www.asix.com.pl

Asix.Evo - Application Parameterization

Doc. No ENP7E012
Version: 2012-08-17

ASKOM

ASKOM® and A**six®** are registered trademarks of ASKOM Spółka z o.o., Gliwice. Other brand names, trademarks, and registered trademarks are the property of their respective holders.

All rights reserved including the right of reproduction in whole or in part in any form. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without prior written permission from the ASKOM.

ASKOM sp. z o. o. shall not be liable for any damages arising out of the use of information included in the publication content.

Copyright © 2012, ASKOM Sp. z o. o., Gliwice

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 Configuration of the Application Workstations

When configuring the Evo type application within a single project, the parameters for all workstations which will participate in the application need to be defined. Generally, each server workstation or stand-alone workstation is explicitly declared, while terminal workstations are parameterized altogether.

When creating the application using the wizard, a basic set of workstations can be created at once. A workstation of name compatible with the computer name is created in each case. The primary objective of this workstation is to edit the application (although it can also be used as a production workstation).

1.1 'Stations' Operating Panel

Stations is the main panel used for managing the application workstations.

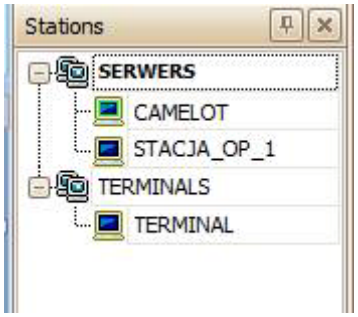

Fig. 'Stations' Operating Panel.

The illustration presents the configuration of workstations for a typical, small application. In the *Serwers* area, the following are defined: single engineering workstation (for developing and testing the applications) and a single production operator workstation. In addition, there is a single terminal workstation defined as well. While the application is running within the context of this station, the number of terminals which may be launched is arbitrary.

Stations panel provides two functions. Firstly, it allows to switch between workstations while the station-dependent parameters are edited. Secondly, the panel is used to create and manage new workstations. Workstation management commands are available through the panel context menu. Additionally, the workstation may be moved between the areas using the 'drag and drop' method.

Locating a few workstations in a single area allows for hierarchical parameterization. The parameters defined in the context of the area apply to all the workstations of the area, if not defined otherwise in the workstation itself.

1.1.1 Creating New Workstation

To create a new workstation, the *New Station ...* command from the panel context menu must be selected.

The workstation parameters are described in the table below:

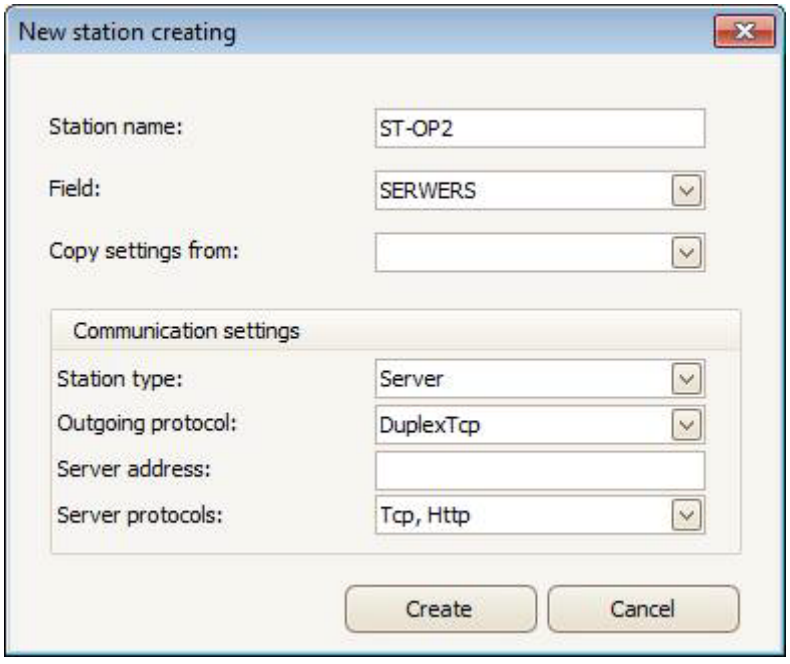

Figure. Window to Create a New Workstation.

Table: Workstation Parameters.

Parameter	Meaning

Workstation name	Workstation identifying name. In the case of server workstations it is preferable to use the name that matches the computer system name on which the workstation will be run.
Field	The name of the area where the workstation is to be located. It will inherit the area settings. It is also possible to create a workstation outside any area.
Copy settings from	The name of another workstation or area from where the settings are to be copied. Used for creating twin workstations.
Station type	A key parameter determining the intended use of the workstation. Depending on the type chosen, the range of possible functions of the workstation may differ. The following types are available: • *Server* - the workstation may share its data with other workstations and it can perform the function of a network controller in the alarm system. • *Terminal* - a window type terminal which performs exclusively the function of data retrieving client in the network system. • *WWW Terminal* - web terminal which collects all data from server workstations by means of a network mechanism. • *Standalone* - the workstation used in configurations without network communication.
Outgoing protocol	The protocol used for the communication of a data client type. The following protocol types are available: • DuplexTcp - the primary mode of communication in the local networks. It can also be used in wide area networks if the transmission is not blocked by a firewall. • Tcp - a protocol similar to DuplexTCP but slightly less efficient. • Http - a protocol intended mainly for the WWW (web) terminals. However, the WWW Terminal may also use the DuplexTcp protocol if the communication is not blocked by a firewall.
Server address	The address of a server that uniquely identifies the workstation in the network. It may be a system name or an IP address. The box may be left blank if the name of the workstation is consistent with the computer name.
Server protocols	The protocols used by a server workstation for communication with clients. A set of selected protocols must comply with the protocols used by other workstations which will connect to the server.

1.2 Setting the Parameters Depending on a Workstation

The application configuration data set contains elements common to all workstations (windows, charts, menus, images, etc.) as well as the parameters separately defined for each workstation. The workstation parameters refer to the configuration of communication channels and data archives, Action scheduler and the so-called *Stations Settings*. The operating panels for all of these setting groups can be opened through *Application Explorer* panel nodes.

The characteristic feature of the operating panels of particular workstations is that the name of the tab displays the name of the workstation (area) to which the displayed settings refer. Switching the operating panel to another workstation is done by clicking on the appropriate workstation panel node.

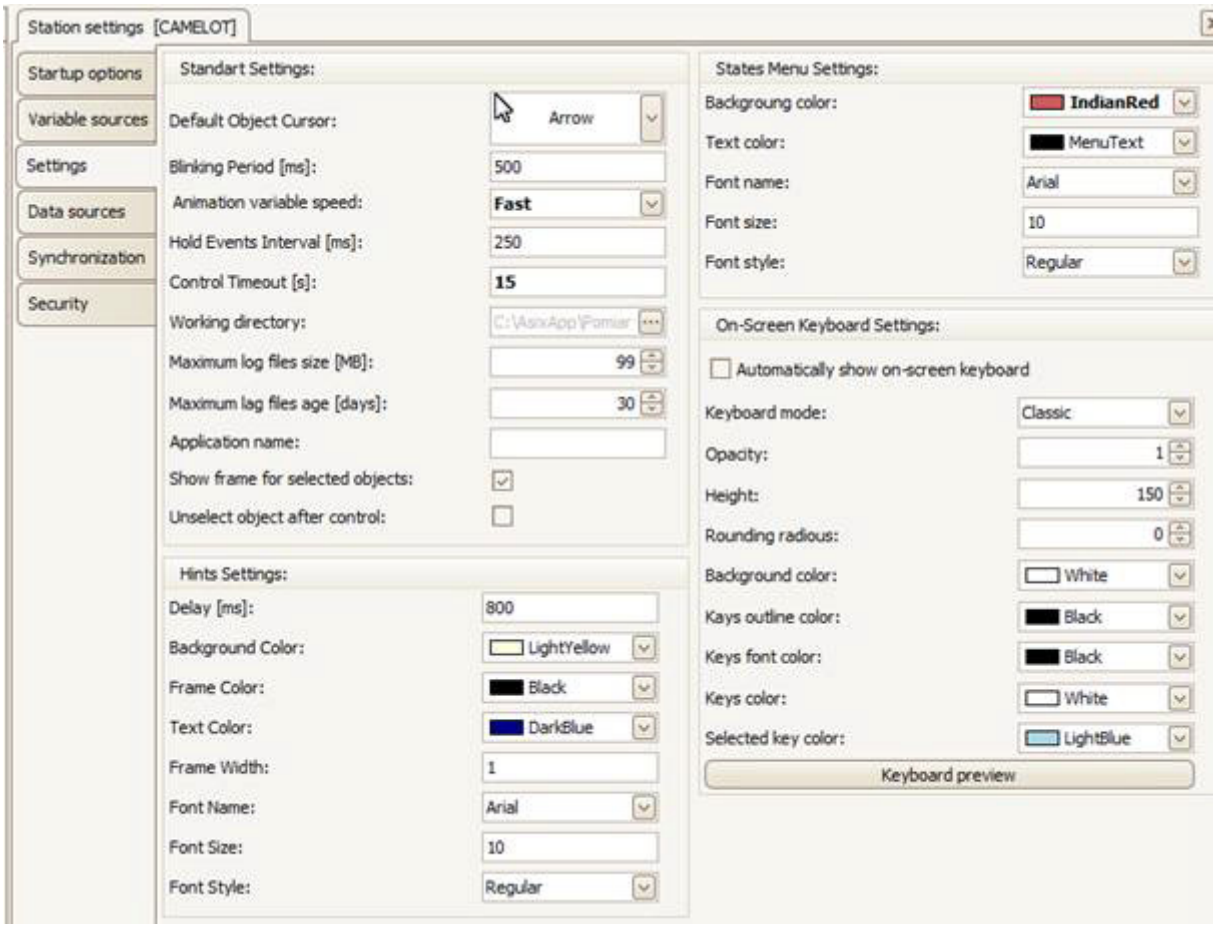

Fig. Editing Panel for Workstation Settings.

Editing panels for workstation parameters always show the current settings of the workstation. The values of these parameters result from the default settings, settings introduced to the parent area or custom settings. If a given workstation setting has been explicitly modified, it is specially indicated in the operating panel. For example, as the above illustration presents, some values (modified) are displayed in bold. Sometimes there are parameters that cannot be changed. It may occur when the

parameter has been defined in the parent area. Such parameters can be changed only in the context of the area in which they have been defined.

The workstation settings can be found in other operating panels, common to all workstations. In such a case, the tables with the parameters for all workstations are displayed. It may be, for example, a table specifying the roles which each of the workstation performs in the alarm management system. Another example is a table of workstation communication settings.

1.3 Configuration of Workstation Communication Parameters

The operating panel of workstation communication parameters is opened by selecting the *Global Settings* node in the *Application Explorer* panel and then selecting the *Communication* tab.

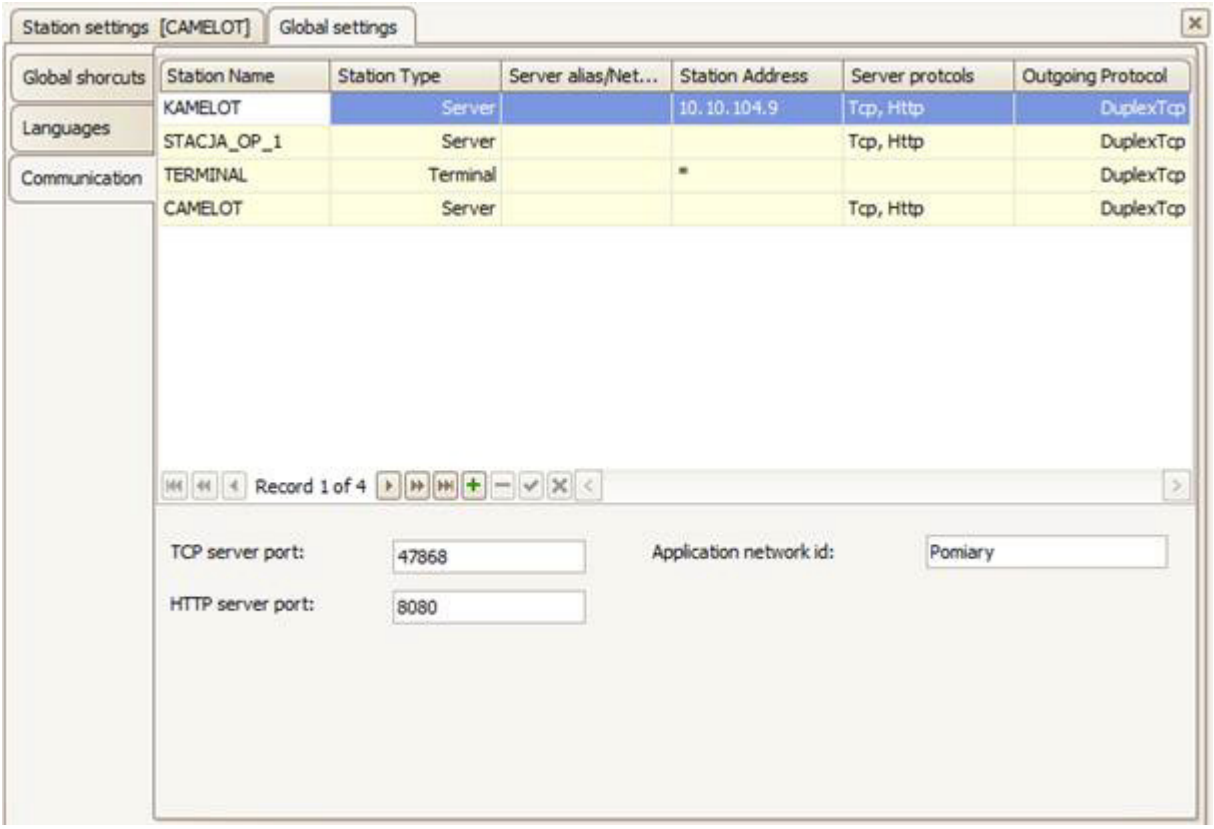

Fig. Workstations Communication Global Settings.

The panel shows the settings of all the application workstations and several global parameters. The global parameters define the port numbers used for the communication between workstations as well as the text identifier used to distinguish the workstations of the various applications running in the shared network.

The workstations shown in the table correspond in a large part to the workstations form the *Stations* operating panel. The parameters are derived from the settings specified during the creation of workstations.

A special type workstation may be added to the list of workstations using the + button. These workstations are not displayed in the *Stations* panel and the application cannot be run in the context of these workstations.

1.3.1 Server Aliases

Server aliases are used when a standard server workstation can be identified in the network by several addresses, e.g.: is equipped with two network cards. Depending on the location of terminal workstations in the network, the server can be accessed via different addresses.

When an alias is added, the name of the server and its alternative address needs to be specified.

1.3.2 Foreign Workstations

Foreign workstations are used when there is a necessity to connect to the server from another Asix.Evo application. Only communication between workstations which are defined within the same application is possible as standard. Foreign workstation mechanism allows for the data exchange between different Asix.Evo applications.

When defining a foreign workstation, the network ID of the application to which the foreign server belongs as well as this server address have to be specified.

