User's Manual for Asix 7

www.asix.com.pl

asix

see and get more ...

Asix.Evo - Visualization Elements

Doc. No ENP7E005
Version: 2012-07-26

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.

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 Visualization Component Hierarchy

Each Asix.Evo Application is displayed inside the so-called visualization windows. The number of windows used is arbitrary, but at least one window must be used. This is the start window of the Application. The visualization windows are Windows system objects and have typical properties of windows, such as the header line, system buttons, system menu, frame. In addition to these typical components of the operating system, the visualization window consists of an optional menu and one or more panels. The following illustration shows the window with menu and two panels in the edit mode.

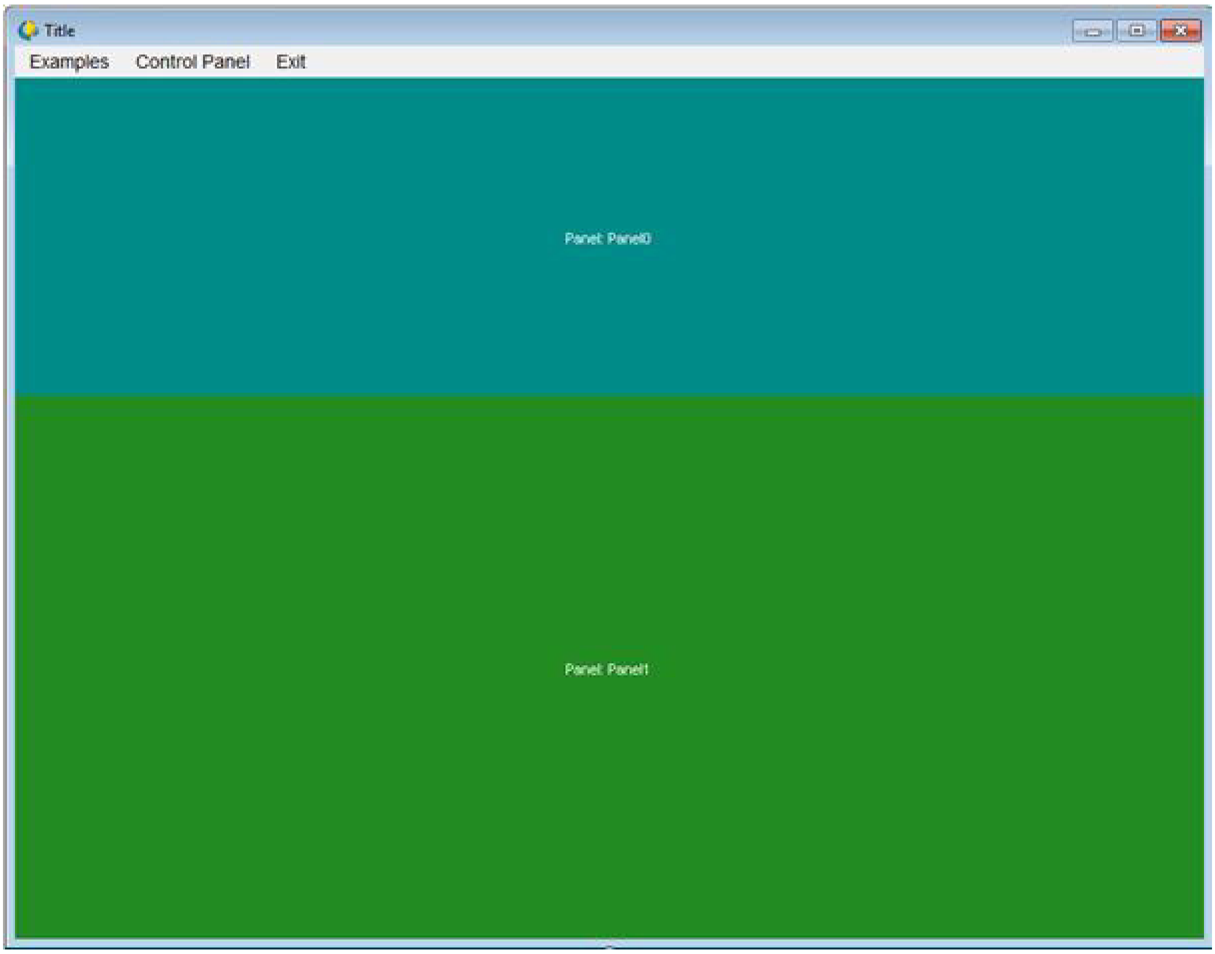


Fig. The Application Window in the Edit Mode.

The window panels are the containers for displaying a synoptic diagrams. During window displaying the diagrams inside the panels can be replaced by using the operator actions.

The synoptic diagram is the application main visualization component used to present the state and handle the controlled process. The diagram consists of the objects and templates embedded in it.

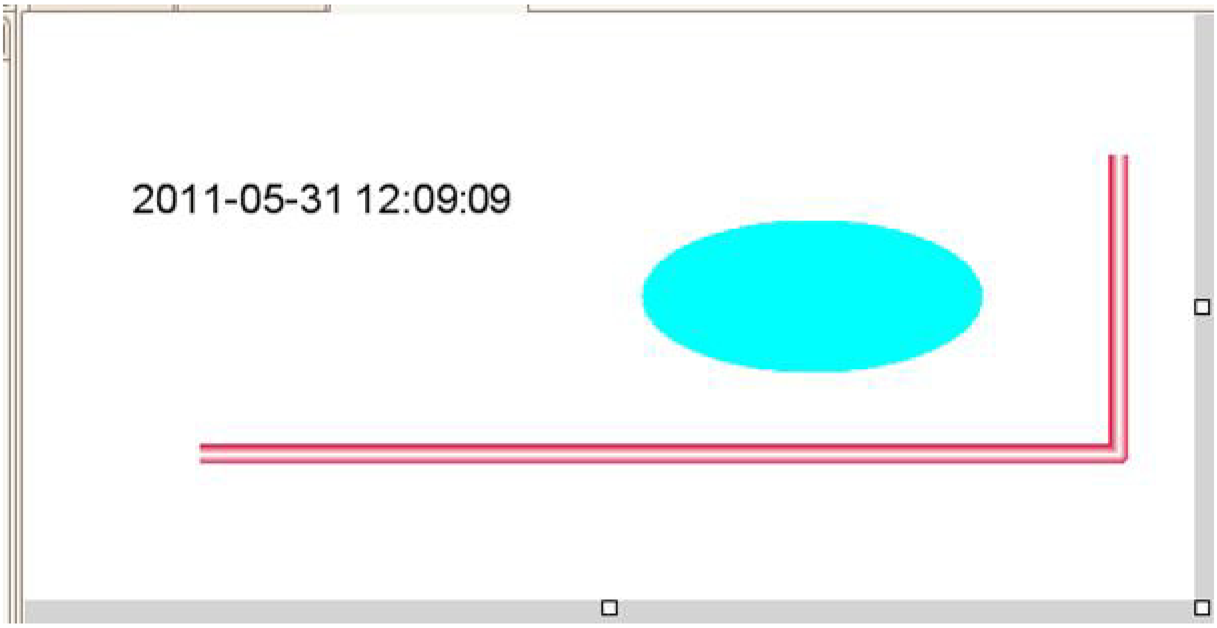


Fig. The Diagram with the Objects Embedded.

An object is a single component of the diagram. The presentation method and behaviour of an object depends on its type. Simple graphic objects and a very complex objects for the presentation of large amount of information, e.g. objects of the alarm table, are available.

The templates are the pre-parameterized groups of objects. They accelerate the process of an application development by multiple embedment of the visualization parts (created previously) on the diagrams.

All visualization components are parameterised by the sets of properties and events. The available sets of properties and events, and the possible methods of their parameterization depend on the type of a component. The so-called 'global properties' are the additional element facilitating the parameterization. They are centrally defined properties which are not associated with any particular application component. The application designer may refer to the global property in the definitions of various components. This facilitates modification of the application appearance or behaviour without changing the definition of properties of many subcomponents.

2 Properties and Events of Visualization Components

The properties and events of visualization components are defined in the *Properties* panel. After selecting a component a relevant set of properties and events is displayed inside the panel.

