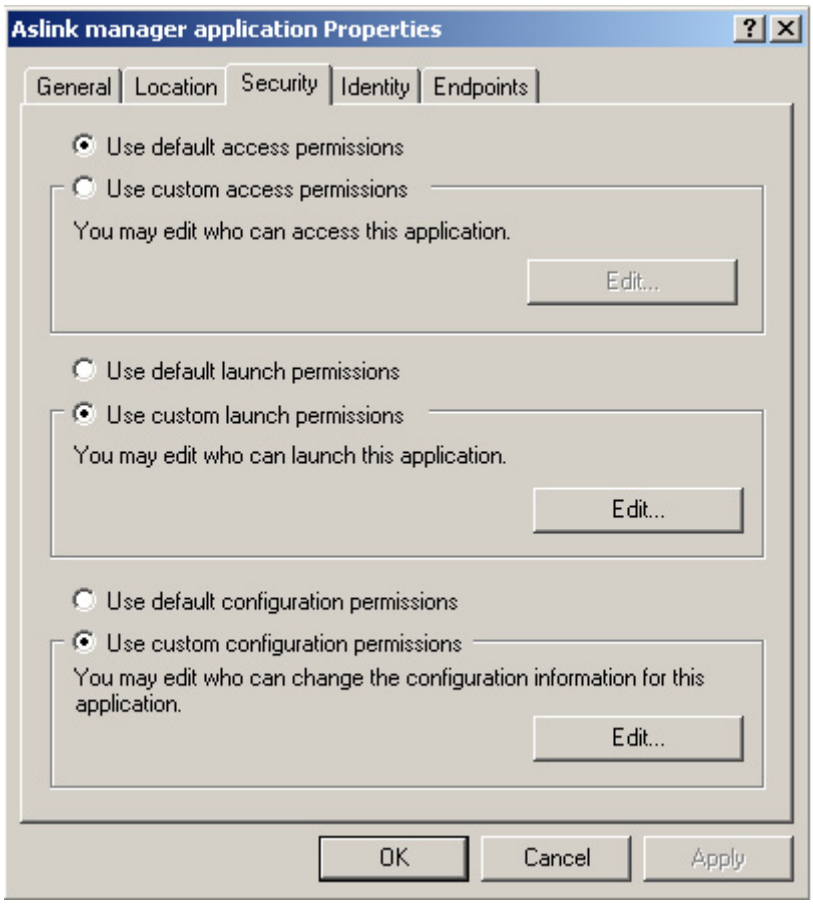


Fig. The 'Aslink Manager Application Properties' Window - Security.

STEP 3

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

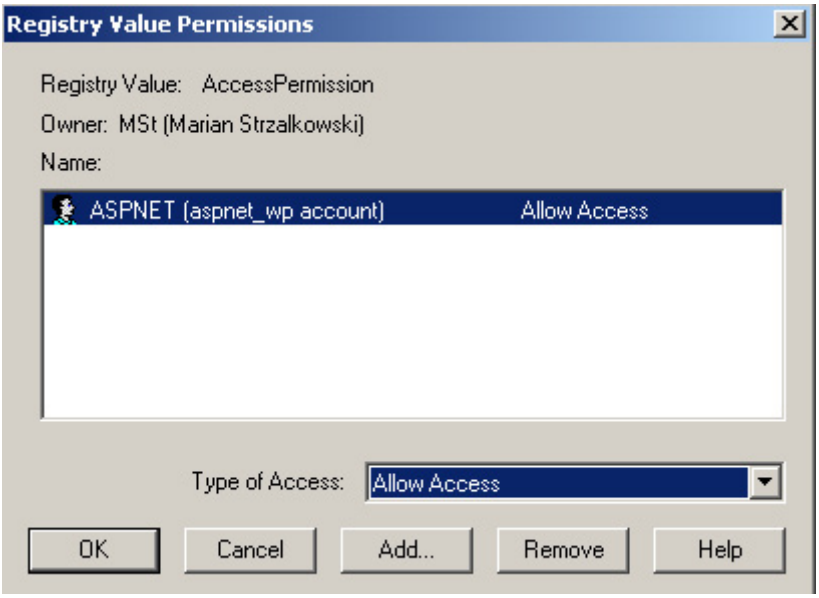


Fig. The 'Registry Value Permissions' Window.