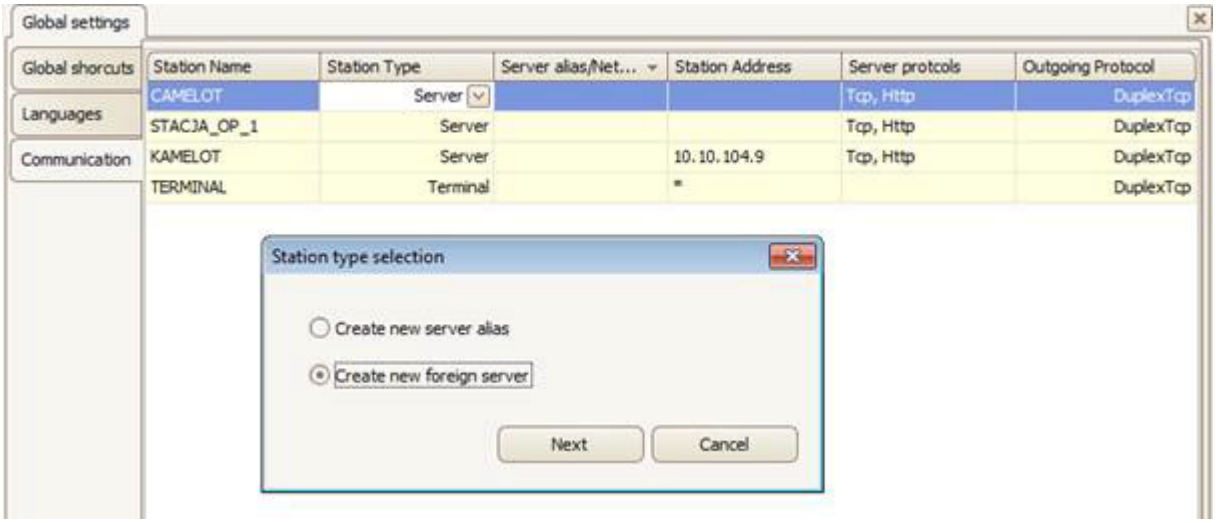

Fig. Adding Foreign Workstation.

1.4 Running the Application in the Context of the Selected Workstation

When running the Asix.Evo program, the name of the workstation in the context of which the program is to be run must be specified. By default, if the name is not specified, the program will attempt to run with the computer name. If the workstation of the name compatible with the computer name is not defined, the user will have to select the workstation from the list of all workstations. However, the workstation name can be forced in the command line using the *as* option.

AsixEvo -as=ST-OP2 "C:\AsixApp\Aplication1"

The example uses the *ST-OP2* workstation. The short-cuts that launch the application may be created by means of the short-cut creator run by a command from the *Tools* menu.

The workstation name, in the context of which the program runs, is displayed in the window title bar of the program.

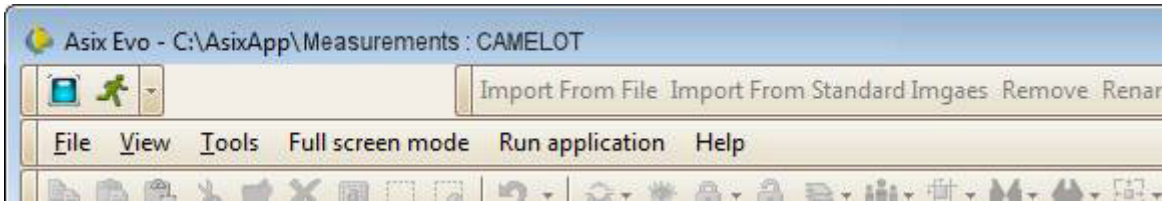

Fig. Displaying Workstation Name in the window title bar of the Asix.Evo program.

The run mode allows to display the workstation name on the diagram using the integrated virtual variable called *StationName*.

1.5 Connection Diagnostics

While the program is running, the communication system state may be monitored by a special diagnostic panel. It is possible to view the status of each active workstation of the application. The basic system information on workstations and the list of active connections is displayed as well.

When the architect mode is active, the panel is opened in the explorer panel of the application, using the *Diagnostics / Connections* node. In the run mode the communication diagnostic panel is available through the control panel window.

2 Current and Historical Data Access Configuration

A key element of any application is to define a set of process variables and the way they can be captured and archived. Parameterization of the access to the process data is performed independently for each workstation of the application. The only exception is the variable definition database which is identical for all workstations, although workstations may use different databases of variable definitions.

2.1 Data Server Configuration

The key decision to be made is the way of collecting the data. To read the variable values from the drivers and to archive them, the Evo type applications use Asmen and Aspad modules, which are also used in the applications of previous type (classic type). These modules are run under the so-called **data server**.

In the application explorer panel, select the *Stations Settings* node and then *Data Sources* tab.

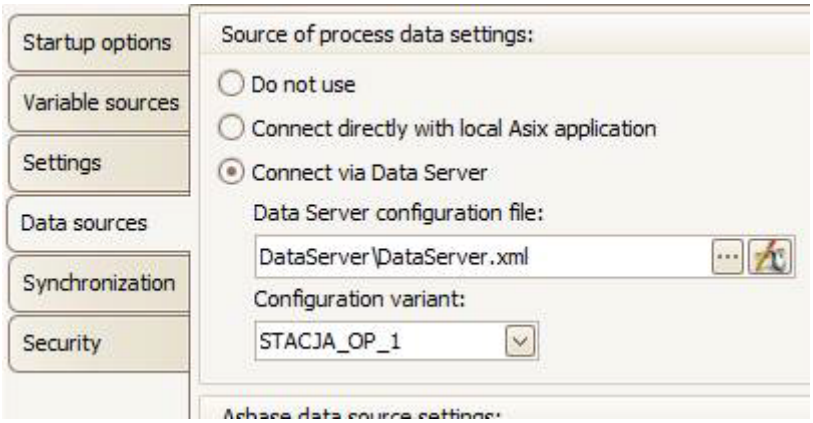

Fig. Workstation Process Data Source Settings.

The name of the workstation which is being parameterized is displayed in the title tab. The workstation may be switched in the *Station* panel.

It is possible to operate using one of the three strategies:

a. *Do not use*
   In this mode, the data server is not used, and thus direct reading of data from the controllers and their archiving is not possible. As a rule, the mode is

used only on the window terminal workstations and WWW workstations - the workstation loads all the data form another application workstation which operates in the server mode.

b. *Connect Directly with local Asix application*
Occasionally used mode which allows accessing the data via the Asix application in the previous type, running on the same computer. Typically used to create the browser version of the application of previous type. Such application is converted into the Evo version, and the Asix.Evo server workstation is launched on the same computer as the original server and is used as a data source for the browser workstations.

In this option, the Asix.Evo data server is initialised within the classic Asix process. To do this, the use of the server in the Architect program should be enabled: the *Programs/Asix.Evo* tab in the *Start parameters* panel.

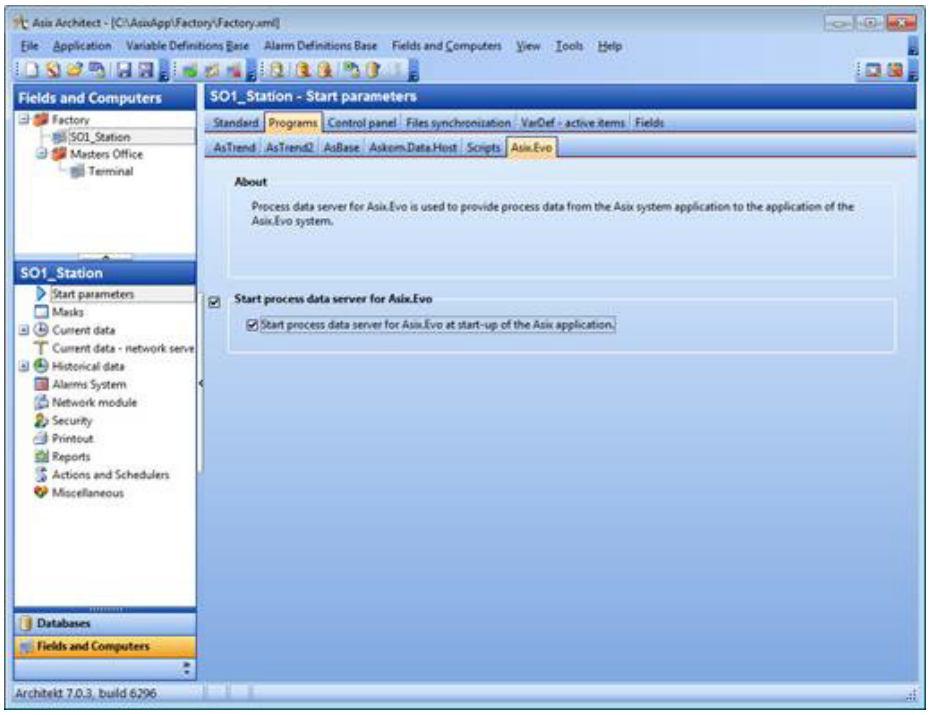

Fig. Architect Program > Start Parameters > Programs > Asix.Evo Tab.

c. *Connect via Data Server*

This is the basic operation mode. Virtually in every application, at least one workstation operates in this mode. The data server reads the data from the controllers which control the process and can archive them. The data server can also be used for the two-way data exchange between the Asix.Evo application and the application of previous type. The data server parameterization requires the configuration file name and the operation mode. It is the same file which is used in the applications of previous type. The Architect program is used to edit the file - program operation is described in the accompanying documentation. The data server configuration file can be created with the wizard,

while developing the new Asix.Evo application. The file used in the applications of previous type may be used as well. With the  button, the Architect program may be run with the relevant configuration file loaded. In a case of the Asix.Evo Application, the Architect functioning is limited to the communication channels and archives configuration and to the variable definition database creation.

2.2 Process Variable Definition Database

In the variable definition database, the information on all the process variables used in the application is stored. This applies to both variables read from the controllers and virtual variables. General rules for the variable definition database use have not changed compared to the previous program versions.

In the Asix.Evo applications, the definition database in the MDB or XML format may be used. The MDB type databases come from the previous program versions. The Architect program is used to edit and generate this type of databases. A detailed description of the rules for the MDB use can be found in the manuals describing the classic application constructions.

The XML databases are used only in the Asix.Evo applications. They are of secondary importance. They are essentially used to define the virtual variables.

2.2.1 Creating a New Database

In the *Application Explorer* panel in the context menu of the *Variable Definitions* node, execute the *Create Variables Definition Base* command.

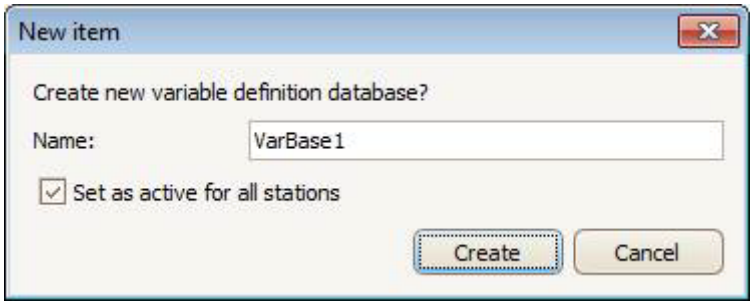

Fig. The Window for Creating a New Variable Definition Database.

Enter the database name and use the *Create* button. The new XML type database with a standard set of attributes will be created.

The MDB type databases are created using the Architect program. If the application has been created using the wizard, the MDB database has been created and added during the wizard operation.

2.2.2 Importing the Previously Created Database

In the *Application Explorer* panel in the context menu of the *Variable Definitions* node, execute the *Import Variable Definition Database* command.

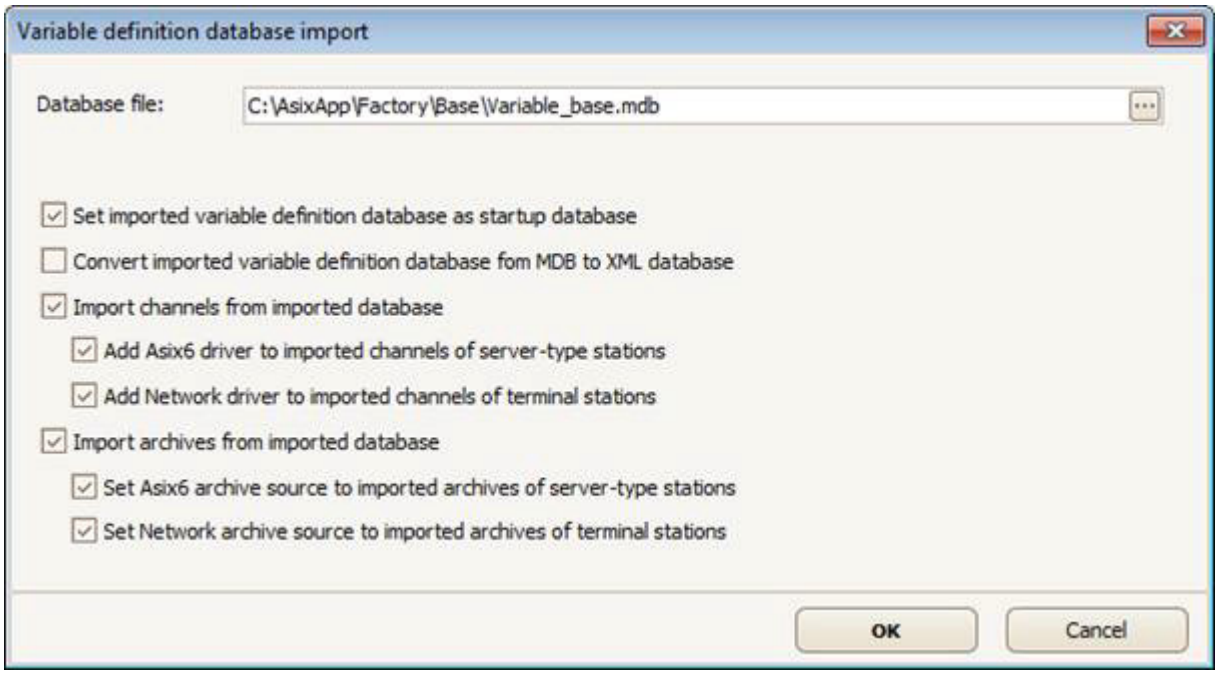

Fig. The Window for Importing the Variable Definition Database.

Enter the existing database name and use the *OK* button.

The databases in the MDB and XML formats can be imported. When the variable definition database file is imported, it is copied to the *VariableSources* subdirectory of the application definition directory.

In the import window, various options controlling the import process are available. They apply mainly to the automatic detection and configuration methods of the channels and archives used in the variable definitions. The default settings are usually optimal. With these, the manual definition process for the channels and archives can be avoided. The channel and archive types are described in the subsequent sections.

2.2.3 Editing and Browsing the Variable Definition Database

In the *Application Explorer* panel, double-click on the selected database in the *Variable Definitions* node. The below operating panel of the variable definition database will open. **In the case of the MDB type databases, only browsing the database contents is possible. For the XML databases, the variable editing functions are available.**

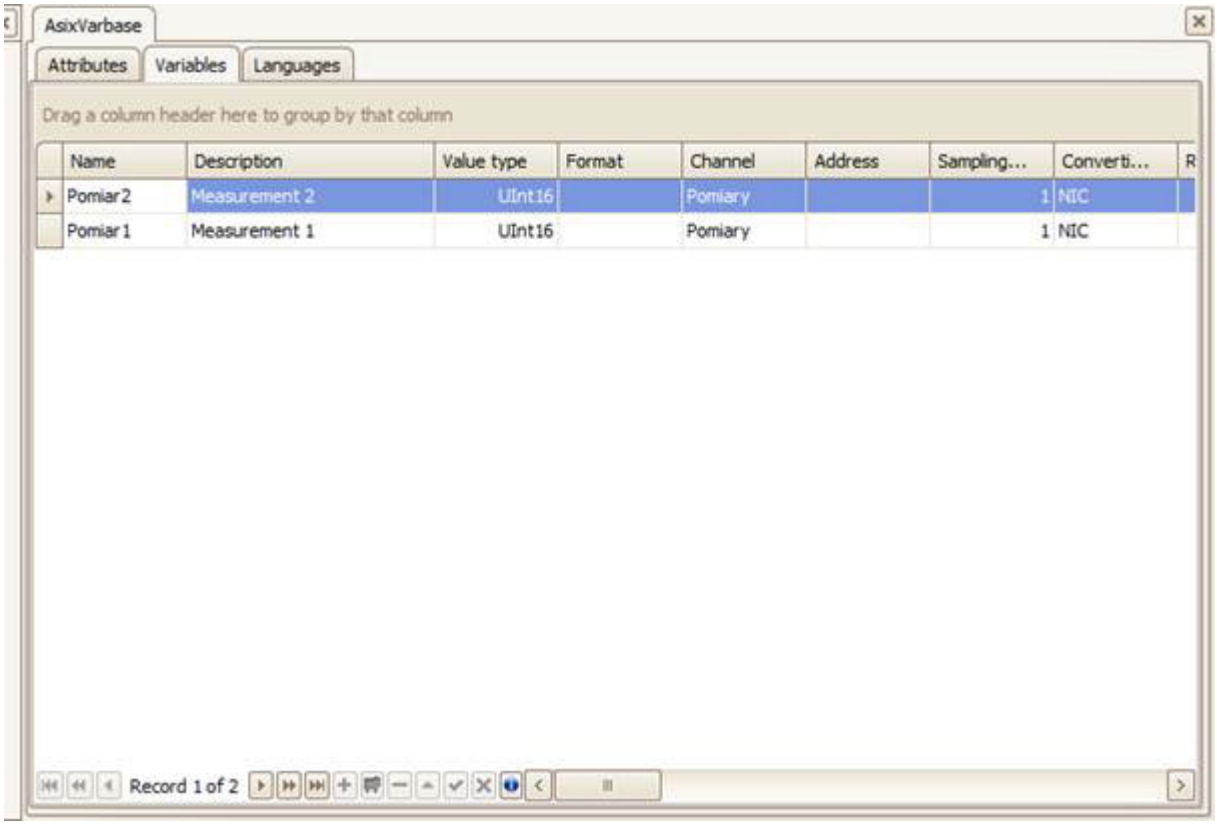

Fig. The Variable Set Panel for the Variable Definition Database.

