The module <VCAEngine> of subsystems "User Interfaces"

Module:	VCAEngine
Name:	Visual control area engine
Type:	User Interfaces
Source:	ui_VCAEngine.so
Version:	1.3.0
Author:	Roman Savochenko
Translated:	Maxim Lysenko
Description:	The main visual control area engine.
License:	GPL

Contents table

Introduction

VCAEngine module provides visual control area engine (VCA) in OpenSCADA system. Module itself does not implement the visualization of the VCA, and contains data in accordance with the ideology of «model/data — Interface». Data visualization of that module is implemented by the visualization modules of VCA, such as Vision and WebVision.

Visual control area (VCA) is an integral part of the SCADA system. It applies to the client stations with a view to providing accessible information about the object and to for the the issuance of the control actions to the object. In various practical situations and conditions the VCA, based on different principles of visualization may by applied. For example, this may be the library of widgets QT, GTK+, WxWidgets or hypertext mechanisms based technologies HTML, XHTML, XML, CSS, and JavaScript, or third-party applications of visualization, realized in various programming languages Java, Python, etc. Any of these principles has its advantages and disadvantages, the combination of which could become an insurmountable obstacle to the use of VCA in a practical case. For example, technologies like the QT library can create highly-reactive VCA, which will undoubtedly important for the operator station for control of technological processes (TP). However, the need for installation of that client software in some cases may make using of it impossible. On the other hand, Web-technology does not require installation on client systems and is extremely multi-platform (it is enough to create a link to the Web-server at any Web-browser) that is most important for various engineering and administrative stations, but the responsiveness and reliability of such interfaces is lower that actually eliminates the using of them at the operator of the TP stations.

OpenSCADA system has extremely flexible architecture that allows you to create external interfaces, including user and in any manner and for any taste. For example, the system configuration OpenSCADA as now available as by means of the QT library, and also the Web-based.

At the same time creation of an independent implementation of the VCA in different basis may cause the inability to use the configuration of one VCA into another one. That is inconvenient and limited from the user side, as well as costly in terms of implementation and follow-up support. In order to avoid these problems, as well as to create as soon as possible the full spectrum of different types of VCA project of the creation of the conception of the VCA is established. The result of this project — the engine module(data model) of the VCA, as well as direct visualization modules Vision and WebVision.

1. Purpose

This module of the engine (data model) of the VCA is aimed to create the logical structure of the VCA and the execution of sessions of individual instances of the VCA projects. Also, the module provides all the necessary data to the final visualizers of the VCA, both through local mechanisms of interaction of OpenSCADA, and through the management Interface of OpenSCADA for remote access.

The final version of the VCA module, built on the basis of this module, will provide:
- three levels of complexity in the formation of visualization interface which let organically to develop and apply the tools of the methodology from simple to complex:
  - formation from the template frames through the appointment of the dynamics (without the graphical configuration);
  - graphical formation of new frames through the use of already made visualization elements from the library (mimic panel);
  - formation of new frames, template frames of the visualization elements in the libraries.
- building of the visualization interfaces of various complexity, ranging from simple flat interfaces of the monitoring and finishing with the full-fledged hierarchical interface used in SCADA systems;
- providing of the different ways of formation and configuration of the user interface, based on different graphical interfaces (QT, Web, Java ...) and also through the standard management interface of OpenSCADA system;
- change of dynamics in the process of execution;

•   building of the new template frames on the user level and the formation of the frames libraries, specialized for the area of application (eg the inclusion of frames of parameters, graphs and other items linking them to each other) in accordance with the theory of secondary using and accumulation;

•   building of the new user elements of the visualization and the formation of the libraries of frames,specialized for the area of application in accordance with the theory of secondary using and accumulation;

•   description of the logic of new template frames and user visualization elements as with the simple links, and also with the laconic, a full-fledged programming language;

•   the possibility of the inclusion of the functions(or frames of computing of the functions) of the object model of OpenSCADA to the user elements of the visualization, actually linking the presentation of the algorithm of computing (for example, by visualizing the library of models of devices of TP for following visual modeling TP);

•   separation of user interfaces and interfaces of visualization of data provides building the user interface in a single environment, and performance of it in many others (QT, Web, Java ...);

•   the possibility to connect to the performing interface for monitoring and corrective actions (for example, while operator training and control in real time for his actions);

•   Visual building of various schemes with the superposition of the logical links and the subsequent centralized execution in the background (visual construction and performance of mathematical models, logic circuits, relay circuits and other proceedings);

•   providing of the the functions of the object API to the OpenSCADA system, it can be used to control the properties of the visualization interface from the user procedures;

•   building of the servers of frames, of elements of the visualization and of the project of the interfaces of the visualization with the possibility to serve the great number of the client connections;

•   simple organization of client stations in different basis (QT, Web, Java ...) with the connection to the central server;

•   full mechanism of separation of privileges between the users which allows to create and execute projects with the various rights of access to its components;

•   adaptive formation of alarms and notifications, with the support of different ways of notification;

•   support of the user formation of the palettes and font preferences for the visualization of the interface;

•   support of the user formation of maps of the events under the various items of equipment management and user preferences;

•   support for user profiles, allowing to define various properties of the visualization interface (colors, font characteristics, the preferred maps of events);

•   flexible storage and distribution of libraries of widgets, frames, and projects of the visualization interfaces in the databases, supported by OpenSCADA; actually users need only to register the database with data.

2. The configuration and the formation of interfaces of the VCA

Module itself does not contain a visual tool for creating interfaces of VCA, based on one of the one of the mechanisms. Such tools can be given by the final visualization modules of the VCA, for example the module Vision of such a tool is provided.

Although the visual tool for the formation of the VCA the module doesn't provide the interface, implemented on the basis of the management interface of the OpenSCADA, to manage the logical structure is provided, and thus it is available for use in any system configurator of the OpenSCADA. Dialogues of this interface are considered further in the context of the architecture of the module and its data.

3. Architecture

Any VCA can operate in two modes — the development and execution. In the development mode the VCA interface and its components are formed, the mechanisms of interaction are identified. While the execution it is carried out the formation of VCA interface and epy interaction, based on the developed VCA, with the final user is made.

VCA interface is formed of the frames, each of which, in its turn, formed from elements of the primitives, or user interface elements. In doing so, the user interface elements are also formed from the primitives or other user elements. That gives us a hierarchy and reuse of already developed components.

Frames and user elements are placed in the libraries of widgets. The projects of the interfaces of the final visualization of the VCA are formed from these libraries' elements. Based on these projects the visualization sessions are formed.

The structure of VCA is shown in Fig. 3.
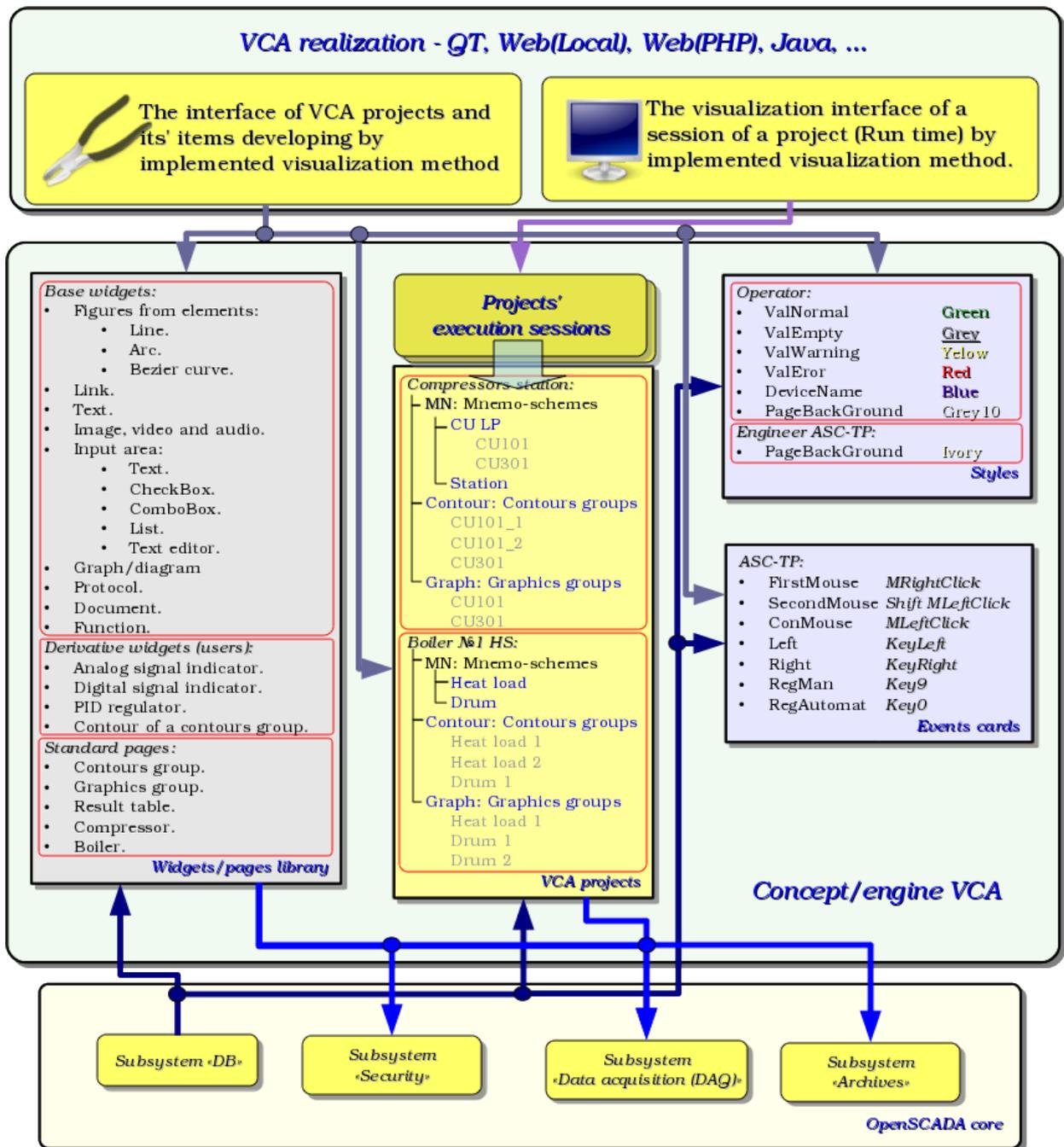
Fig.3 Generalized structure of the VCA.

This architecture of the VCA allows the support of three levels of complexity of the developing process of the management interface:

• Forming of the VC interface (visualization and control) using the library of template frames by placing the templates of the frames in the project and by the assignment of the dynamics.

• In addition to the first level the own creation of frames based on the library of derivatives and basic widgets is to be done. Perhaps as a direct appointment of the dynamics in the widget, and the subsequent appointment of it in the project.

• In addition to the second level is performed the independently forming of derivatives widgets, new template frames and also the frames with the use of mechanism of describing the logic of interaction and handling of events in one of the languages of a user programming of OpenSCADA system.

3.1. Frames and elements of visualization (widgets)

Frame is the window which directly provides information to the user in a graphical or text form. The group of interconnected frames creates whole user interface of VC.

The contents of the frame is forming from the elements of visualization (widgets). Widgets may be the basic primitives (different flat shapes, text, trend, etc.) and derivatives (formed from the basic or other derivatives of widgets). All the widgets are grouped into the libraries. In the process, you can build your own library of derivative widgets.

Actually the frame is also a widget that is used as a final element of visualization. This means that the widget libraries can store the blanks of frames and the templates of the resulting pages of the user interface.

Frames and widgets are passive elements that do not normally contain links to the dynamics and other frames, but only provide information about the properties of the widget and the nature of the dynamics (configuration), connected to the properties of the frame. Activated frames, ie containing links to the dynamics and active connections, form the user interface and are stored in the projects. In some cases, it is possible the direct appointment of the dynamics in the blanks of frames.

Derivative frames/widgets can contain other widgets (attached), which can be glued (associated) with the logic of one another by one of the languages of programming available in the OpenSCADA system (Fig.3.1.1).
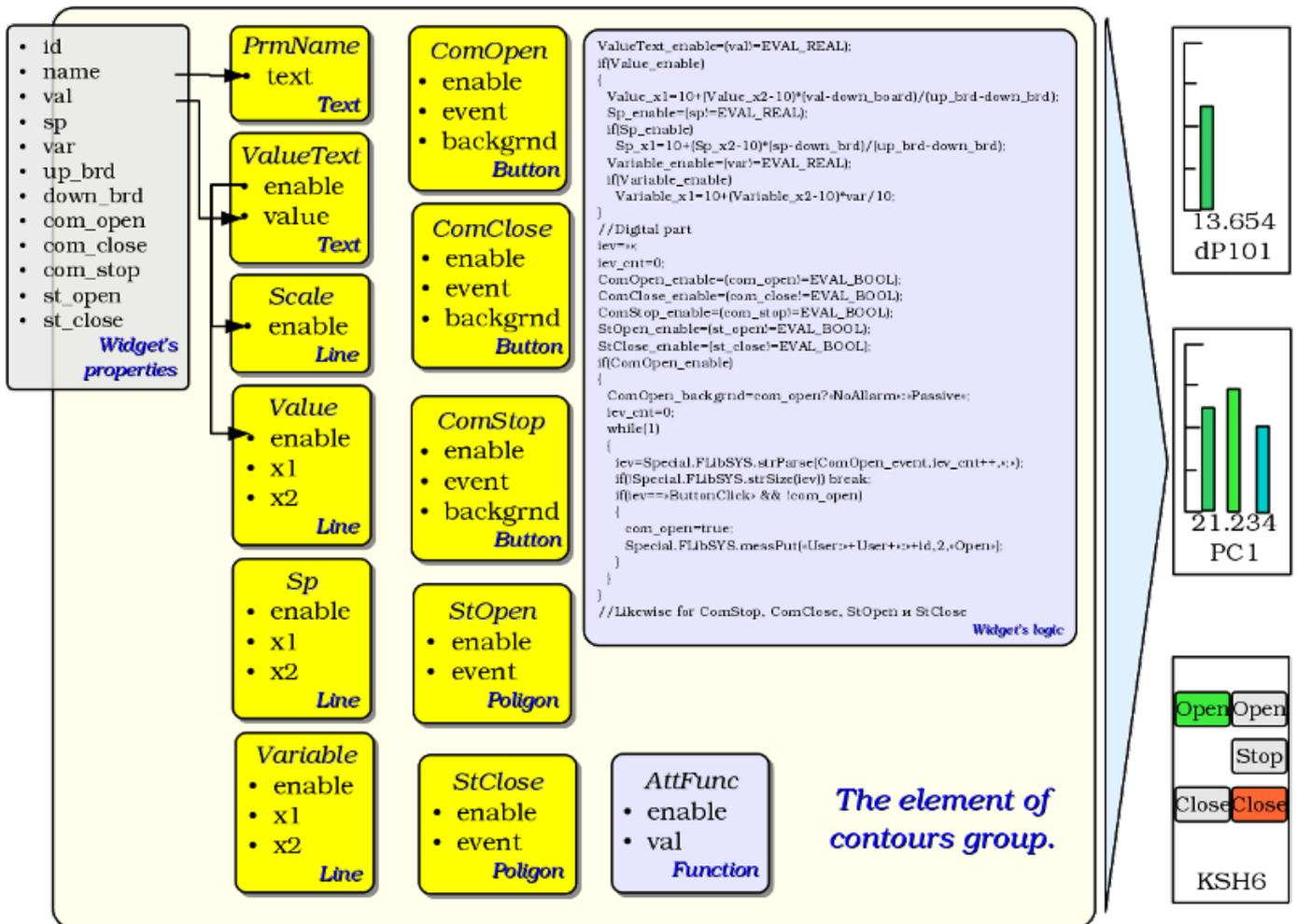
Fig.3.1.1 Example of the structure of the derived widget.

The widget is an element, by means of which it is provided:
- visualization of operational and archive information about TP;
- alarm about a violation of conduction of TP;
- switching between the frames of TP;
- management of technological equipment and the parameters of conduction of TP.

Tuning and linkage of the widgets is done through their properties. Parent widget and the widgets it contains, can be complemented by user properties. Then the user and static attributes are associated with the properties of embedded widget by internal logic. To show the dynamics (ie, current and archived data), properties of widgets are dynamized, that is linked with the attributes of the parameters of OpenSCADA or properties of other widgets. Using to link of the nested widgets by means of the internal logic with the available programming language of the OpenSCADA system eliminates the question of the implementation of complex logic of visualization, thus providing high flexibility. Practically, you can create fully dynamized frames with complex interactions at the level of the user.

Between widgets at different levels of hierarchy complex inheritance relations are arranged, which are defined by the possibility of using some widgets by other ones, beginning with the library widget, and finishing with the widget to the session. To clarify these features of the interaction in Fig. 3.1.2 comprehensive map of «uses» inheritance is shown.

Fig.3.1.2 Map of «uses» inheritance of the the components of conception/engine

At the session level widget contains a frame of values of calculation procedure. This frame is initiated and used in the case of presence of the calculation procedure. At the time of the initialization the list of parameters of the procedure is created and a compilation of procedure is performed with these parameters in the module, implementing the selected programming language and encoded with the full name of the widget. A compiled function is connected to the frame of values of the calculation procedure. Further the calculation is performed with the frequency of session.

Calculation and processing of the widget as a whole runs in the following sequence:
- the events, which are available at the time of computation, are selected from the attribute "event" of the widget;
- events are loaded into the parameter "event" of the frame of computation;
- values of the input connections are loaded in the frame of calculation;
- values of special variables are loaded in the computation frame (f_frq, f_start and f_stop);
- values of selected parameters of the widget are loaded in the frame of computation;
- computation;
- uploading of the computation frame values into the selected parameters of the widget;
- uploading of the event from the parameter "event" of the computation frame;
- processing th events and transfer the unprocessed events at the level above.

3.2. Project

Direct configuration and properties of the final visualization interface are contained in the project of the visualization interface of the VCA. It may be created a lot of projects of the visualization interfaces.

Each project includes frames from the libraries of the frames/widgets. A frame provides a tool for the dynamics to the properties described therein. All properties of the frame may be associated with dynamics or authorized by the constants, and can act as a template for the formation of derivative pages. In fact, each frame may contain multiple pages with their own dynamics. This mechanism allows to extremely simplify the process of creating the same type of the frames by the ACS-TP engineer or by the user of OpenSCADA for easy monitoring. An example of such one-type frames may be: groups of contours, groups of graphs, reports and various tables. Mnemonic schemes of technological processes rarely come under this scheme and will be formed directly in the description of the frame.

To provide the possibility of creation of a complex hierarchical interfaces of VC the frames, placed into the project, can be grouped by name in the hierarchical form and by the appropriate visualization in the form of a tree. In addition to this a mechanism of associative description of the calling of the frames through regular expressions is provided.

Example of hierarchical representations of components of the project of the classical interface of VC of the technological process with the description of standard expressions is given in Fig. 3.2.

```
SO1: Low pressure stage
 MN: Mnemo-schemes
   KU101                      SO1.MN.KU101
   KU201                      SO1.MN.KU201
   KU301                      SO1.MN.KU301
 Cont: Contours group         SO1.Cont
   KU101_1                    SO1.Cont.KU101_1
   KU101_2                    SO1.Cont.KU101_2
   KU201                      SO1.Cont.KU201
   KU301                      SO1.Cont.KU301
 Graph: Graphics group        SO1.Graph
   KU101                      SO1.Graph.KU101
   KU201_1                    SO1.Graph.KU201_1
   KU201_2                    SO1.Graph.KU201_2
   KU301                      SO1.Graph.KU301
OC2: High pressure stage
 MN: Mnemo-schemes
   KU102                      SO2.MN.KU102
   KU202                      SO2.MN.KU202
   KU302                      SO2.MN.KU302
 Cont: Contours group         SO2.Cont
   KU102_1                    SO2.Cont.KU102_1
   KU102_2                    SO2.Cont.KU102_2
   KU202                      SO2.Cont.KU202
   KU302                      SO2.Cont.KU302
 Graph: Graphics group        SO2.Graph
   KU102                      SO2.Graph.KU102
   KU202_1                    SO2.Graph.KU202_1
   KU202_2                    SO2.Graph.KU202_2
   KU302                      SO2.Graph.KU302

Container    — Widgets' virtual container
Cont         — Real widget
Page         — Paget of real widget

                Call examples
  •   Page type independent from signal object (SO):
        Current:     SO1.MN.KU101
        Call:        *.Cont
        Result:      SO1.Cont.KU101_1
  •   Second SO with type page saving:
        Current:     SO1.MN.KU101
        Call:        SO2.*
        Result:      SO2.MN.KU102
```

Fig.3.2 Hierarchical view of components of the project of classical interface of VC of the technological process.