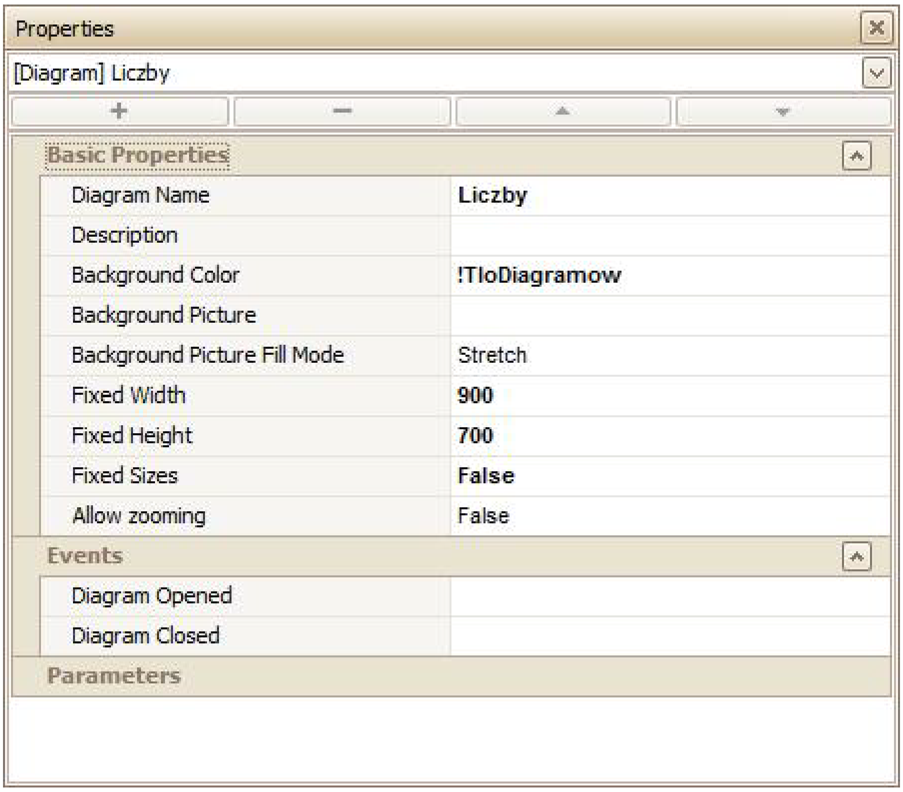


Diagram Name	Liczby
Description	
Background Color	!TloDiagramow
Background Picture	
Background Picture Fill Mode	Stretch
Fixed Width	900
Fixed Height	700
Fixed Sizes	False
Allow zooming	False
Diagram Opened	
Diagram Closed	

Fig. The Panel of Visualization Component Properties.

The properties of components are responsible for its appearance and way of operation. A typical example are the properties used to define the color used to display the component. The events, on the other hand, are used to execute the actions defined by the designer in response to the occurrence of certain events during the application runtime. A typical example is a mouse click event in the area of an object.

Both properties and events may be parametrised in such a way, that their value changes during the application runtime. These changes may depend on the changes in the controlled process state, component opening method or on the logged user permissions and its activities.

The property or event definition is a text that is interpreted depending on the first symbol of a definition and the context of use. Definitions can be entered manually or using the boxes available in the *Properties* panel.

The table below describes interpretation method for the properties definition depending on the used prefix symbol.

Prefix	**Meaning**	**Description**
~ or none	Direct value	The property value is specified directly and does not change while using the component. The value specification method depends on the property type. Examples: *Red* The properties defining the color accept the color name. *True* The boolean properties accept *True* and *False* texts. *Flat* The special properties may feature their own set of allowed values. *10* A large part of the properties accepts numeric values.
#	Main variable	The property value is equal to the current value of the object main variable. It can change in the application runtime mode. It is possible to use notation such as: #*Suffix*. It means loading the process variable value of a name consisting of the main variable name and specified suffix. The main variable name is defined in the *Main variable* property of the object. References to the main variable can be used only in the object context. The variable value must be compatible with the type of property in the context of which it was used. Examples: # Returns the main variable value. If in the *Text* property the *Text* object is used, the variable value converted to text form will be displayed. # _limit If the main variable name is *Z1*, the variable value of *Z1_limit* will be loaded.
&	Process variable	Loading of any process variable value. The variable name should be specified after the prefix symbol. The variable value must be compatible with the type of property

		in the context of which it was used. Example: &z1 Loading of the value of the variable *Z1*.
@	Variable Attribute	Loading of the attribute value of the process variable from the variable definition database. The name of the process variable and attribute should be specified following the prefix symbol. If the variable name is not specified, then the object main variable is used. If the variable name is not specified, then the property default attribute is loaded. The attribute value must be compatible with the type of property in the context of which it was used. Example: @ Loads the main variable default attribute. This form can be used only if the property has a defined default attribute. @Description Loads object main variable description. @Z1.Description Loads the *Z1* variable description.
%	Parameter	Loading of parameter value of template or diagram. The parameter name should be specified after the prefix symbol. The parameters provide the mechanism that allows creating templates and diagrams, the functioning of which depends on the parameters provided only when the component is used. The parameter value must be compatible with the type of property in the context of which it was used. Example: % variable Loading the diagram or template parameter *zmienna* value.
!	Global property	Loading the global property value. The property name should be specified after the prefix symbol. The global property value must be compatible with the type of property in the context of which it was used.

		Example: !KolorLowLow Loading the global property *KolorLowLow* value.
=	Expression	The calculation of an arithmetic expression value. The expression contents should be specified after the prefix symbol. The expression result may depend on many factors (e.g. process variables values). It is calculated in on a real time basis during the application runtime. See: ***Expressions and Functions (ExpressionandFunction.CHM/PDF)***. The expression calculation result must be compatible with the type of property in the context of which it was used. Example: = "Temperature =" + Variable (t1) The expression result is a text combining the text: *"Temperature ="* with the *t1* variable value.
^	Operator action	Defines the contents of the operator action which will be executed when the event occurs. The action contents should be given after the prefix symbol. Expressions can be used internally in the action parameters. This creates the possibility of executing actions which operation depends on factors such as process variable etc. See: ***Operator Actions (OperatorActions.CHM/PDF)***. Example: ^ SetVariable ("#", Variable()+1) The action increases the current value of the object main variable by 1.

NOTE:

Not all prefixes are permitted in every context of use. In the event definitions only the operator action prefix can be used. Some parameters can be entered directly only - this relates mainly to the window and panel parameters.

3 Windows and Panels

A window is a key component of the application inside which diagrams (used for presentation of the controlled process status) are displayed. Most of the window parameters is used to determine its appearance and available system functions.

A window may have its own menu, which name is specified in the *Menu* property. Depending on the menu definition, it can be displayed along any edge of the window.

Each window consists of panels inside which, during the operation, the diagrams are inserted. Diagrams to be displayed inside individual panels may be selected:

- at the stage of the panel definition process, in the *Default Diagram* property.
- when the application is running, in the *OpenWindow* operator action contents.

When defining the panels, it is important to determine their size, especially if the designed window is permitted to resize. The panels can match their size to the window being resized. *Dock Style* and *Anchor* panel properties are used for this purpose. It is recommended to check the functioning of these parameters experimentally. For the docked panels the sequence of defining panels is of key importance.

The window may be opened by:

- selecting the window as start-up window in the station settings - in this case the window is opened along with the application start-up.
- Use of the *OpenWindow* action
- Use of the *OpenDiagram* action - in this case a temporary window consisting of a single panel, covering the entire window, of the size matched to the diagram size is created.