In the window *’Aslink Manager Application Properties'*, click on **(!)** the button ***Apply***.

STEP 4

Identity of the user who will be authorized to start the Aslink module should be defined. It is done in the *Identity* tab of the *’Aslink manager application Properties'* window. Set *Administrator* as the *User* and the relevant password. If the Asix system does not operate on the same computer, the option *The Launching User* may be selected. The modifications are accepted by clicking on the button *Apply*.

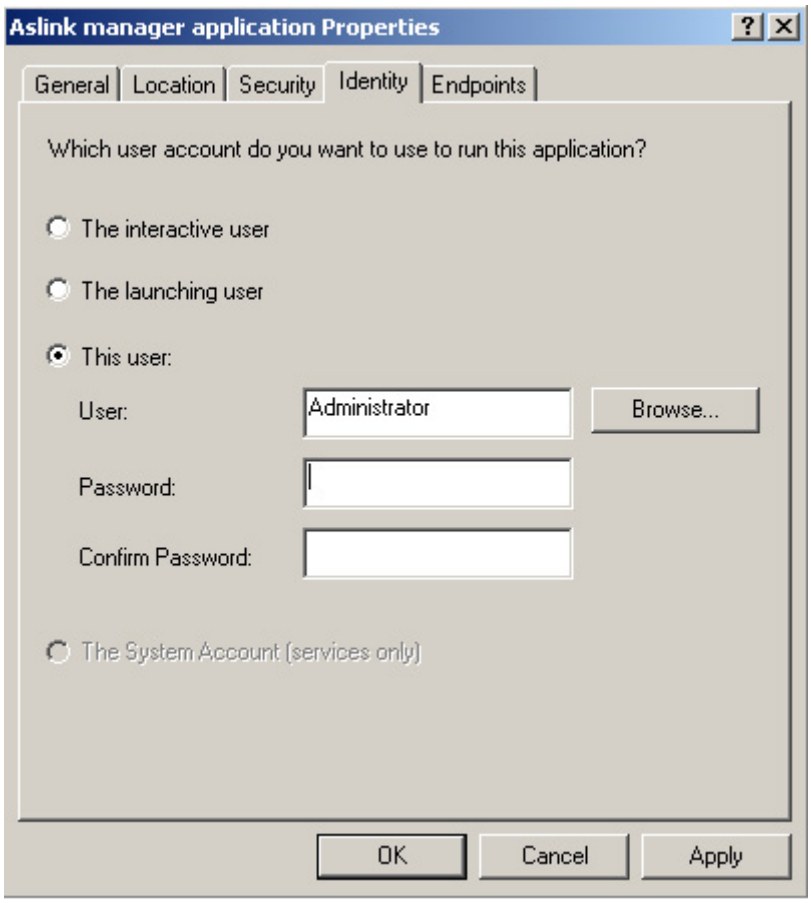


Fig. The 'Aslink Manager Application Properties' Window - Identity.

6.6 OPC Server

6.6.1 Technical Specification

OPC server of the 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 the 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 the Asix system, registers in the 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 the Asix system, registers in the Windows system under the name *ServerCTOPC.App*. Server registers also in the system registry of object category as the one complying with the specification *OPC Data Access 2.0*. Furthermore, the 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 server (*out-of-process*), i.e. in the form of EXE file. OPC calls are handled in the main thread of server process. An 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 the Asix system.

The 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 the thread

handling model *APARTMENT_THREADED*. In this model, objects are created and handled by main thread of process and all callings to objects are handled in this thread.

The 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.

The *OPC_STATUS_NOCONFIG* constant is returned as the server status if either no Variable Definition Database was defined in server options or loading the defined Variable Definition Database failed. Then descriptive variable identifiers are handled only.

6.6.2.3 Browsing of Variable Definition Database

The 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 Definition Database. By default, when this option is disabled, the branches of the tree that represent the Variable Definition 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, the function *MatchPattern* working as the *LIKE* function in Visual Basic language is recommended by the OPC specification.

The table presents special characters allowed in the pattern and possibility of their matching.

Table. Special Characters Allowed in the Pattern for the 'r;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 the list of characters.
[!lista_znaków]	Any single character off the list of characters.