In accordance with the Fig.3.1.2 objects of the session of the project inherit from an abstract object "Widget" and use the appropriate objects of the project. Thus, the session ("Session") uses the project ("Project") and forms expand tree on its basis. Project page "Page" is directly used by the session page "SessPage". The remaining objects ("SessWdg") are deployed in accordance with the hierarchy of page elements (Fig.3.1.2).

In addition to the standard properties of an abstract widget ("Widget") elements of the pages of session themselves get the following properties: storage of the frame of values of computational procedure, calculation of the procedures and mechanism for processing of the events. Pages of the session, in addition, contain a container of the following by the hierarchy pages. The session generally is computed with the frequency and in the consistency:
   •   «Page of the top level» -> «Page of the lower level»
   •   «Widget of the lower level» -> «Widget of the top level»

This policy allows you to traverse the pages in accordance with the hierarchy, and to rise on the top during the one iteration for the widget events.

The session supports the special properties of pages:
- *Container* — page is a container for the underlying pages;
- *Template* — page is a template for the underlying pages;
- *Empty* — empty, inactive, page; this feature is used in conjunction with the property *Container* for logical containers organization.

Based on these properties the following types of pages are realized:
- *Standard* — The standard page (none property is set). It is the full final page.
- *Container* — Full page with the feature of the container (*Container*).
- *Logical container* — Logical container is actually not a page (*Container|Empty*). Performs property of the intermediate and bunching element in the tree of pages.
- *Template* — Template page (*Template*). Pure template page is used to describe the common properties and hipping them in privately order in nested pages.
- *Container and template* — The template and a container page (*Template|Container*). Combines the functions of the template and the container.

Switching, opening, substitution and navigation through the pages is based on processing of the events by the scenario in the attribute of the active widget "evProc". The scenario of this attribute is stored as a list of commands with the syntax:**<event>:<evSrc>:<com>:<prm>**. Where:
- *event* — the expected event;
- *evSrc* — the path of the nested widget-source of the event;
- *com* — session command;
- *prm* — parameter of the command.

The following commands are implemented:
- *open* — Opening page. Page to open is specified in the parameter <prm> both: in direct way and as a template (example: /pg_so/1/*/*).
- *next* — The opening of the next page. Page to open is specified in the parameter <prm> as a template (example: /pg_so/*/*/$).
- *prev* — Opening of the previous page. Page to open is specified in the parameter <prm> as a template (example: /pg_so/*/*/$).

Special characters of the template are deciphered as follows:
- *pg_so* — direct name of the desired page with the prefix. Requires the compulsory accordance and is used to identify the last open page;
- *1* — name of a new page in a general way, without a prefix. It is ignored when it detects a previous open pages;
- *** — the page is taken from the name of a previous opened page or the first available page is substituted, if the previous opened page is missing;
- *$* — points the place of the opened page relative to which you are to go to the next or to the previous one.

To understand the mechanism of the templates lets cite some real examples:
- *Changing the signal object:*
    Command: open:/pg_so/2/*/*
    In was: /pg_so/pg_1/pg_mn/pg_1
    It is: /pg_so/pg_2/pg_mn/pg_1
- *Switching of the type:*
    Command: open:/pg_so/*/gkadr/*
    It was: /pg_so/pg_1/pg_mn/pg_1
    It is: /pg_so/pg_1/pg_gkadr/pg_1
- *Next/previous page of the type:*
    Command: next:/pg_so/*/*/$
    It was: /pg_so/pg_1/pg_mn/pg_1
    It is: /pg_so/pg_1/pg_mn/pg_2

As an example lets cite the scenario of operation of the main page of the user interface:

```
ws_BtPress:/prev:prev:/pg_so/*/*/$
ws_BtPress:/next:next:/pg_so/*/*/$
ws_BtPress:/go_mn:open:/pg_so/*/mn/*
ws_BtPress:/go_graph:open:/pg_so/*/ggraph/*
ws_BtPress:/go_cadr:open:/pg_so/*/gcadr/*
ws_BtPress:/go_view:open:/pg_so/*/gview/*
ws_BtPress:/go_doc:open:/pg_so/*/doc/*
ws_BtPress:/go_resg:open:/pg_so/rg/rg/*
ws_BtPress:/so1:open:/pg_so/1/*/*
ws_BtPress:/so2:open:/pg_so/2/*/*
ws_BtPress:/so3:open:/pg_so/3/*/*
ws_BtPress:/so4:open:/pg_so/4/*/*
ws_BtPress:/so5:open:/pg_so/5/*/*
ws_BtPress:/so6:open:/pg_so/6/*/*
ws_BtPress:/so7:open:/pg_so/7/*/*
ws_BtPress:/so8:open:/pg_so/8/*/*
ws_BtPress:/so9:open:/pg_so/9/*/*
ws_BtPress:*:open:/pg_control/pg_terminator
```

In conjunction with the mechanism, above described, on the side of the visualization (RunTime) there is the logic regulating how to open the pages. The logic is built on the following attributes of the basic element "Box":

• *pgOpen* — Sign "The page is opened".
• *pgNoOpenProc* — Sign "Perform the page, even if it is not opened".
• *pgOpenSrc* — Contains the address of the widget or of the page which has opened the current. In the case of the nested container widget here it is contained the address of the included page. To open the pages from the script here it is enough to indicate the address of the widget-source of the opening.
• *pgGrp* — Group of pages. Used for conjunction of the containers of the pages with the pages in accordance with the general group.

The logic of the method of the opening the pages work in the following way:

• if the page has the group "main" or coincides with a group of the page in the main window or there is no page on the main window, then open the page in the main window;
• if the page has a group which coincides with the group one of the containers of the current page, then open it in the container;
• if the source of the opening of the page coincides with the current page, then open it as an additional window over the current page;
• transmit a call for request for the opening to the additional windows with the processing in each of the first three paragraphs;
• if any one of the relative windows doesn't open a new page, then open it as a related window of the main window.

3.3. Styles

We know that people can have individual characteristics in the perception of graphical information. If these features are not taken into account, it is possible to obtain the rejection and seizure of the user to the interface of VC. This rejection and seizure can lead to fatal errors in the management of TP, as well as traumatize the human by the continuous work with such interface. In SCADA systems the agreements are adopted, which regulate the requirements for creating a unified interface of VC normally perceived by most people. This is actually eliminates the features of people with some deviations.

In order to take this into account and allow centralized and easy to change the visual properties of the interface module is scheduled to implement a theme manager of the visualization interface.

User can create many themes, each of which will keep the color, font and other properties of the elements of the frame. Simple changing of the theme will allow you to change the interface of VC, and the

possibility of appointing an individual theme in the user's profile allows to take into account his individual characteristics.

To realize this opportunity, when you create a frame, it is necessary for the properties of color, font and others set the «Config» (of the table if the «process» tab) in the value of «From style» (Fig. 3.7). And in the parameter «Config template» to specify the identifier of the style field. Further, this field will automatically appear in the Style Manager and will be there to change. Style Manager is available on the project configuration page in the tab «Styles» (Fig. 3.3). On this tab you can create new styles, delete old ones, change the field of the style and delete unnecessary.
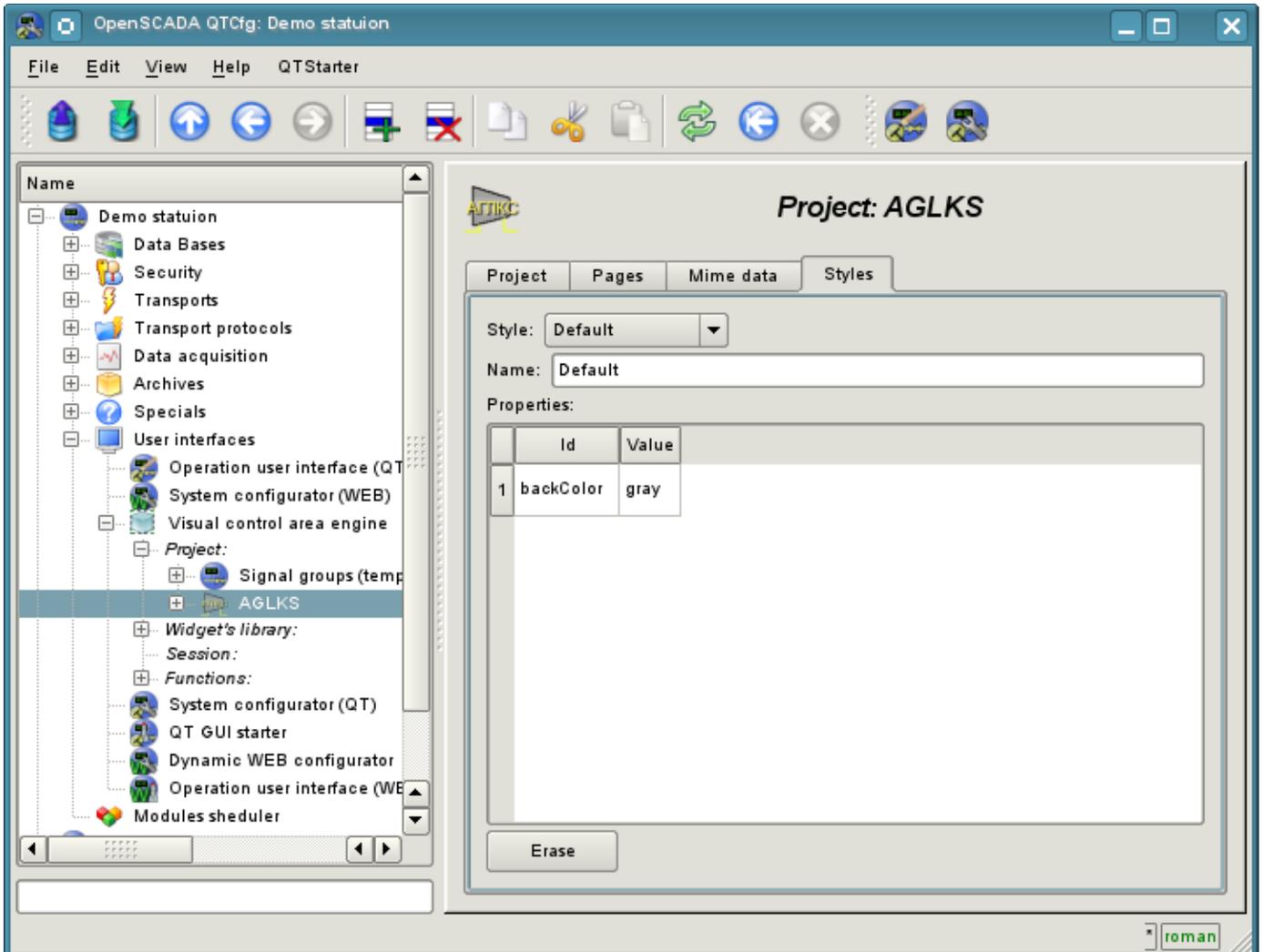
Fig. 3.3 "Styles" tab of the configuration page of the project.

In general the styles are available from the project level. At the level of libraries of widgets you can only define styles fields of widgets. At the project level, at the choice of style it is started the work with styles, which includes access to the fields of styles instead of direct attribute values. In fact, this means that when reading or writing a widget attribute these operations will be carried out with the corresponding field of the chosen style.

When you run the project execution it will be used the set in the project style. Subsequently, the user can select a style from the list of available ones. The user's style will be saved and used next time you run the project.

3.4. Events, their processing and the events' maps

Given the range of tasks for which the OpenSCADA system may be used, it is necessary to provide a tool for management of interactive user events. This is due to the fact that in dealing with individual tasks of embedded systems, input and control devices can greatly vary. But it is enough to look at the regular office keyboard and notebook one, that would remove any doubt about the necessity for the manager of events.

Event manager must work using the maps of events. Map of the events — is the list of named events, indicating their origin. The origin of the events can be a keyboard, mouse, paddle, joystick, etc. If you have any event manager of the events is looking for it in the active map and compares with the name of the event. A comparison name of the event is placed in the queue for processing. Widgets in this case must process the given queue of events.

The active map of events is specified in the profile of each user or is set by default.

In general, four types of events are provided:
  • events of the images of VCA (prefix: ws_), for example, pressing of the button event — *ws_BtPress*;
  • keyboard events (prefix: key_) — all events from mouse and keyboard in the form of — *key_presAlt1*;
  • user events (prefix: usr_) are generated by the user in the procedures of the calculation of widgets;
  • mapping of the event (prefix: map_) — events from the map of events.

Event itself represents little information, especially if its processing occurs at higher level. For the unequivocal identification of the event and its source in the whole the event is recorded as follows: "ws_BtPress:/curtime". Where:
  ws_BtPress — event;
  /curtime — the path to the child element that has generated the event.

Table 3.4 provides a list of standard events, the support of which should be provided in visualizers of VCA.

Table 3.4. Standard events

Id	Description
Keyboard events: **key_[pres|rels][Ctrl|Alt|Shift]{Key}**	
*SC#3b	Scan code of the kye.
*#2cd5	Code of the unnamed key.
*Esc	"Esc".
*BackSpace	Removing of the previous character — "<--".
*Return, *Enter	Enter — "Enter".
*Insert	Insertion — "Insert".
*Delete	Deleting — "Delete".
*Pause	Pause — "Pause".
*Print	Print of the screen — "Print Screen".
*Home	Home — "Home".
*End	End — "End".
*Left	Left — "<-".
*Up	Up — '^'.

Id	Description
*Right	Right — "->".
*Down	Down — '\/'.
*PageUp	Page up — "PageUp".
*PageDown	Page down — "PageDown".
*F1 - *F35	Function key from "F1" to "F35".
*Space	Space — ' '.
*Apostrophe	Apostrophe — '`'.
*Asterisk	Asterisk on an additional field of the keyboard — '*'.
*Plus	Plus on an additional field of the keyboard — '+'.
*Comma	Comma — ','.
*Minus	Minus — '-'.
*Period	Period — '.'.
*Slash	Slash — '\'.
*0 - *9	Number from '0' to '9'.
*Semicolon	Semicolon — ';'.
*Equal	Equal — '='.
*A - *Z	Keys of Latin alphabet from 'A' to 'Z'.
*BracketLeft	Left square bracket - '['.
*BackSlash	Backslash — '/'.
*BracketRight	Right square bracket — ']'.
*QuoteLeft	Left quote — '''.
Keyboard focus events.	
ws_FocusIn	Focus is obtained by a widget.
ws_FocusOut	Focus is lost by a widget.
Mouse events:	
key_mouse[Pres|Rels][Left|Right|Midle]	Pressed/released the mouse button.
key_mouseDblClick	Double-click the left mouse button.
Events handshake on the side of the visualizer.	
ws_alarmLev	Acknowledgment of all violations by all means notice.
ws_alarmLight	Acknowledgment of all violations of the notification by flashing/light.
ws_alarmAlarm	Acknowledgment of all violations of the notification buzzer.
ws_alarmSound	Acknowledgment of all violations of the notification sound/speech.
Events of the primitive of elemental figure **ElFigure**:	
ws_Fig[Left|Right|Midle|DblClick]	Activating of the figures (fills) by the mouse button.

Id	Description
ws_Fig{n}[Left|Right|Midle|DblClick]	Activating of the figure (fill) {n} by the mouse button.
*Events of the primitive of form elements **FormEl**:*	
ws_LnAccept	A new value in the input line is set.
ws_TxtAccept	The value of the the text editor is changed.
ws_ChkChange	The state of the flag is changed.
ws_BtPress	The button is pressed.
ws_BtRelease	The button is released.
ws_BtToggleChange	Button toggle is changed.
ws_CombChange	The value of the combo box is changed.
ws_ListChange	The current list item is changed.
ws_SliderChange	Changing of the the slider position.
*Events of the primitive of media content **Media**:*	
ws_MapAct{n}[Left|Right|Midle]	Media area with the number {n} is activated by the mouse button.
ws_MediaFinished	Media-stream finish play.

Events are the main mechanism of notification and is actively used for user interaction. For the event processing there are two mechanisms: the script used to control the opening of the pages and the computational procedure of the widget.

The mechanism "Scripts for the control the opening of pages" based on the basic attribute of the widget "evProc" and is described in detail in section 3.2.

The mechanism "Processing the event with the help of computational procedure of the widget" is based on the attribute "event" and the user procedure of calculating written with the help of the language of the user programming of OpenSCADA. Events, in process of receipt, are accumulated in the attribute "event" till the moment of call of computational procedure. Computational procedure is called with the specified frequency of calculating the widget and receives a value for the attribute "event" as the list of events. In the calculation procedure the user can: analyze, process and delete the processed events from the list, and add to the list new events. The remaining, after the procedure execution, events are analyzed for compliance with the conditions of the call by means of script of the first mechanism, after which the remaining events are transmitted to the upper by the hierarchy widget to be processed by it, with the correction of the path of events in accordance with the hierarchy of the penetration of the event.

The contents of the attribute "event" is a list of events in the format **<event>:<evSrc>**, with the event on the separate line. Here is an example of processing events in the Java-like programming language of the OpenSCADA:

```
using Special.FLibSYS;
ev_rez = "";
off = 0;
while(true)
{
  sval = strParse(event,0,"\n",off);
  if( sval == "" ) break;
  else if( sval == "ws_BtPress:/cvt_light" ) alarmSt = 0x1000001;
  else if( sval == "ws_BtPress:/cvt_alarm" ) alarmSt = 0x1000002;
  else if( sval == "ws_BtPress:/cvt_sound" ) alarmSt = 0x1000004;
  else ev_rez+=sval+"\n";
}
event=ev_rez;
```

3.5. Signaling (Alarms)

An important element of any visualization interface is the user notification about the violation — alarm. To simplify the perception, but also in mind the close connectivity of visualization and notification (typically notification is amplified with the visualization) it is decided to integrate the interface of a notification in the visualization interface. To do this, all the widget provides two additional attributes (of the session level): "alarm" and "alarmSt". Attribute "alarm" is used to form the signal by the widget, according to his logic, and attribute "alarmSt" is used to control the signaling fact of the branch of the tree of the session of the project.

Attribute "alarm" is a line and has the following format: *{lev|categ|message|type|tp_arg}* Where:
   - *lev* — signaling (alarm) level; number from 0 to 255;
   - *categ* — alarm category; parameter of the acquisition subsystem, object, path, or a combination;
   - *message* — signaling (alarm) message;
   - *type* — type of notification (visual, speech, and beep) is formed as a the integer number, which contains the flags of notification methods:
      - *0x01* — visual;
      - *0x02* — beep, is frequently made through the PC-speaker;
      - *0x04* — sound signal from the sound file or the speech synthesis, and if in the <tp_arg> the name of the resource of the sound file is specified, then play it, or in other case the speech synthesis from the text specified in <message> is made.
   - *tp_arg* — argument of the type; it is used in the case of the audible signal to indicate the resource of the sound alarm (file of the sound format).

Attribute "alarmSt" is an integer number that represents the maximum alarm level and the fact of the quittance of the branch of the tree of the session of the project. Format of the number is as follows:
   - first bite (0-255) characterizes the level of the alarm of the branch;
   - the second byte indicates the type of notification (as well as in the attribute "alarm");
   - the third byte indicates the type of notification without quittance (as well as in the attribute "alarm");
   - the first bit of the the fourth byte has a special appointment, setting this bit is the fact of the quittance of the notification referred to the first byte.

Alarm formation and receipt of it by the visualizer.

Alarm formation is performed by the widget by setting its own attribute "alarm" in appropriate way and in accordance with it the attribute "alarmSt" of current and the parent widget is set. Visualizers receive notification of the alarm using a standard mechanism for notifications of the changes of attributes of widgets.

This mechanism provides the ability to build the signaling (alarm) interfaces at the level of subsystems "data acquisition", or directly at the level of representation.

Taking into account that the processing of conditions of the signaling is made in the widgets, the page containing the objects of signaling should be performed in the background, regardless of their openness to the moment. This is done by setting a flag of the background execution of the page.