3.1 Window properties

Name	**Value Type**	**Description**
Window Name	Name	Name identifying the window. The first character must be a letter, and the next one can be a letter, a number or character _.
Window Title	Text	Description of the window contents displayed in the title bar

Background Color	Color	Color used to display the window background. It is important in the edit mode or when the window area is not completely covered by panels.
X	Number	The X coordinate for the window opening in the runtime mode, expressed in pixels. The opening coordinate may be changed in the operator action that opens a window or may result from the opening mode specified in the *Window Initial State* property.
Y	Number	The Y coordinate for the window opening in the runtime mode, expressed in pixels. The opening coordinate may be changed in the Operator action that opens a window or may result from the opening mode specified in the *Window Initial State* property.
Width	Number	Window width in pixels.
Height	Number	Window height in pixels.
Menu	Menu name	Optional menu name used in the window.
Show Title Bar	Logical	Specifies whether the window title bar is to be displayed.
Show Frame	Logical	Specifies whether the window frame is to be displayed. Some other settings can also force the frame to be displayed.
Fixed Position	Logical	Specifies whether the user can change the position of a window.
Fixed Size	Logical	Specifies whether the user can change the size of a window. Even if the size is locked, it can be changed with the *SetWindowSize* operator action.
Tool Window	Logical	It changes the title bar appearance and at the same time blocks the system buttons displaying.
Show Bar Buttons	Logical	It blocks the system buttons displaying on the title bar.
Show Maximize Button	Logical	Specifies whether the maximize button is to be displayed on the title bar.
Show Minimize Button	Logical	Specifies whether the minimise button is to be displayed on the title bar.
Background Window	Logical	Specifies whether the window is to be always displayed underneath other windows (of the Asix.Evo applications and others). Usually used for large start-up windows of the application. It reduces the use of the *Top Most* mode.
Top Most	Logical	Specifies whether the window is to be always displayed above the other windows. This causes the inactive window will not be covered. This mode should not be used for large windows because they can cover other windows completely. An alternative

		method is to use the *Background Window* mode for the start-up windows.
Window Initial State	Enumerated type: Normal, Minimized, Maximized, Center	Specifies the window opening mode. The *normal* mode opens the window according to the *X, Y Width* and *Height* parameters. The *Minimized* mode minimizes the window, and the *Maximized* enlarges (or reduces) the window to the full screen size. This option is especially useful when the Application should be adjusted automatically to the different screen resolutions. The *Center* option displays a window in the middle of the screen while maintaining the original size.
Opacity	Number <0.1>	Specify the windows transparency level in the range from 0 to 1 The value 1 means an untransparent window, a value of 0 means a completely transparent window.
Show in Taskbar	Logical	Specifies whether information about the open window is to be displayed on the taskbar.
Closable	Logical	Prevents the user from closing the window. Blocked window can be closed only through the *CloseWindow* action.

3.2 Panel Parameters

Name	**Value Type**	**Description**
Panel Name	Name	Name identifying the panel. The first character must be a letter, and the next one can be a letter, a number or character _.
Default Diagram	Diagram name	The name of diagram which should be automatically opened inside the panel. Used when the window is a start-up window of the application or when the diagram name was not specified in the *OpenWindow* action.
Background Color	Color	Color used to display the panel background. It is of significance only in the edit mode. In the runtime mode it is used only when the diagram is not connected to the panel.
Dock Style	Enumerated type: None, Top, Right,	Specifies the panel dock mode. The *Fill* mode means that the whole space available will be occupied. The

	Fill, Left, Right	dock to the edge mode means the space at the selected edge will be occupied - the panel size is then specified by the *Width* or *Height* properties.
Anchor	Multiple enumerated type: None, Top, Bottom, Left, Right	Specifies the method for calculation the panel size when a window is being resized. Anchoring to the selected edge means that the constant distance between a window and this edge is kept. A panel can be anchored to several edges simultaneously.
X	Number	The X coordinate of the panel position expressed in pixels and calculated with reference to the upper-left corner of the window. It is of no significance for docked panels.
Y	Number	The Y coordinate of the panel position expressed in pixels and calculated with reference to the upper-left corner of the window. It is of no significance for docked panels.
Width	Number	The initial panel width in pixels. It can be changed when the window is resized while using the dock or anchor modes.
Height	Number	The initial height of the panel in pixels. It can be changed when the window is resized while using the dock or anchor modes.

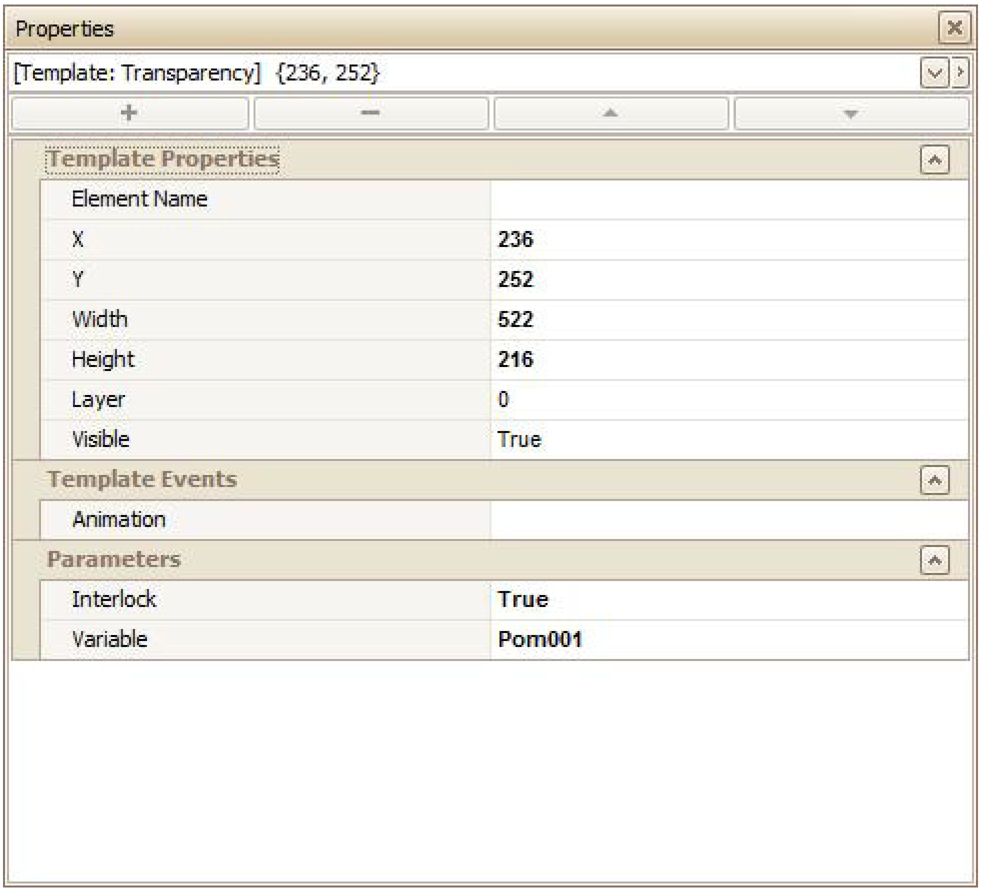
4 Diagrams

The diagrams are the main visualization components of the application and are used to present the state and to handle the controlled process. The diagrams are always displayed inside the window panel. The window can be opened by:

- declaring the diagram as the panel default diagram - in this case when the window is opened (if there are no other dispositions available), the diagram will be displayed automatically,
- using the diagram name in the *OpenWindow* action contents - the diagram will be opened in the indicated window panel; the action may relate to the newly opened window or to the one opened previously (diagram replacement),
- using the diagram name in the *OpenDiagram* action contents - the diagram will be opened in an automatically created temporary window.

An important aspect of the diagram definition is to determine how the diagram size is adjusted to the panel size. The *Fixed Sizes* parameter is used to perform this function. If it is set to *False*, the diagram size and the panel size are always identical - if the panel is resized, location and dimensions of all diagram components are rescaled. If the fixed sizes mode has been selected, the diagram will not be scaled, it is displayed in its original size, and if necessary, panel is completed with a sliders allowing to choose a part of the diagram to be visible. The fixed size mode does not exclude the manual scaling of the diagram. Setting the *Allow zooming* property to the *True* value allows the user to scale the diagram size with the mouse wheel (while holding down the *Control* key), but still the panel size change does not affect the way the window is displayed.

The diagram consists of objects, groups of objects and templates embedded in it. The object groups allow to connect several objects localised on the diagram together. It facilitates the objects editing (in particular the selection, moving and copying objects of the group). In addition, the motion animation of the object groups by handling the *Animation* event is easier. Moreover, the objects of the group behave identically as ungrouped objects. Different situation is in the case of embedded template. The template similarly as a group, consists of a set of objects. However, in this case, the definitions of these objects are not included in the definition of the diagram. The templates are a separately defined components of the application. Only the information on a template embedment method is stored in the diagram. This information consists of the properties which define the size and position of the template and the parameters that were used in the template.



Properties

[Template: Transparency] {236, 252}

Template Properties	
Element Name	
X	236
Y	252
Width	522
Height	216
Layer	0
Visible	True
Template Events	
Animation	
Parameters	
Interlock	True
Variable	Pom001

Fig. Template Properties Panel.

Similarly to the group, the template has the *Animation* event, which allows for animation of the template motion.