In the *Variables* tab, editing of the variable attributes is performed. Adding or removing a variable is performed with the integrated toolbar buttons or with the context menu commands.

The XML databases are created with a standard set of variable attributes. This set can be changed in the *Attributes* tab.

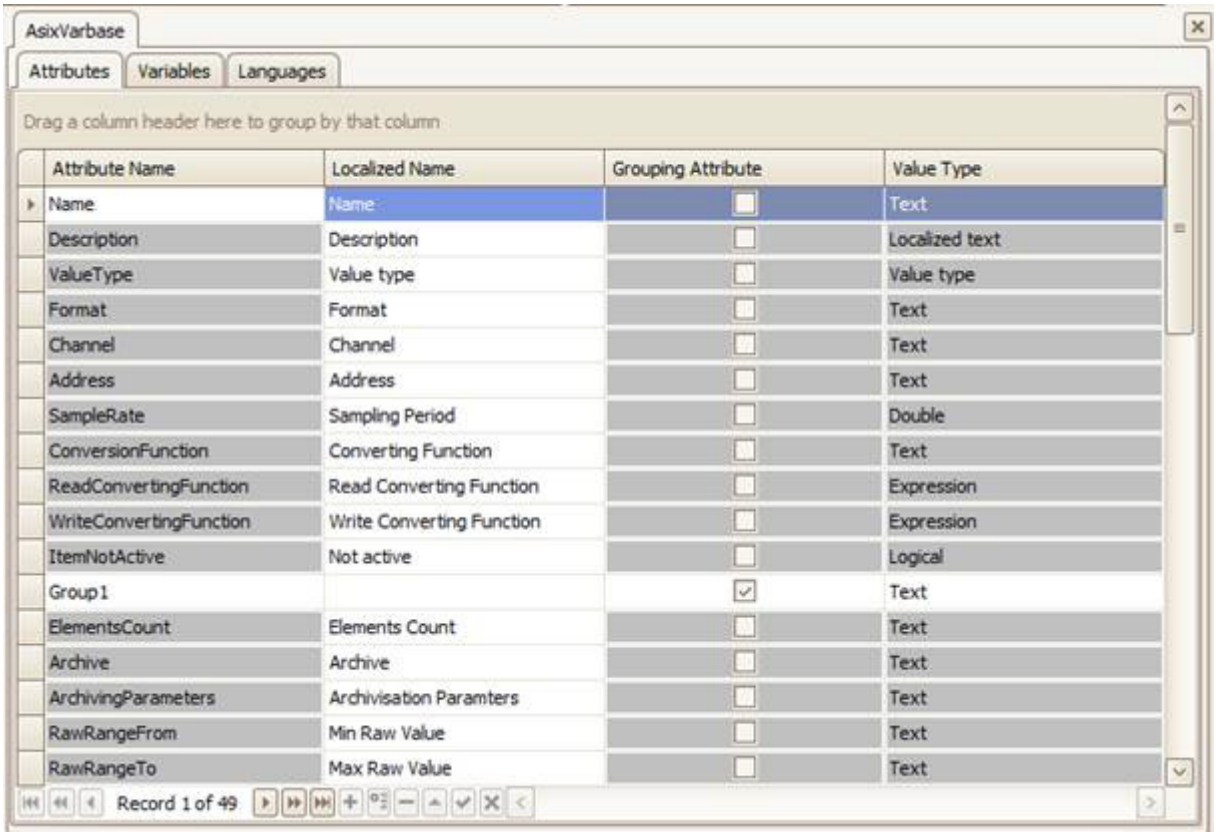

Fig. The Panel of Variable Attributes.

The panel displays the descriptions of all the attributes currently used in the variable definition database. Each attribute is defined by:

a. *Attribute Name* - attribute name independent of the language used during the parameterization of diagrams in the *Attribute* function and in the @ abbreviated notation.

b. *Localized Name* - multilingual descriptive name used in the variable selection window and in the variable definition panel.

c. *Grouping Attribute* - the option marking the attribute as a grouping attribute, i.e. used to crate a hierarchy tree in the variable selection window.

d. *Value Type* - specifying the attribute type, allows selecting the appropriate field editor.

The custom attribute of any significance can be added to the existing database using the toolbar  button. Typically, it is used to add grouping attributes. Predefined attributes can also be added using the  button.

The *Languages* tab allows declaring in which languages the attribute and variable descriptive properties are given.

The MDB database editing and generating is performed with the Architect program. If the MDB database is modified while developing the Asix.Evo application, it can be reloaded with the *Reload Active Databases* command from the context menu in the *Application Explorer* panel.

2.2.4 Variable Attributes

Most of the attributes used in the variable definition databases of the Asix.Evo application are identical as in the case of classic applications. The list below describes only the differences in the attributes interpretation. A complete list of the variable attributes can be found in the Architect.pdf/chm documentation, in *Appendix 1*.

Name	Description
Value Type	Specifies the variable value type in the Net. Convention. If the MDB database without this attribute is used, it is virtually created based on the converting function used in the variable definition and based on the format.
Format	The value text formatting method in the .Net Convention. A brief description of the .Net format can be found in the *AsixEvo_Techniques_of_Diagram_Creation.pdf/chm* manual. In the case of MDB databases, the attribute is virtually created according to the following principles: a. If the database has the *FormatEvo* attribute, it is directly treated as the Asix.Evo *Format* attribute definition. b. If the database does not have the *FormatEvo* attribute, the *Format* attribute of the MDB database (specified in the convention of classic application formats) is translated into the corresponding .Net format when loaded.
Readout conversion function	The attribute has the form of an expression that pre-calculates the variable raw value loaded from a driver. The reference to the raw

	value is performed in the expression contents using the *RawValue* function. If the data server (Asix6 channel) is used, regardless of the attribute definition of the function converting a readout, the variable value is subject to the Asmen standard conversion, based on the *ConversionFunction* attribute. If the MDB database without the *ReadConvertingFunction* attribute is used, it is virtually created based on the format used in the variable definition.
Record conversion function	The attribute has the form of an expression that calculates the control value before sending it to the driver. The rest of the rules are the same as for the *ReadConvertingFunction* attribute.

2.2.5 Setting the Database Activity Status

Not all the variable definition databases defined in the application must be used on every workstation. In the *Application Explorer* panel, double-click on the *Stations Settings* node, and then select the *Variable Sources* tab. In the *Stations* panel select an area or workstation.

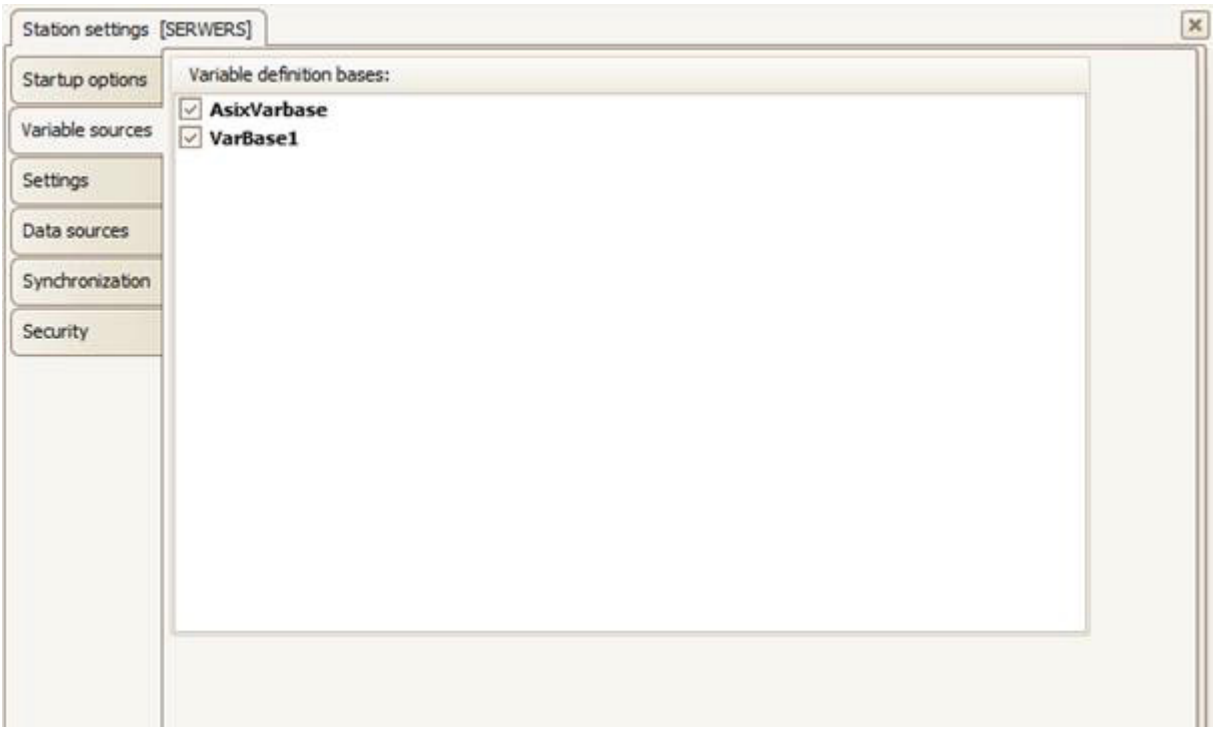

Fig. Setting the Database Activity Status in the Workstation Settings.

In the open operating panel, the variable definition databases which are to be used on a selected workstation (loaded at the start-up) may be selected.

2.3 Communication Channels

Each communication channel used in the process variable definitions should be parameterized in terms of communication drivers used by the channel. Operating panel used for parameterization of the channel is opened by double-clicking on the *Channels* node of the *Application Explorer* panel.

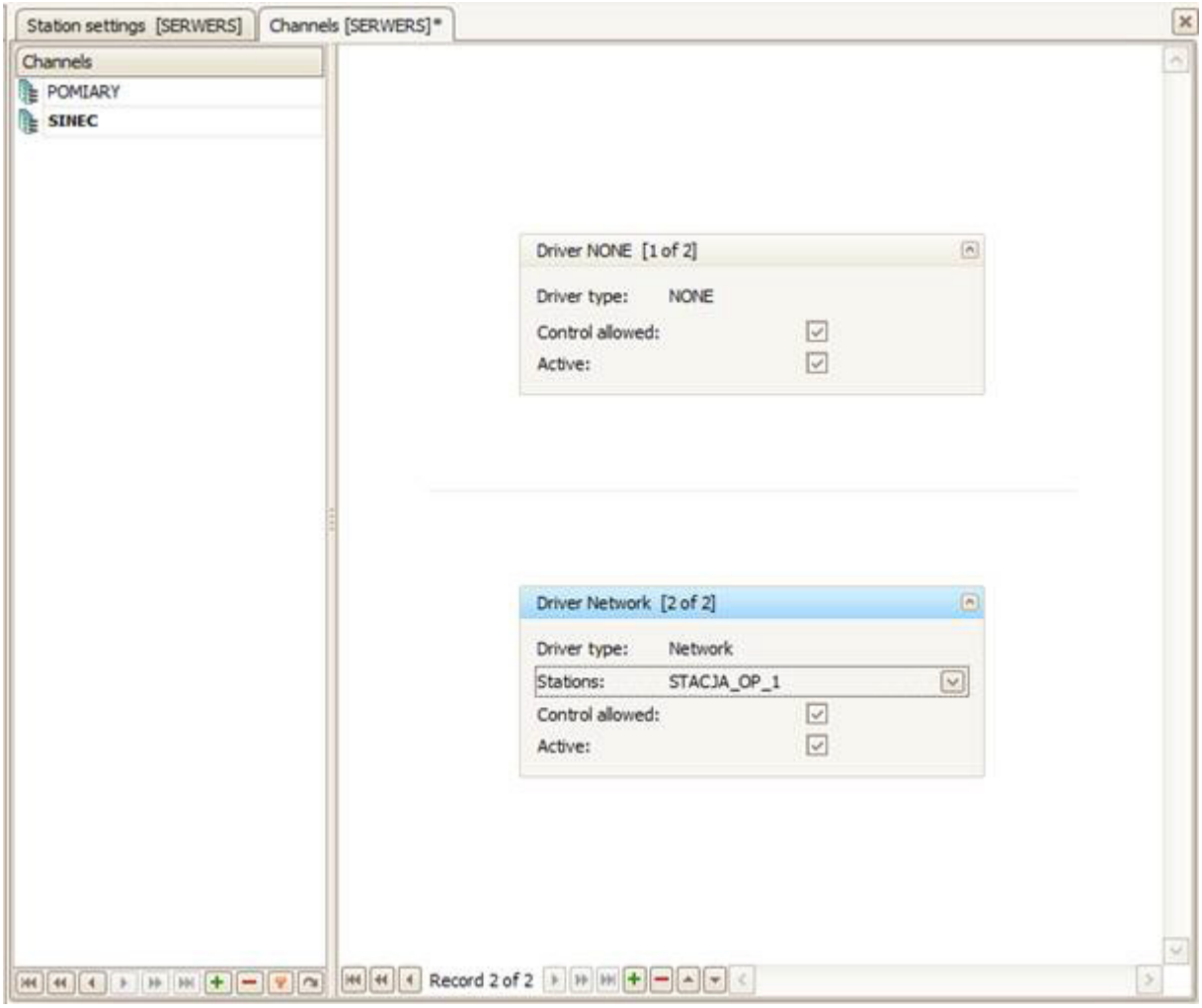

Fig. Panel for Parameterization of Communication Channels.

The channel panel shows the parameterization in the context of a particular workstation or area whose name is shown in the panel title tab. The workstation can be switched via the *Stations* panel.

The channel panel consists of two parts. In the left one, the list of all the application channels is shown. This list can be created while the application is being created using the wizard or when importing the variable definition database, based on the database contents scanning. It is also possible to, at any time, execute the *Import channels and archives* function from the context menu of the variable definition database node in the *Application Explorer* panel in order to update the channel list. Regardless of the automatic methods, the channels may be manually added, deleted, and copied using the toolbar buttons and context menu commands.

In the right side of the panel, the list of drivers along with parameters associated with the selected channel is located. One driver of each type can be added to each channel. During the application runtime, the first driver from the list which has an activity option set and was correctly activated will be used. If the active driver is not of the highest priority and during its operation the system detects that one of the drivers of the higher priority may operate, switching to that driver will occur. Adding and removing the drivers as well as reordering them can be performed with the toolbar buttons or the context menu commands.