Although the mechanism of signaling is built in the visualization area the possibility of formation of visual signaling elements remains, for example by creating the page that will never be opened.

Quittance

Quittance is done by specifying the root of the branch of the widgets and the types of notification. This allows to make quittance on the side of visualizer both as by groups, for example by the signaling objects as well as individually by the objects. It is possible to independently quit different types of alarms. Setting of the quittance is made by the simple modification of the attribute "alarmSt".

Example of the script to work with the signals is listed below:

```
//Allocation of the existence of alarms of different ways of notification
cvt_light_en = alarmSt&0x100;
cvt_alarm_en = alarmSt&0x200;
cvt_sound_en = alarmSt&0x400;
//Allocation of the existence of not quited alarms of different ways notification
cvt_light_active = alarmSt&0x10000;
cvt_alarm_active = alarmSt&0x20000;
cvt_sound_active = alarmSt&0x40000;
//Processing of the event  buttons of quittance and quittance of different ways of
notification
ev_rez = "";
off = 0;
while(true)
{
  sval = strParse(event,0,"\n",off);
  if( sval == "" ) break;
  else if( sval == "ws_BtPress:/cvt_light" ) alarmSt = 0x1000001;
  else if( sval == "ws_BtPress:/cvt_alarm" ) alarmSt = 0x1000002;
  else if( sval == "ws_BtPress:/cvt_sound" ) alarmSt = 0x1000004;
  else ev_rez+=sval+"\n";
}
event=ev_rez;
```

3.6. Rights management

For the separation of access to the interface of VC and its components every widget contains information about the owner, about its group and access rights. Access rights are recorded as is the convention in the OpenSCADA system, in the form of a triad: <user><group><rest> where each element consists of three attributes of access. For the elements of the VCA the following interpretation is taken:
- $'r'$ — the right to review the widget;
- $'w'$ — the right to control over the widget.

In the development mode a simple scheme of access "root.UI:RWRWR_" is used, which means — all users can open and view the libraries, their components and projects, and all users of group "UI" user interfaces) can edit.

In the performance mode the right described in the components of interface work.

3.7. Linkage with the dynamics

To provide relevant data in the visualization interface the data of subsystems "Data acquisition (DAQ)" must be used. The nature of these data as follows:
1. parameters that contain some number of attributes;
2. attributes of the parameter can provide information of four types: Boolean, Integer, Real and String;
3. attributes of the parameter can have their history (archive);
4. attributes of the parameter can be set to read, write, and with full access.

Considering the first paragraph it is necessary to allow the possibility of the group of destination links. To do this we use the conception of of the logic level.

In accordance with paragraph 2, links provide transparent conversion of connection types and do not require special configuration.

To satisfy the opportunities for access to archives, in accordance with paragraph 3, links make check of the type of the attribute, and in the case of connection to the "Address", the address of linkage is put into the value.

In terms of the VCA, the dynamic links and configuration of the dynamics are the one process, to describe a configuration of which the tab "Processing" of the widgets is provided (Fig.3.7.a). The tab contains a table of configuration of the properties of the attributes of the widget and the text of calculation procedure of the widget.
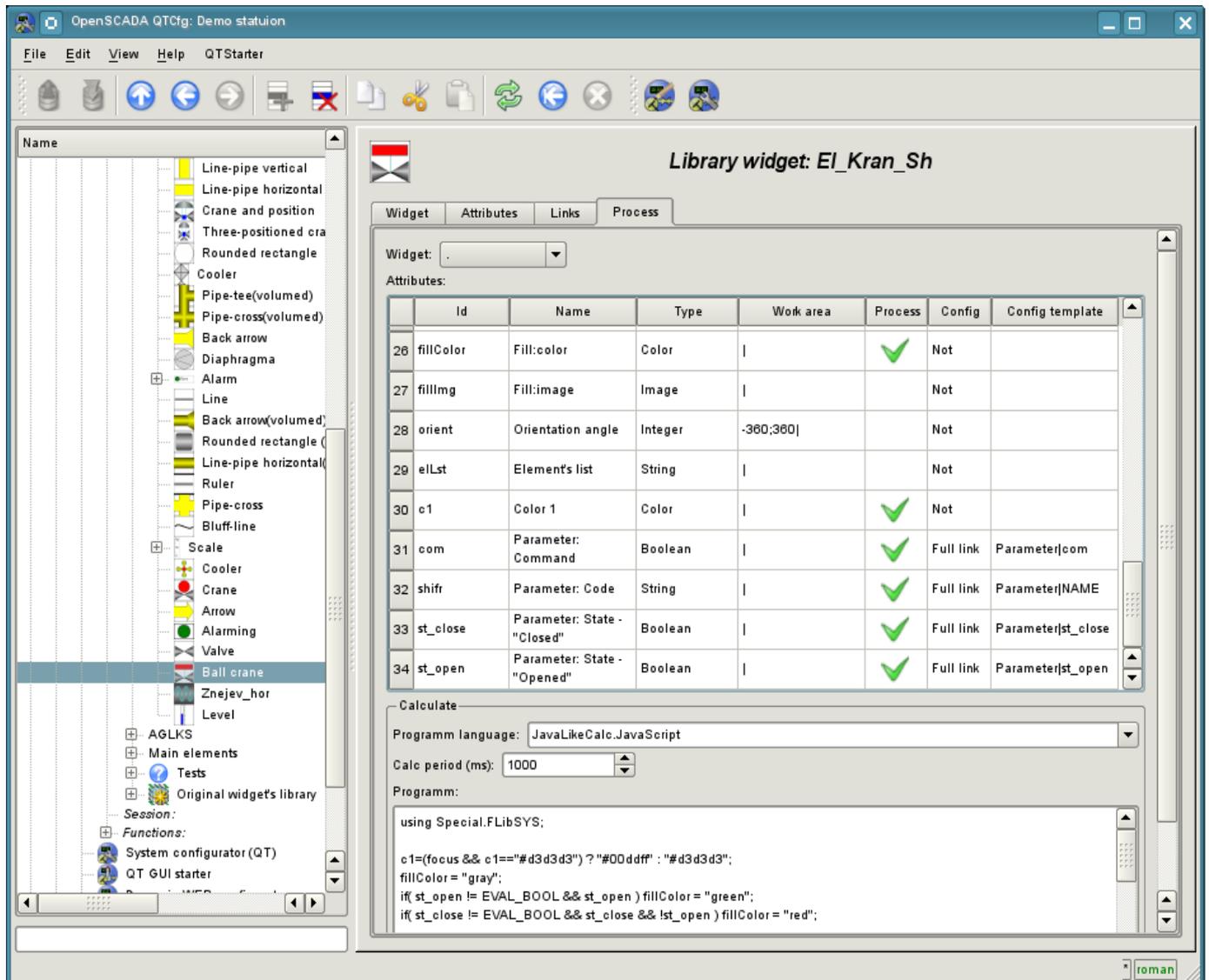
Fig. 3.7.a The tab "Processing" of the configuration page of the widget.

In addition to configuration fields of the attributes the column "Processing" in the table is provided, for selective using of the attributes of the widgets in the computational procedure of the widget, and the columns "Configuration" and "Configuration template", to describe the configuration of links.

Column "Configuration" allows you to specify the linkage type for the attribute of the widget:
- *Constant* — in the tab of widget links the field for indication of a constant appears, for example of the special color or header for the template frames;
- *Input link* — linkage with the dynamics for a read-only;
- *Output link* — linkage with the dynamics just for the record;
- *Full link* — complete linkage with dynamic (read/write).

Column "Configuration template" makes it possible to describe the groups of dynamic attributes. For example it may be different types of parameters of subsystem "DAQ". Furthermore, in the case of correct formation of this field, the mechanism of automatically assign of the attributes with the only indication of

the parameter of subsystem "DAQ" is working, which simplifies and accelerates the configuration process. The value of this column has the following format: **<Parameter>|<identifier>**, where:

- *<Parameter>* — the group of the attribute;
- *<Identifier>* — identifier of the attribute, this value is compared with the attributes of the DAQ parameters with automatic linkage, after the group link indication.

Installation of the links may be of several types, which are determined by the prefix:

- *val:* — Direct download of the value through the links mechanism. For example, link: "val:100" loads in the attribute of the widget the value of the 100. It is often used in the case of absence of end point of the link, in order to direct value indicating.
- *prm:* — Link to the attribute of the parameter or parameter, in general, for a group of attributes, of subsystem "Data acquisition". For example, the link "prm:/LogicLev/experiment/Pi/var" implements the access of the attribute of the widget to the attribute of the parameter of subsystem "Data acquisition". Sign "(+)" at the end of the address signals about successful linking and presence of the target.
- *wdg:* — Link to an attribute of another widget or a widget, in general, for a group of attributes. For example, the link "wdg:/ses_AGLKS/pg_so/pg_1/pg_ggraph/pg_1/a_bordColor" implements the access of the attribute of one widget to the attribute of another one. Supported absolute and relative link's path. Start point of absolute point is root object of module "VCAEngine", then the first item of absolute address is a session or a project identifier. On session side first item is passed then set into a project links there work. For relative links by start point used widget with the link set. The item ".." of parent node is special item of relative links.
- *arh:* — A special type of link is only available for a particular attribute such as "Address," which allows you to connect directly to the archive values ("arh:CPU_load"). It may be useful to specify the archive as a source of data for primitive "Diagram".

Processing of the links occurs at a frequency of calculating the widget in the following order:

- Receiving of the data from input links.
- The implementation of calculating of the script.
- Transmission of the values by the output links.

In the Fig. 3.7.b the tab of links with the group assignment of the attributes by the only specifying the parameter is presented, and in Fig. 3.7.c — with the individual appointment of the attributes.
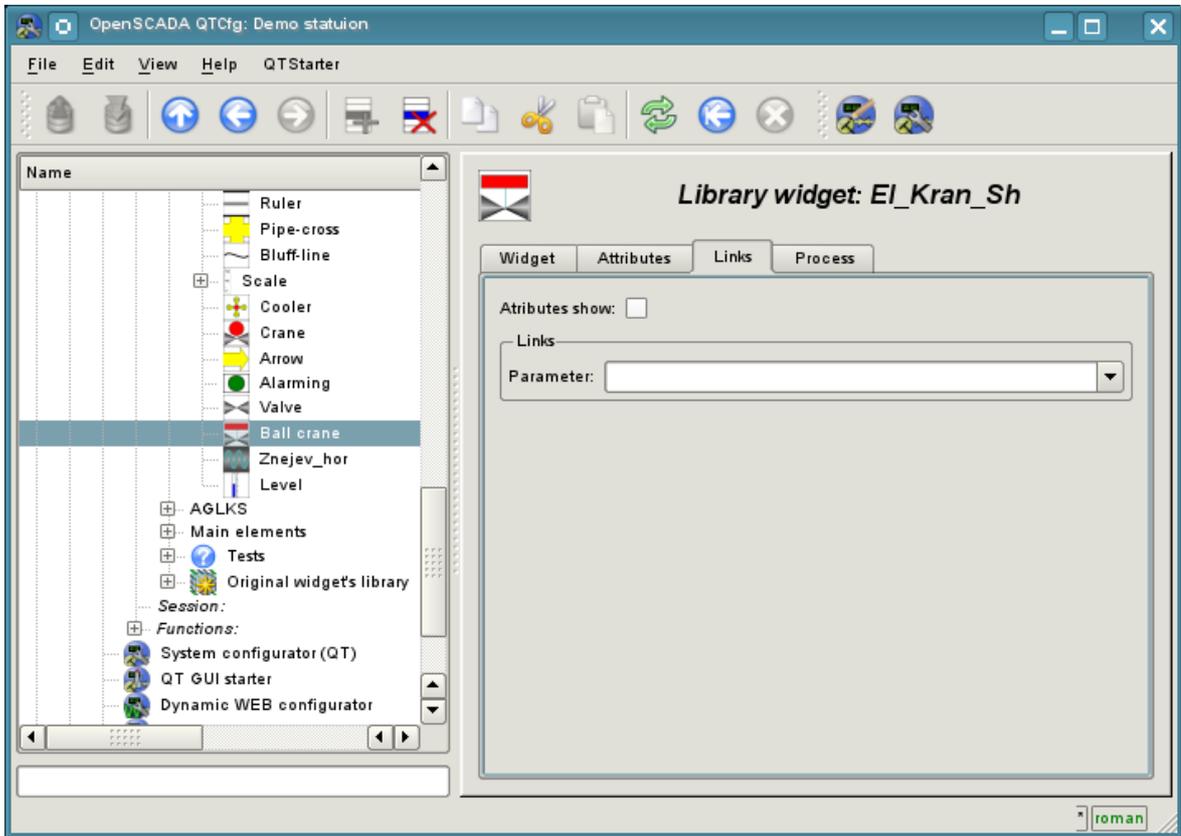
Fig. 3.7.b Tab "Links" of the page of configuration of the widget with the group assignment of the attributes by the only specifying of the parameter.
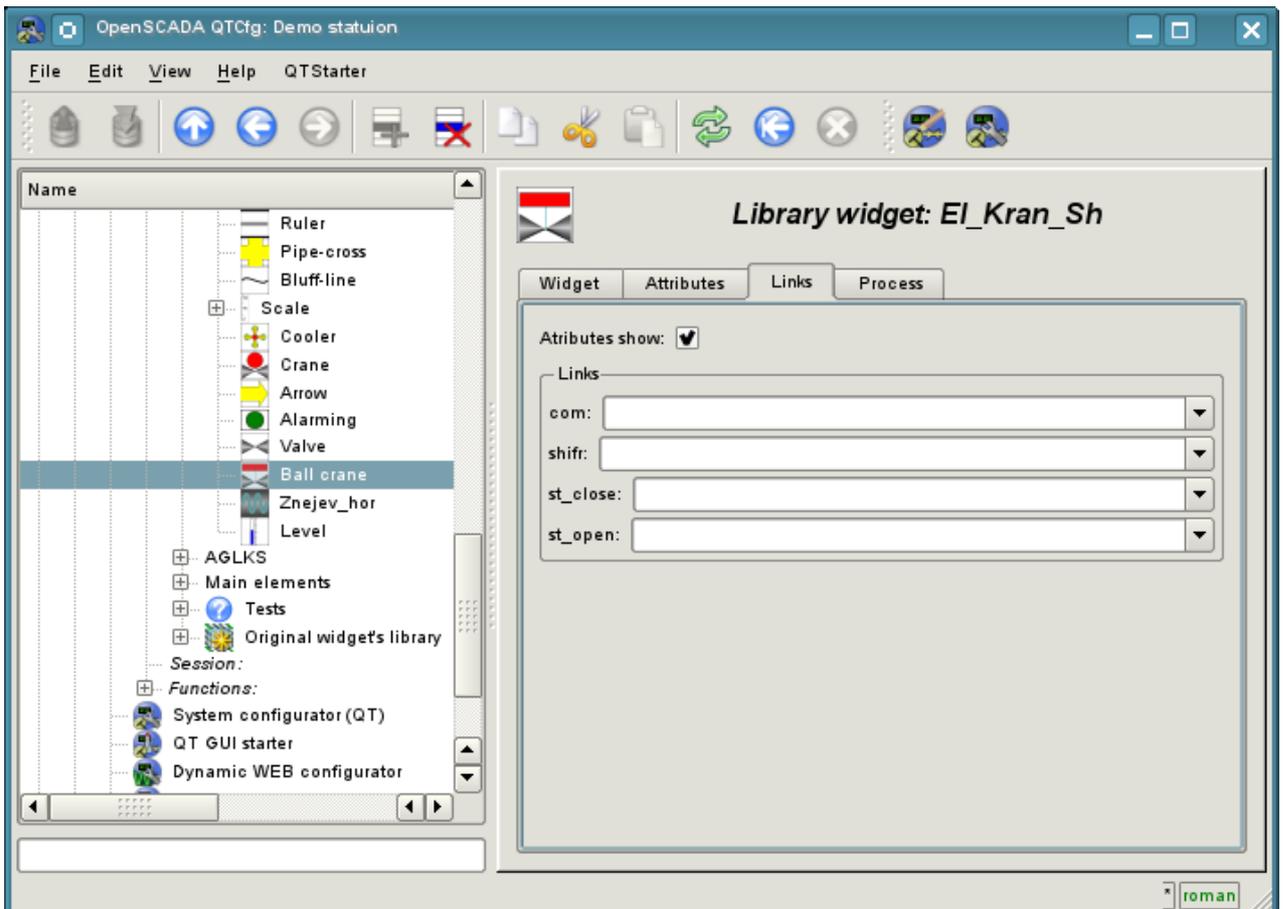
Fig. 3.7.c Tab "Links" of the page of configuration of the widget with the individual appointment of the attributes.

When the widget that contains the configuration of links is placed to the container of widgets, all links of the source widget is added to the list of resulting links of the widgets' container (Fig. 3.7.d)
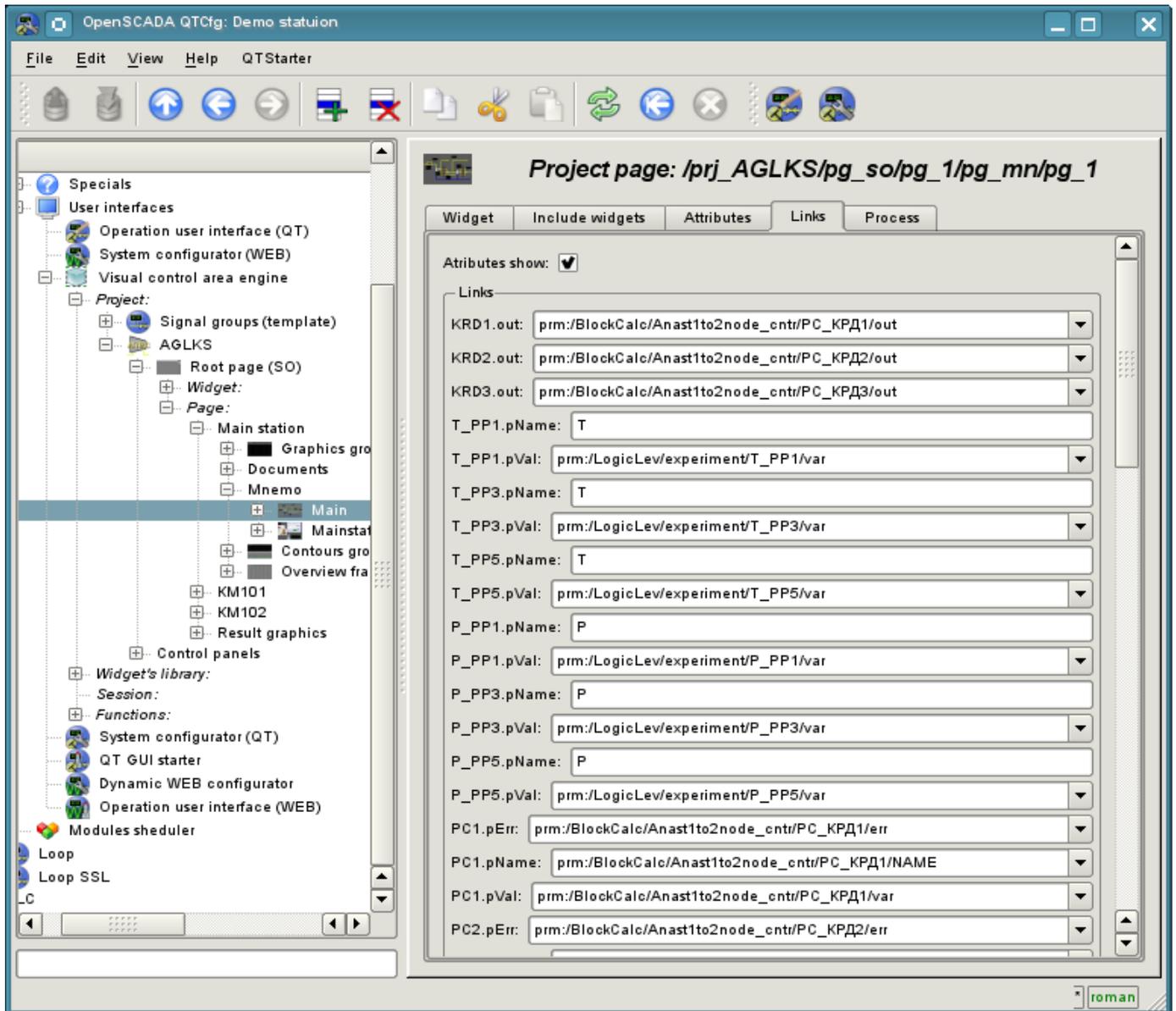
Fig. 3.7.d Tab "Links" of the page of configuration of the container of widgets, including widgets with links.