The diagram can be of a parametrised type. This allows using the same diagram to display information from various sources or to modify the appearance without changing the diagram definition. The correct use of parameters consists of two steps:

- At the stage of the diagram defining process, the diagram parameters are defined. The definition consists of the parameter name and its default value. The default value is used during the diagram edition and when, in the diagram opening action other parameter values have not been specified. On completion the parameter defining process, the parameter may be called from the object properties with the suffix notation *%name* or by calling the *Parameter* function.
- In the application runtime mode the parameter values should be specified in the diagram opening actions called *OpenWindow* and *OpenDiagram*. This parameters are specified as a sequence of pairs: *parameter_name* = *parameter_value* separated by a semicolon, e.g., *Color= Red;Variable=v1*.

The diagram supports events which allow performing of any operator actions at the time of the diagram opening and closing.

4.1 Diagram Properties

Name	**Value Type**	**Description**
Diagram Name	Name	Name identifying the diagram. The first character must be a letter, and the next one can be a letter, a number or character _.
Description	Text	Diagram description of the information importance only.
Background Color	Color	Color used to fill the diagram background
Background Picture	Image Name	The image name, which will be used as the diagram background.
Background Picture Fill Mode	Enumerated type: Tile, Stretch, Center, Original	How to display the background image. The Original type means that the picture will be displayed in original dimensions in the upper left corner of the diagram. In the Tile type, the picture is displayed in multiple instances, in order to cover the entire diagram area. In the centring mode the picture is displayed in original size in the middle of the diagram. In the stretching mode, picture size is adjusted to the diagram size.
Fixed Width	Number	The initial diagram width, expressed in pixels, in the *Fixed Sized* mode.
Fixed Height	Number	The initial diagram height, expressed in pixels, in the *Fixed Sized* mode.
Fixed Sized	Logical	Determines whether the diagram size is independent of the panel size, inside which the diagram is displayed. *False* value means that the diagram is always the same as the panel size - the panel resizing results in the diagram rescaling. The *True* means that the diagram is displayed according to the dimensions specified in the *Fixed Height* and *Fixed Width* properties. If the panel size is smaller than the diagram size, the sliders will be displayed.
Allow zooming	Logical	Determines whether the user can scale the diagram size with the use of a mouse wheel (while holding down the *Control* key) in the *Fixed Sized* mode.

4.2 The Diagram Events

Name	Description
Diagram Opened	The event triggered when the diagram is opened. A typical example of use is an establishment of the connection with AsBase through the *AsbaseOpen* action.
Diagram Closed	The event triggered when the diagram is closed. A typical example of use is a termination of the connection with AsBase through the *AsbaseClose* action.

4.3 Group Properties

Name	Value Type	Description
Element Name	Name	The name identifying the object group.
X	Number	The X coordinate of the object group position expressed in pixels and calculated with reference to the upper-left corner of the diagram.
Y	Number	The Y coordinate of the object group position expressed in pixels and calculated with reference to the upper-left corner of the diagram.
Visible	Logical	Specifies whether the object group is to be visible.

4.4 Group Events

Name	Description
Animation	The event run periodically. Designed for the animation of an object groups motion.

4.5 The Properties of the Embedded Templates

Name	Value Type	Description
Element Name	Name	The name identifying the embedded template.

X	Number	The X coordinate of the template position expressed in pixels and calculated with reference to the upper-left corner of the diagram.
Y	Number	The Y coordinate of the template position expressed in pixels and calculated with reference to the upper-left corner of the diagram.
Width	Number	The width of template embedding area
Height	Number	The height of template embedding area
Layer	Number	The base layer number, in reference to which the number of the template component objects layer is calculated - the template layer number is added to the layer number of the object. Objects with a higher number of layers cover the objects with a lower number. The way the objects with the same number are displayed results from the order in which they have been defined.
Visible	Logical	Specifies whether the object group is to be visible.

4.6 The Events of the Embedded Templates

Name	**Description**
Animation	The event run periodically. Designed for the animation of a template motion.

5 Menu

The menus of two types may be used in the Asix.Evo applications. The first type menu is a menu associated with a window. The menu structure and appearance may be freely defined. The menu is connected to the window via the window *Menu* property. The *Dock Style* property determines how the menu is connected to the window.

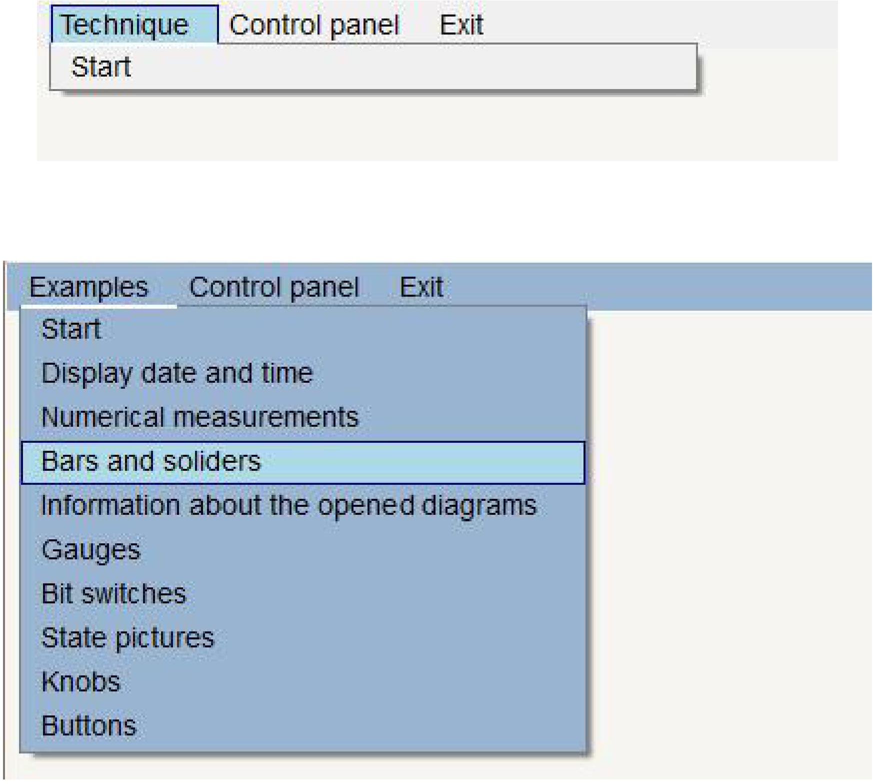


Fig. Example of a Menu Associated with the Application Window.

The second type menu is a context menu, usually connected to objects. The context menu is opened on the *ShowMenu* operator action execution.

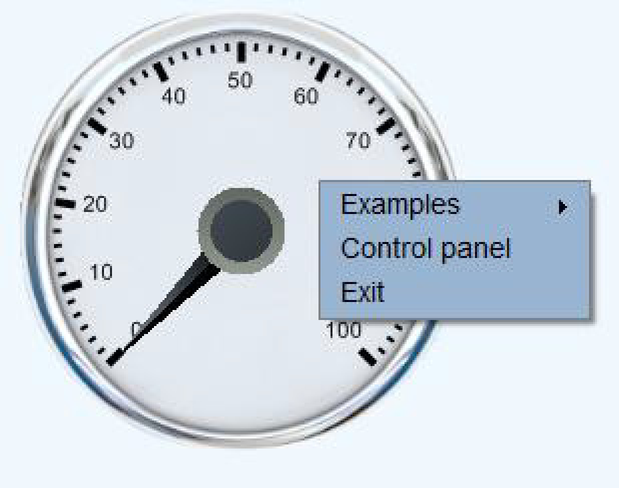


Fig. An Example of a Context Menu Associated with an Object.

The context menu may be of parameterized type. This allows to use the same menu in many places and make its operation dependent on parameters transmitted in the menu opening action. General rules for the use of the properties are the same as for the diagrams properties.

The menu appearance can depend on the state of the application. In particular, the *Visible* property of a menu item allows to hide unwanted elements, and the *Active* property controls the ability to execute the action associated with the menu item. The menu functioning may be interlocked with the user permissions:

HasRole(superoperator)

, or with the process state:

Variable(v1)==0

5.1 Menu Properties

Name	**Value Type**	**Description**
Menu Name	Name	The name that identifies the menu. The first character must be a letter, and the next one can be a letter, a number or character _.
Font Color	Color	Color used to display the text of the menu items.
Background Color	Color	The background color of menu items.
Border Color	Color	Border color around the menu items.