Filtering according to a variable type was not implemented because the server may pass practically every variable in any format and an appropriate conversion may be performed by the *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 the Asix system.

According to the OPC specification, the function *GetItemID* returns for a 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 the Asix system identifiers are unique and providing them with path is not required.

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

6.6.2.4 Browsing Variable Properties

The 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. The sServer 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 the variable parameter is interpreted as a name of an Asix computer, which the variable is to be retrieved from. For description of channel definition, see 3.1. Channels.

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

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

If an error occurs during initialization of updating, the server will cyclically try to begin updating until this action will be successful. The current status of a variable is returned by functions reading the variable value from the server cache 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. The server is provided with internal buffer for variable values, from which values are retrieved when they are read from the cache memory. Data are read from the device via the Asix system network. Time of read operation from the device depends on the time necessary to find the data server of the 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 a 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 a queue and due to this asynchronous operations end their action immediately and client may continue its operation.

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

In general, the server uses the main thread for handling the requests of OPC clients and to send and receive data to/from the 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 Writing New Value, Quality and Time Stamp

The *Write function* of *ISyncIO* interface was extended, with regard to the 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 a respective array item of the *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 the 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 Definition Database of the Asix system from which the current data are to be retrieved or generate such a base.
- 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 ServerCT. In this program, you need to:
  - create object of ServerCT type, giving the name of the previously configured channel as the parameter;
  - 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 the *channelName* parameter, the channel name should be given (see: *3. Connection Configuration*).

6.3.2 Dispose Function

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

This function is used for releasing the resources used by the *ServerCT* object. This function must be called after the use of the *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 chapters.

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);
```

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

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

Table. 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 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 the value of *CTDataSetFlags* type.

Table. 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 the *itemIDs* parameter. The table contains

	one row with current values of each variable. The value is of *Object* type.
CTDataSetFlags.ItemValueAsString	The variable value is turned 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 (*dsDevice* parameter equal to *DataSource.Device*) is supported by Asix in versions supplied after 2000/06/01 (Netsrv module in v. 3.1.0 or later). In earlier versions, readout from the Asix cache memory is automatically replaced by readout from the controller.

6.7.6 Write Function

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

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

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

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

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

6.7.7 Write Function - Extended Write Operation

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

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 variable value. It is applied by the *Read* and *Write* functions and the *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. The Contents of the 'r;ItemState' Structure (for the .NET Server / Current Data).

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

6.7.9 SetItemActive Function

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

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

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

The *ActiveState* parameter should contain the value *true* if variables are to be active and the value *false* if variables are to be inactive.

If the variable is active then:

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

6.7.10 ItemsChange Event

Declaration of event source:

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

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

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

The *OnItemsChange* event is triggered after every change of the process variable value.

To trigger an event:

- the process variable must be active (see section 6.7.9. SetItemActive Function);
- the *Active* server property has to have the 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 active variables from server to client. A variable is active when the function *SetItemActive* with the *itemIDs* parameter containing the name of the variable and the parameter *activeState* equal to *true* was called. In order the current values would be transferred,the *Active* property has to take the value *true*. A default value of the *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 the *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 the *ServerCT.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 (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);
    }     }
```

The pool of servers:

- creates several objects of the *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 the cache memory of the ASP.NET application;
- makes the objects available for successive calls under the ASP.NET application;
- reports the *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 is sufficient for unique identification of a process variable, i.e. the name used in the Asix system application. All other information on the variable needed for communication with the Asix system is retrieved from the Variable Definition Database. In order to obtain the Variable Definition Database, you should address to the administrator of Asix system application 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 Definition Database should be determined interactively or programmatically. For details, see the chapter 3.6.2 Variable Definition Database.

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

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

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

STRING type is available in a .NET server only.

7.2 Operation Without Variable Definition Database

NOTE: In general, using the Variable Definition Database is recommended. Using the information contained in this chapter should be exceptional.

The AsixConnect server of archive variables accepts also descriptive identifiers of variables and no Variable Definitions Database loading is needed when only the identifiers are used.

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

The descriptive identifier takes the following form:

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

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

The Asix system in version 2.68, 3.x and 4.x makes the following archive types 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 the 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. The List of Supported Aggregates.