The aforesaid shows that the links are set by the user in the configuration interface. However, for the possibility of creation of the frames for general use, with the function of providing detailed data of various sources of the same type, a dynamic linkage mechanism is necessary. Such an mechanism is provided through a reserved key identifier "**<page>**" of the group of attributes of links in the frames of general purpose and dynamic linkage with the identifier "**<page>**" in the process of opening of the frame of general purpose by means of the signal from another widget.

Lets examine the example when we have the frame of general-purpose "Control panel of graph" and a lot of "Graphs" in different tabs. "Control panel of graph" has links with the templates:

- tSek --> "<page>|tSek"
- tSize --> "<page>|tSize"
- trcPer --> "<page>|trcPer"
- valArch --> "<page>|valArch"

At the same time, each widget "Graph" has the attributes tSek, tSize, trcPer and valArch. In the case of a calling of the opening signal of "Control panel of graph" from any widget "Graph" it is happening the linkage of the attributes of the "Control panel of graph" in accordance with the attribute specified in the

template with the attribute of the widget "Graph". As a result, all changes in the "Control panel of graph" will be displayed on the graph by means of the link.

In the case of presence of external links to the parameters of subsystem "Data acquisition" in the widget "Graph", the links of "Control panel of graph" will be installed on an external source. In addition, if in the "Control panel of graph" will be declared the links to the missing attributes directly in the widget "Graph", it will be made the search for the availability of such attributes from an external source, the first to which the link is directed, performing, thus, the addition of missing links.

To visualize this mechanism the table 3.7 is cited.

Table 3.7. The mechanism of the dynamic linkage.

Attributes of the "Control panel of graph" (the template of dynamic linkage)	"Graph" attributes	Attributes of an external "Parameter"	The resulting link or an value of the linking attribute
tSek (<page>|tSek)	tSek	-	"Graph".tSek
tSize (<page>|tSize)	tSize	-	"Graph".tSize
trcPer (<page>|trcPer)	trcPer	-	"Graph".trcPer
valArch (<page>|valArch)	valArch	-	"Graph".valArch
var (<page>|var)	var	var	"Parameter".var
ed (<page>|ed)	-	ed	"Parameter".ed
max (<page>|max)	-	-	EVAL
min (<page>|min)	-	-	EVAL

3.8. The primitives of the widget

Any newly created widget is based on one of several primitives (finite element of the visualization) by installing of the related link as directly to the primitive, as well as through the several intermediate user widgets. Each of the primitives contains a mechanism (logic) of data model. A copy of the widget keeps the values of the properties of configuration of the the primitive specially for itself.

The purposes of the visualization interface includes support and work with the data model of the primitives of widgets. Primitives of the widget must be carefully developed and unitized in order to cover as many opportunities in the as possible to a smaller number of weakly connected with each other by their purpose primitives.

Table 3.8.a shows the list of primitives of widgets (basic elements of visualization).

Table 3.8.a. The library of the primitives of widgets (basic elements of visualization)

Id	Name	Function
ElFigure	Elementary graphic figures	Primitive is the basis for drawing basic graphic shapes with their possible combinations in a single object. The support of the following basic figures is provided: • Line. • Arc. • Bézier curve. • Fill of the enclosed space. For all the figures contained in the widget it is set the common properties of thickness, color, etc., but this does not exclude the possibility of indicating the above attributes for each figure separately.
FormEl	Elements of the form.	Includes support for standard form components: • Line edit. • Text edit. • Check box. • Button. • Combo box. • List. • Slider. • Scroll bar.
Text	Text	Text element (labels). Characterized by the type of font, color, orientation and alignment.
Media	Media	Element of visualization of raster and vector images of various formats, playback of animated images, playback of audio segments and playback of video fragments. Perhaps it should be included the OpenGL support!
Diagram	Diagram	Element of the diagram with the support of the visualization of the flow of several trends, the spectrum ... .
Protocol	Protocol	Element of the protocol, visualizer of the system messages, with support for multiple operating modes.
Document	Document	The element of generating the reports, journals and other documentation on the basis of available in the system data.
Box	Container	Contains the mechanism fro other widgets placement with the purpose of creation of new, more complex widgets and pages of final visualization.

Id	Name	Function
Function	Function of API of the object model of OpenSCADA	Not visual, on the side of execution, widget which allows to include a computing function of the object model of OpenSCADA in the VCA.

Each primitive, and the widget at all, contains the common set of properties/attributes in the composition which is shown in Table 3.8.b:

Table 3.8.b. The common set of properties/attributes in the widget

Id	Name	#	Value
id	Id	-	Id of the element. The attribute is read-only, designed to provide information on the ID of the element.
path	Path	-	The path to the widget. The attribute is read-only and disigned to provide information about the location of the element.
parent	Parent	-	Path to parent widget. The attribute is read-only and designed to provide information about the location of ancestor from which the widget is inherited from.
owner	Owner	-	The widget owner and group in form "**[owner]:[group]**". By default the "root:UI".
perm	Access	-	Permission to the widget in form "**[user][group][other]**". Where "*user*", "*group*" and "*other*" is: • "__" — no any access; • "R_" — read only; • "RW" — read and write. By default the 0664(RWRWR_).
root	Root	1	Id of the widget-primitive (basic element) which underlies the image of visualization of the widget.
name	Name	-	Name of the element. Modifiable element name.
dscr	Description	-	Description of the element. Text field, serves for attachment to the widget of the brief description.
en	Enabled	5	The state of the element — "Enabled". Disabled element is not shown in the execution mode.
active	Active	6	The state of the element — "Active". Active element may receive focus in the execution mode, and thus receive keyboard and other events with their subsequent processing.
geomX	Geometry:x	7	Geometry, coordinate 'x' of the element position.
geomY	Geometry:y	8	Geometry, coordinate 'y' of the element position.
geomW	Geometry:width	9	Geometry, the width of the element.
geomH	Geometry:height	10	Geometry, the height of the element.
geomXsc	Geometry:x scale	13	The horizontally scale of the element.
geomYsc	Geometry:y scale	14	The vertical scale of the element.
geomZ	Geometry:z	11	Geometry, coordinate 'z' (level) of element on the page. It also defines how to transfer the focus through active elements.

Id	Name	#	Value
geomMargin	Geometry:margin	12	Geometry, the fields of the element.
tipTool	Tip:tool	15	The text of a brief help or tip on this element. Usually is realized as a tool tip, while keeping your mouse cursor over the element.
tipStatus	Tip:status	16	Text information on the status of the element or guide to action over the element. Usually is realized in the form of a message in the status bar while keeping your mouse cursor over the element. * Modifications from session the attribute of the root page will record the message in the status bar of the visualization window session.

contextMenu	Context menu	17	Context menu in form strings list: "**[ItName]:[Signal]**".
			Where:
			  • "ItName" — item name;
			  • "Signal" — signal name and result signal name is "**usr_[Signal]**".

evProc	Events process	-	Attribute for storing of the script of the processing of event of direct control of user interface. Script is the list of commands to the visualization interface generated at the event receipt (attribute **event**). Direct events processing for pages manipulation in form: "**[event]:[evSrc]:[com]:[prm]**". Where:
			  • "event" — waiting event;
			  • "evSrc" — event source;
			  • "com" — command of a session (open, next, prev);
			  • "prm" — command parameter, where used:
			    • pg_so — direct name of the desired page with the prefix;
			    • 1 — name of a new page in a general way, without a prefix;
			    • * — the page is taken from the name of a previous page;
			    • $ — points the place of the opened page relative.
			Examples:
			  • ws_BtPress:/prev:prev:/pg_so/*/*/$
			  • ws_BtPress:/next:next:/pg_so/*/*/$
			  • ws_BtPress:/go_mn:open:/pg_so/*/mn/*
			  • ws_BtPress:/go_graph:open:/pg_so/*/ggraph

Additional attributes for items placed into the project in the role of a page.			
pgOpen	Page:open state	-	Sign "The page is open". * Modifications from session provides an immediate opening/closing page.
pgNoOpenProc	Page:process no opened	-	Sign "Execute the page, even if it is closed".
pgOpenSrc	Page:open source	3	Full address of the page which has opened this one. * Write/clear address of the opening initiator — widget performs an immediate opening/closing page. In the case of write the address and on certain conditions carried the dynamic linking of the current widget to the initiator.
pgGrp	Page:group	4	The group of the page.

Id	Name	#	Value
			Additional attributes of the execution mode.
event	Event	-	Special attributes for the collection of events of the widget in the list, which is divided by the new line. This attribute is only available in the session. Access to the attribute is protected by the resource in order to avoid loss of events. An attribute is always available in the script of widget.
load	Load	-1	A virtual command of the group data download.
focus	Focus	-2	Special attribute of the indicating the fact of receiving the focus by an active widget. This attribute is only available in the session. Attribute of the widget and of the the embedded widgets is available in the script of widget.
perm	Permition	-3	Virtual attribute of the rights verification of active user on the viewing and control over the widget.

* — Special function the widget attribute running in the session of the project when user modification.

3.8.1. Elementary graphic figures (ElFigure)

Primitive is the basis for drawing basic graphic shapes with their possible combinations in a single object. Taking into account the wide range of various shapes, which must be maintained by the primitive, and at the same time the primitive must be simple enough for using and, if possible, for implementation, it was decided to limit the list of the basic figures used for the construction of the resulting graphics to these figures: line, arc, Bézier curve and fill of the enclosed spaces. Based at these basic figures, it is possible to construct derived figures by combining the basic. in the context of the primitive, there is possibility to set the transparency of the color in the range [0 .. 255], where '0' — complete transparency.

A list of additional properties/attributes of the primitive is given in Table 3.8.1.

Table 3.8.1. A list of additional properties/attributes of the primitive ElFigure

Id	Name	#	Value
lineWdth	Line:width	20	Line width.
lineClr	Line:color	21	Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.
lineStyle	Line:style	22	Line style (solid, dashed, dotted).
bordWdth	Border:width	23	Line border width. The zero width indicates the lack of border.
bordClr	Border:color	24	Border color (detailed in attribute 21).
fillColor	Fill:color	25	Fill color (detailed in attribute 21).

fillImg	Fill:image	26	Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			  • file — direct from local file by path;
			  • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";
			• "*backLogo*" — like previous;
			• "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".

orient	Orientation angle	28	The rotation angle of the content of widget.

Id	Name	#	Value
elLst	Element's list	27	List of graphic primitives in the following format: • Line. Record form in the list: **line:(x|y)|{1}:(x|y)|{2}:[width|w{n}]:[color|c{n}]:[bord_w|w{n}]:[bord_clr|c{n}]:[line_stl|s{n}]** • Arc. Record form in the list: **arc:(x|y)|{1}:(x|y)|{2}:(x|y)|{3}:(x|y)|{4}:(x|y)|{5}:[width|w{n}]:[color|c{n}]:[bord_w|w{n}]:[bord_clr|c{n}]:[line_stl|s{n}]** • Bézier curve. Record form in the list: **bezier:(x|y)|{1}:(x|y)|{2}:(x|y)|{3}:(x|y)|{4}:[width|w{n}]:[color|c{n}]:[bord_w|w{n}]:[bord_clr|c{n}]:[line_stl|s{n}]** • Fill. Record form in the list: **fill:(x|y)|{1},(x|y){2},...,(x|y)|{n}:[fill_clr|c{n}]:[fill_img|i{n}]** Where: *(x|y)* — direct point (x,y) coordinate in float point pixels; *{1}...{n}* — dynamic point 1...n; *width, bord_w* — direct line and border width in float point pixels; *w{n}* — dynamic width 'n'; *color, bord_clr, fill_clr* — direct line, border and fill color name or 32bit code whith alpha: {name}-AAA, #RRGGBB-AAA; *c{n}* — dynamic color 'n'; *line_stl* — direct line style: 0-Solid, 1-Dashed, 2-Dotted; *s{n}* — dynamic style 'n'; *fill_img* — direct fill image in form "[src%3Aname]", where: "src" — image source: *file* — direct from local file by path; *res* — from DB mime resources table. "name" — file path or resource mime Id. *i{n}* — dynamic fill image 'n'. For example: • line:(50|25):(90.5|25):2:yellow:3:green:2 • arc:(25|50):(25|50):1:4:(25|50)::#000000-0 • fill:(25|50):(25|50):c2:i2 • fill:(50|25):(90.5|25):(90|50):(50|50):#d3d3d3:h_31
*The attributes for each point from the list of graphic figures **elLst***			
p{n}x	Point {n}:x	30+n*6	Coordinates 'x' of the point {n}.
p{n}y	Point {n}:y	30+n*6+1	Coordinates 'y' of the point {n}.
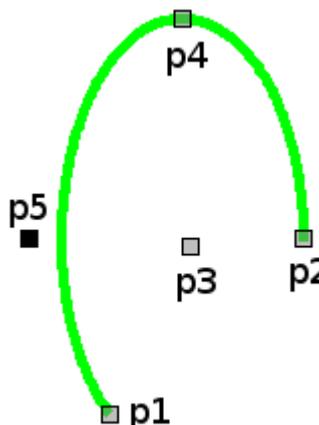
Id	Name	#	Value
w{n}	Width {n}	30+n*6+2	Width {n}.
c{n}	Color {n}	30+n*6+3	Color {n} (detailed in attribute 21).
i{n}	Image {n}	30+n*6+4	Image {n} (detailed in attribute 26).
s{n}	Style {n}	30+n*6+5	Style {n}.

3.8.2. Element of the form (FormEl)

Primitive is intended to provide the standard form elements to the user. The general list of attributes depends on the type of element. A list of additional properties/attributes of the primitive is given in Table 3.8.2.

Table 3.8.2. A set of additional properties/attributes of primitive FormEl

Id	Name	#	Value
elType	Element type	20	Type of element (Line edit, Text edit, Check box, Button, Combo box, List, Slider, Scroll bar). On its value it is depended a list of additional attributes.
			Line edit:
value	Value	21	The contents of the line.
view	View	22	Type of the editing line (Text; Combobox; Integer; Real Time, Date, Date and Time).
cfg	Config	23	Configuration of the line. The format of the value of the field for different types of lines: *Text* — the formated input configuration with parameters: **A** — ASCII alphabetic character required. A-Z, a-z. **a** — ASCII alphabetic character permitted but not required. **N** — ASCII alphanumeric character required. A-Z, a-z, 0-9. **n** — ASCII alphanumeric character permitted but not required. **X** — Any character required. **x** — Any character permitted but not required. **9** — ASCII digit required. 0-9. **0** — ASCII digit permitted but not required. **D** — ASCII digit required. 1-9. **d** — ASCII digit permitted but not required (1-9). **#** — ASCII digit or plus/minus sign permitted but not required. **H** — Hexadecimal character required. A-F, a-f, 0-9. **h** — Hexadecimal character permitted but not required. **B** — Binary character required. 0-1. **b** — Binary character permitted but not required. **>** — All following alphabetic characters are uppercased. **<** — All following alphabetic characters are lowercased. **!** — Switch off case conversion. \\ — Use to escape the special characters listed above to use them as separators. *Combobox* — contains a list of the values of the editable combobox. *Integer* — contains the configuration of input field of integer in the format: **<Minimum>:<Maximum>:<Step of change>:<Prefix>:<Suffix>**. *Real* — contains the configuration of input field of real in the format: **<Minimum>:<Maximum>:<Step of change>:<Prefix>:<Suffix>:<The number of digits after the decimal point>**. *Time, Date, Date and time* — to form the date following the the template with parameters: **d** — number of the day (1-31); **dd** — number of the day (01-31); **ddd** — acronym of the day ("Mon" ... "Sun"); **dddd** — the full name of the day ("Monday" ... "Sunday"); **M** — number of the month (1-12);

Id	Name	#	Value
			MM — number of the month (01-12); **MMM** — acronym of the month ("Jan" ... "Dec"); **MMMM** — the full name of the month ("January" ... "December"); **yy** — last two digits of the year; **yyyy** — full year; **h** — hour (0-23); **hh** — hour (00-23); **m** — minutes (0-59); **mm** — minutes (00-59); **s** — seconds (0-59); **ss** — seconds (00-59); **AP,ap** — to display AM/PM or am/pm.
confirm	Confirm	24	Enable confirm mode.
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**", where: • "*family*" — font family, for spaces use symbol '_', like: "Arial", "Courier", "Times_New_Roman"; • "*size*" — font size in pixels; • "*bold*" — font bold (0 or 1); • "*italic*" — font italic (0 or 1); • "*underline*" — font underlined (0 or 1); • "*strike*" — font striked (0 or 1). Examples: • "Arial 10 1 0 0 0" — Arial font size 10 pixels and bolded.
			Text edit:
value	Value	21	The contents of the editor.
wordWrap	Word wrap	22	Automatic division of text by the words.
confirm	Confirm	24	Enable confirm mode.
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**" (details above).
			Check box:
name	Name	26	Bame/label of the checkbox.
value	Value	21	Value of the checkbox.
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**" (details above).
			Button:
name	Name	26	Name, the inscription on the button.
value	Value	21	The value for the settled button.

img	Image	22	The image on the button. Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			  • file — direct from local file by path;
			  • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";

Id	Name	#	Value
			• "*backLogo*" — like previous; • "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".
color	Color	23	Color of the button. Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.
colorText	Color:text	27	The color of the text. (details above)
checkable	Checkable	24	Sign of functioning as a settled button.
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**" (details above).
			Combo box:
value	Value	21	Current value of the list.
items	Items	22	The entries of the list.
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**" (details above).
			List:
value	Value	21	The selected list value.
items	Items	22	The entries of the list.
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**" (details above).
			Slider and the scroll bar:
value	Value	21	Slider position.
cfg	Config	22	Configuration of the slider in the format: "**[VertOrient]:[Min]:[Max]:[SinglStep]:[PageStep]**". Where: • "*VertOrient*" — sign of a vertical orientation, the default is the horizontal orientation; • "*Min*" — minimum value; • "*Max*" — maximum value; • "*SinglStep*" — the size of a single step; • "*PageStep*" — the size of the page step.

3.8.3. Text element (Text)

This primitive is designed to display the plain text used as labels, and different signatures. With the aim of creating a simple frequent decorations the primitive must support the surrounding of the text by frame. A list of additional properties/attributes of the primitive is given in Table 3.8.3.

Table 3.8.3. The list of additional properties/attributes of the primitive Text

Id	Name	#	Value
backColor	Background: color	20	Background color. Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.

backImg	Background: image	21	Background image. The image on the button. Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			  • file — direct from local file by path;
			  • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";
			• "*backLogo*" — like previous;
			• "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".

bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color (detailed in attribute 20).
bordStyle	Border:style	24	Border style (None;Dotted;Dashed;Solid;Double;Groove;Ridge;Inset;Outset).
font	Font	25	Font name form "**{family} {size} {bold} {italic} {underline} {strike}**", where: • "*family*" — font family, for spaces use symbol '_', like: "Arial", "Courier", "Times_New_Roman"; • "*size*" — font size in pixels; • "*bold*" — font bold (0 or 1); • "*italic*" — font italic (0 or 1); • "*underline*" — font underlined (0 or 1); • "*strike*" — font striked (0 or 1). Examples: • "Arial 10 1 0 0 0" — Arial font size 10 pixels and bolded.
color	Color	26	Text color (detailed in attribute 20).
orient	Orientation angle	27	Orientation of text, rotation on angle.
wordWrap	Word wrap	28	Automatic division of text by words.
alignment	Alignment	29	Alignment of the text (Top left, top right, top center, top justify, the bottom left, bottom right, bottom justify; V center left, V center right, center ; V center justify).

Id	Name	#	Value
text	Text	30	Text value. Use "%{n}" for argument {n} (from 1) value insert.
numbArg	Arguments number	40	Arguments number.
Attributes of the arguments			
arg{x}val	Argument {x}:value	50+10*x	Argument value.
arg{x}tp	Argument {x}:type	50+10*x+1	Argument type: "Integer", "Real", "String"
arg{x}cfg	Argument {x}:config	50+10*x+2	Argument configuration: • *string*: **[len]** — string width; • *real*: **[wdth];[form];[prec]** — value width, the form of ('g', 'e', 'f'); • *integer*: **[len]** — value width.