Background Highlight Color	Color	The background color of the menu items being selected.
Background Highlight Color	Color	Border color around the menu items being selected.
Font	Font Name	The system font name used to display menu items.
Font Style	Multiple enumerated type: Regular, Bold, Italic, Underline, Strikeout	Determines the used font style. Multiple styles can be used simultaneously.
Font Size	Number	Determines the used font size.
Dock Style	Enumerated type: None, Top, Bottom, Fill, Left, Right	Determines at which edge of the window the menu is to be located. The *None* mode should be used for the context menu of the objects.

5.2 Menu Items Properties

Name	**Value Type**	**Description**
Caption	Text	The text displayed in the menu item
Visible	Logical	Determines whether the menu item is to be visible. As a rule, the option is calculated with the expression taking into account the logged user permissions or the process variable values.
Enabled	Logical	Specifies whether the menu item is to be active. Inactive elements are displayed as shaded and will not run any operator action. As a rule, the option is calculated with the expression taking into account the logged user permissions or the process variable values.
Separator	Logical	Specifies whether the menu item is to be a separator. If yes, in the item position only the line separating the remaining menu items is displayed.
Action	Operator action	The operator action performed in response to selection of a menu item by the user.

6 Objects

The objects are the basic visualization components. They are placed on the diagrams and inside the template. The appearance and functionality of the objects depend on the object class. There following object classes are available:

- *Log Viewer* - displays the system messages table
- *Keyboard* - on-screen keyboard
- *Messages* - displays the text loaded from the message definition file
- *Chart Controller* - an object used to control the operation of objects of *Chart* class.
- *Shape* - displays shapes based on the rectangle and ellipse
- *Line* - a line displaying, also these ones of multisegment type
- *Gauge* - a gauge displaying
- *Picture* - a graphical picture displaying
- *Authorization box* - a user login control
- *Web Browser* - web pages and document contents displaying
- *Button* - various styles of buttons displaying
- *Pipe* - pipeline type object
- *Bar* - displays any value in the form of bar with an optional control slider
- *Active Alarms Viewer* - displays a table with active alarms
- *Historical Alarms Viewer* - displays a table with historical alarms
- *Variable table* - displays the values of process variables in a tabular form
- *Conveyor* - a conveyor type object
- *Text* - displays information in a text form
- *Polygon* - displays any polygon shape
- *Chart* - displays a characteristics of archival variables or model trends
- *Tank* - a tank type object

There is the possibility to add own object classes to the application by expanding the system with a properly coded software modules.

The basic functionality of the object results from its class. However, the application designer has a great freedom concerning details of objects functioning. The vast majority of the object properties may be dynamically changed during the application runtime. This allows getting virtually any animation effects.

Some object classes have built-in interactive functions (such as *Text* , *Bar*, *Button* objects, alarm tables). For non-interactive objects (e.g. Line, Shape, Pipe), the designer may add this functionality via the event handling. For example, each object can handle the event of a mouse click and execute any operator action at that moment.

Object classes are divided into two types:

- Graphic - a basic type
- Control - objects based on the use of standard system controls. The control type objects are: *Log Viewer*, *Chart Controller*, *Authorisation Box*, *Web Browser*, *Active Alarms Viewer*, *Historical Alarms Viewer*, *Variable Table*, *Chart*.

The most significant limitation for the control type objects is that they cannot be overlayed. Graphical objects can be placed in the same area with the mutual overlaying control.

Objects are parameterized by sets of properties and events. There are the sets of so-called standard properties and events that are present virtually in all classes. Each class can introduce its specific additional properties and events.

6.1 Properties Groups

Objects properties are always placed in some group. There are three types of the property groups described in the following chapters.

6.1.1 Basic Properties

Group of properties that occur once. The *Main variable* is the most important property of this group. It sets out the context of the object. With the correct definition of the object (using a function in the contextual versions, e.g. *Variable ()*, or suffix notation of variable names), the property allows creating the objects which after copying may be switched to show other data by changing only the main variable name.

The property values may be dynamically changed by using in their definitions expressions and references to the elements which value changes.

Example:

Property: *Active*

Definition: = HasRole (DIR)

Activity state of the object changes depending on the logged user.

Property: *Hint*

Definition: @Description

The text displayed in a hint will be a description of the main variable of the object

Property: *Value* of the *Gauge* object

Definition: #

The pointer position of the *Gauge* object will depend on the value of the object main variable.

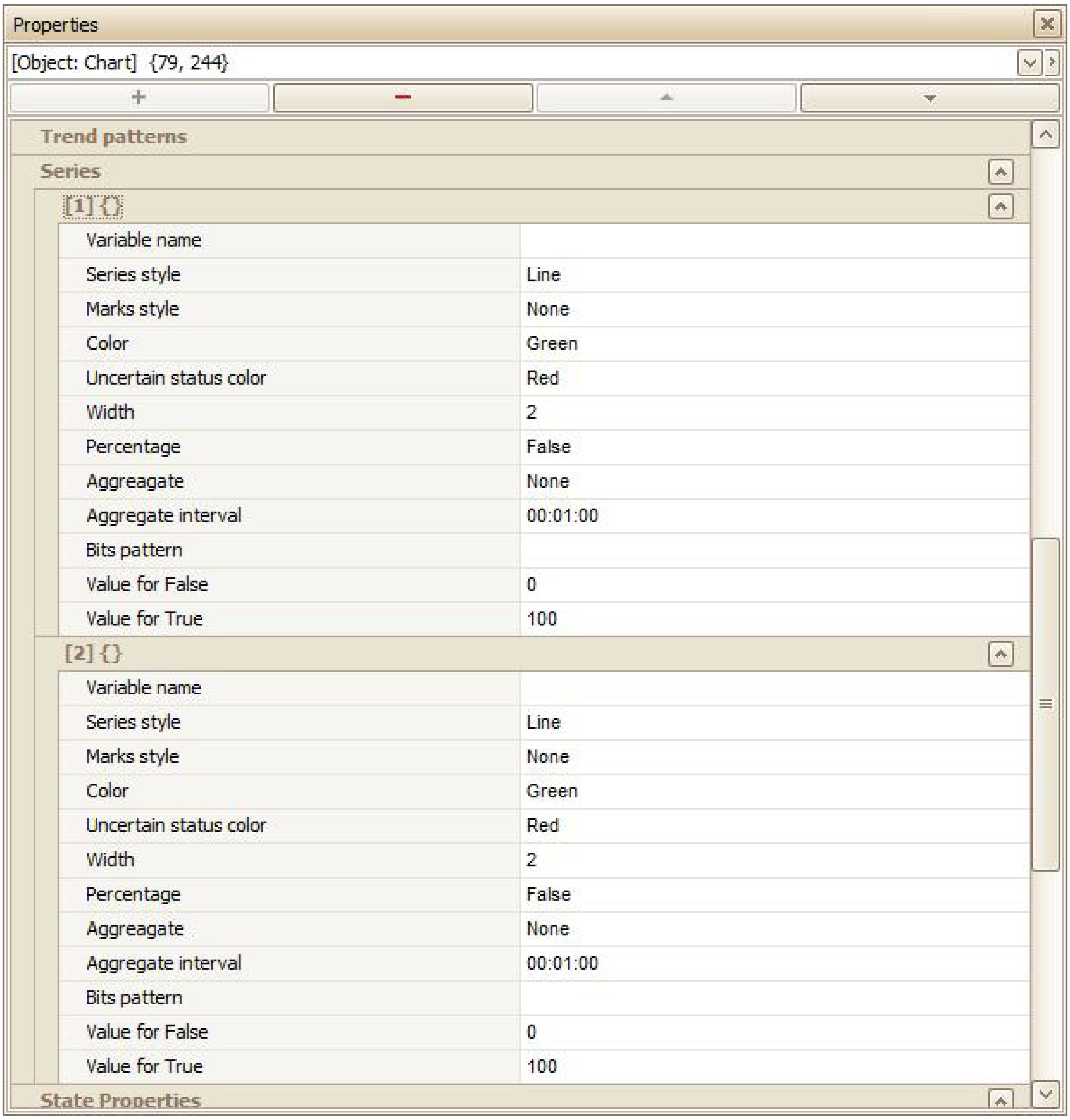
Property: *Layer*

Definition: = Variable (r1)> 0? 10:0

If the variable *r1* value is greater than 0, the layer number is set to 10. Otherwise it is equal to 0. The above expression allows controlling the mutual overlaying of the objects in a dynamic manner.