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

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

Quality and time stamp of aggregate is set depending on the value of the *Aggregate calculation algorithm* option.

7.3.2 Askom Algorithm

In the algorithm *Askom*, when calculating aggregates, the time period from which data is retrieved is divided into intervals of fixed length. The unit is calculated for each interval using the data archived for the duration of this interval. It is assumed that each interval is a left-closed and right-open interval.

The aggregate time stamp is equal to the end of an interval, except the aggregates *Start* and *Previous_Known* - for which it is equal to the interval start.

The quality of the sample has the value *qualityGood* if the percentage of all samples for the quality *qualityGood* used to calculate the aggregate is equal to or exceeds the *Threshold of good quality* option. The default threshold is 80%.

7.3.3 OPC Algorithm

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

An aggregate time stamp is equal to the beginning time of interval, except the aggregates *End* and *Last* - for which it is equal to the interval end.

The quality of the sample has the value *qualityGood* if qualities of all samples used for aggregate calculation have the value *qualityGood*. If one or more samples used for calculation of aggregate have qualities that differ from *qualityGood*, then the aggregate quality takes the value *qualityUncertainSubNormal*. If the status of all samples used for aggregate calculation has the value *qualityBad*, then the aggregate status is equal to the value *qualityBad*.

7.3.4 Raport Algorithm

In the algorithm *Raport*, when calculating aggregates, the time period from which data is retrieved is divided into intervals of fixed length. The aggregate is calculated for each interval using the data archived for the duration of this interval. It is assumed that each interval is a left-closed and right-open interval.

The aggregate time stamp is equal to the start of an interval.

The quality of the sample has the value *qualityGood* if the percentage of all samples for the quality *qualityGood* used to calculate the aggregate is equal to or exceeds the *Threshold of good quality* option. The default threshold is 80%.

7.4 OPC Time Format

OPC syntax of relative time is as follow:

```
keyword +/- offset +/- offset &ldots;
```

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 an integer number that specifies its multiplication factor and direction.

Table. 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. 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 the beginning time of data for a day report generated in the current day (DAY = first time stamp of today, -1D first time stamp of yesterday, +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 the last day of a 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 part of keyword in the described format should be omitted.

7.5 Automation Server

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

When converting a 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*. When using a Visual Basic application that uses the AsixConnect package in version 6, the name of the server object *XConnect11.ServerHT* should be changed into *XConnect.ServerHT*. When converting from any version, you should change the name of your library of types on the *AsixConnect Type Library*.

7.5.2 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 the AsixConnect package.
- Obtain the Variable Definition Database of an 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 a 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 Definition Database may be loaded) using the *Init* function.

7.5.3 LoadChannel Function

Function calling syntax:

```
LoadChannel ChannelName
```

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

7.5.4 Init Function

Function calling syntax:

```
Init InitString
```

This function is designed for setting the server parameters. See 3.5 Program Configuration for description of the function. Available options are described in the following sections.

7.5.5 ReadRaw Function

Function syntax:

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

The ReadRaw function reads values, statuses and time stamps of a variable from an archive of a 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 the period from which data are to be read. These parameters should contain value of *DATE* or *STRING* type. A value of *DATE* type directly contains time stamp; local time should be used. A value of *STRING* type is considered as relative time. Two formats of relative time are recognized: *OPC* and *asix-raporter*. OPC syntax is described in 7.4 OPC Time Format. The format *asix-raporter* is described in the Asix system documentation (Asix.hlp) in the chapter on Raporter module.

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

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

7.5.6 ReadProcessed Function

Function syntax:

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

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

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

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

The *AggregateName* parameter should contain the aggregate name.

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

7.6 OLE DB Server

7.6.1 OLE DB Server

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

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

7.6.2 Identification and Configuration

OLE DB of the Asix system registers in the operating system under the shorted name *ServerHTOleDB.App* and under the 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.