3.8.4. Element of visualization of media materials (Media)

This primitive is designed to play different media materials, ranging from simple images to the full audio and video streams. Taking into the account the variety of ways and libraries for playing a full audio and video streams as well as a serious laboriousness of implementing of all of these mechanisms in this widget, it was decided at the initial stage, only to realize the work with images and with simple animated images and video formats. A list of additional features/attributes of the primitive is given in Table 3.8.4.

Table 3.8.4. A set of additional properties/attributes of primitive Media

Id	Name	#	Value
backColor	Background:color	20	Background color. Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.

backImg	Background:image	21	Background image. The image on the button. Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			  • file — direct from local file by path;
			  • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";
			• "*backLogo*" — like previous;
			• "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".

bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color (detailed in attribute 20).
bordStyle	Border:style	24	Border style (None; Dotted; Dashed; Solid; Double; Groove; Ridge; Inset; Outset).

src	Source	25	Media source name in form "[src:]name", where:
			• "src" — source:
			  • file — direct from local (visualizator or engine) file by path;
			  • res — from DB mime resources table;
			  • stream — Stream URL for video and audio play.
			• "name" — file path or resource mime Id.
			Examples:
			• "res:workMedia" — from DB mime resources table for Id "workMedia";
			• "workMedia" — like previous;
			• "file:/var/tmp/workMedia.mng" — from local file by path "/var/tmp/workMedia.mng";
			• "stream:http://localhost.localhost:5050" — video and audio stream play from URL.

Id	Name	#	Value
type	Type	27	Media type variant: • "Image" — raster or vector(can not support) image, like: PNG, JPEG, GIF; • "Animation" — simple animation image, like: GIF, MNG; • "Full video" — full video, audio or stream, like: OGG, OGM, AVI, MKV, MPG, MP3, MP4.
areas	Map areas	28	Number of active areas.
The attributes of the image (Image)			
fit	Fit to widget size	26	Sign "Coordinate the contents with the size of the widget".
The attributes of the video (Movie)			
fit	Fit to widget size	26	Sign "Coordinate the contents with the size of the widget".
speed	Play speed	29	The speed of playback, as a percentage from the original speed. If the value is less than 1%, the playback stops.
The attributes of the full video (Full video)			
play	Play	29	Video/audio - "Play".
roll	Roll play	30	Roll play on finish.
pause	Pause	31	Playing pause.
size	Size	32	Total video size (in milliseconds).
seek	Seek	33	Seek video playing (in milliseconds).
volume	Volume	34	Sound volume (0...100).
Active areas			
area{x}shp	Area {x}:shape	40+3*x	Type of the area (Rect;Poly;Circle).
area{x}coord	Area {x}:coordinates	40+3*x+1	The coordinates of areas. Coordinates are separated by commas: "x1,y1,x2,y2,xN,yN"
area{x}title	Area {x}:title	40+3*x+2	Title of the area.

3.8.5. Element of constructing diagrams/trends (Diagram)

This primitive is designed to construct various diagrams, including graphs/trends showing ongoing process and archive data. At this time, the following types of diagrams are realized:

• "Graph" — allows you to build a one-dimensional graphs of the values of parameters of subsystems "Data acquisition" in time, as well as direct use of historical data to graph. It supports the tracing of current values and the values of the archive modes. It supports also the possibility of building the graphs of the parameters which have no archive of values.

• "Spectrum" — builds the frequency spectrum of values from the subsystem "Data acquisition". Window of the data of frequency spectrum is formed on the basis of the size of the widget horizontally, in pixels, and the available data of the parameters imposed on the horizontal grid size. In this regard, the minimum frequency is determined by the value of the attribute tSize (1/tSize), and maximum frequency of allocated frequencies is determined by half-width of the graph in pixels multiplied by the minimum frequency (width/(2*tSize)). It is supported the formation of the spectrum in the tracing mode.

The process of access to archive data is optimized, by means of an intermediate buffer for the display, as well as the package of traffic data in the query. A list of additional properties/attributes of the primitive is given in Table 3.8.5.

Table 3.8.5. A list of additional properties/attributes of the primitive Diagram

Id	Name	#	Value
backColor	Background:color	20	Background color. Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.

backImg	Background:image	21	Background image. The image on the button. Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			  • file — direct from local file by path;
			  • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";
			• "*backLogo*" — like previous;
			• "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".

bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color (detailed in attribute 20).
bordStyle	Border:style	24	Border style (None; Dotted; Dashed; Solid; Double; Groove; Ridge; Inset; Outset).
trcPer	Tracing period (s)	25	Mode and frequency of tracing.
type	Type	26	Diagram type: "Trend".

Id	Name	#	Value
			Attributes of the trend/graph (Trend)
tSek	Time:sek	27	Current time, seconds.
tUSek	Time:usek	28	Current time, microseconds.
tSize	Size, sek	29	Size of the trend, seconds.
curSek	Cursor:sek	30	Cursor position, seconds.
curUSek	Cursor:usek	31	Cursor position, microseconds.
curColor	Cursor:color	32	Cursor color.
sclColor	Scale:color	33	Color of the scale/grid (detailed in attribute 20).
sclHor	Scale:horizontal	34	Horizontal mode of the scale/grid: "No draw", "Grid;Markers" и "Grid and markers".
sclVer	Scale:vertical	35	Vertical mode of the scale/grid: "No draw", "Grid", "Markers", "Grid and markers", "Grid (log)", "Marker (log)", "Grid and markers (log)".
sclVerScl	Scale:vertical scale (%)	40	Graphic's vertical scale in percents.
sclVerSclOff	Scale:vertical scale offset (%)	41	Offset of graphic's vertical scale in percents.
sclMarkColor	Scale:Markers:color	36	Color of markers of the scale/grid (detailed in attribute 20).
sclMarkFont	Scale:Markers:font	37	Font of markers of scale/grid. Font name form "**{family} {size} {bold} {italic} {underline} {strike}**", where: • "*family*" — font family, for spaces use symbol '_', like: "Arial", "Courier", "Times_New_Roman"; • "*size*" — font size in pixels; • "*bold*" — font bold (0 or 1); • "*italic*" — font italic (0 or 1); • "*underline*" — font underlined (0 or 1); • "*strike*" — font striked (0 or 1). Examples: • "Arial 10 1 0 0 0" — Arial font size 10 pixels and bolded.
valArch	Value archivator	38	Value archivator in form "**ArchMod.ArchivatorId**".
valsForPix	Values for pixel	42	The number of values per pixel. Increase to enhance the accuracy of export at large time intervals.
parNum	Parameters number	39	The number of parameters that can be displayed on the one trend.
			Individual attributes of the parameters of trend/graph

Id	Name	#	Value
prm{X}addr	Parametr {X} :address	50+10*{X}	Full address to DAQ attribute of a parameter {X} or to an archive. Example: • "/DAQ/System/AutoDA/MemInfo/use" — address to attribute "use" of parameter "MemInfo" of controller "AutoDA" of DAQ module "System"; • "/Archive/va_CPULoad_load" — address to archive "CPULoad_load".
prm{X}bord L	Parametr {X} :view border:lower	50+10*{X} +1	The lower limit of the parameter {X}.
prm{X}bord U	Parametr {X} :view border:upper	50+10*{X} +2	The upper limit of the parameter {X}.
prm{X}color	Parametr {X} :color	50+10*{X} +3	Color for display of trend of the parameter {X} (detailed in attribute 20).
prm{X}width	Parametr {X} :width	50+10*{X} +6	Line width for display of trend of the parameter {X}, in pixels.
prm{X}val	Parametr {X} :value	50+10*{X} +4	Value of the parameter {X} under the cursor.
prm{X}prop	Parametr {X} :properties	50+10*{X} +7	Real archive properties in form **"BegArh:EndArh:DataPeriod"**, where: • *BegArh*, *EndArh*, *DataPeriod* — begin, end and period archive's data in seconds, real up to microseconds (1e-6).

3.8.6. The element of building the protocols based on the archives of messages (Protocol)

This primitive is designed to visualize the data of the archive of messages through the formation of protocols with different ways of visualization, starting from a static scanning view and finishing with dynamic tracing of protocol of message. A list of additional properties/attributes of the primitive is given in Table 3.8.6.

Table 3.8.6. A list of additional properties/attributes of the primitive Protocol

Id	Name	#	Value
backColor	Background:color	20	Background color. Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.

backImg	Background:image	21	Background image. The image on the button. Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			  • file — direct from local file by path;
			  • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";
			• "*backLogo*" — like previous;
			• "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".

font	Font	22	Font of markers of scale/grid. Font name form "**{family} {size} {bold} {italic} {underline} {strike}**", where: • "*family*" — font family, for spaces use symbol '_', like: "Arial", "Courier", "Times_New_Roman"; • "*size*" — font size in pixels; • "*bold*" — font bold (0 or 1); • "*italic*" — font italic (0 or 1); • "*underline*" — font underlined (0 or 1); • "*strike*" — font striked (0 or 1). Examples: • "Arial 10 1 0 0 0" — Arial font size 10 pixels and bolded.
headVis	Header visible	23	Show header for table or not.
time	Time, sek	24	Current time, seconds.
tSize	Size, sek	25	Query size, seconds. Set value to '0' for get all alarms, for "lev" < 0.
trcPer	Tracing period (s)	26	Mode and frequency of tracing.
arch	Archivator	27	Messages archivator in form "**ArchMod.ArchivatorId**".

Id	Name	#	Value
tmpl	Template	28	Category template or regular expression "/{**re**}/". For template reserved special symbols: • '*' — any multiply symbols group; • '?' — any one symbol; • '\\' — use for shield special symbols.
lev	Level	29	The level of messages. Set value to < 0 for get current alarms.
viewOrd	View order	30	View order ("By time", "By level", "By category", "By messages", "By time (reverse)", "By level (reverse)", "By category (reverse)", "By messages (reverse)").
col	View columns	31	Visible and order columns list separated by symbol ';'. Supported columns: • "*pos*" — row number; • "*tm*" — date and time of the message; • "*utm*" — microseconds part of time of the message; • "*lev*" — level of the message; • "*cat*" — category of the message; • "*mess*" — the message text.
itProp	Item properties	32	Item's properties number.
	Individual attributes of item's properties		
it{X}lev	Item {X}:level	40+5*{X}	Criterion: element's level {X}. More or equal for pointed.
it{X}tmpl	Item {X}:template	41+5*{X}	Criterion: element's category template {X}. (detailed in attribute 28).
it{X}fnt	Item {X}:font	42+5*{X}	Element {X} font (detailed in attribute 22).
it{X}color	Item {X}:color	43+5*{X}	Element {X} color (detailed in attribute 20).

3.8.7. Element of formation of documentation(Document)

Primitive is designed to create report, operational and other documents based on templates of documents. A list of additional properties/attributes of the primitive is given in Table 3.8.7.

Table 3.8.7. A list of additional properties/attributes of the primitive Document

Id	Name	#	Value
style	CSS	20	CSS rules in rows like "**body { background-color:#818181; }**".

tmpl	Template	21	Document's template in XHTML. Start from tag "body" and include procedures parts:
			`<body docProcLang="JavaLikeCalc.JavaScript">`
			`    <h1>Value<?dp return wCod+1.314;?></h1>`
			`</body>`

doc	Document	22	Final document in XHTML. Start from tag "body".
font	Font	26	Basic font of the text. Font name form "**{family} {size} {bold} {italic} {underline} {strike}**", where: • "*family*" — font family, for spaces use symbol '_', like: "Arial", "Courier", "Times_New_Roman"; • "*size*" — font size in pixels; • "*bold*" — font bold (0 or 1); • "*italic*" — font italic (0 or 1); • "*underline*" — font underlined (0 or 1); • "*strike*" — font striked (0 or 1). Examples: • "Arial 10 1 0 0 0" — Arial font size 10 pixels and bolded.
bTime	Time:begin	24	Start time of the document, seconds.
time	Time:current	23	Time of the document generation, seconds. Write time for document generation from that point.
n	Archive size	25	Number of documents or the depth of the archive.
Attributes of the enabled archival mode			
aCur	Archive:cursor:current	-	The position of the current document in the archive. Record of the value <0 produces the archiving of this document.
vCur	Archive:cursor:view	-	Current visual document of the archive. Writing a value of -1 — to select the next document, -2 — to select the previous instrument.
aDoc	Archive:current document	-	Current archive document in XHTML. Start from tag "body".
aSize	Archive:size	-	Real archive documents size.

Features of the primitive "Document":
• Flexible formation of the structure of the document based on Hypertext Markup Language. This will provide support of wide formatting opportunities of documents with the subsequent implementation of the GUI form of the document formation.
• Formation of documents on command or on a plan into the with the archive with the subsequent viewing of the archive.
• Document formation in real-time mode, fully dynamic and based on the archives for the specified time.
• Using the attributes of the widget to pass values and addresses to the archives in the document. Allows you to use the widget of document as the template for generating reports with other input data.

The basis of any document is XHTML-template. XHTML-template is the tag "body" of the WEB-page which contains the document's static in the standard XHTML 1.0 and elements of the executable instructions in one of the languages of the user programming of OpenSCADA in the form of **<?dp {procedure} ?>**. The resulting document is formed by the execution of procedures and insert of their result into the document.

The source for values of the executable instructions are the attributes of the widget of the primitive, as well as all the mechanisms of the user programming language. Attributes may be added by the user and they can be linked to the actual attributes or parameters or they can be autonomous, values of which will be formed in the script of the widget. In the case of linked attributes the values can be extracted from the history, archive.

Fig. 3.8.7.a shows a block diagram of the widget of the primitive "Document". According to this structure "Document" includes: XHTML-template, the resulting documents and the processing script. The data source for the script and for the resulting documents are the attributes of the widget.
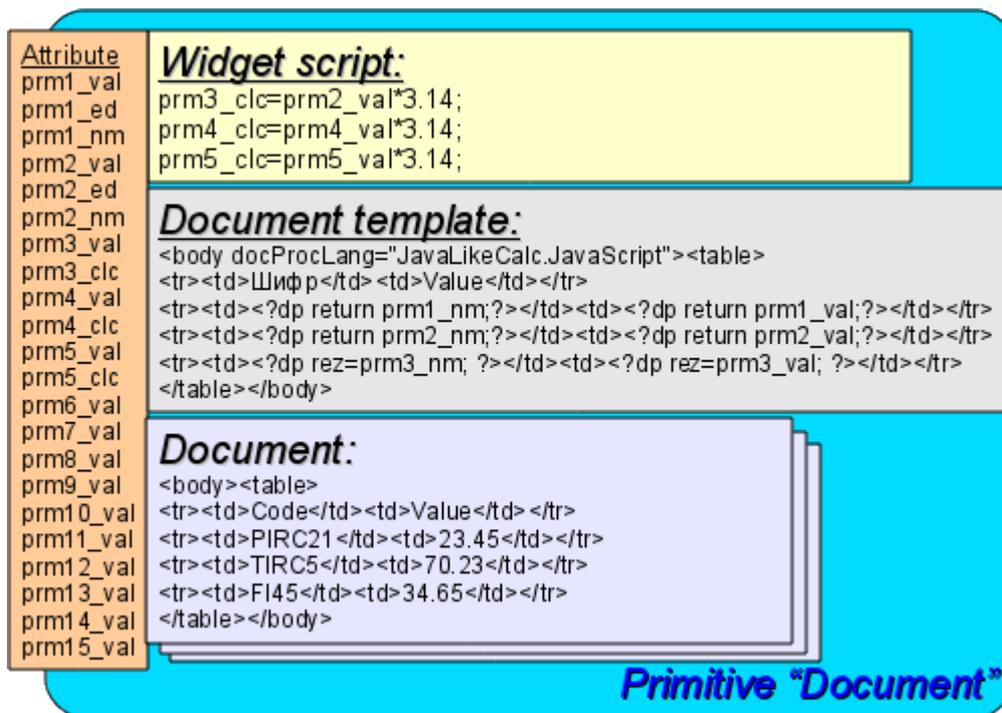
Fig. 3.8.7.a The block diagram of the primitive "Document".

It i provided the work of widget in two modes: Dynamic and Archive. The difference between archive mode is the availability of the archive of the specified depth and attributes which allow you to control the process of archiving and viewing of the document in the archive.

Generation of the document is always made at the time of installation of the time attribute <time> relatively to the set start time of the document in the attribute <bTime>. With the archive turned off the resulting document is placed directly in the attribute <doc>. When the archive is turned on the resulting document is placed in the cell under the cursor, the attribute <aCur>, as well as in <doc> if the value of the archive cursor <aCur> and the cursor of visualized document <vCur> match. Attributes of the archival cursors provide several command of values:

• *aCur<0* — Moves the archiver cursor for the following position, thereby leaving the previous document in the archive and clearing the document under the cursor.
• *vCur==-1* — Select of the next document to be displayed. The selected document is copied to the attribute <doc>.
• *vCur==-2* — Select of the previous document to be displayed. The selected document is copied to the attribute <doc>.

As it was stated above dynamics of the document's template is defined by the inserts of executable instructions of the form **<?dp {procedure} ?>**. The procedures may use the same attributes of the widget

and functions of the user programming interface of OpenSCADA. In addition to the attributes of the widget special attributes (Table 3.8.7.a) are reserved.

In addition to special attributes in XHTML template tags and tags' attributes of special assignment are reserved (Table 3.8.7.a).

Table 3.8.7.a. Special and reserved elements of the template.

Name	Assignment
Attributes	
rez	Attribute of the results of the procedure execution, the contents of which is placed to the document tree.
lTime	Last formation time. If the document is formed for the first time, <lTime> is equal to the <bTime>.
rTime	Contains the time for the selected values in seconds. It is defined inside the tags with the attribute "docRept".
rTimeU	Contains the time for the selected values in microseconds. It is defined inside the tags with the attribute "docRept".
rPer	Contains the periodicity of the selection of values (the attribute "docRept").
mTime, mTimeU, mLev, mCat, mVal	It is defined inside the tags with an attribute "docAMess" when parsing messages of the messages' archive: mTime — message time; mTimeU — message time, microseconds; mLev — message level; mCat — message category; mVal — message value.
Special tags	
Special attributes of the standard tags	
body.docProcLang	Language of executable procedures of the document. By defaults it is JavaLikeCalc.JavaScript.
*.docRept="1s"	Tag with the specified attribute, while the formation it multiplies through the time offset in the attribute "rTime" to the value, specified in this attribute.
.docAMess="1:PLC"	Indicates the necessity of the tag multiplication with an attribute of message from the archive of messages for the specified interval of time and in accordance with the level of (1) and template of request (PLC*). The template request may specify a regular expression in the form of /{**re**}/. For this tag in the process of multiplication the following attributes: mTime, mTimeU, mLev, mCat and mVal are defined.
*.docRevers="1"	Points to invert of the order of multiplication, the last from the top.
*.docAppend="1"	The sign of the necessity of addition of the procedure execution result in the tag of the procedure. Otherwise, the result of execution replaces the contents of the tag.
body.docTime	Time of formation of the document. It is used to set the attribute <lTime> in the time of the next formation of the document. It is not set by the user!
table.export="1"	Enable for selected table content allow for export to CSV-file and other table formats.

3.8.8. Container (Box)

Primitive container is used to build composite widgets and/or the pages the user interface. A list of additional properties/attributes of the primitive is given in Table 3.8.8.

Table 3.8.8. A list of additional properties/attributes of the primitive Box

Id	Name	#	Value
pgOpenSrc	Page:open source	3	Full address of the page, which is included inside of the container.
pgGrp	Page:group	4	The group of container of pages.
backColor	Background:color	20	Background color. Background color. Color name form "**color[-alpha]**", where: • "*color*" — standard color name or digital view of three hexadecimal digit's number form "#RRGGBB"; • "*alpha*" — alpha channel level (0-255). Examples: • "*red*" — solid red color; • "*#FF0000*" — solid red color by digital code; • "*red-127*" — half transparent red color.

backImg	Background:image	21	Background image. The image on the button. Image name in form "**[src:]name**", where:
			• "*src*" — image source:
			    • file — direct from local file by path;
			    • res — from DB mime resources table.
			• "*name*" — file path or resource mime Id.
			Examples:
			• "*res:backLogo*" — from DB mime resources table for Id "backLogo";
			• "*backLogo*" — like previous;
			• "*file:/var/tmp/backLogo.png*" — from local file by path "/var/tmp/backLogo.png".

bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color (detailed in attribute 20).
bordStyle	Border:style	24	Border style (None; Dotted; Dashed; Solid; Double; Groove; Ridge; Inset; Outset).