6.1.2 Multiple Groups

Properties multiple groups are available only in a certain object classes, e.g. *Chart, Variable Table*. They are always used in a cases, where some repetitive elements are created within an object. The classic example are definitions of displayed trends in the *Chart* object.



[Object: Chart] {79, 244}

Trend patterns

Series

[1] {}

Variable name	
Series style	Line
Marks style	None
Color	Green
Uncertain status color	Red
Width	2
Percentage	False
Aggreagate	None
Aggregate interval	00:01:00
Bits pattern	
Value for False	0
Value for True	100

[2] {}

Variable name	
Series style	Line
Marks style	None
Color	Green
Uncertain status color	Red
Width	2
Percentage	False
Aggreagate	None
Aggregate interval	00:01:00
Bits pattern	
Value for False	0
Value for True	100

State Properties

Fig. Multiple Groups of the Object Properties.

The designer can create just as many properties groups as needed. Each group is independent of the others. The rules for calculating the values are the same as for the properties of the basic group.

6.1.3 State Properties

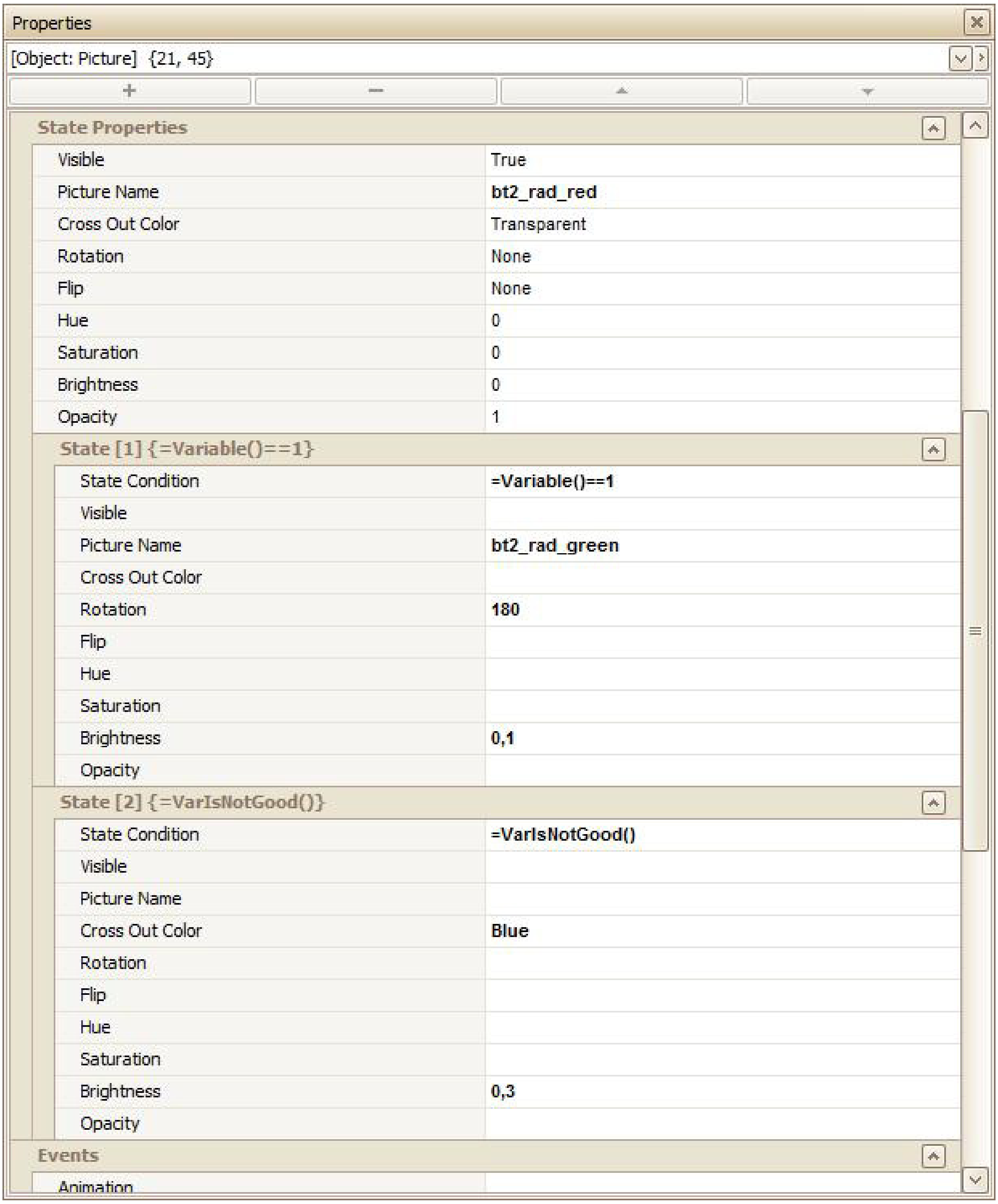
The state properties are special properties that allow easy creation of an objects which change their appearance depending on the application and controlled process state. It is possible to achieve similar results without the use of state groups, but this requires much more complicated expressions.

The idea of the state properties is to create many groups of identical properties. In each of these groups the properties can be defined in different ways. Moreover, the group feature an additional boolean property called *State Condition* that determines whether the group is taken into account when calculating the effective value of the property used for the object displaying.

The algorithm for calculating the property value is as follows:

- The group whose state condition is *True, and* the definition of property is not blank.
- If there is only one such group, the value of property results from the definition used in this group.
- If there are more such groups, then the property value is loaded from the definition of the last group.
- If none of the group is active, or in none of the active groups the property is defined, then the value is determined on the basis of the definitions in the basic (primary) group.

Example:



Properties

[Object: Picture] {21, 45}

State Properties	
Visible	True
Picture Name	**bt2_rad_red**
Cross Out Color	Transparent
Rotation	None
Flip	None
Hue	0
Saturation	0
Brightness	0
Opacity	1
State [1] {=Variable()==1}	
State Condition	**=Variable()==1**
Visible	
Picture Name	**bt2_rad_green**
Cross Out Color	
Rotation	**180**
Flip	
Hue	
Saturation	
Brightness	**0,1**
Opacity	
State [2] {=VarIsNotGood()}	
State Condition	**=VarIsNotGood()**
Visible	
Picture Name	
Cross Out Color	**Blue**
Rotation	
Flip	
Hue	
Saturation	
Brightness	**0,3**
Opacity	
Events	
Animation	

Fig. Example of a State Properties for the Picture Object.

In the presented fragment of the definition of the *Chart* class object, it is shown how the object appearance depends on the variable value and its status. The state group number 1 is active when the value of the main object variable is set to 0. The state group number 2 is active when the status of the main variable is incorrect. The property value specification method is as follows:

The value of the main variable is not equal to 1, and the variable state is correct	The values of all properties are loaded from the basic group.
The value of the main variable is equal to 1, variable state is correct	The displayed picture is bt2_rad_green, the picture is rotated by 180 degrees and its brightness is changed with the 0.1 ratio. All the other properties are loaded from the basic group.
The value of the main variable is not equal to 1, variable state is correct	The picture is crossed out and its brightness is changed with the 0.3 ratio. The other settings are loaded from the basic group, in particular, the picture ibt2_rad_red is displayed.
The value of the main variable is equal to 1, variable state is incorrect	The displayed picture is bt2_rad_green, the picture is rotated by 180 degrees, is crossed out and its brightness is changed with the 0.3 ratio. The brightness ratio has been loaded from the group controlling the variable state since it has been defined later.