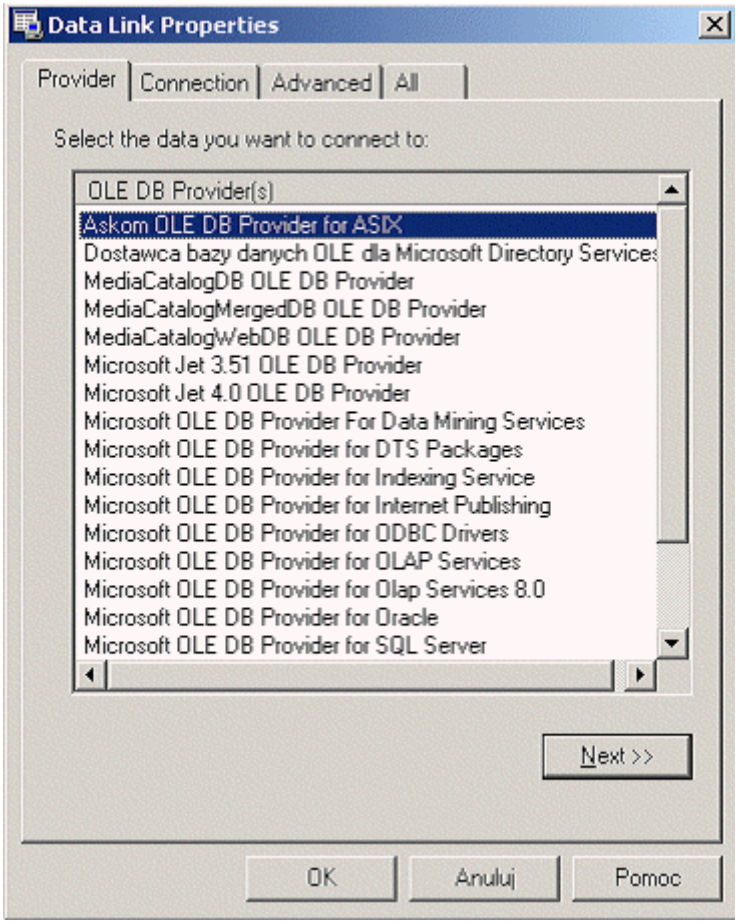


Figure. 'Data Link Properties' Window for Parameterization of OLE DB Server - Provider Tab.

The server handles the standard parameter named *Data Source*. As the *Data Source* parameter full directory path to the Variable Definition Database should be entered.

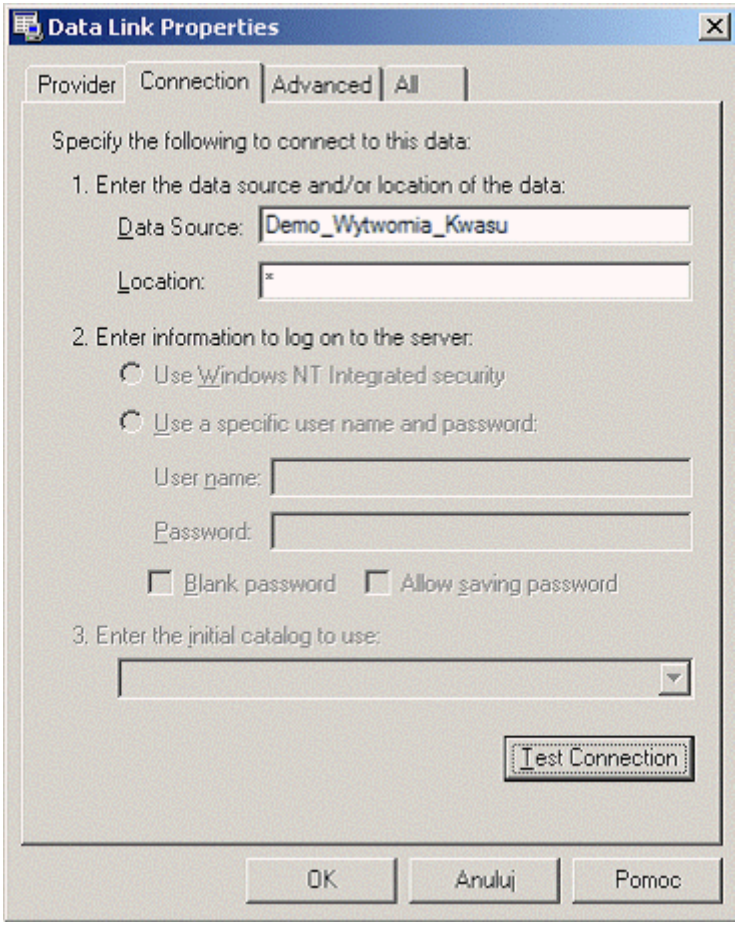


Figure. 'Data Link Properties' Window for Parameterization of OLE DB Server - Connection Tab.