2.3.1 Communication Drivers

Each of the drivers features standard parameters and an optional set of parameters specific to it.

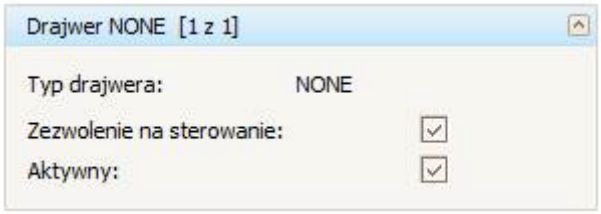

Fig. Standard Parameter Window of the Communication Driver.

The standard parameters are described in the table below.

Table: Description of the Communication Driver Standard Parameters.

Name	Description
Driver Type	Specifies the type of used communication driver.
Control Allowed	Specifies whether it is possible to perform control functions using defined driver on the selected workstation (within the selected area).
Active	Specifies whether the parameter which allows disabling the driver without removing its definition should be used.

2.3.1.1 NONE driver

The NONE driver is a virtual driver that does not perform any communication operations. It is used only to store the variables whose values are set using the scripts or the operator actions. If no active driver is defined in the channel, the NONE implicit driver will be used.

The driver does not have any non-standard parameters.

2.3.1.2 Network Driver

The Network driver is used for the network data exchange between the Asix.Evo application workstations. It is used on terminal type workstations to read the data from server type workstations. (The servers can also retrieve the data from other servers).

Table: Network Driver Non-Standard Parameters.

Name	Description
Stations	The list of server type workstations of the Asix.Evo application to which the driver can connect to obtain the data.

2.3.1.3 Asix6 Driver

The Asix6 driver should be selected for all the channels whose variables are handled via a data server, i.e. in practice, all the physical channels. The actual channel type is specified in the data server configuration file. This file is parameterized in the Architect program. Rules of its parameterization are the same as for the classic application and are described in the Architect program manual.

See also: 2.1. *Data server configuration*

The driver does not have any non-standard parameters.

2.3.2 Types of Virtual Channels

The declaration method for virtual channels of None type requires some further explanation. This can be done by three methods. The table below describes features specific to each declaration method.

Table: The None Type Virtual Channel Declaration Methods.

Declaration Method	Description
Variable without specifying the communication channel name	Variables without the channel name are automatically assigned to a default virtual channel. A variable type can be any .Net type. These variables, however, can not be stored in the Aspad long-term archive. The variable visibility range is limited to a local workstation - they can not be shared via network mechanisms.
Variables in the channels with the None type driver or without explicitly defined driver.	A variable type can be any .Net type. These variables can not be stored in the Aspad long-term archive. It is possible to share the variable values with the other workstations with the Asix.Evo application running.
The variables in the channels with the Asix6 driver and Asmen None type channel	A variable type can be any of Asmen type. The variable values can be stored in the Aspad long-term archive. It is possible to share the variable values with the other workstations with the classic type or Asix.Evo application running.

2.4 Declaration of the Process Variable Value Archives

Each archive used in the definitions of process variables should be parameterized in terms of the archive type. The operating panel used for the archive parameterization is opened by double-clicking on the *Archives* node of the *Application Explorer* panel.

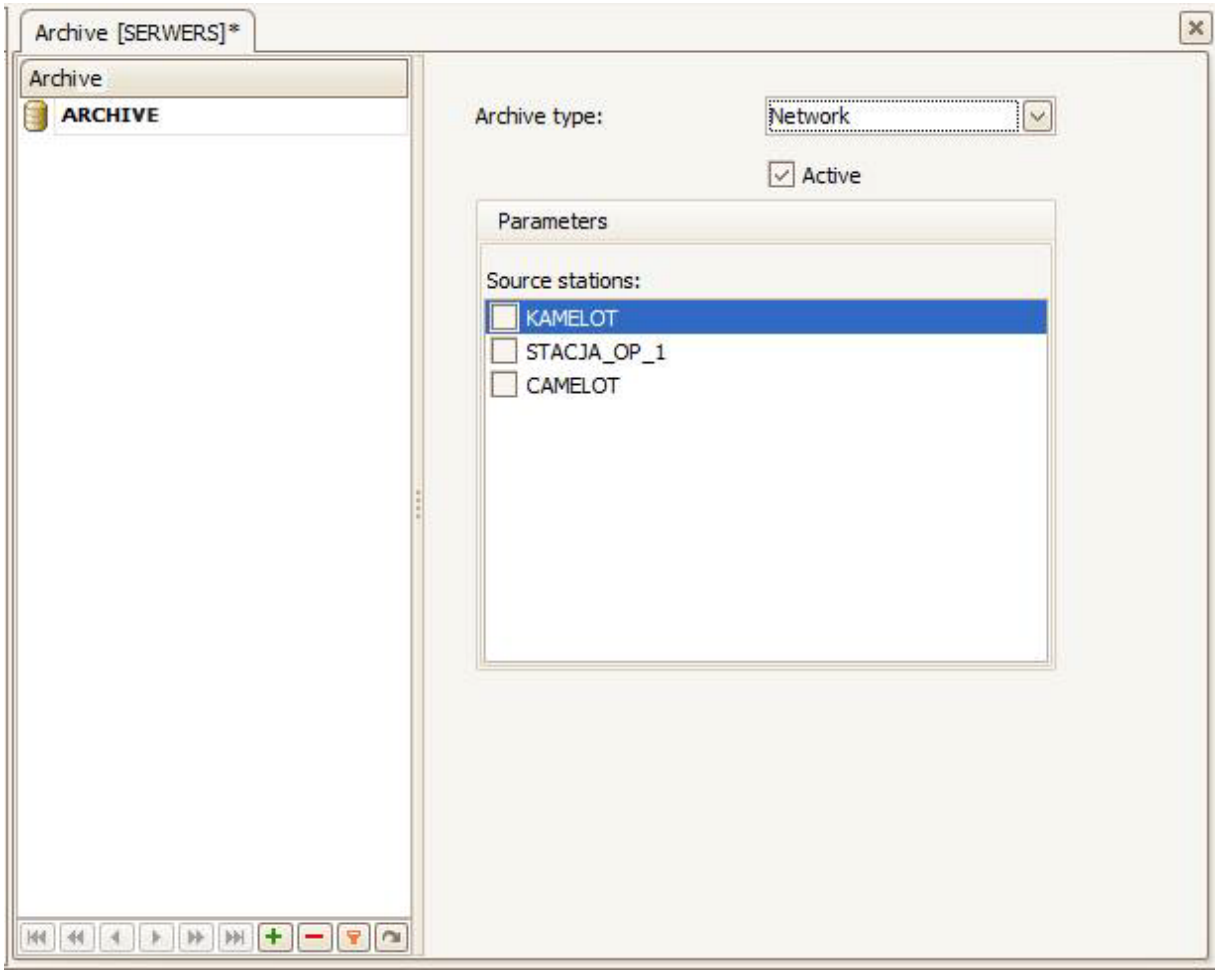

Fig. Panel for Declaration of the Process Variable Value Archives.

The archive panel shows the parameterization in the context of a particular workstation or area whose name is shown in the panel title tab. The workstation can be switched via the *Stations* panel.

The archive panel consists of two parts. In the left one, the list of all the archives used in the application is shown. This list can be created while the application is being created using the wizard or when importing the variable definition database, based on the database contents scanning. It is also possible to, at any time, execute the *Import channels and archives* functions from the context menu of the variable definition database node in the *Application Explorer* panel in order to update the archive list. Regardless of the automatic methods, the archives may be manually added, deleted, and copied using the toolbar buttons and context menu commands.

The right part of the panel is used to select the archive type and to define its operating parameters. Please note that in addition to the archive parameters, each variable has individual archiving parameters specified in the *ArchivingParameters* attribute.

2.4.1 Archives Types

2.4.1.1 Memory Archive

An auxiliary archive in which the variable values are stored only in the operating memory, without saving it into the disk file. The archive stores data only from the moment the application is started.

The archive does not feature any configuration parameters.

The archive horizon defined as a number of minutes should be defined in the *ArchivingParameters* attribute of the variable.

2.4.1.2 Network Archive

Network Archive is used to read the archive data from the other workstations of the Asix.Evo application. It is used on terminal type workstations to read the data from server type workstations.

The server workstations from which the archive data are to be retrieved should be specified in the archive parameters.

The variable *ArchivingParameters* attribute contents is of no significance for the network archive.

2.4.1.3 Asix6 Archive

The Asix6 archive is designated for long-term variable value archiving in a file or database archives. The Asix6 archives are handled through the data server. Detailed parameters of the archive operation mode are defined in the data server configuration file. This file is parameterised in the Architect program. Rules of its parameterization are the same as for the classic application and are described in the Architect program manual.

See also: *2.1. Data Server Configuration*

The archive does not feature any configuration parameters.

The variable *ArchivingParameters* attribute is interpreted identically as for the classic applications - it should be defined using the Architect program.

2.4.1.4 Random Data Archive

An auxiliary archive which creates a simulated variable value trends. Designated to demonstrate or test the application operation, without creating the actual value archive.

The archive does not feature any configuration parameters.

Three integers separated by spaces constitute the *ArchivingParameters* variable attribute contents. The first one specifies the time intervals between return samples [s], the second one specifies the low limit of the value, and the third one the high limit of the value.

3 Scheduler Configuration

The Scheduler mechanism allows for automatic execution of the operator actions. The actions can be executed in a specified time cycle, in response to the occurrence of freely defined conditions or when certain system events take place.

The scheduler tasks are defined in the operating panel that is opened via the *Scheduler* node in the *Application Explorer* panel.

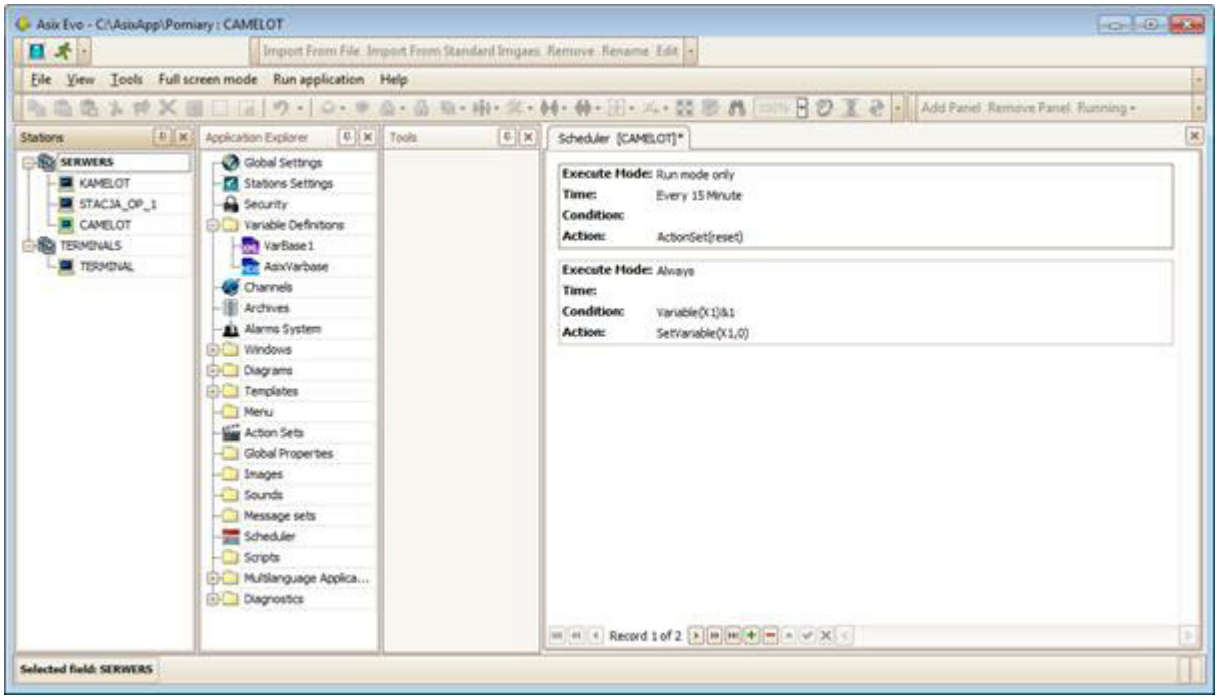

Fig. Scheduler Panel.

The scheduler tasks belong to the settings linked to a workstation. Each workstation can have its own set of tasks. If no task is defined on a workstation, it inherits the tasks from the parent area.

A new task is added to the scheduler with the toolbar '**+**' button or with the context menu command.

The three following properties are responsible for selection of the execution time for the operator action defined in the task: *Execute Mode*, *Time* and *Condition*. The values of the *Execute Mode* property specified in the table below are available.

Table: The Values of the Operator Action Execution Mode Defined in the Task.

Value	**Meaning**
Always	There are no restrictions, the execution time is determined by the *Time* and *Condition* property settings.
Run mode only	The action is only executed when the application is in the full run mode, and the execution time is determined by the *Time* and *Condition* properties.
Always except architect mode	The action is only executed when the application is in the full run mode or simplified edit mode, and the execution time is determined by the *Time* and *Condition* properties.
Once at application start	The action is executed once, at the time of the application loading, regardless of the start-up mode. This mode is often used to start the so-called resident scripts of the application.
At application start in run mode	The action is executed once, at the start of the application in the run mode (with the *run* parameter). This mode is often used to start the so-called resident scripts of the application.
At current user change	The action is executed whenever the logged user is changed.

The *Time* property is used to determine the operator action execution cycle.

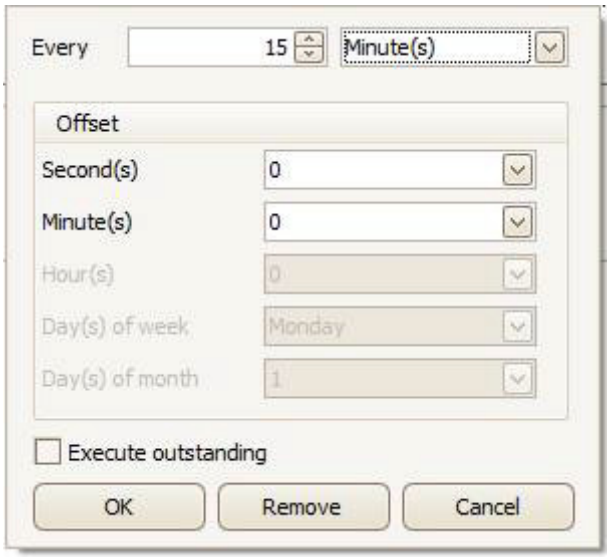

Fig. The Operator Action Execution Cycle Declaration.

In addition to the basic execution cycle, expressed in any time unit, a start point offset is also declared. In the example presented, the action will be executed every hour, at 5 minutes past each full hour (0:05, 1:05, etc.). The property value will be saved as: *Every 1 Hour Offset: M(5)*.

Examples of use:

Every 15 Minute - every 15 minutes (every 0, 15, 30 and 45 minute of an hour)

Every 1 Hour - every hour, exactly at full hour

Every 8 Hour Offset: H(6) - every 8 hours, at 6:00, 14:00 and 22:00

Every 1 Week Offset: H(8)D(Monday) - every week, on Monday at 8:00

In the time definition window, the *Execute outstanding* command may be selected as well. In consequence, at the time of the application start it is verified if the previously planned execution was performed. If not, the single outstanding execution is performed, out of normal sequence.

The *Condition* property allows defining a task the execution of which depends on meeting the specific condition. The property is in the standard Asix.Evo expression form and should return the boolean *true / false* value (or eventually *not equal to zero/equal to zero*). The interdependence of the *Time* and *Condition* properties is explained in the table below.

Table: Interdependence of the Time and Condition Properties.