6.2 Object Standard Properties

Name	**Value Type**	**Description**
Element Name	Name	The name of the embedded object. It allows to identify the object and change its properties using the *SetProperty* action.
X	Number	The X coordinate of the object position expressed in pixels and calculated with reference to the upper-left corner of the diagram.
Y	Number	The Y coordinate of the object position expressed in pixels and calculated with reference to the upper-left corner of the diagram.
Width	Number	The width of the object embedment area expressed in pixels.
Height	Number	The hight of the object embedment area expressed in pixels.
Layer	Number	The layer number on which the object is displayed. Objects with a higher number of layers cover the objects with a lower number. The way the objects with the same number are displayed results from the order in which they have been defined.
Active	Logical	Specifies whether the object is active. In the case of objects with built-in control functions, the *False* value blocks this functionality. For objects without a standard interactive functions (e.g. *Shape*), the property value is

		of no direct significance, however, may be controlled in the actions and functions used in the object properties and events - the activity state can be loaded via the *IsActive* function.
Minimum Visible Width	Number	The minimum width of the object in pixels at which the object is displayed. If as a result of a diagram scaling the object width falls below the limit value, the object is automatically hidden.
Minimum Visible Height	Number	The minimum height of the object in pixels at which the object is displayed. If as a result of a diagram scaling the object height falls below the limit value, the object is automatically hidden.
Main Variable	Variable name	The object main variable name. This property allows central entering of the name of variable which value and definition determine the object functioning. In other properties of the object the main variable can be called via contextual function of *Variable* and *Attribute* type, brief calls to the name and value of the main variable in the form of *#*, or derivative names of the variables may be created by using the suffix notation *# suffix*.
Control Variable	Variable name	The variable name that is saved as a result of the use of the object control functions (for objects with built-in control functions e.g *Text*). Example: # The controlled variable is identical to the main variable of the object. This is the only case when the use of notation *#* indicates the main variable name and not its value. #_s The controlled variable name is created by combining the main variable name and the suffix *_s*. @ControlVariable The controlled variable name loaded from the attribute of the main variable.
Hint	Text	A definition of the text displayed in the hint. The hint contents may depend on the values of attributes from the variable definition database or on the process variable values.

		Example: @Description Displays a description of the object main variable =Format(”{0} = {1}”, Attribute(Name),Variable()) Displays the name and value of the object main variable.
Cursor	Cursor Name	The system name of the cursor type used when the cursor hovers over an object. This way the object readiness to execute interactive operations can be signalled, e.g. execution of an action on a mouse click.
Position Lock	Logical	Specifies whether a change of the object position is possible. It applies only to the diagram edit mode.
Size Lock	Logical	Specifies whether a change of the object size is possible. It applies only to the diagram edit mode.
Visible	Logical	Specifies whether the object is to be displayed. In the edit mode, the property value is ignored - the object is always visible.
Opacity	Number <0.1>	Specifies the object transparency level in the range from 0 to 1. The value 1 means an opaque object, the value of 0 means a completely transparent object.

6.3 The Standard Object Events

Name	**Description**
Animation	The event run periodically. Designed for animation of an object motion.
Control Send Event	This event is triggered upon the *SendControls* operator action execution. It applies only to the active objects.
Control Cancel Event	This event is triggered upon the *CancelControls* operator action execution. It applies only to the active objects.
Left Button Down	This event is triggered when the left mouse button is pressed while the cursor is within the object area.
Right Button Down	This event is triggered when the right mouse button is pressed while the cursor is within the object area.
Middle Button Down	This event is triggered when the middle mouse button is pressed while the cursor is within the object area.
Left Button Up	This event is triggered when the left mouse button is released while the cursor is within the object area.
Right Button Up	This event is triggered when the right mouse button is released while the cursor is within the object area.
Middle Button Up	This event is triggered when the middle mouse button is released while the

	cursor is within the object area.
Left Button Click	This event is triggered when the left mouse button is clicked (pressed and released) while the cursor is within the object area.
Right Button Click	This event is triggered when the right mouse button is clicked while the cursor is within the object area.
Middle Button Click	This event is triggered when the middle mouse button is clicked while the cursor is within the object area.
Left Button Double Click	This event is triggered when the left mouse button is double clicked while the cursor is within the object area.
Right Button Double Click	This event is triggered when the right mouse button is double clicked while the cursor is within the object area.
Middle Button Double Click	This event is triggered when the middle mouse button is double clicked while the cursor is within the object area.
Left Button Hold	This event is triggered when the mouse cursor is within the object area and the left mouse button is held down.
Right Button Hold	This event is triggered periodically when the mouse cursor is within the object area and the right mouse button is held down.
Middle Button Hold	This event is triggered periodically when the mouse cursor is within the object area and the middle mouse button is held down.
Mouse Enter	This event is triggered when the mouse cursor is moved into the object area.
Mouse Leave	This event is triggered when the mouse cursor is moved outside the object area.
Mouse Move	This event is triggered when the mouse cursor moves within the object area.
Key Press	This event is triggered when the keyboard key is pressed and the object is selected. The last key pressed may be checked with the function *LastKeyPressed*. Example: Perform (LastKeyPressed()==q, SetVariable (v1, 1), Nothing()) The variable v1 is set to 1 if the *q* key has been pressed.
Select Object	This event is triggered when the object is selected. It applies only to the active objects.

7 Templates

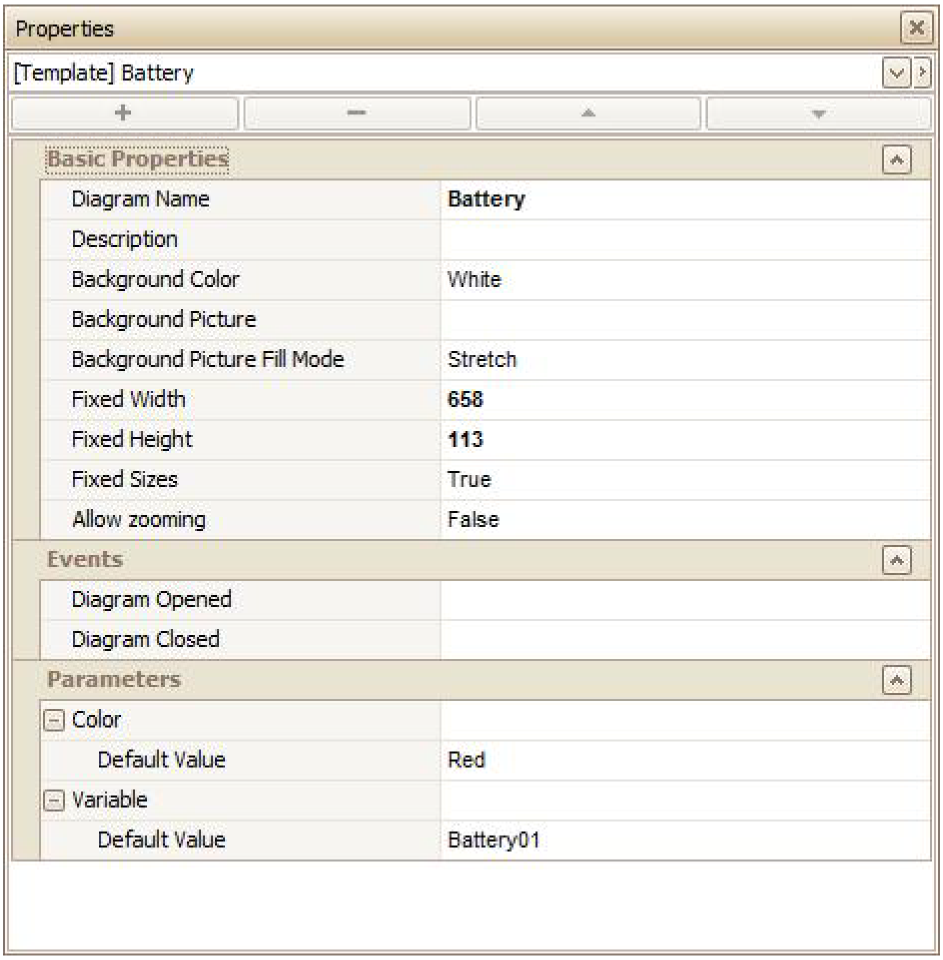
The templates are the pre-parameterized groups of objects. The templates are identified by a name. The template, after it was created, may be embedded on the diagrams repeatedly. If the definition of any template component is changed, it will automatically affect all diagrams using the template.

The diagram can also consist of a single object. This allows to prepare the objects that implement certain action scheme. Change of this scheme does not require the change of diagrams.

An embedded template can be converted into ordinary objects. This creates a copy of all template objects on the diagram. From this moment the objects from the template are treated as any other object of the diagram.

The key element of the template definition are the parameters. This allows to embed a template repeatedly with entering the data on which the template will operate, in particular, the names of process variables.