7.6.3 Tables

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

7.6.4 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. Characters of Meta Language for asix.SQL Queries.

Characters of Meta Language	Meaning
UPPERCASE LETTERS	Keywords of Asix.SQL language.
Italics	Parameters supplied by the 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 the command SELECT 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 { = | < > | ! =  } status_value
        | Value { = | < > | ! = | > | > = | ! > | < | < = | ! < }
number
        | Value IS [NOT] NULL }
status_value    ::=    {    qualityGood    |    qualityUncertain    |
qualityUncertainSubNormal |
                              qualityBad    |    qualityBadNoData    |
qualityBadNoBound }
aggregate_name ::= { Start | End | Delta | Min | Max |
                                    Range  |  Total  |  Average  |
Average0 }
```

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

In the result of a query:

- column *TimeStamp* will contain the sample time stamp;
- column *Quality* will contain the sample time quality;
- column *Value* will contain the value of the sample provided if the quality of the sample is good. If sample quality takes the value *qualityBad*, *qualityBadNoData* or *qualityBadNoBound*, the value of the sample will be equal to NULL.

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

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

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

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

If the clause *AGGREGATE* is used, the format of read data is the same as given in description of the *ReadProcessed* function (7.5.6 ReadProcessed Function) of Automation server of archive data. If the clause AGGREGATE is omitted, format of read data is the same as given in description of the *ReadRaw* function (7.5.5 ReadRaw Function).

7.5.6 Examples of Queries

Read raw values of the variable K8_11U14 from the 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 the variable K8_11U14 from the 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 the variable K8_11U14 from the previous hour. The variable name in example is closed in brackets. Brackets should be used when the 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 a variable from the current hour.

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

Read minimum values of a 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 a 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 a 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 a 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 a 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 the 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 the 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 the AsixConnect package.
- Obtain the Variable Definition Database of the 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 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 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 an object of the *ServerHT* type, giving as a 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 an object of the *ServerHT* class.

As the *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 the *ServerHT* object. This function must be called when the use of the *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);
```

The *ReadRaw* function is used to read archive, raw values of process variables from a specified time 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 is 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 a period from which data are to be read.

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

The function returns limit values for given period of time as well. If in the archive the sample, corresponding exactly to the time point defined by the parameter *dateTimeFrom*, was not registered, the latest sample before that time is returned. If in the archive the sample, corresponding exactly to the time point defined by the 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 the 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);
```

Thje *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 Asix system archive.

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

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

is used and its value should be a table of aggregates. How to calculate aggregates, see *7.3 Aggregates*. The list of supported aggregates is given in the table below.

Table. 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 the 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 the value of variables is not accessible, 0 value is used.

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

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

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

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

Constant	Description
HTDataSetFlags .Default	There is one column, named as a variable name, created in the *HTData* table for each value in the *itemIDs* parameter. If a 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 a time stamp of samples.
HTDataSetFlags .ShowQuality	For all variables (without the value column) will be inserted the column with a sample quality into the *HTDataSet* table. This column has a name composed of a variable name with the text '-Quality' added.

HTDataSetFlags.ShowErrorStringWhenReadError	There is an error description inserted into the column if read error occurs. If the flag is not used, the null value is inserted.

After the read operation has been finished, the result is returned in the form of the *ReadProcessedResult* class object for the first version of the function or in the form of array of the *ReadProcessedResult* class objects 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 DataSet class object 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);
```

The *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 RelativeTimeSpan 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 Class

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

Field	Content
ReadSucceeded()	The function turns back the value *true* if the value of the *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 the 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 period from which archived data were read.
Samples	Read archival data.

7.7.11 ReadProcessedResult Class

The object of the *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. The Contents of the Structure for the 'r;ReadProcessedResult'r; Class Declaration (for the .NET Server / Archive Data).

Field	Content
ReadSucceeded()	The function turns back the value *true* if the value of the *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 the *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 a 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 the execution of the archive data reading function has been finished, the contents of the structure is as follows.