3.9. Using the database to store the library of widgets and projects

Storage of widgets and widget libraries is implemented in the databases accessible in the OpenSCADA system. DB is organized on the data belonging to the library. Ie a separate library is stored in a separate group of tables of one or of the different databases. The list of libraries of widgets is stored in the index table of the libraries with the name "VCALibs" and with the structure "Libs". A copy of this table is created in each database, which stores data of the module with the list of libraries which are hold in a given database. To the composition of the tables belonging to the library of widgets, are included:
- {DB_TBL} — Table of widgets belonging to the library (structure "LibWigets").
- {DB_TBL}_io — Table with the working properties of the widget in this library and of the embedded widgets of the container ones (structure "LibWidgetIO").
- {DB_TBL}_uio — Table with the user properties of the widgets of this library and the embedded widgets of container ones (structure "LibWidgetUserIO", раздела БД).
- {DB_TBL}_incl — Table of embedded widgets in the widgets-containers of the Library (structure "LibWidgetIncl").
- {DB_TBL}_mime — Table with the resources of the library and its widgets (structure "LibWidgetMime").
- {DB_TBL}_ses — Table for store data of project's run mode, session (structure "PrjSesIO").

Projections (structures) of basic tables are as follows:
- Libs(ID, NAME, DSCR, DB_TBL, ICO) — Libraries of widgets <ID>.
  - *ID* — identifier;
  - *NAME* — name;
  - *DSCR* — description;
  - *DB_TBL* — DB with widgets;
  - *ICO* — coded (Base64) image of the icon of the library.
- LibWigets(ID, ICO, PARENT, PROC, PROC_PER, USER, GRP, PERMIT, ATTRS) — Widgets <ID> of the library.
  - *ID* — identifier;
  - *ICO* — coded (Base64) image of the icon of the widget.
  - *PARENT* — address of the basic widget */wlb_originals/wdg_Box* ;
  - *PROC* — internal script and script language of the widget;
  - *PROC_PER* — frequency of the computation of the script of the widget;
  - *ATTRS* — list of attributes of the widget, modified by the user.
- LibWidgetIO(IDW, ID, IDC, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) — Work attributes <ID> of the widget <IDW>.
  - *IDW* — identifier of the widget;
  - *ID* — identifier of the IO;
  - *IDC* — child widget identifier;
  - *IO_VAL* — value of the attribute;
  - *SELF_FLG* — internal flags of the IO;
  - *CFG_TMPL* — template of the configuration element based on this attribute;
  - *CFG_VAL* — value of the configuration element (link, constant ...).
- LibWidgetUserIO(IDW, ID, IDC, NAME, IO_TP, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) — User attributes <ID> of the widget <IDW>.
  - *IDW* — identifier of the widget;
  - *ID* — identifier of the IO;
  - *IDC* — child widget identifier;
  - *NAME* — name of the IO;
  - *IO_TP* — type and main flags of the IO;
  - *IO_VAL* — value of the IO;
  - *SELF_FLG* — internal flags of the IO;
  - *CFG_TMPL* — template of the configuration element based on this attribute;
  - *CFG_VAL* — value of the configuration element (link, constant ...).

- LibWidgetIncl(<u>IDW</u>, <u>ID</u>, PARENT, ATTRS, USER, GRP, PERMIT) — Included into the container *<IDW>* widgets *<ID>*.

    IDW — identifier of the widget;

    ID — Identifier of the copy of the embedded widget;

    PARENT — address of the basic widget */wlb_originals/wdg_Box* ;

    ATTRS — list of attributes of the widget, modified by the user.

- LibWidgetMime(<u>ID</u>, MIME, DATA) — Audio, video, media and other resources of widgets of the library.

    ID — identifier of the resource.

    MIME — Mime data type of the resource (in the format — <mimeType;Size>).

    DATA — Resource data coded with Base64.

- Project(<u>ID</u>, NAME, DSCR, DB_TBL, ICO, USER, GRP, PERMIT, PER, FLGS) — Projects of visualization interfaces *<ID>*.

    ID — identifier of the project;

    NAME — name of the project;

    DSCR — description of the project;

    DB_TBL — Database with project pages.

    ICO — coded (Base64) image of the icon of the project;

    USER — owner of the project;

    GRP — users group of the project;

    PERMIT — rights of access to the project;

    PER — frequency of the computation of the project;

    FLGS — flags of the project.

- ProjPage(<u>OWNER</u>, <u>ID</u>, ICO, PARENT, PROC, PROC_PER, USER, GRP, PERMIT, FLGS, ATTRS) — The pages *<ID>* which are hold in the project/page *OWNER>*.

    OWNER — project/page — owner of the page (in the format — "/AGLKS/so/1/gcadr")

    ID — identifier of the page;

    ICO — coded (Base64) image of the icon of the page;

    PARENT — address of the basic widget of the page in the format: */wlb_originals/wdg_Box* ;

    PROC — internal script and script language of the page;

    PROC_PER — frequency of the computation of the script of the widget;

    FLGS — flags of the page;

    ATTRS — list of attributes of the widget, modified by the user.

- ProjSess(<u>IDW</u>, <u>ID</u>, IO_VAL) — Project table *<IDW>* for data storage of the sessions, performing project.

    IDW — the full path of the element of the project;

    ID — attribute of the element;

    IO_VAL — value of the element.

- ProjPageIO(<u>IDW</u>, <u>ID</u>, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) — Working attributes of the pages. The structure actually corresponds to the table LibWidgetIO.

- ProjPageUserIO(<u>IDW</u>, <u>ID</u>, NAME, IO_TP, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) — User attributes of the pages. The structure actually corresponds to the table LibWidgetUserIO.

- ProjPageWIncl(<u>IDW</u>, <u>ID</u>, PARENT, ATTRS, USER, GRP, PERMIT) — Enabled widgets on the page. The structure actually corresponds to the table LibWidgetIncl.

- PrjSesIO(IDW, ID, IO_VAL) — Attributes *<ID>* of the session's element *<IDW>*.

    IDW — identifier of the session's element;

    ID — identifier of the IO;

    IO_VAL — value of the attribute.

3.10 API of the user programming and service interfaces of the OpenSCADA

3.10.1. API of the user programming

API of the user programming of API of the visualization engine are represented by OpenSCADA objects directly, which build user interface, and same "Session" and "Widget/page". These objects provide the set of control functions for the user:

Object "Session" (this.ownerSess())
- *string user( )* — The session user.
- *string alrmSndPlay( )* — The widget's path for that on this time played the alarm message.
- *int alrmQuittance(int quit_tmpl, string wpath = "")* — alarm quittance *<wpath>* with template *<quit_tmpl>*. If *<wpath>* is empty string then make global quittance.

Object "Widget" (this)

- *TCntrNodeObj ownerSess( )* — the object-session is getting for current widget.
- *TCntrNodeObj ownerPage( )* — the parent object-page is getting for current widget.
- *TCntrNodeObj ownerWdg(bool base = false)* — the parent object-widget is getting for current widget. If set *<base>* then will include return the parent object-page.
- *TCntrNodeObj wdgAdd(string wid, string wname, string parent)* — add new widget *<wid>* with name *<wname>* and based at library widget *<parent>*.
    ```
    //New widget add, which based at text primitive
    nw = this.wdgAdd("nw", "New widget", "/wlb_originals/wdg_Text");
    nw.attrSet("geomX", 50).attrSet("geomY", 50);
    ```
- *bool wdgDel(string wid)* — delete widget *<wid>*.
- *TCntrNodeObj wdgAt(string wid, bool byPath = false)* — attach to child or global, by *<byPath>*, widget. In the case of global connection, you can use absolute or relative path to the widget. For starting point of the absolute address acts the root object of module "VCAEngine", which means the first element of the absolute address is session identifier, which is omitted. The relative address takes the countdown from the current widget. Special element of relative address is an element of parent node "..".
- *bool attrPresent(string attr)* — the attribute *<attr>* of widget checking to allow fact.
- *ElTp attr(string attr)* — the attribute *<attr>* of widget value getting. For disallow attributes will return empty string.
- *TCntrNodeObj attrSet(string attr, ElTp vl)* — the attribute *<attr>* of widget value setting to *<vl>*. The object is returned for the function concatenation.
- *string link(string attr, bool prm = false)* — link return for widget's attribute *<attr>*. At set *<prm>* requested link for attributes block (parameter), represented by the attribute.
- *string linkSet(string attr, string vl, bool prm)* — set link for widget's attribute *<attr>*. At set *<prm>* made set link for attributes block (parameter), represented by the attribute.
    ```
    //Set link for eight trend to parameter
    this.linkSet("el8.name", "prm:/LogicLev/experiment/Pi", true);
    ```

Object "Widget", of primitive "Document" (this)
- *string getArhDoc(integer nDoc)* — get archive document text to "nDoc" (0-{aSize-1}) depth.

Deprecated API of the user programming of the visualization engine are represented by the group of functions directly in the engine module of the VCA. Calling of these functions from the scripts of widgets can be performed directly by the ID of the function, since their area of names is indicated for the context of the scripts of widgets.

Widget list (WdgList)

Description: Returns a list of widgets in the container of widgets or a list of child widgets. If <pg> is set it returns a list of pages for projects and sessions.

Parameters:

ID	Name	Type	Mode	By default
list	List	String	Return	
addr	Address	String	Input	
pg	Pages	Bool	Input	0

Presence of the node (NodePresent)

Description: Check for the presence of the node, including widgets, attributes and others.

Parameters:

ID	Name	Type	Mode	By default
rez	Result	Bool	Return	
addr	Address	String	Input	

Attributes list (AttrList)

Description: Returns list of attributes of the widget. If <noUser> is set then only not user attributes are returned.

Parameters:

ID	Name	Type	Mode	By default
list	List	String	Return	
addr	Address	String	Input	
noUser	Without user	Bool	Input	1

Request of the attribute (AttrGet)

Description: Request of the value of the attribute of the widget. The request can be done as by indicating the full address of the attribute in <addr>, and by indicating separately the address of the widget in <addr>, and the ID of the attribute in the <attr>.

Parameters:

ID	Name	Type	Mode	By default
val	Value	String	Return	
addr	Address	String	Input	
attr	Attribute	Bool	Input	

Setting of the attribute (AttrSet)

Description: Setting of the value of the attribute of the widget. Setting can be done as by the indicating the full address of the attribute in <addr>, and by indicating separately the address of the widget in <addr>, and the ID of the attribute in <attr>.

Parameters:

ID	Name	Type	Mode	By default
addr	Address	String	Input	
val	Value	String	Input	
attr	Attribute	Bool	Input	

Session user (SesUser)

Description: Return session user by session's widget path.

Parameters:

ID	Name	Type	Mode	By default
user	User	String	Return	
addr	Address	String	Input	

3.10.2. Service interfaces of the OpenSCADA

Service interfaces are interfaces of access to the OpenSCADA system by means of OpenSCADA control interface from external systems. This mechanism — is the basis of all the mechanisms for sharing within OpenSCADA, implemented through weak ties, and standard exchange protocol of OpenSCADA.

Access to the values of attributes of the visualization elements (widgets)

In order to provide uniform, group, and relatively fast access to the values of attributes of the visual elements the service function of the visual element "/serv/attr" and get/set command of the attributes' values are provided: <get path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattr"/> and <set path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattr"/>. Attributes of these commands, which provide the various mechanisms of the request, are presented in the Table 3.10.2.a.

Table 3.10.2.a. Attributes of commands of get/set of the the attributes of visual elements

Id	Name	Value
Request command of the visual attributes of the widget: <get path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattr"/>		
tm	Time/counter of changes	Time/counter of changes set up for the query of the only changed attributes.
<el id="{attr}" p="{a_id}">{val}</el>	The formation of the child elements with the results of the attributes	In the child element are specified: string ID {attr} of the attribute, index {a_id} of the attribute and its value {val}.
The set command of the visual attributes of the widget: <set path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattr"/>		
<el id="{attr}">{val}</el>	Set of the ettributes	In the child elements the ID of the attribute {attr} and its value {val} are specified.

Group access to the values of attributes of the visualization elements (widgets)

In order to optimize network traffic by eliminating small queries, but use one, but a large the group query of the attributes' values of visual elements is made. Grouping of this query involves a request of attributes of the entire branch of the widget, including embedded elements. For this request the service command "/serv/attrBr". Request of this service command is equivalent to the service command "/serv/attr" and looks as follows:
<get path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattrBr"/>

> *tm* — Time/counter of changes. Time/counter of changes set up for the query of the only changed attributes.

> Result:

> > *<el id="{attr}" p="{a_id}">{val}</el>* — Elements with the results of the attributes. In the element are specified: string ID {attr} of the attribute, index {a_id} f the attribute and its value {val}.

> > *<w id="{wid}" lnkPath="{lnk_path}">{childs+attrs}</w>* — Elements with child widgets and their attributes. The identifier of the child widget {wid} and the path to the widget on which the current widget links to, if its is the link {lnk_path}, are specified in the element.

Access to the pages of the session

In order to unify and optimize the access to the pages the service function of the session "/serv/pg" and commands of the query of the list of open pages (<openlist path="/UI/VCAEngine/ses_{Session}/%2fserv%2fpg"/>); of opening the pages (<open path="/UI/VCAEngine/ses_{Session}/%2fserv%2fpg"/>); and closing of the pages <close path="/UI/VCAEngine/ses_{Session}/%2fserv%2fpg"/>) are provided.

The result of the query of the list of open pages are child elements <el>{OpPage}</el> which contain the full path of the open page. In addition to the list of open pages, the query returns the value of the current counter for calculating the session in the attribute <tm>. If this attribute is set during the query, then for each open page it is returned the list of changed, since the moment of the specified value of the counter, widgets of the open page.

Signaling (alarm) management

To provide a mechanism for global control of the signaling of the session the service function of the session "/serv/alarm" and commands of the query of the signals status (<get path="/UI/VCAEngine/ses_{Session}/%2fserv%2falarm"/>); and of the quittance (<quittance path="/UI/VCAEngine/ses_{Session}/%2fserv%2falarm"/>) are provided.

Request for the status of signals returns generalized condition of the signals, as well as additional information for the sound signaling. Additional information by sound signal is provided by the current resource, a sound file, for playback and it provides monitoring of the sequence of signaling and quittance of individual files of sound messages.

Request for the quittance performs quittance of the specified widget, attribute *<wdg>*, in accordance with the template, *attribute <tmpl>*.

Manipulation with the sessions of the projects

To provide a uniform mechanism for manipulation of the sessions by the visualizers of VCA in the module of the VCA engine (VCAEngin) are provided: the service function "/serv/sess" and query commands of the list of open sessions, connection/creation of the new session and disconnection/deleting of the session:<list path="/UI/VCAEngine/%2fserv%2fsess"/>, <connect path="/UI/VCAEngine/%2fserv%2fsess"/> and <disconnect path="/UI/VCAEngine/%2fserv%2fsess"/> accordingly. Attributes of these commands, which provide the various mechanisms of the request, are presented in Table 3.10.2.b.

Table 3.10.2.b. Attributes of commands of the mechanism of manipulation with sessions

Id	Name	Value
Command of request of the list of open sessions for the project: <list path="/UI/VCAEngine/%2fserv%2fsess"/>		
prj	Indication of the project	Specifies the project for which to return the list of open sessions.
<el>{Session}</el>	Control of the sessions' list	In the child element the open for the requested project sessions are specified.
The command of the connection/opening of the session: <connect path="/UI/VCAEngine/%2fserv%2fsess"/>		
sess	Installation and control of the session name	If the attribute is defined, then connecting to an existing session is to be made, else — creation of the new session is to be made. In the case of opening of the new session in this attribute its name is is placed.
prj	Setting the name of the project	It is used to open a new session for indicated project and when the attribute {sess} is not specified.
The command of disconnection/closing of the session: <disconnect path="/UI/VCAEngine/%2fserv%2fsess"/>		
sess	Setting the name of the session	Specify the name of the session from that it is made the disconnection or closing. Sessions, not the background, and to which none of the visualizers is not connected, are automatically closed.

The group request of the tree of widget libraries

In order to optimize the performance of local and especially network interaction the service function "/serv/wlbBr" and command of the query of the tree of widget libraries: <get path="/UI/VCAEngine/%2fserv%2fwlbBr"/> are provided. The result of the query is the tree with the elements of the libraries of widgets, tags *<wlb>*. Inside the tags of libraries of widgets are included: icon tag *<ico>* and widgets library tags *<w>*. Widgets tags, in their turn, contain the icon tag and tags of the child widgets *<cw>*.