Definition method	**Meaning**
Only the *Time* property is defined	An action executed in accordance with the specified time cycle
Only the *Condition* property is defined	An action performed when the condition value is changed from the *false (equal to zero)* to *true (non-zero)* value
Defining the *Time and Condition* properties	The action is executed in accordance with the specified time cycle, provided that at the time of the planned execution the condition expression is of *true* (non-zero) value

3.1 Other Methods of the Operator Action Automatic Execution

In addition to the scheduler mechanism, the operator action automatic execution is also possible through:

- executing the operator action with the *ExecuteAction* method in the application script code.
- the operator actions of the *Diagram Opened* and *Closed Diagram* events, declared in the diagrams properties - the actions executed at the diagram opening / closing.

4 Parameterization of Multilingual Applications

Asix.Evo allows for a simple creation of multilingual applications, i.e. applications which during their runtime, at the request of a user, switch the language of all the displayed text information. Since the texts are saved in the Unicode standard, it is possible to use all the languages supporting that standard.

4.1 Application Languages Declaration

The first operation to be executed while creating the multilingual application is to determine the languages that will be used. This can be performed when the application is being created, using the wizard. Alternatively, it is possible to use the *Language settings* operating panel which is opened via the *Global Settings* node of the *Application Explorer* panel.

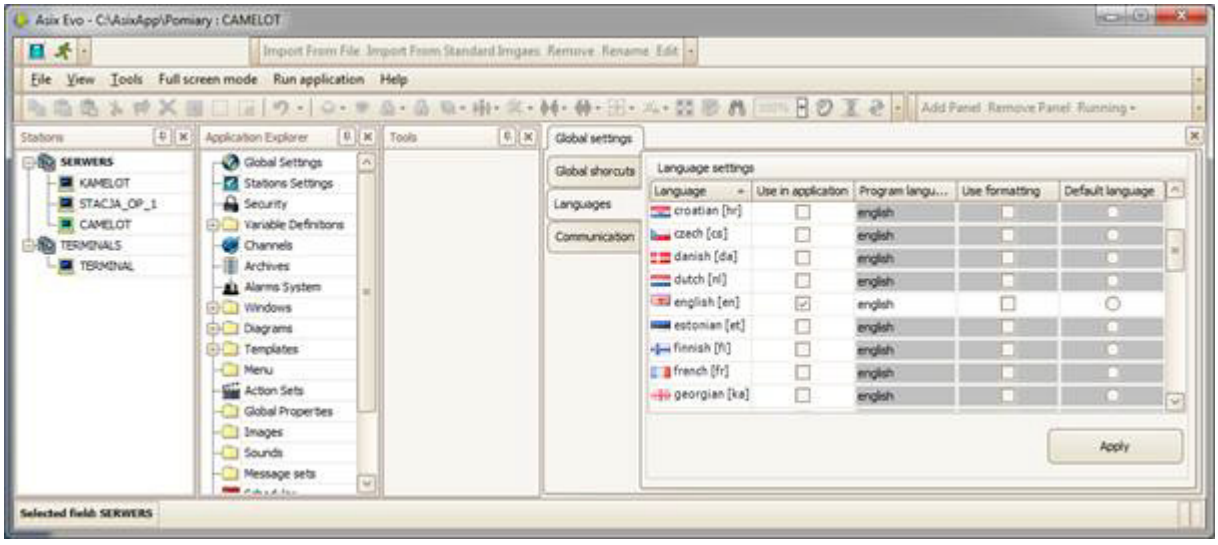

Fig. The Application Languages Declaration Panel.

In the language setting panel, the check box in the *Use in application* column should be checked - for each language which will be used in the application. In addition, the program language should be specified for the languages used. The program language specifies the fixed texts embedded in the program code, e.g. descriptions of fields in dialogue boxes, text of messages and commands in the menu. Generally, the two predefined program languages may be used: Polish and English. The designer, however, can define the program texts in other languages as well.

The *Use formatting* column determines whether data formatting according to the local settings should be used when switching the application to the specific language. This applies mainly to

displaying of dates. In the *Default language* column it should be determined which language of the application will be used when it is started.

4.2 Switching the Operating Language

When editing the application, the current language may be switched using the *Select Language* ... command from the *Tools* menu.

Fig. Program and Application Language Switching Window.

Using the displayed window, both the application and program language can be freely set.

In the application run mode, the language can be changed using the control panel window. A custom language switching system integrated with the application diagram can be developed. To do this, the *SwitchLanguage* operator action should be used. The action requires specifying a parameter in the form of a two-letter language code. The language codes are displayed in the language settings panel in the *Language* column.

4.3 Program Texts

As mentioned previously, Asix.Evo supports two working environment languages: Polish and English. However, other languages can be used, but it requires an explicit determination of the program texts in these languages. To this end, the *Program texts* work panel opened through the *Multilanguage Applications / Application Texts* node of the *Application Explorer* panel should be used.

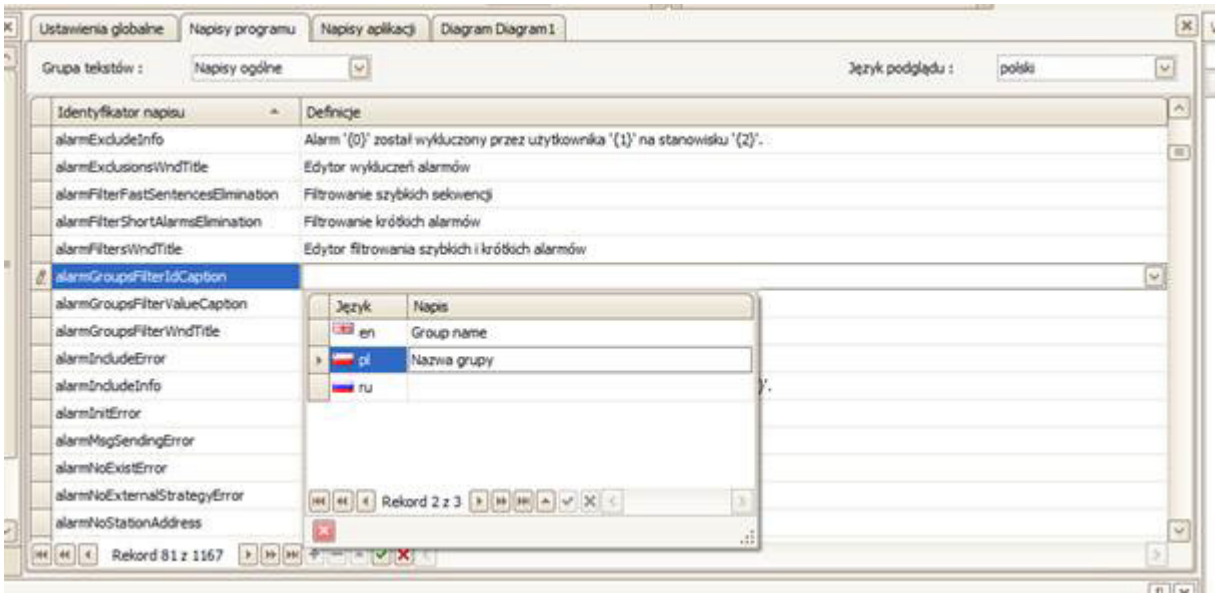

Fig. Operating Panel for Declaration of Program Texts.

The panel displays all the texts of the program. In the *Text Id* column, the fixed text ID is specified, and in the *Definitions* column, a specific text contents in the language selected in the *Preview language* box is given. After entering the text edit mode, a window for defining text in all the program languages used is displayed.

Providing translations for all the program texts is not required. If the text is not translated, and it needs to be used, the default form will be used: Polish or English (depending on the system settings or selected settings in the *SelectLanguage* program of the Asix package).

4.4 Application Texts

The application texts are the texts of any types, used in the components of the application being developed. These types include: text displayed on diagrams, alarm definitions, process variable attributes, application menu commands. Each of these components is defined in a manner specific to it, but in general, texts are entered in the multilingual text window which was already shown in the

example of the program text definition. In the case of variable and alarm definition database creation based on the Excel spreadsheets, texts are inserted in appropriately named separate columns of the sheets. The details for sheet handling are enclosed in the alarm system manual and in the Architect program user's manual.

In any case, in the application runtime, the selection of a text version compatible with the current application working environment language is entirely automatic.

4.4.1 General Purpose Texts

When displaying multilanguage texts on a diagram or creating a multilanguage application menu is necessary, the application text mechanism should be used. Its operation is similar to the program text mechanism described earlier.

The *Application Texts* operating panel is opened through the *Multilanguage Applications / Application Texts* node of the *Application Explorer* panel.

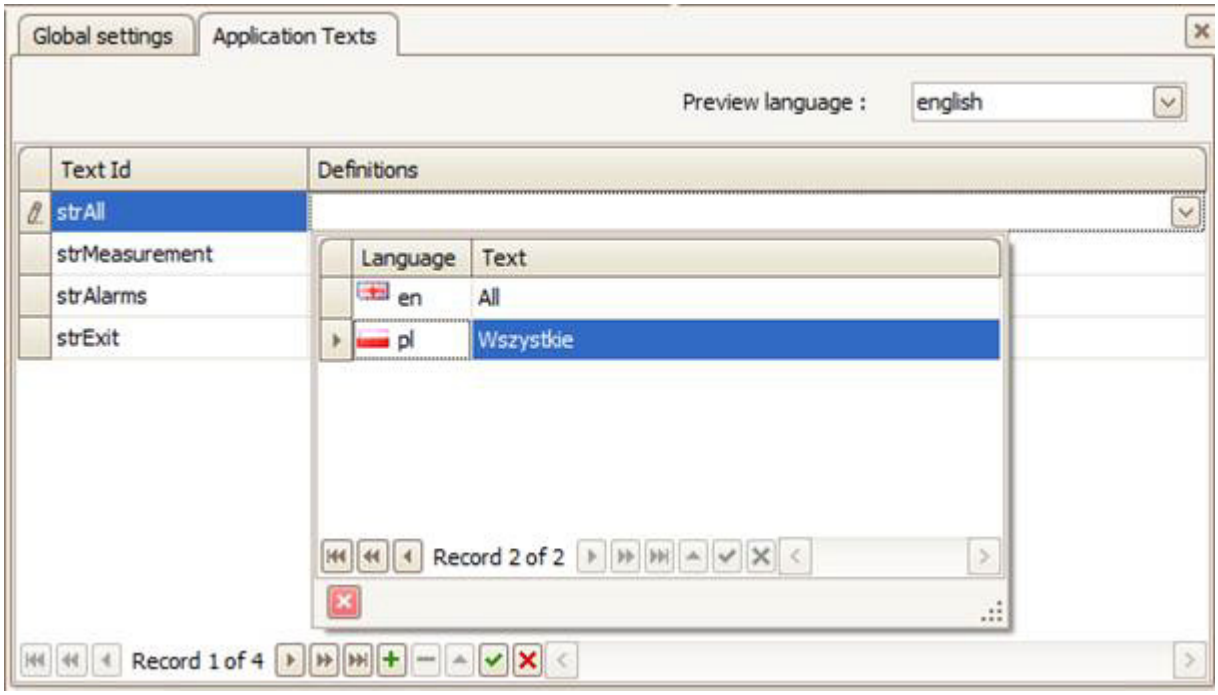

Fig. The Operating Panel for Declaration of the Application Texts.

For application text in the *Text Id* column, any unique, language-independent text ID should be specified. All the text language versions should be entered in the *Definitions* column using the standard multilingual text window.

In order to use the application text in its location of use (e.g. in the *Text* property of the menu item or in the *Off text* property of the *Button* object) the reference to the text should be made using the

expression with the *Text* function, e.g. = *Text (AllOff)*. The parameter of the *Text* function is the ID of the application text created.

In the script it is possible to reference to the application text, using the *ApplicationText* method of the *IApplication* interface.

5 Security System

The Asix.Evo authorization system is based on the user role system. Each application user is assigned to one or several roles. The specific user privileges results from the fact of the assignment to a particular role. In other words, the user can execute a protected operation, if at least in one of its role, this operation is permitted. The action scheme is possible in which the mere fact of being assigned to the role, allows for execution of certain operations.

The operating panel for permissions parameterization is opened through the *Security* node of the *Application Explorer* panel.

5.1 Defining Users

The list of currently defined users is displayed on the *Users* tab of the security panel.

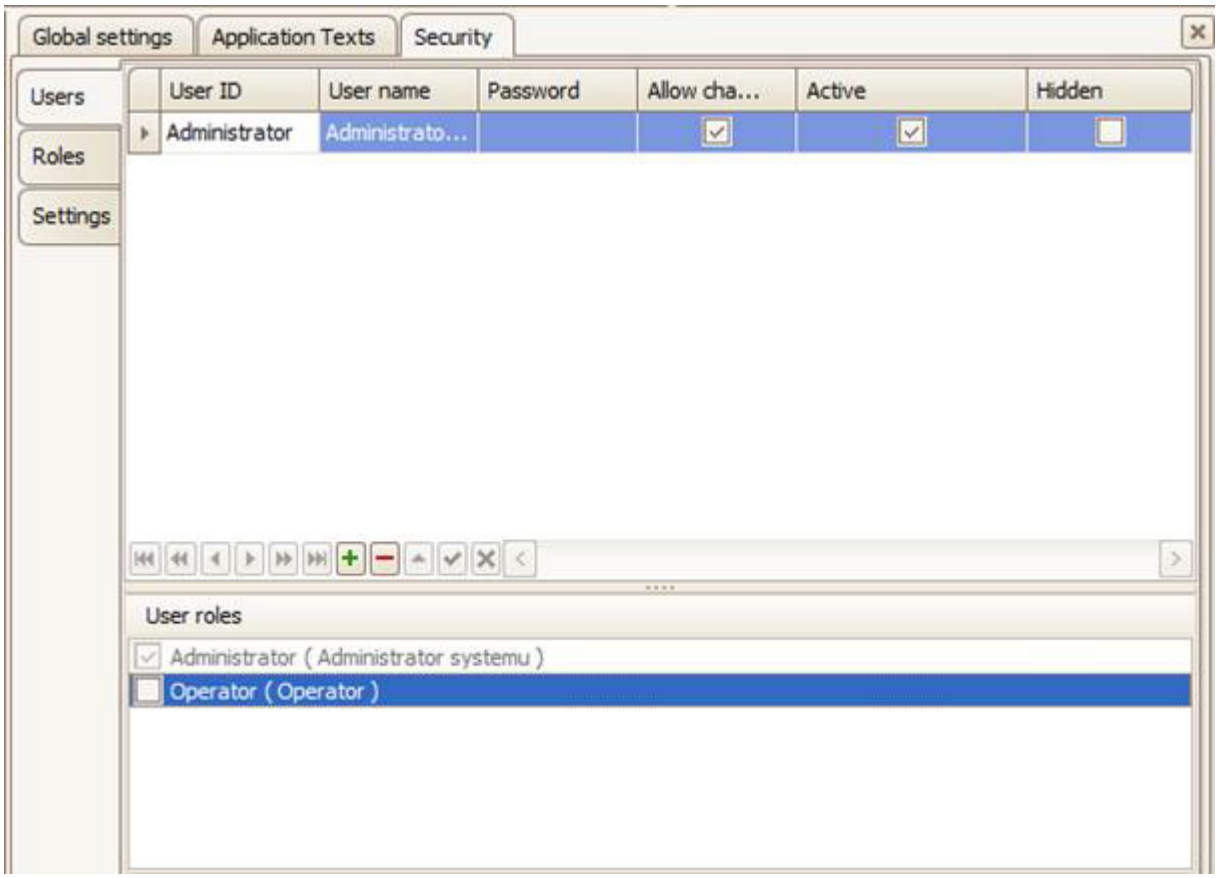

Fig. The Defining Panel of Application Users.

When the application is created, a user of the *Administrator* identifier which is assigned to the *Administrator* role is created as well. The user has full privileges and cannot be deleted. With the toolbar '*+*' button, additional users can be added.