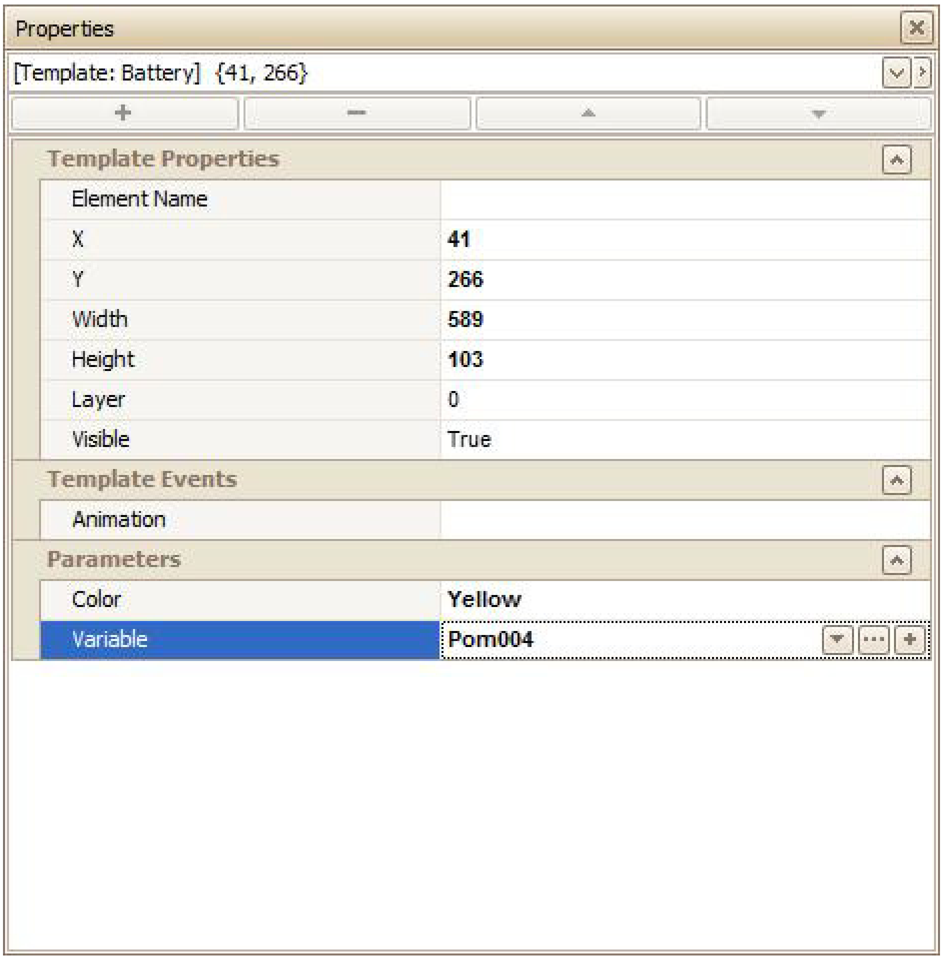
The template parameters are defined at the stage of the diagram definition. The definition consists of the parameter name and its default value. The default value is used when editing the template, and also when the other parameter values are not entered at the template embedment place. On completion of the parameter defining process, the parameter may be called from the template object properties with the suffix notation *%name* or by calling the *Parameter* function.



Basic Properties	
Diagram Name	**Battery**
Description	
Background Color	White
Background Picture	
Background Picture Fill Mode	Stretch
Fixed Width	**658**
Fixed Height	**113**
Fixed Sizes	True
Allow zooming	False
Events	
Diagram Opened	
Diagram Closed	
Parameters	
⊟ Color	
Default Value	Red
⊟ Variable	
Default Value	Battery01

Fig. Template Basic Properties.

In the place of the template embedment correct values of the properties should be entered.



Properties

[Template: Battery] {41, 266}

Template Properties

Element Name	
X	**41**
Y	**266**
Width	**589**
Height	**103**
Layer	0
Visible	True

Template Events

Animation	

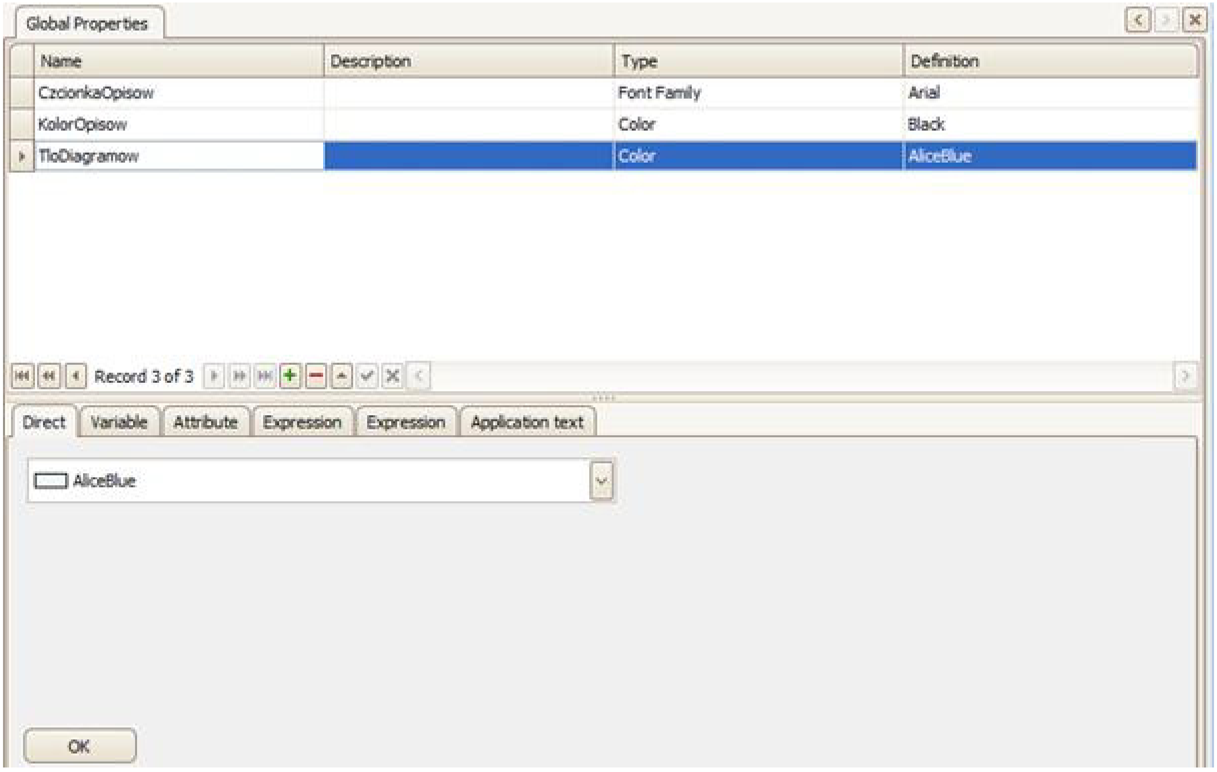
Parameters

Color	**Yellow**
Variable	**Pom004**

Fig. Template Properties.

8 Global Properties

The global properties are centrally-defined properties not associated with any particular component of the application. The application designer may create multilinks to the global property in the definitions of various components, e.g. in the object properties. This facilitates the application appearance/behaviour modification without changing the definition of properties of many subcomponents. The global properties are defined in the *Global Properties* panel.



Name	Description	Type	Definition
CzcionkaOpisow		Font Family	Arial
KolorOpisow		Color	Black
TloDiagramow		Color	AliceBlue

Record 3 of 3

AliceBlue

Fig. Global Properties Panel.

The global property is defined by its name, optional description, property type and value definition. The type is only of secondary importance and is used to select the appropriate editor of the direct mode. The actual interpretation of the property values definition depends on the location of its use. If an example property *Text Color* is used in the *Text* object in the *Color* property, it will change the color of the text displayed. If the *Text Color* property is used in the *Text* property, the *AliceBlue* text will be displayed.

References to the global property are executed with the *Property* function or with the prefix notation *!property_name*.

The definition of the property values is usually entered directly. Other methods may be used, e.g references to the variable attributes or expressions. In these cases the calculation of the property values is carried out at the location and context of its use.

Example:

The *Limit* property is defined as:

= Attribute (LimitHi) * 1.1

If the *Limit* property is used in the *Text* object in the *Text* property, by the **!L*imit*** reference, the high limit value of the object main variable multiplied by 1.1 will be displayed.

It should be noted that in this case, the use of the *=Property(Limit)* expression is not equivalent. The *Limit* property value would be displayed in the text form of *Attribute(LimitHi) * 1.1*.. The correct use has the following form: ***=Evaluate (Property (Limit))***.