4. Configuring the module via the control interface of OpenSCADA

Through the management interface of OpenSCADA, components that use it, can be configured from any system configurator OpenSCADA. This module provides an interface to access all of the data object of the VCA. Main inset of configuration page of the module provides access to widgets libraries and projects (Fig. 4.1). The inset "Sessions" provides access to opened sessions of projects (Fig. 4.2). To adjustment of the speech synthesis engine it is provided the relevant page (Fig. 4.3).
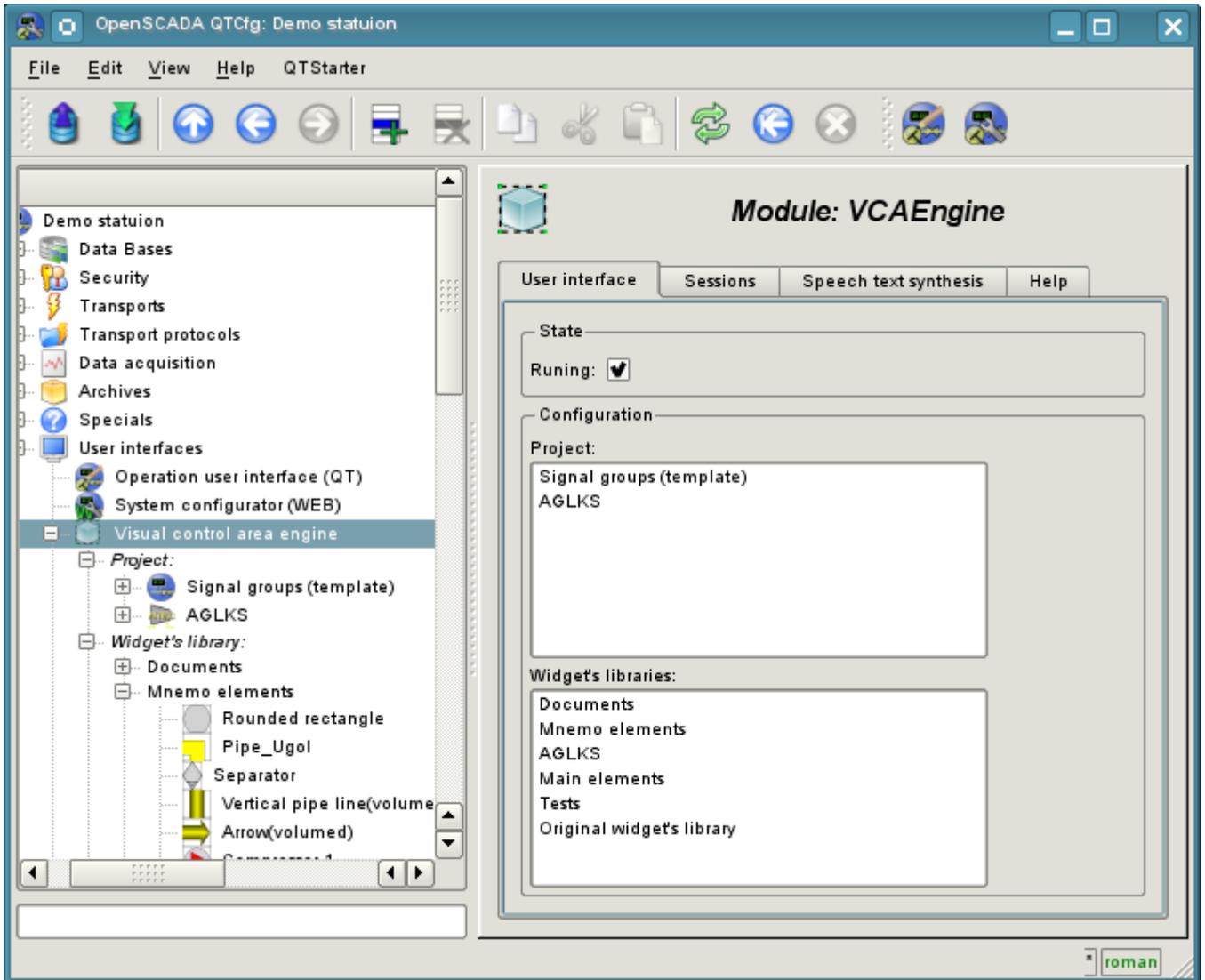
Fig.4.1 Main configuration page of the module.
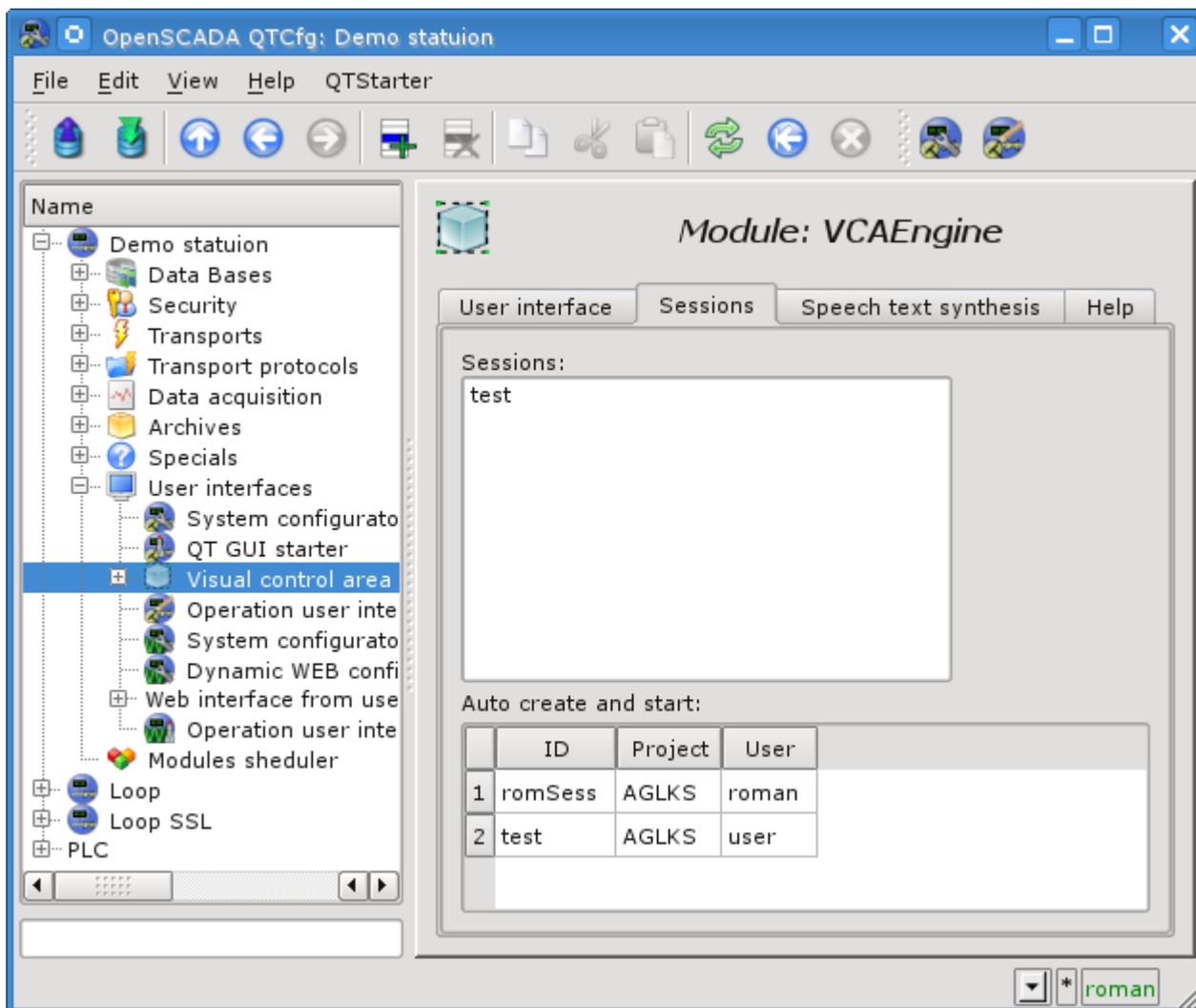
Fig.4.2 The inset "Sessions" of configuration page of the module.

In addition to the list of open sessions tab in Figure 4.2 contains a table with a list of sessions that must be created and run at boot time OpenSCADA. Creation of sessions through this tool can be useful for Web-based interface. In this case, when connecting Web-user data is ready and ensures the continuity of the formation of archival documents.
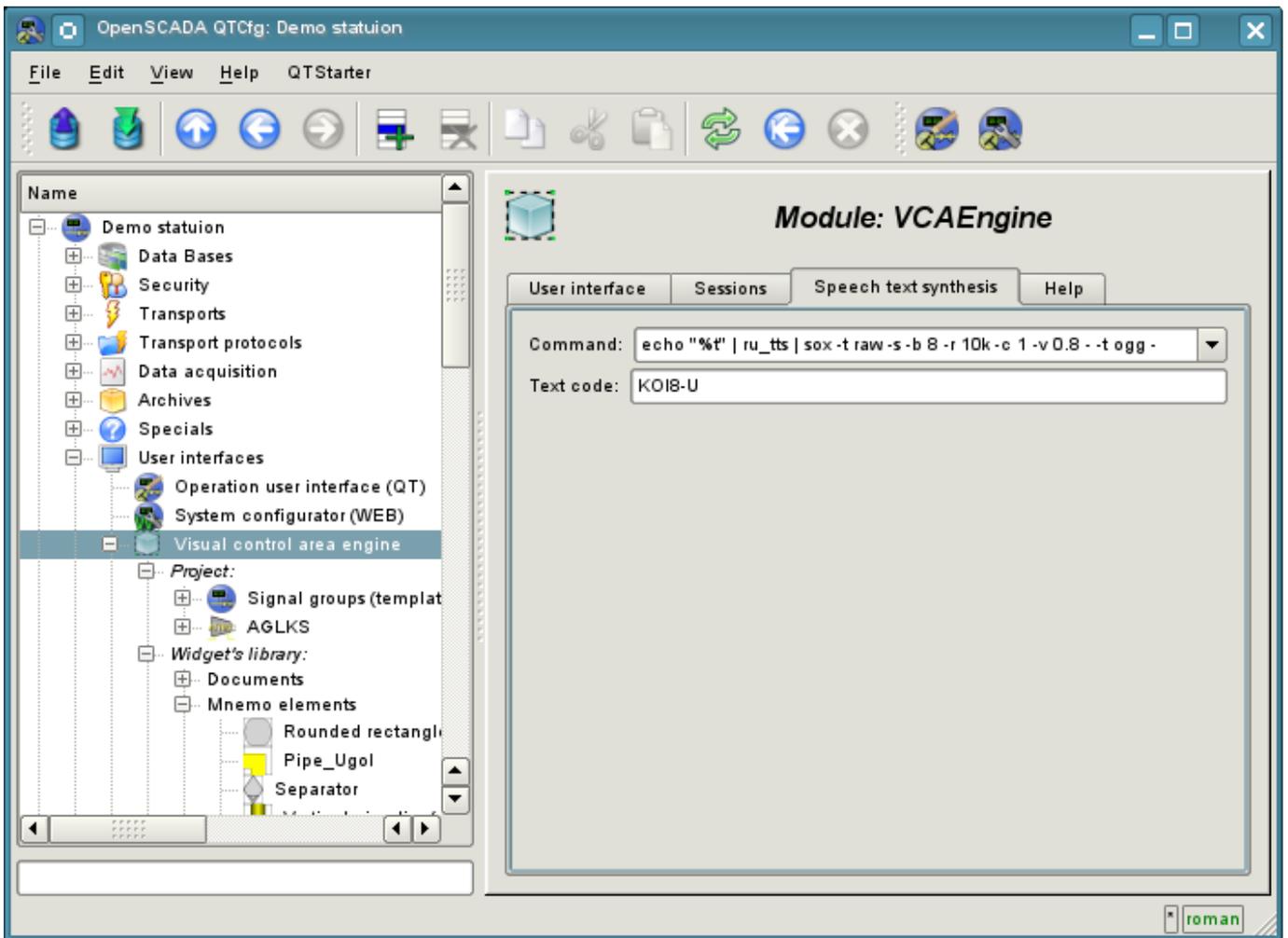
Fig.4.3 The inset for speech synthesis engine configuration.

The configuration of container widgets in the face of libraries and widget projects is done through pages in Fig. 4.4 (a project) and Fig.4.5 (a library of widgets). Widget library contains widgets, and the draft — page. Both types contain a tab container configuration Mime-data used widgets (Fig.4.6).
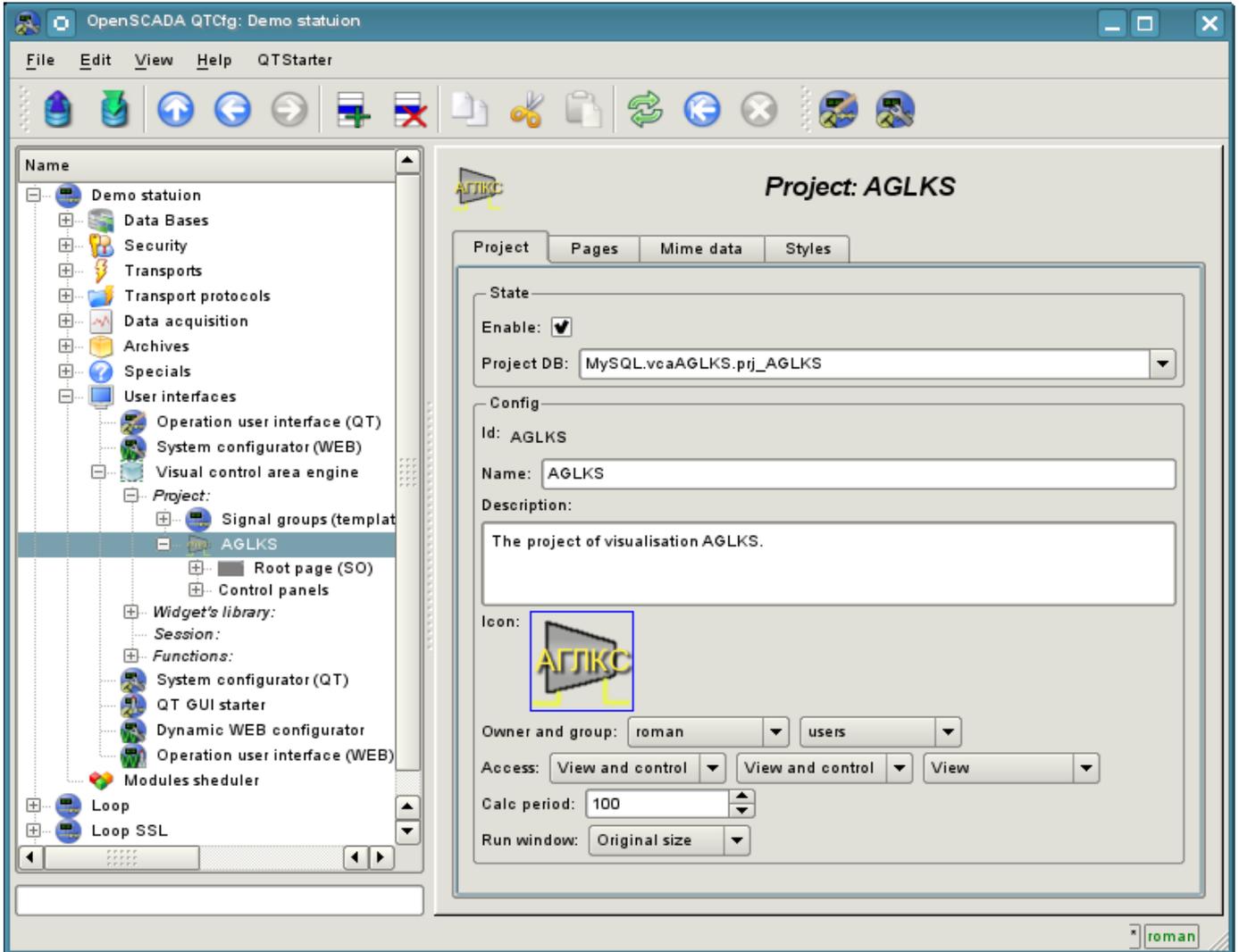
Fig.4.4 The configuration page of the projects.

From this page you can set:
- The state of the container, namely: «Enabled», the name of the database containing the configuration, the owner and group of the container.
- Id, name, description and icon of the container.
- Access rights to the container.
- The period for computing of the sessions based on the given project.
- Method for opening the main window of execution (original size, maximization and the full screen).
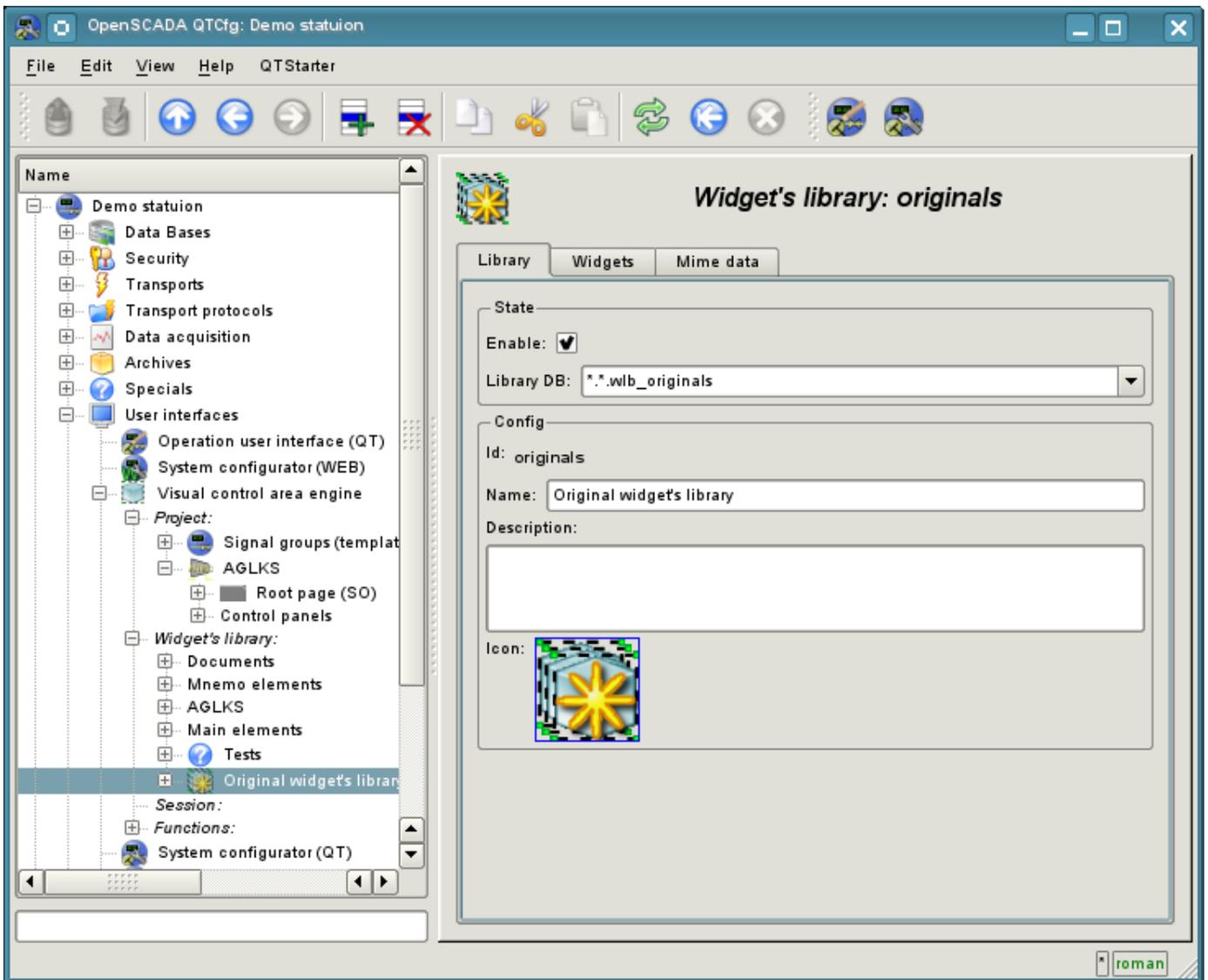
Fig.4.5 The configuration page of the widgets libraries.

From this page you can set:
- The state of the container, namely: «Enabled», the name of the database containing the configuration.
- Id, name, description and icon of the container.
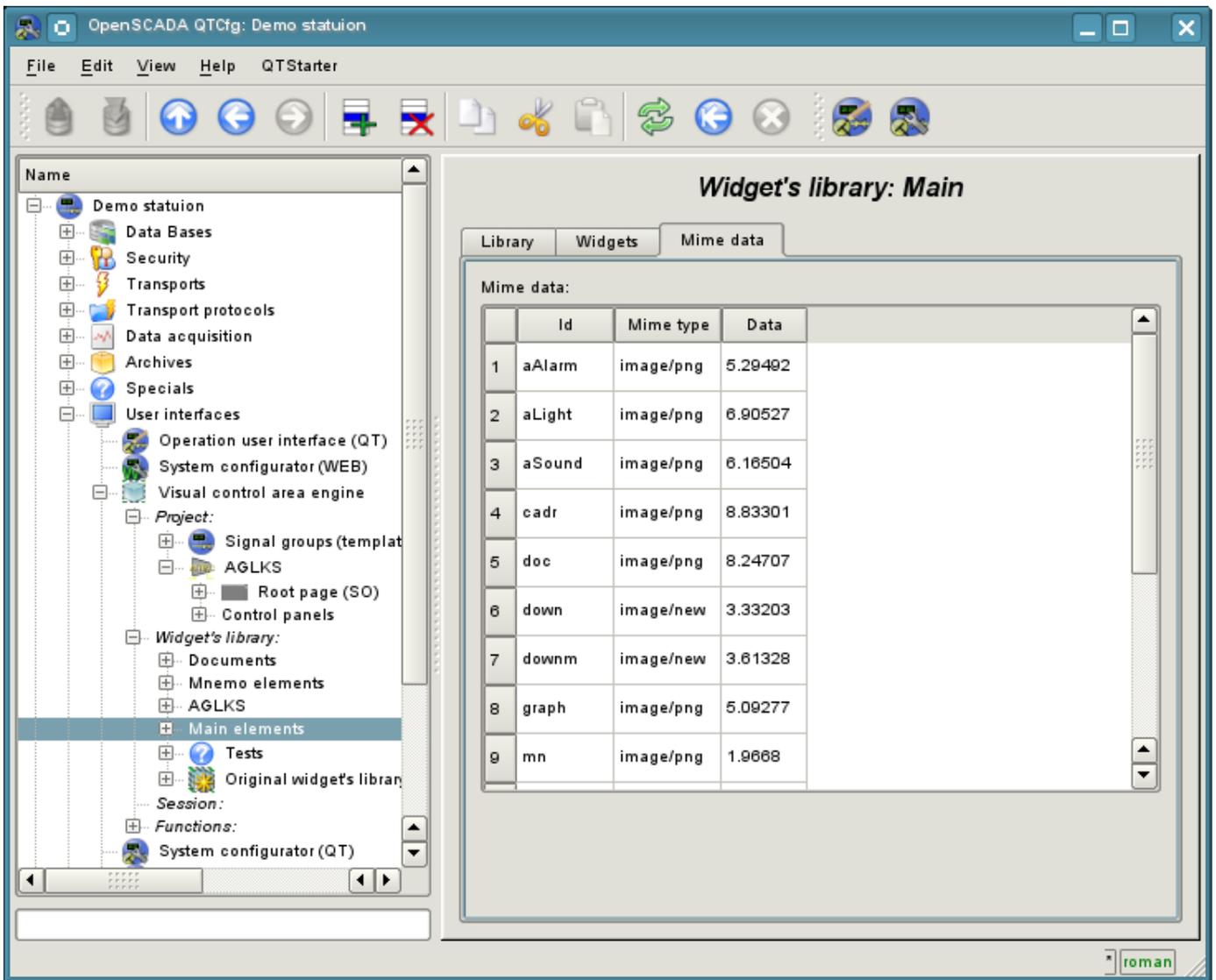
Fig.4.6 The configuration tab of the Mime-data of the container.