The user is provided with a text identifier which is used in the login process, and with the name performing just an informative function. The login and password should be also defined as well as it should be selected whether the user will have the rights to change the password independently. The *Administrator* user, initially has a blank password. In the *Active* column it is possible to disable the user login privilege without deleting it completely. The check box in the *Hidden* column should be checked if the user ID is not to be shown on the suggestions list in the login window - it increases the level of security. In addition to these parameters, the roles to which the user is assigned in the application should be specified.

5.2 Defining Roles

The role management is performed in the *Roles* tab.

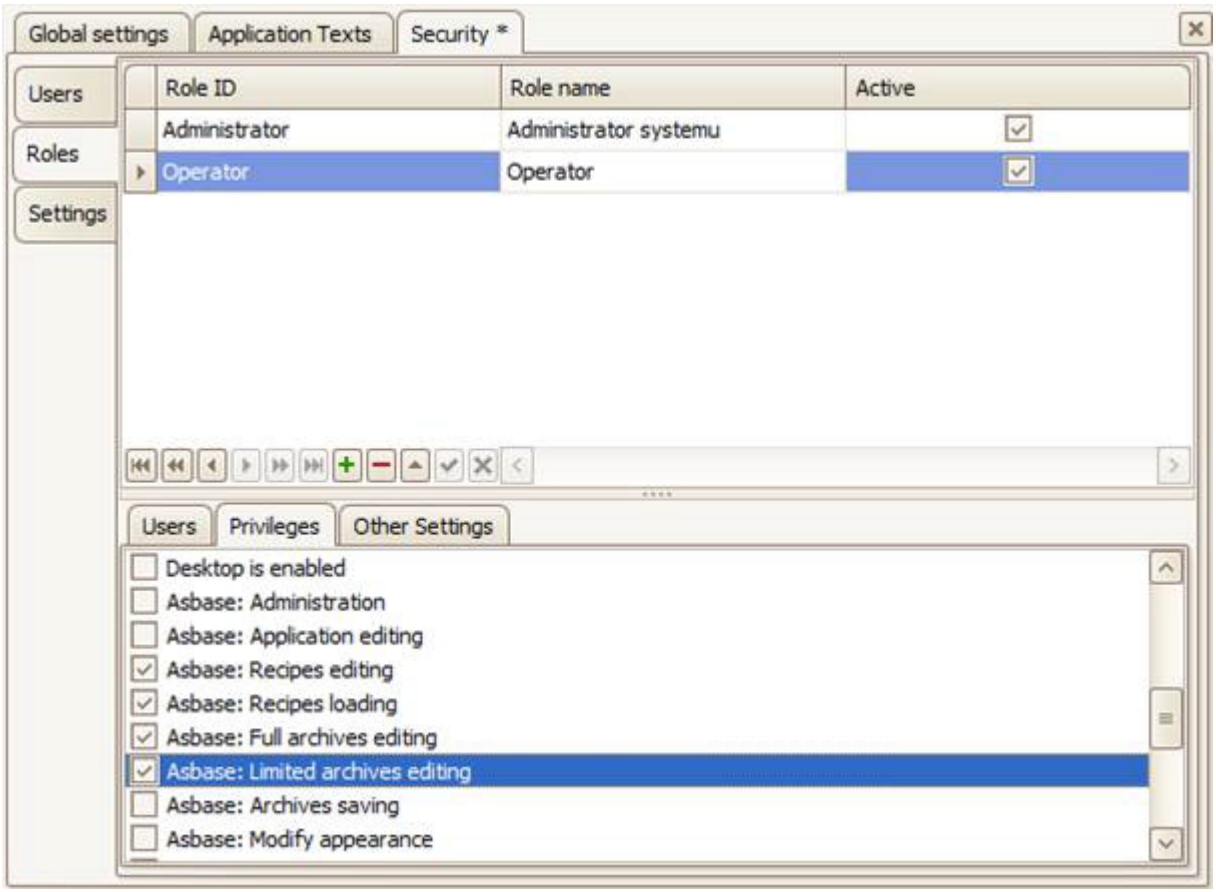

Fig. The Application Users' Role Management Panel.

By default, the *Administrator* role with all the detailed permissions is created. This role can not be changed. With the toolbar '*+*' button, additional roles can be added.

Each role has the text ID which is used e.g. in the *HasRole* function and in the *CofirmRole* action, and the name performing just an informative function. In the *Active* column, it is possible to disable the role without deleting it completely. At the bottom of the panel, the list of all the detailed permissions is located - the permissions to be assigned to the role should be checked.

5.2.1 The "Runtime Administration Right" Permission

The permission specifies whether the communication diagnostic and script windows, as well as the variable preview window in the application run mode (accessed via the control panel window) can be opened.

5.2.2 The "Control Send Right" Permission

The permission specifies whether the user can execute control commands (values recording into the process variables). In a case of control commands executed through the communication channel of the *Network* type, an additional control lock, set up at the driver level is possible - it allows locking the control operations on the selected work stations. The control operations lock in the network channel has priority over the users' permissions.

5.2.3 The "Alarms Acceptation Right" Permission

The permission specifies whether the alarms can be acknowledged. Irrespective of this permission, in the alarm system configuration, the right to execute acknowledgement on the selected workstations can be blocked - this lock is of the highest priority.

5.2.4 The "Desktop Is Enabled" Permission

The permission specifies whether the user assigned to the role has full access to all the system functions. Lack of the permission blocks functions such as the Windows taskbar, Windows buttons,

Ctrl-Esc, *Ctr-Alt-Del* etc. However, the locks are used only when the application was run with the dynamic security measures option specified in the program start-up command line.

The protections controlled dynamically by the permission are supplement by the additional protections configured with the *StaticPolicies* program included in the package.

5.3 The Security System Operating Mode

The operating mode of the security system, can be selected in the *Settings* tab of the *Security* operating panel.

5.3.1 Standard Mode

The standard mode is a simple system in which the complete parameterization of the system is stored in the *security.xml* file, located in the application definition directory. Therefore, the changes made on a single workstation require copying the file to the other application workstations. The application definition synchronization mechanism can be also used.

The standard mode should be used in single-station applications or when the permission parameters are set at the time of the application development and the subsequent changes occur occasionally.

Although the permissions parameters are stored in the XML file, its contents cannot be changed outside the control of the Asix.Evo program. The checksum is stored in the file - any unauthorized change to the contents will be detected.

5.3.2 Central Mode

In the central mode, all the information concerning the permissions are stored in the Microsoft SQL Server database. All the changes to the permissions and users are immediately shown on all workstations. Additionally, the central register of all the events related with the authorization system operation is created.

The central mode operation is similar to the AsAudit module operation, in previous versions of the Asix applications.

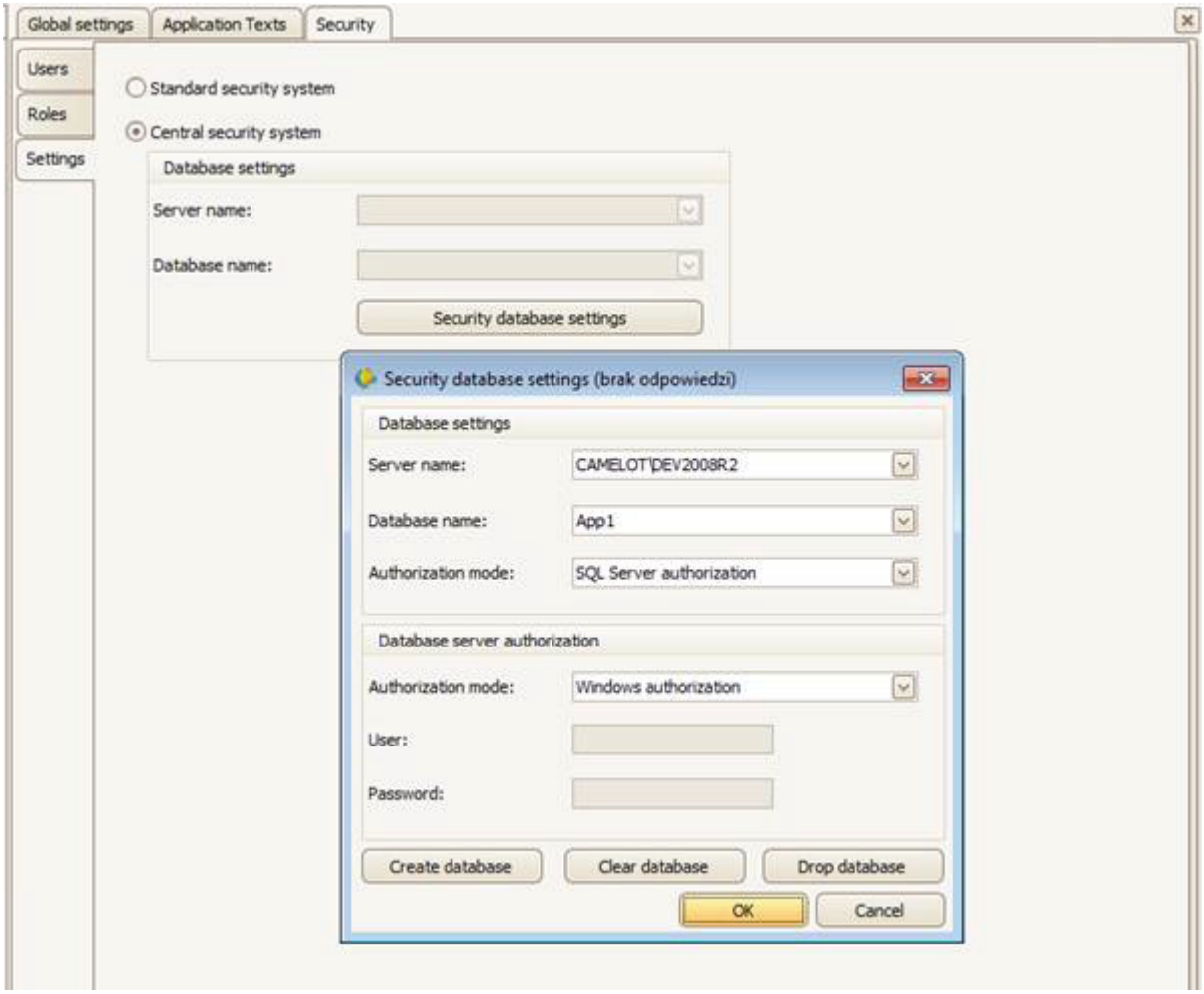

Fig. The Parameterization Panel for the Security System Central Mode.

When configuring the central mode, the SQL Server name, authorization database name and authorization mode used should be specified. Since creating the new database requires administrative privileges to the SQL server, the login method to the SQL server with the administrative privileges should be specified in the *Database server authorization* frame.

5.4 Settings Specific for a Workstation

Most of the security system parameters are defined globally for the entire application. However, there are several specific settings for individual workstations. To alter them, an appropriate workstation or area should be selected in the *Stations* panel, the *Stations Settings* operating panel should be opened via *Application Explorer* panel and then the *Security* tab selected.

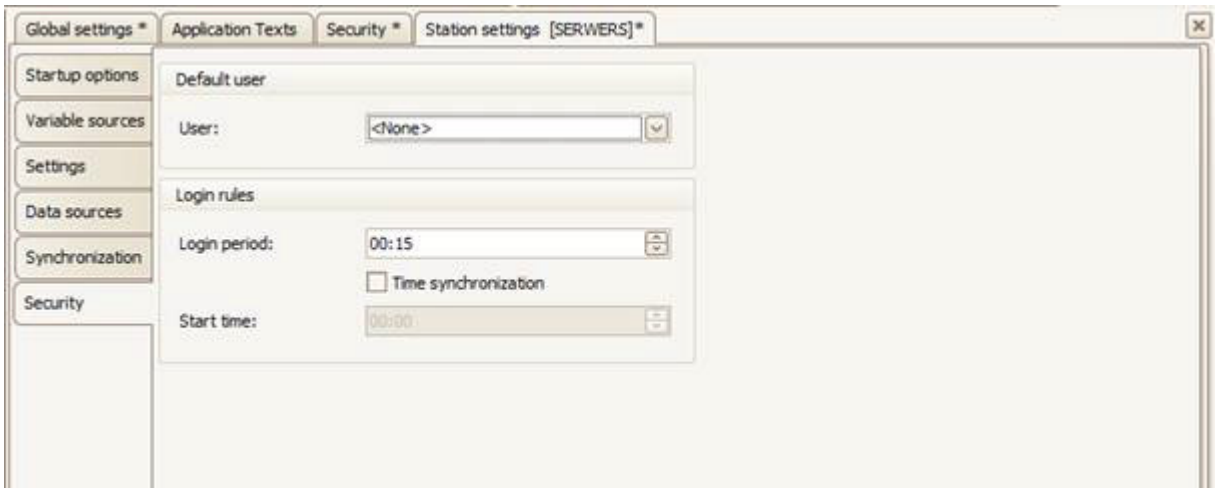

Fig. The Panel for Defining Individual Security Settings for an Individual Workstation.

The Asix.Evo application may operate with no user logged in - all the unprotected functions are available, whereas the functions controlled by the security system are locked. If we want the permission level of general availability to be higher, the so-called default user should be defined. The role with a basic permission set should be defined and then the user who performs this role. This user is specified as a default user. In consequence, after starting the application the user will be automatically logged in - unless it had no password defined. In addition, if the other user is logged out, a return to the default user will occur.

It is also possible to specify the login expiration time - after the specified time since the moment a user have logged in elapses, it will be automatically logged out. The period of 00:00 means logging without time limit.

The login start time which allows defining logging scheme forced at a specific points in time may be specified. For example, the logging time is at 08:00 a.m. and the start time is at 6:00 a.m., it means that the user will be logged out at 6:00 a.m., 2:00 p.m. and 10:00 p.m..

5.5 Using the Security System in the Application

5.5.1 User Login

Generally, users can login to the system via the control panel window. However, the application designer can directly add login mechanism to the synoptic diagrams. The first of these mechanisms is the *Authorization box* class object, which ensures a full support for the login, directly on the diagram. The alternative is to use the *Login* operator action, which displays the user login window. It is supplemented with the *Logout* action used to log off the user and the *ChangePassword* action which opens the window used for password change.

The information about the current authorization system status can be displayed on the diagram. The following predefined process variables provide this functionality:

- *CurrentUser* - ID of the currently logged user.
- *UserRoles* - identifiers of all the roles performed by the currently logged user.
- *UserRemainingTime* - time remaining until an automatic logout of the user.

5.5.2 Verification of Permissions

In the scope of standard permission control, the program operation is fully automatic. However, it is possible to extend the application with custom functions. This is provided with the *ConfirmRole* operator action and the *HasRole* function.

The *ConfirmRole* action allows for additional authorization of the user outside the standard logging system. It can be used in the action scheme, in which an execution of any operation requires authorization, or when an additional confirmation required from the user who is not logged currently, is necessary.

The *HasRole* function allows linking the object appearance or functioning on a diagram, depending on whether the user is assigned to the role specified in the call.

Examples of use are included in the *"Techniques of Diagram Creation"* (Asix.Evo_*Techniques_of_Diagram_Creation*.PDF/CHM) user manual, under the *Controlling Permissions* section.

5.5.3 Changes in the Permission Parameters in the Application Run Mode

Typically, the authorization system is configured when the application is developed, during its operation in the edit mode. However, the full permission management (defining users and roles, changing permissions and passwords) in the run mode is possible as well. This is enabled by the *SecurityManager* action which opens the window below.