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

Field	Content
TimeStamp	Time stamp of the variable sample; local time is used.
Quality	Quality of the 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 the *true* value if the value of *Quality* field indicates the variable state quality to be good.

The sample quality may take one of the values described in section 5.2 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. 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 corresponded to 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 the archive when reading the data (reading the data from the

		period that does not occur yet or no timer synchronization between a client computer and the Asix system station).

7.7.14 ItemStringSample Structure

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

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

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

7.7.15 ItemProcessedSample Structure

The object of the *ItemProcessedSample* type is designed for transferring the values of archive variables. The *ItemProcessedSample* structure is used by the *ReadProcessedResult* class.

Declaration of the *ItemProcessedSample* structure is as follows:

```
public struct ItemProcessedSample
{
    public DateTime StartTimeStamp;
    public DateTime EndTimeStamp;
    public Int32    Quality;
    public double   ItemValue;

    public bool     IsQualityGood();
};
```

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

Table: *The Result of the 'ItemProcessedSample' Structure Declaration After Completion of the Archival Data Read Function (for the .NET Server / Archive Data).*

Field	Content
StartTimeStamp	The time stamp of the interval beginning of data used to calculate the sample; local time is used.
EndTimeStamp	The time stamp of the interval end of data used to calculate the sample; local time is used.
Quality	Quality of the variable sample. Good or uncertain quality means that the *ItemValue* field is filled.
ItemValue	Variable values; the field is of double type.
IsQualityGood()	Function turns back the *true* value if the value of the *Quality* field indicates the variable state quality to be good.

Jakość próbki może przyjąć jedną z wartości opisanych w rozdziale Jakość pomiaru (patrz: punkt 5.2.). Ponadto w przypadku jakości złej mogą być ustawione flagi specyficzne dla danych archiwalnych. Flagi te mieszczą się w zakresie numerów bitów 16-31 (szczegóły - patrz: poniższa tabela).

The sample quality may take one of the values described in section 5.2 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. See the table below.

Table: 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 corresponded 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 no timer synchronization between a client computer and the Asix system station).

7.7.16 DataSet Object

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

7.7.17 Operation in ASP.NET Eenvironment

In case of operation in the ASP.NET environment, the object should be created using the *ServerHT.ServerPool.Get()* expression. The *ServerPool* object is a static field of the *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 the *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 the *ServerHT* class for the application (the channel name is retrieved from Web.Configfile, see 3.2 How to Specify the Channel Name);
- stores the objects in a 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.

8 Alarms

8.1 .NET Server

8.1.1 Application of Server

The *ServerAL* class enables access to the functional part of the 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 the AsixConnect package.
- Using *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 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 an object of the **ServerAL** type, giving as a 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 initialize the object of the *ServerAL* class.

As the *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 the *ServerAL* object. This function must be called after the use of the *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 an 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 alarm type. One of the constants described in the table below or the bit sum of these constants should be given as its value.

Table. Constants of the 'alarmType' Parameter for the ’r;ReadActive’ Function for Retrieving the Information on Alarms.

Constant	Description
AlarmType.NoFiltering	Filtering by the 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 the alarm. As its value one of the constants described in the table below or the bit sum of these constants should be given.

Table. 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 the *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 a 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 the 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 the *DataSet* class to be created on the basis of the data stored in the table of the *Alarm* type structures.

The table of the *Alarm* type structures 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 a *DataSet* object as sorted out in the ascending or descending order. Sorting out takes place by the alarm time.

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

8.1.8 Alarm Structure

The objects of the *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. The Contents of the 'r;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 the *ASP.NET* environment, the object should be created using the *ServerAL.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 *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 the *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 the *ServerAL* 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 the cache memory of the 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.

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 the *ServerRP* type, giving as a 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 the report group name 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. 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 - report group name (it corresponds to tree nodes);
Length - duration of report generation (it corresponds to ”r;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 the *ServerRP* object has been finished. Calling must take place from the same thread the object was created in.

9.1.5 Init Function

Type topic text here.

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. 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 the *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. Contents of the 'ReportInfoNet' Structure.

Field	Content
FileName	Name of the report file
ReportName	Report name
ReportDate	Report date
FileDate	Date of report file creation
FileSize	File size
ReportFormat	Format of the 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 the 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 the 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 the 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

10.1 Web Service Server

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

10.2 Installation

The WebService server is located in c:\asix\WebService directory. In order to make it available within the network, run the *IIS* 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.