Configuration of the project session differs significantly from the configuration of the project (Fig. 4.7), but also contains pages of the project.
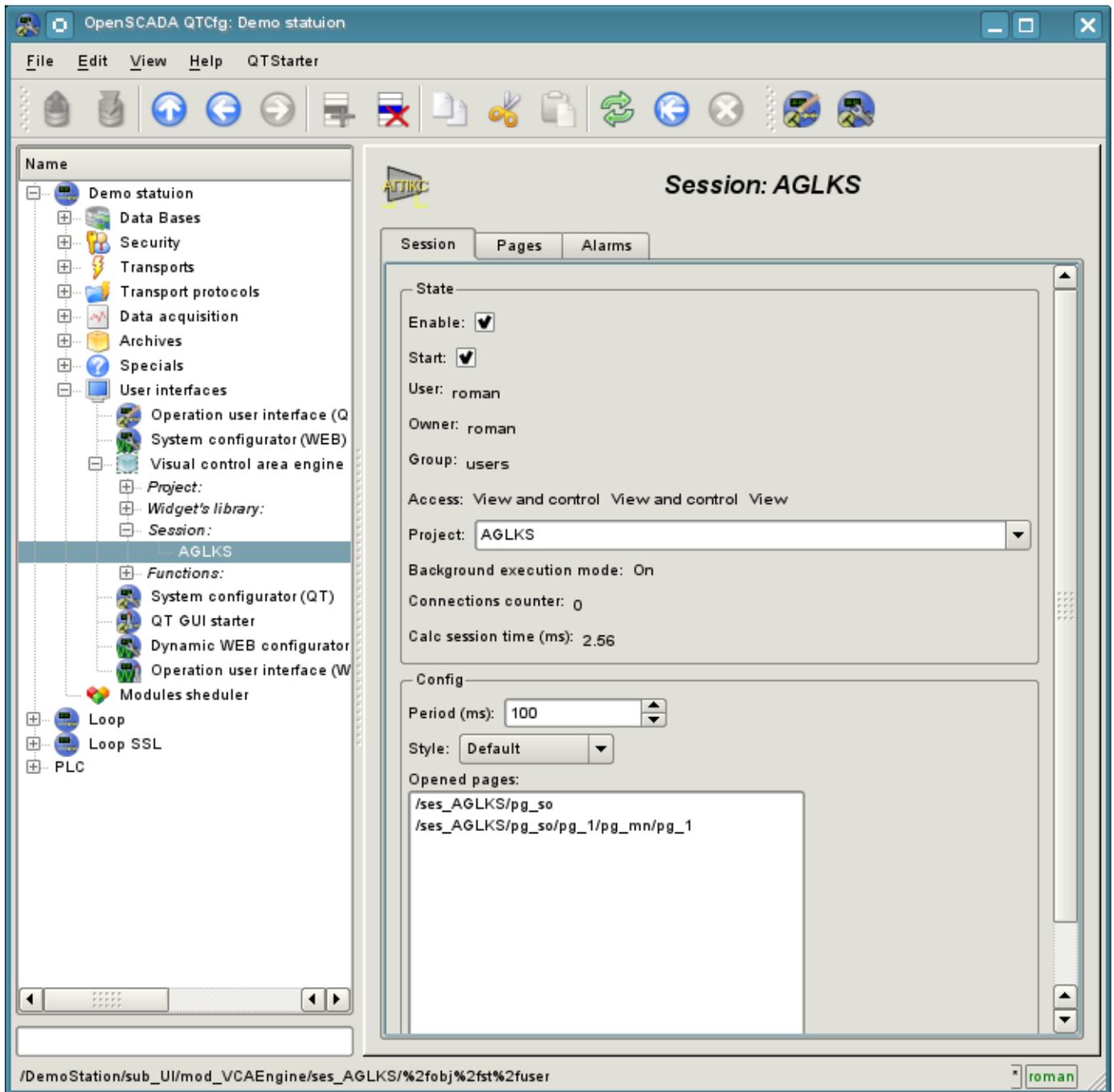
Fig.4.7 The configuration page of the sessions of the projects.

From this page you can set:
- The state of the session, namely: "Enabled", "Started", the user, owner, user group, access, source project, mode of execution in the background, the counter of client connections and execution time of the session.
- Period of calculation of the session.
- The list of opened pages.

The configuration pages of visual elements, placed in different containers, may be very different, but this difference is the presence or absence of individual tabs. The main tab of visual elements in fact is the same everywhere, differing in one configuration field (Fig. 4.8). The pages contains the tabs of the child pages and embedded widget. Container widgets contains the tab of the embedded widgets. All visual elements contain attributes tab (Figure 4.9), except the logical containers of the projects. Elements, at the level of which it is possible to build the user procedure and to determine the links, contain the tabs "Process" (Fig. 4.10) and "Links" (Fig.4.11).
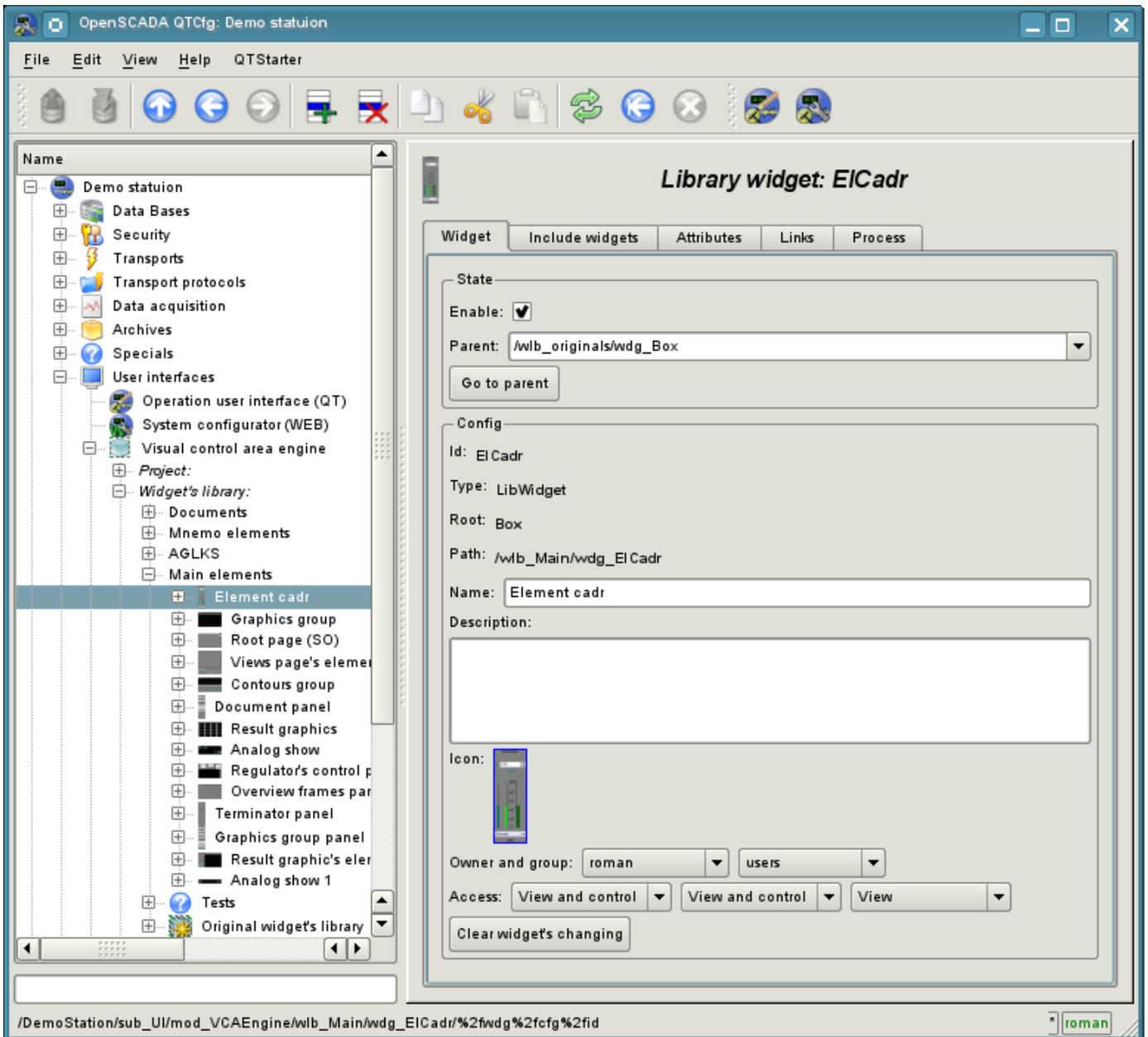
Fig.4.8 Main tab of the configuration of visual elements.

From this page you can set:
- The state of element, namely: «Enabled», parent element and jump to it. For the page in the state it is indicate the type of the page.
- Id, type, root, path, name, description and icon of the element.
- The owner, a group of users and access rights to the element.
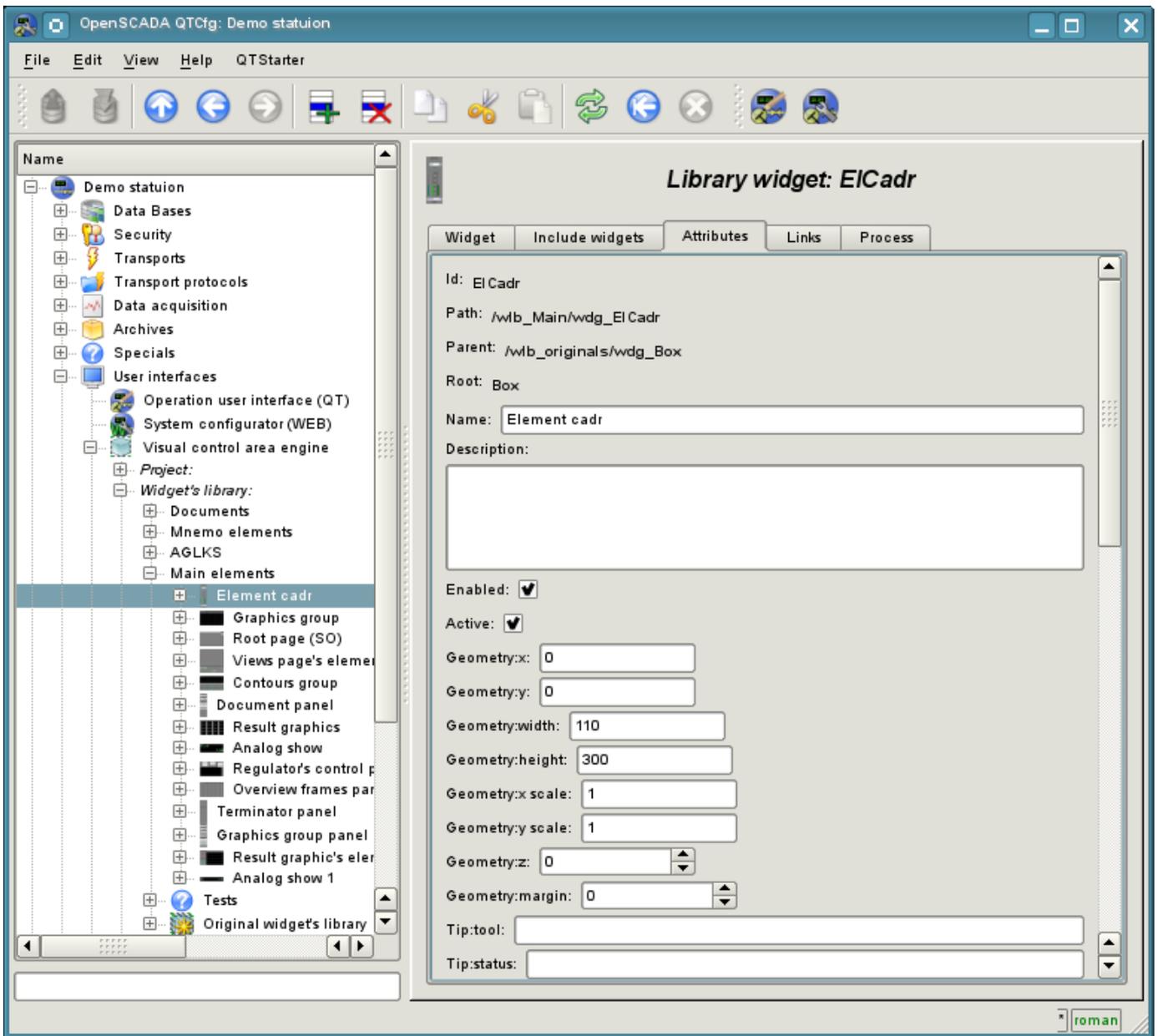
Fig.4.9 Tab of the attributes of visual elements.
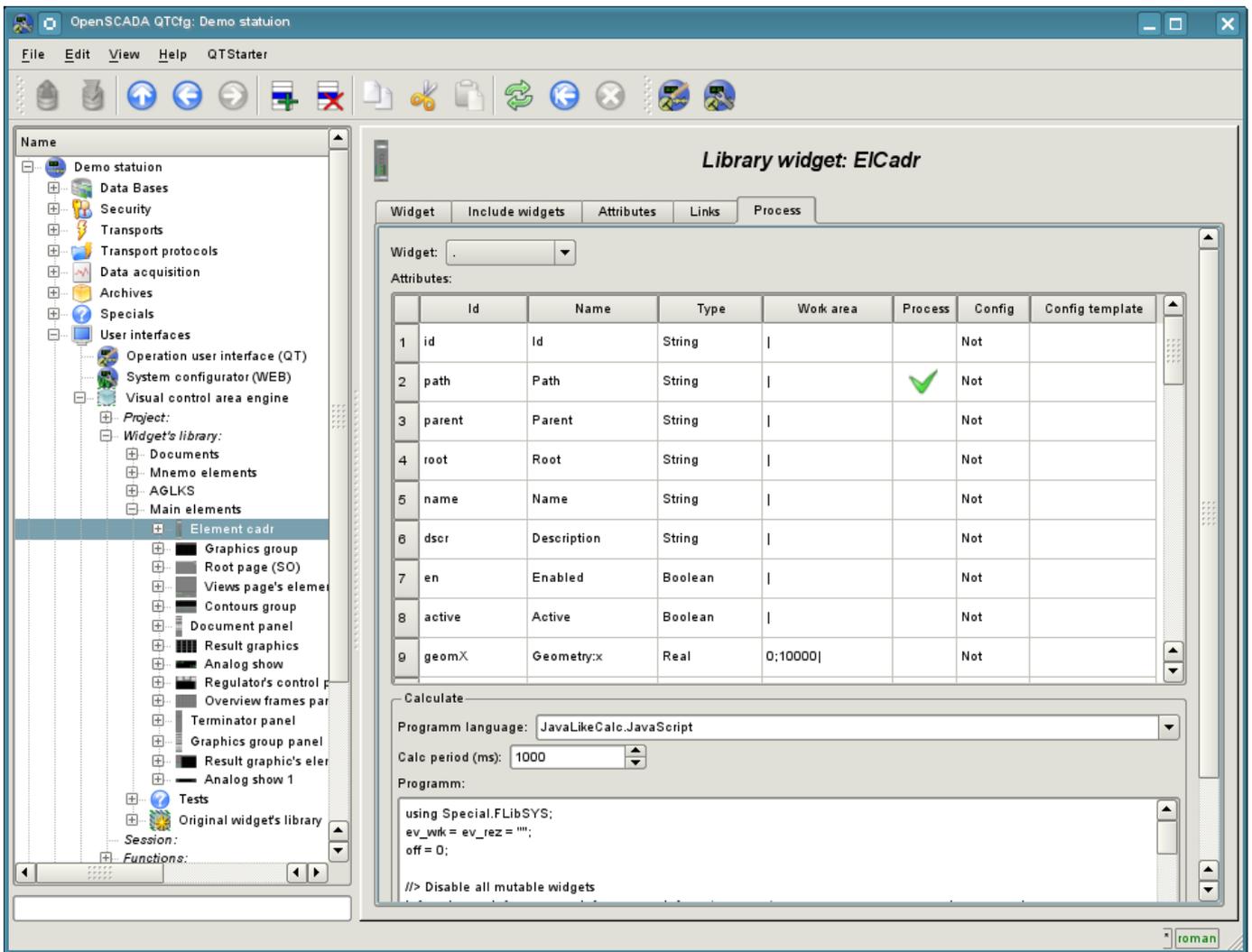
Fig.4.10 Tab of the processing of visual elements.
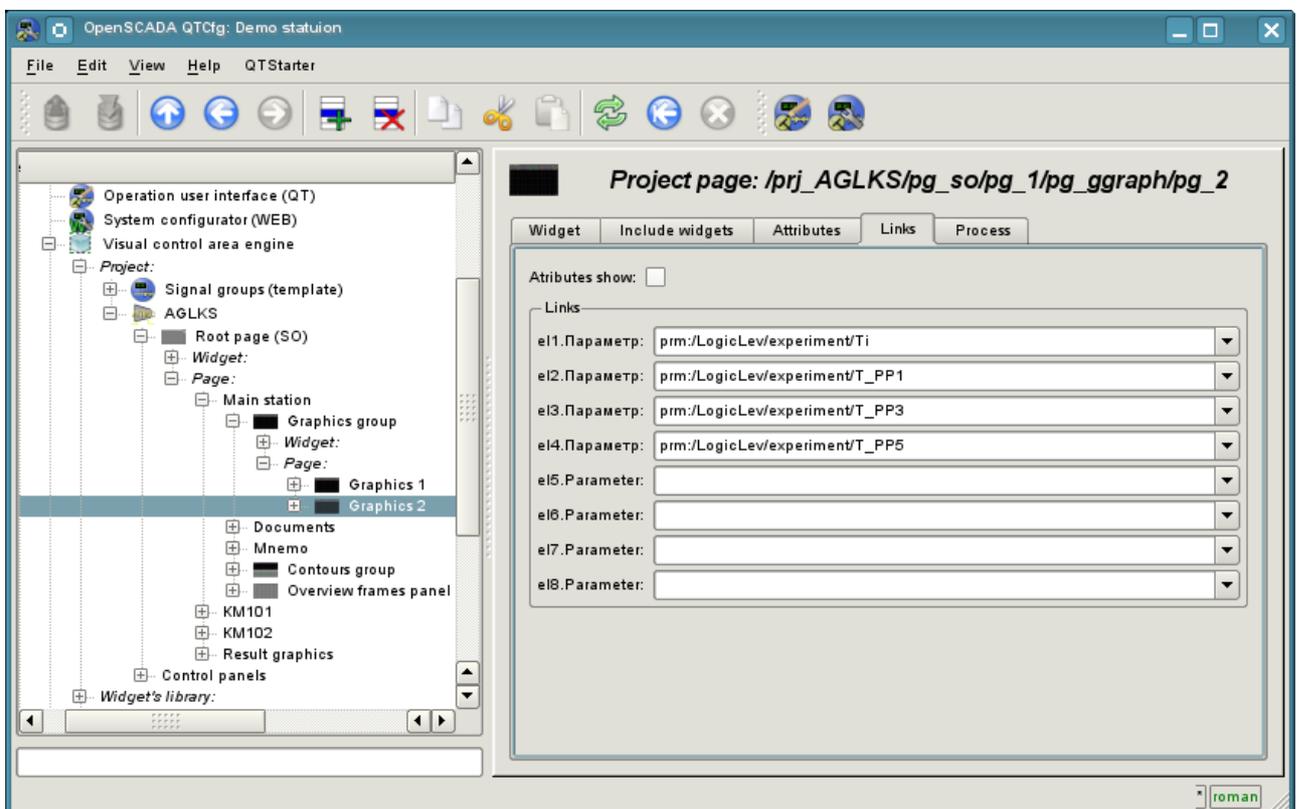
Fig.4.11 Tab of the links of the visual elements.