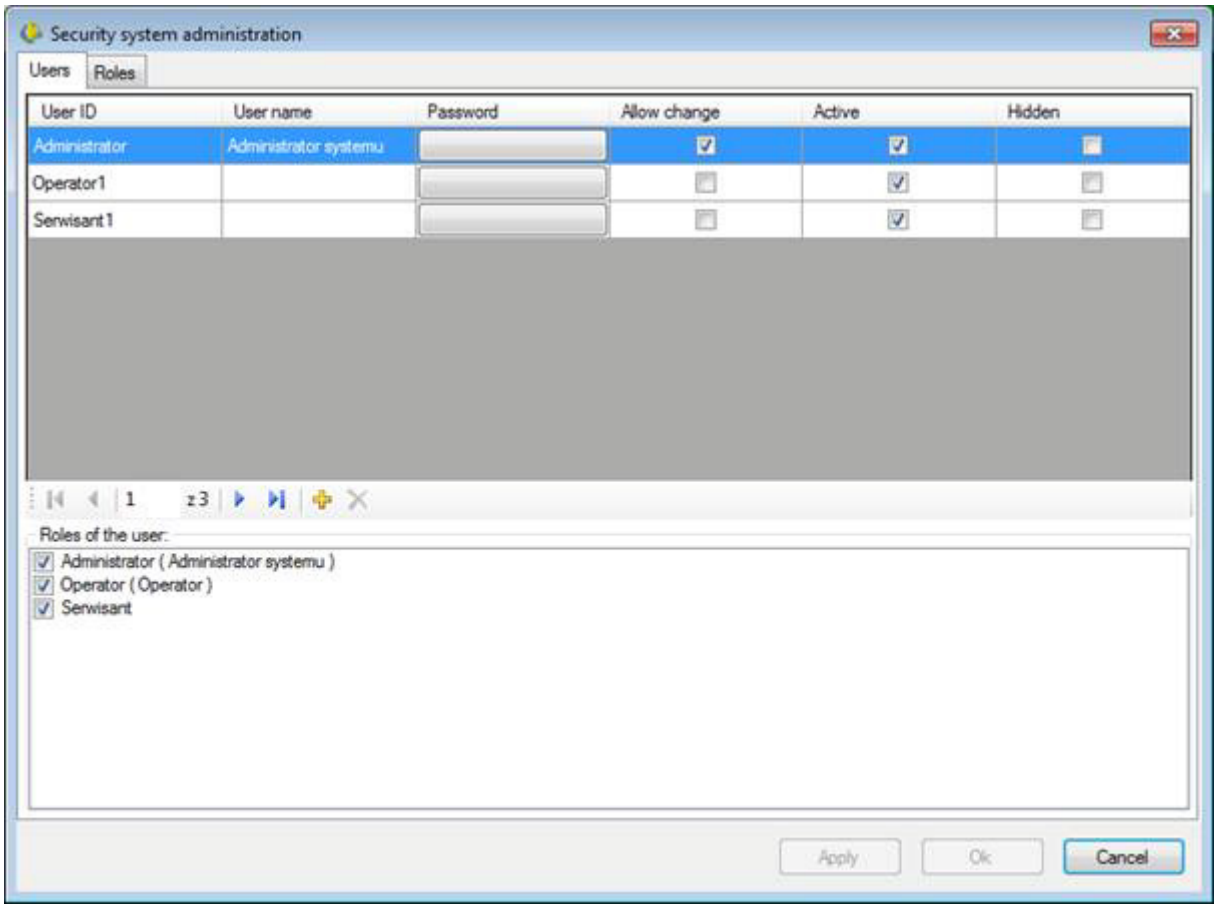

Fig. The Permission Parameterization Window in the Application Run Mode

Please note that the changes made on one workstation are immediately visible on the other workstations, only if the authorization system operates in the central mode. In the standard mode, the changes are visible only when the authorization file will be copied and synchronized.

5.5.4 Viewing the Login Event Log

In the standard mode, the user login events are recorded only in the normal message Log. They only relate to the current workstation. In the central mode, a separate event log relating to the authorization system is created. Events from all the workstations are recorded in it.

The overview window with the authorization system event log can be opened with the *SecurityBrowser($ MessagesLog)* operator action.

6 Recording Control Operation

It is possible to record the control operations (the process variable recording) within the Asix.Evo application. In order to initiate the recording, the two preconditions have to be met:

- The application security system must operate in the central mode - the control operations are stored in the database used also in the security system.
- The Asix system license with the *AsAudit* extension is required.

The variables for which all the executed control operations are to be recorded can be selected in the variable definition database. The database must feature the *ControlLogging* attribute. All the variables, for which the value of this attribute is not equal to zero, are registered. The other settings are not required.

For each registered control operation, the following information are recorded: who, when and on which workstation has executed the operation. The variable value before the control operation, and the new variable value are also recorded.

The overview window with the control operation log can be opened with the *SecurityBrowser($ CpmtrolLog)* operator action.

In a case of bit-based control operations, it is possible to display the text description of the executed control operation. The current and new variable values are compared, and for the bits which have been changed the text description of the operation is created. The text descriptions are created based on the variable definition database contents. The *StateNames*, *StateSet* and *StateValue* attributes are used for this purpose. The control operation description calculation algorithm for the single bit is as follows:

- The *StateNames* attribute value for the controlled variable, is loaded.
- In the variable definition database, all the variables whose *StateSet* attribute is identical with the *StateNames* attribute value loaded in the first point are searched. Typically, these are inactive variables used only to define the state names.
- Among the variables selected in the previous point, the variable whose *StateValue* attribute value is identical with the bit number is searched.
- The *StateNames* attribute of the variable selected in the previous point should constitute a sequence in the form of: *0=state0;1=state1*. Depending on a new value of the controlled variable bit, an appropriate description is selected.

Example

Name	*Variable Inactive*	*State Names*	*State Set*	*State Value*
ENGINE_1	**0**	**ENGINE_CONTROL**		

EC_BIT0	**1**	**0=OFF;1=ON**	**ENGINE_CONTROL**	**0**
EC_BIT1	**1**	**0=FWD;1=BWD**	**ENGINE_CONTROL**	**1**
EC_BIT1	**1**	**0=SLOW;1=FAST**	**ENGINE_CONTROL**	**2**

If the *Engine1* variable is changed from the 2 to 3 value (the change of the bit of number 0 to the value of 1), then in the control operation description, the text *ON* will be used.

7 Operation with AsTrend Program

The Asix.Evo applications can use the AsTrend program to display the trend of the process variable archive data providing their advanced analysis. Interoperability of the Asix.Evo application and the AsTrend program is based on the use of *AstrendDisplay* and *AstrendPrint* operator actions. However, the preconfiguration of the AsTrend program use is required. The preconfiguration is performed independently for each workstation of the application. Through the *Stations* node of the *Application Explorer* panel, the workstation setting operating panel should be opened and then the *Data Sources* tab should be selected. For the AsTrend program configuration, the shown below panel fragment is responsible.

Fig. The AsTrend Program Configuration Parameters.

If the *Astrend...* family actions will be used, the *Start AsTrend* option should be selected. In a case of window application it also launches the AsTrend program at the start of the application - this will allow further for faster execution of the operator actions.

Generally, the list of all the MDB databases of the variable definitions, defined in the application, is transferred to the AsTrend program. If this is an unwanted operation, a different name for the variable definition database in the *Alternative variable definition database* field can be specified.

The user of the application will not be able to change the trend configuration files (trnx files), if the *Do not allow saving any files* option is selected. The user will automatically acquire all the permissions to the AsTrend program functions, if the *Always work as AsTrend Administrator* option is selected. Both options are of significance only if the application runs in the window mode.

7.1 Operation in the Browser Version of the Application

In a case of Asix.Evo application operation in the browser mode, the interoperability with AsTrend is also based on the browser version of this program. This means that the Asix package should be installed along with Internet components. This will create the AsTrend program directory on the IIS server. The installation must be performed on the same computer on which the Asix.Evo application is published.

The AsTrend program usually loads historical data directly from the process data server which is initialised within the Asix.Evo application. It is also possible to use the data available in the Asix classic application servers. If the *Data servers addresses for Web Browser* parameter is not explicitly defined, AsTrend will load the data from the computer on which the Asix.Evo application was published. However, other sources of historical data can be specified - in the *Data servers addresses for Web Browser* parameter, the system names or IP addresses (separated by commas) of all the computers that provide historical data should be entered.

8 Operation with the AsBase Program

In a case of the Asix.Evo application, the interoperability with the AsBase program is similar to the traditional applications. The first step in developing the application is a standard parameterization of the AsBase application - the registration sets and / or groups of recipes and sets of variables used for the data exchange between AsBase, the PLCs controllers and the Asix.Evo application should be defined.

If the Asix.Evo application is to interoperate with the AsBase program, an appropriate interface should be enabled. The preconfiguration is performed independently for each workstation of the application. Through the *Stations Settings* node of the *Application Explorer* panel, the operating panel of workstation settings should be opened and then the *Data Sources* tab should be selected. For the AsBase program configuration, the panel fragment shown below is responsible.

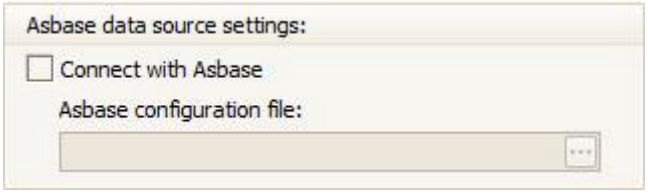

Fig. Setting Up a Connection with the AsBase Program.

The *Connect with AsBase* option enables the interoperability with the AsBase program. In addition, it launches the AsBase program at the start of the application. In the *AsBase configuration file* filed the xml start up file name of the AsBase application should be specified.

Further interoperability of Asix.Evo and AsBase is performed via the variables used in the sets of variables of AsBase and the *Asbase...* family actions. The process variables used for data exchange should be defined in the channel of *None* type of the Asmen module. This results from the fact that AsBase directly communicates with the process data server, initialized within the Asix.Evo application. To conclude, the variable data exchange channel is defined as *None* type in the data server configuration file (using the Architect program), and as *Asix6* type in the Asix.Evo application.

The operator actions used to interoperate with the AsBase program are divided into two groups: connection actions and connectionless actions. The connectionless actions perform auxilliary function. These are the AsbaseLoad, AsbaseShow and AsbasePrint actions. They can be executed at any time.

The connection mode actions allow browsing data from the AsBase database using the synoptic diagram objects. The action scheme is as follows:

- **Opening a Connection**

The connection opening is executed via the *AsbaseOpen* action. In this action, the data subset which will be displayed within the connection and the variable set which will be used to transfer the data to the w Asix.Evo application should be specified. In the action, the connection ID should be also specified which will be used by the other connection mode actions.

The *AsbaseOpen* action is generally used to handle the *Diagram Opened* event of the diagram, on which the AsBase data will be shown. In classical applications the *AsBase* class object was responsible for establishing the connection.

- **Data Visualization on a Diagram**

The data presentation is performed by using standard diagram objects.

The process variables, constituting the variable set specified in the *AsbaseOpen* action are used for this purpose. These variables assume the values compatible with the AsBase data set field values for the active record.

- **Navigating Within the Data of the Connection**

The *AsbaseAdd*, *AsbaseDelete*, *AsbaseUpdate*, *AsbaseNavigate* actions allow performing operations on the connection data. Each of these functions has a parameter specifying the connection ID. The operations performed by these actions, generally relate to the connection active record. The *AsbaseNavigate* action is of particular importance, because the action is used in general to change the active record, but it can also change the data filtering (initially set out while establishing the connection).

- **Connection Termination**

The connection is terminated with the *AsbaseClose* action. The action is generally used to handle the *Diagram Closed* event of the diagram, on which the Asbase data were shown.

8.1 Connection Filter

One of the *AsbaseOpen* and *AsbseNavigate* action parameters (in the *$Filter* option) is the filter which allows selecting the data to be used in a connection.

The filter syntax is compatible with the WHERE clause of the SQL language, with additional optional extensions which allow inserting process variable values and time constraints into this clause. The words WHERE are not specified in the filter definition. There is also a possibility to add the ORDER BY clause for data sorting to the filter syntax.

As a rule, in the filter definition the references to the fields defined in an archive set or in a recipes group are used. In order to use a registration set, recipe or load history field value as a filter element, the identifier of the field should be preceded by the prefix "V_". In the case of a registration set archive, a value status and a timestamp of this value may be associated with each field value. To use these components, the field ID of the registration set should be preceded by the prefix "S_" or "T_". In the case of recalculated recipes, the percentage value field can be called in a load record by using the prefix "VP_".

AsbaseOpen(conn1, $Archive, Paints, PaintsDgr, null, null, ”V_Akt=1”)

In the example above, the used filter retrieves data only from the *Paints* egistration set for which the field value with the *Act* identifier is equal to 1.

In addition to the references to the field values defined by the application designer, the static names of the AsBase table fields, i.e. such fields that are always present in a table can be used. Static fields are described in the table below.

Table: AsBase Table Static Fields.

Name	Meaning
LOCALTIME	Local time of a record logging or recipe loading
UTCTIME	UTC time of a record logging or recipe loading
STATUS	Recipe load operation status
SOURCE	Source of the record logged
NAME	Name of the recipe
LOADBY	Specifies the user who has loaded the recipe
TARGET	Recipe load target
CREATED BY	Recipe creator
BATCH	The batch value for loading the recipe being recalculated.

8.1.1 Inserts

In the filter contents it is possible to use inserts which allow including the current process variable values and dynamic limiting of a time range.

The insert of the filter is contained in curly brackets.

The expansion insert which allows using the process variable value has the following form:

{V:variable_name}

The variable value replaces the insert in its position.

The insert which defines the time filter elements has the following form:

{T:code [:*field_id*]}

This code defines the time range. The following codes are available:

t	-	today
y	-	yesterday
sy	-	since yesterday
tw	-	this week
lw	-	last week
slw	-	since last week
tm	-	this month
lm	-	last Month
slm	-	since last month
ty	-	this year
ly	-	last year
sly	-	since last year

The optional *Field_id* filter specifies the table field for which the time filter is to be used. If this element is omitted, it is assumed that the filter applies to the time the record was logged in the table. If the ID field is specified, the field should be of *Data* type. When this insert is used, the entire insert text will be replaced with an expression (compatible with SQL language syntax) limiting the set of table records to those the value (time) of which is within the time range specified by the code parameter.

The filter containing inserts is not modified during the connection. For example, if the time filter of "today" type is defined and the connection is open for a second day, the records from the previous day will be shown. Only the connection re-opening or re-applying of the same filter in the operator action will update the data.

The inserts can not occur inside the filter elements contained within square brackets [and] and within quotes ' and ". To contain the insert content within these delimiters, the delimiters must be placed between curly brackets.

{'}{V: variable_name }{'}

The above filter fragment will be replaced with the variable value enclosed in single quotes.

The use of inserts was necessary in classic applications. In the Asix.Evo applications, dynamically varying filters may be created using expressions.

”V_Akt={V:Z1}”

”V_Akt=”+Variable(Z1)

Both the above versions of the filter definition will function the same way. The records for which the *Act* field value is equal to the *Z1* variable value will be selected (at the time of the action execution).

8.2 Users Permissions

If the AsBase program is run within the linked Asix.Evo application, then the security settings defined in the Asix.Evo are used. In this case, the settings selected directly within the AsBase application are of no significance. Nevertheless, if the AsBase application is run independently, the settings will be used. In a typical configuration, in the AsBase application a single administrative operator is defined – it protects the AsBase database against unauthorised access. The other users have such permissions as specified in the Asix.Evo security system configuration.

8.3 AsBase in Browser-Based Applications

As in the case of classic applications, the latest Asix.Evo program version does not enable integration with the AsBase program in the browser mode. All workstations exchanging data with AsBase have to operate in the Window mode.

9 Synchronization of the Application Files

In the case of multiply workstation installations, maintaining consistent file versions of the application on all workstations becomes a significant issue. This applies mainly to the phase in which the application is still in development and the definition files are frequently modified. In the case of applications operating in the browser mode this is of less significance - a changed version of the application is published on the server once and from there it is automatically transferred to all Web terminals. In the case of window applications, the application synchronization system of the Asix.Evo comes to help.

The concept of the synchronization system operation is based on defining a network template directory which is available to all the application workstations. The changed files are loaded to the template directory and from there they are automatically transferred to all the application workstations.

The synchronization module parameters are set out in the workstation settings work panel in the *Synchronization* tab.