10.3 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 supplies data for dynamic HTML pages, than the *Control variable* and *Limits of variables* options should be enabled in the channel.

10.4 Clients

10.4.1 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 / *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).

10.4.2 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 do so, 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 the client to handle the *cookie* mechanism. To enable this mechanism, after the server object of the *Asix.XConnectWebService* type is created in the application, you should assign the object of the *System.Net.CookieContainer* type to its *CookieContainer* properties.

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

The object of the *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 = "";
}
```

10.5 Variable Definition 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 the *ServerHT* class (see: 4.3.1.5 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 the *ServerHT* class (see: 4.3.1.6 ReadAttributesN Function).

10.6 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 a parameter of the *Read* function the variable name is assigned and the function returns the value of this variable in the form of the *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 the *ItemStateWS* structures.

ItemStateWS Structure

Objects of the *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. Contents of the 'ItemStateWS' Structure (Web Service Server).

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

10.7 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 the *ServerHT* object described in section 7.7.5 ReadRaw Functions.

10.8 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 a given process variable in a defined period. Its operation is the same as of the *ReadProcessed* object of the *ServerHT* class (see: 7.5.6 ReadProcessed Function). It only differs in the 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.

10.9 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 an Asix system application. The operation of the *ReadActiveAlarms* function is the same as of the parameterless *ReadActive* function of the *ServerAL* class.

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

10.10 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 an Asix system application.

The operation of the *ReadHistoricalAlarms* function is the same as of the 3-parameter *ReadHistorical* function of the *ServerAL* class. The operation of the *ReadHistoricalAlarmsEx* function is the same as of the 8-parameter *ReadHistorical* function of the *ServerAL* class (see: 8.1.6 ReadHistorical Functions). It only differs in the 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 operation. Filename of this log is:

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

As an identifier the following values may passed.

Table. 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 the <*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 a new name will be created. In case of lack of disk memory, the old log files are automatically removed.

EXAMPLE

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 the *Log* subdirectory of the same directory where the AsixConnect package was installed; the most frequently in c:\asix\Log. The 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 the 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. Error Codes Returned by OPC Server.

Name of Error Code	Error Code	Description
ASKOM_E_NetInit	0xC0048100L	Error in initiating network of the 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
ASKOM_E_NetDeinstallClient	0xC0048103L	Error during closing the client of network of Asix
ASKOM_E_NetFindServer	0xC0048104L	Function calling error *Searching of data server*
ASKOM_E_NetFindServer_NotFound	0xC0048105L	Data server in network of the Asix system was not found
ASKOM_E_NetLinkToServer	0xC0048106L	Error when connecting to data server of Asix
ASKOM_E_NetLinkToServerTimeout	0xC0048107L	Timeout during connecting to data server of Asix
ASKOM_E_NetLinkToServerNegativeAnswer	0xC0048108L	Error during connecting to data server of Asix - negative response
ASKOM_E_NetCloseLink	0xC0048109L	Error during closing connection with data server of Asix
ASKOM_E_NetSendData	0xC004810aL	Data transmission error in network of Asix
ASKOM_E_NetAnswerTimeout	0xC004810bL	Timeout in waiting on response in network of Asix
ASKOM_E_ItemUnknown	0xC0048120L	Error - variable unknown in Asix
ASKOM_E_TheSameItemWrittenMoreThanOnce	0xC0048121L	Errors in parameters during write operation. A variable is written more then once.
ASKOM_E_WriteError	0xC0048128L	The Asix system returned error during attempt of write operation
ASKOM_E_WriteError_NetworkServerNotFound	0xC0048129L	The Asix system returned error during an attempt of write operation- server servicing the variable was not found
ASKOM_E_WriteError_IllegalCommand	0xC004812aL	The Asix system returned error during attempt of write operation -a 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	The Asix system returned error during attempt of write operation - transmission error during communication with device
ASKOM_E_WriteError_DeviceRequestIllegal	0xC004812dL	The 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 the HASP key
ASKOM_E_HaspNotFound	0xC0048048L	The 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 the 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_ErrorDuringConfirmingReadingArchiveVarDat	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.