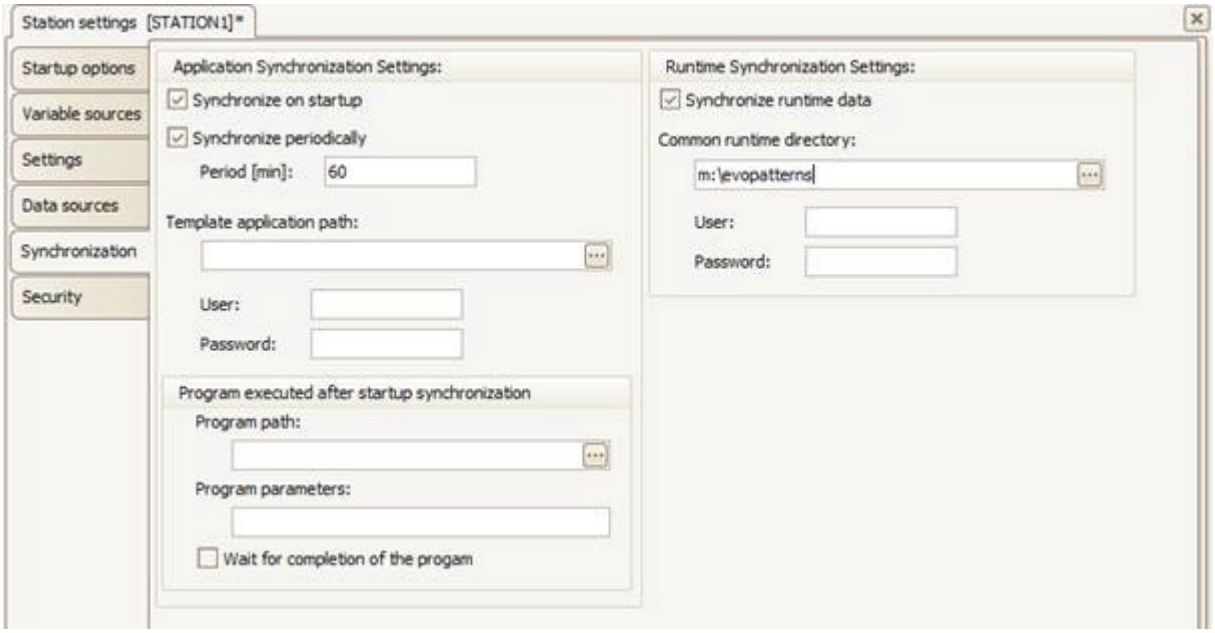

Fig. Synchronization Module Parameters.

9.1 Synchronisation of the Application Definition Directory

The application definition directory can be synchronised once at the start of the application and/or periodically at a determined period of time. In the *Template application path* field, the template

location should be specified. It is possible to reference to the directory on the locally mapped disc or use the *UNC* name. If, in order to access the template directory, a special system authorization is required, then in the *User* and *Password* fields the ID and password of the User with the required permissions should be entered.

In the case of a synchronization executed at the start-up of the application, the program which will be executed on the synchronization completion may be declared in the *Program executed after startup synchronization* box. This program is able to handle any extra tasks required by the application for correct completion of the synchronization.

Automatic synchronization involves transferring the updated files of the template directory into a local definition directory. The files which are not present in the local directory are transferred as well. Copying in the other direction - into the template directory is performed manually. The system tools or the Asix.Evo synchronization window (opened using the *Synchronization* command from the *Tools* menu) may be used.

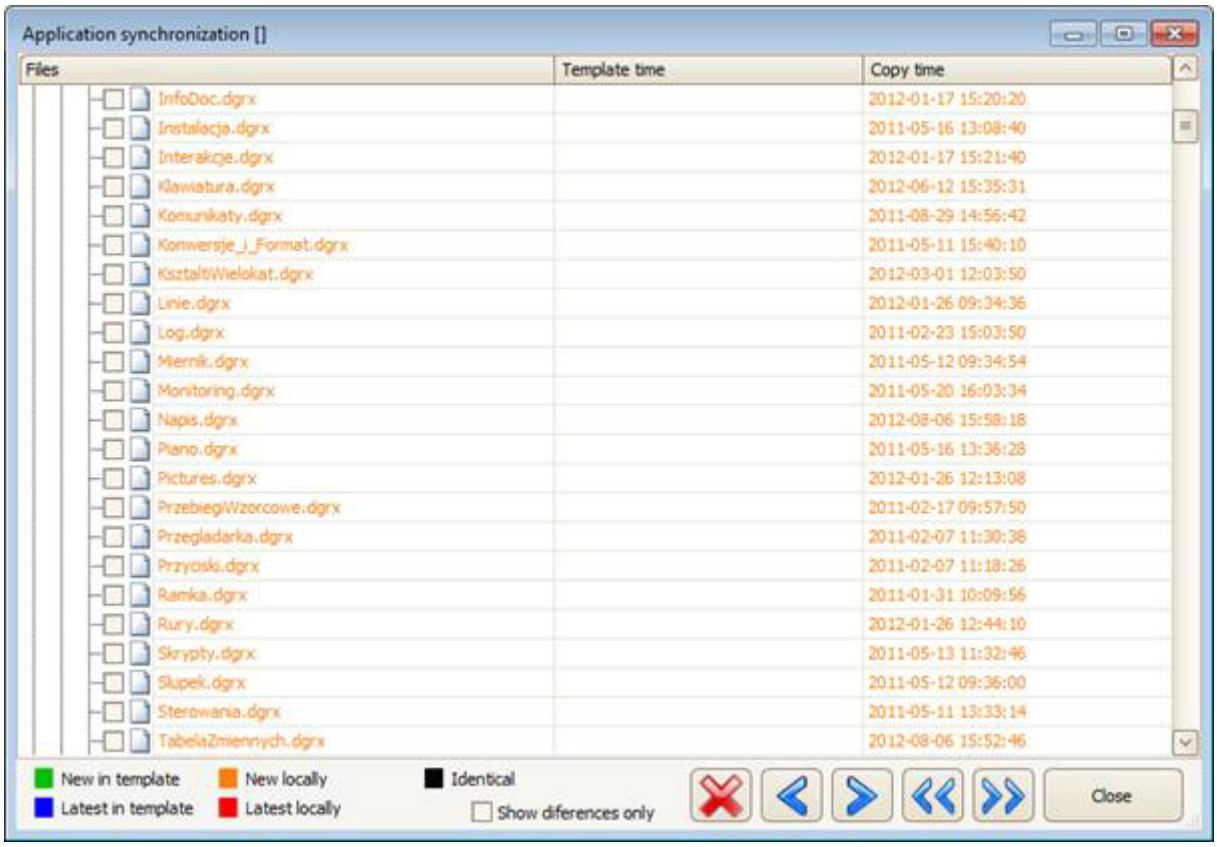

Fig. The 'Application Synchronization' Window.

In this window all the definition directory files are shown. The differences between the template and local directory are highlighted with colours. Using the window buttons it is possible to copy selected or updated files in either direction and to delete selected template files. During the synchronization operation one may write a note describing the changes implemented. Notes are saved to the *SynchronizationNotes.xml* files in the template directory.

9.2 Synchronization of Working Directory

The parameterization of the working directory synchronization requires just to specify the template directory and possibly the details of the user with access permissions to the directory.

The working directory synchronization requires just synchronization of the local trend patterns stored in the *TrendPatterns* subdirectory. The synchronizer operation is fully automatic. In the case of changing or adding a local pattern trend, the file which describes the trend is immediately transferred into the template directory. The functions used for reading the pattern trends always check if the updated definitions exist in the template directory and, if necessary, copy them to the local drive.

10 Using Pattern Trends

Pattern trends have two typical uses:

- Visual comparison of the actual trends of the process variables with the ideal pattern trend.
- Transfer of data determining process running method to the controller.

In the classic applications, the pattern trends were stored in a special type of archive of the ASPAD module. Additionally, selection and edition of the trends was supported by the *PEdit* and *PSelect* programs. All this functionality is now directly integrated into the Asix.Evo program.

10.1 Trends Edition

Editing and managing of the pattern trends is carried out in the trend editor which can be opened with the *Trend patterns editor* command of the *Tools* menu. During operation in the application run mode, a window may be opened via the *EditPatterns* operator action.

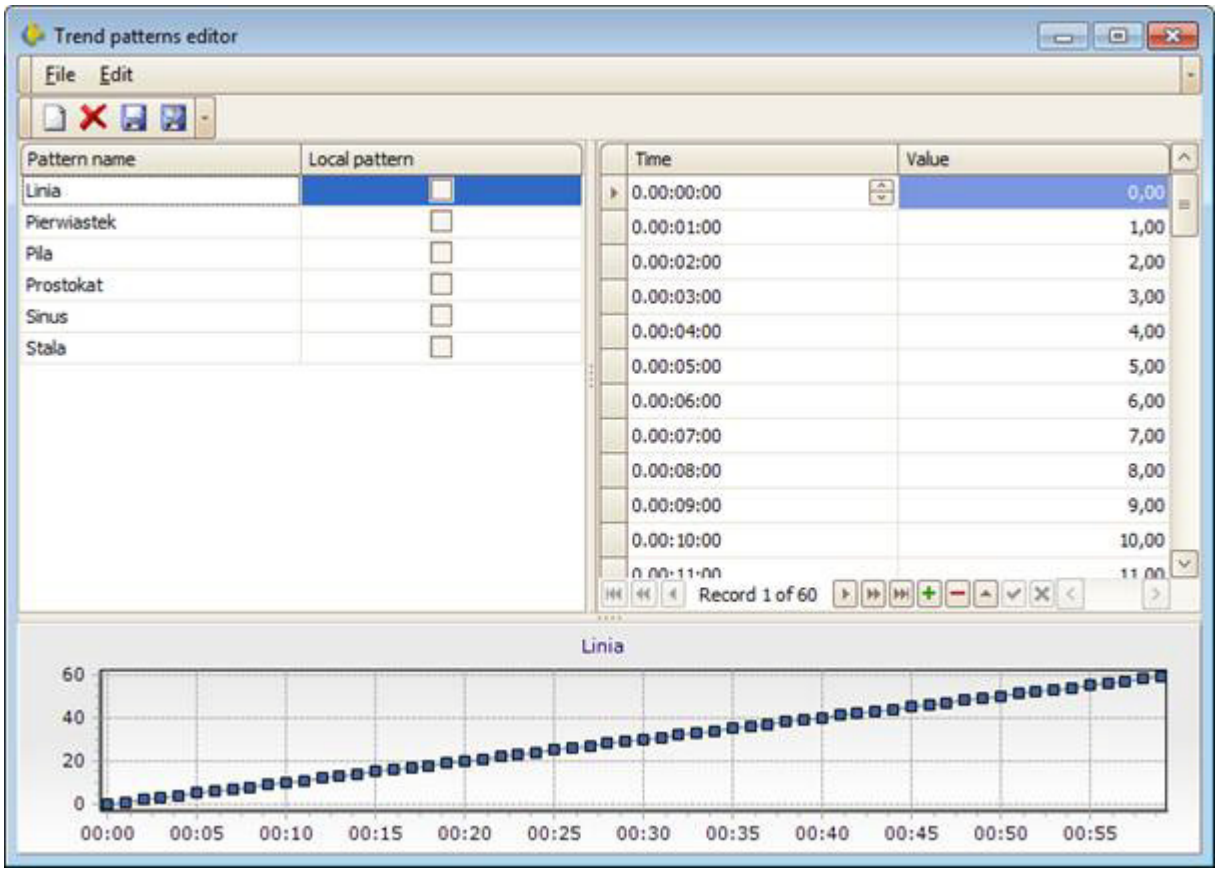

Fig. Pattern Trend Editor.

The pattern trends are divided into common trends and user local trends. The trend type should be determined as early as during its creation. Common trends must be created in the application edit mode. The file describing such trend is stored in the *TrendPatterns* subdirectory of the application definition directory. The common trend can not be changed by the user in the application run mode. The local trends are saved to the *TrendPatterns* subdirectory of the application working directory. Local trends can be set up and edited in the application run mode. In general, they are used on a specific application workstation. However, if synchronization of the working directory content is enabled, the local trends can be transferred to the other application workstations.

The pattern trend points can be edited in the table of points. It is also possible to move the location of the points directly on a preview chart. The left mouse button is used to move the points, the right one to move the chart and double-clicking adjusts the chart scale to the definition of the points.

If a conversion of the pattern trends from a classic application is necessary, the CSV file import function may be used.

10.2 Displaying Pattern Trends

Pattern trends can be displayed as a curve in the *Chart* class object. The pattern trends displayed on the *Chart* object may be pre-parameterized. The object also has its own interfaces which, in the application run mode, allow selecting the pattern trend and its first node.

It is also possible to automatically control the pattern trends parameters in the *Chart* object. An example of such operation is described under the *Controlling Trend Pattern Displaying* section in the *Asix.Evo-Techniques of Diagram Creation* user manual.

10.3 Trend Transfer

Pattern trends may also be used as the base to control the process under control of the PLC. To do this, it is necessary to transfer the trend definition to the PLC. As no standards exist, Asix.Evo has no built-in trend transfer functions. Operations of this type should be executed via scripts. The example of a pattern trend transfer to the PLC is described under the *Transferring Pattern Trends* section *of Asix.Evo - Scripts* user's manual.

11 Keyboard

If the application is run on a computer without a keyboard, the use of an on-screen keyboard to enter the settings may be necessary. A keyboard integrated with the operating system may be used. However, Asix.Evo provides two other mechanisms which are discussed in the subsequent chapters.

11.1 *Keyboard* Object

An object of the *Keyboard* class should be located on the diagram which contains the objects designed for entering settings. The keyboard may be displayed as fixed or may operate in the so-called auto-hide mode (the keyboard is displayed on the diagram only when the edit mode is active).

When the s edit mode of ettings is active (e.g. for the object of the *Text* class) subsequent characters can be entered using keyboard fields.

11.2 Box Keyboard

A more flexible mechanism is the keyboard displayed in its own box. In addition to entering settings in the diagram objects it also allows editing fields in the dialogue windows.

The appearance and operation mode of the on-screen keyboard is defined by the workstation settings of the working panel in the *Settings* tab.

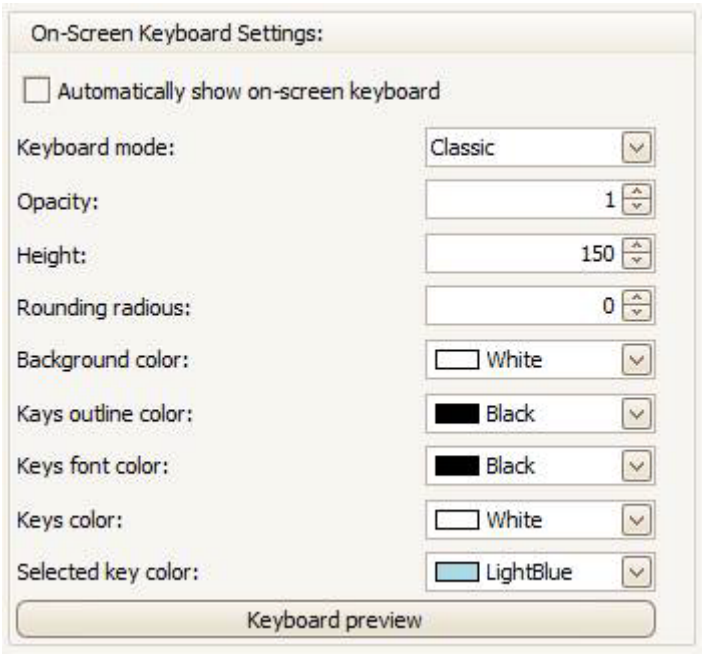

Fig. The On-Screen Keyboard Settings.

Selecting the *Automatically show on-screen keyboard* check box automatically displays the keyboard window when entering into the edit mode.

If the automatic mode is not enabled or entering into editing occurs for the application component which is not directly controlled, the keyboard window may be activated with the use of *ShowKeyboard* and *ShowNumericKeyboard* operator actions.