RadiCS Platform software

u7 Integrated Development Environment

User Manual

[1. u7ide program’s functions and features 3](#_Toc422835502)

[2. Installing the program 4](#_Toc422835503)

[3. Running and setting up the u7ide program 7](#_Toc422835504)

[4. Application Project Management 9](#_Toc422835505)

[5. Hardware configuration 14](#_Toc422835506)

[6. Application logic signal editor. Adding, editing and deleting 22](#_Toc422835507)

[7. AFB library 28](#_Toc422835508)

[8. Application Logic Editor 31](#_Toc422835509)

[9. Редактор технологических видеокадров (Workflow schemes) 41](#_Toc422835510)

[10. Редактор диагностических видеокадров (Diag schemes) 41](#_Toc422835511)

[11. Project building 42](#_Toc422835512)

# u7ide program’s functions and features

* 1. The u7ide program is an integrated development environment for the RadiCS Platform. It is the main CAD part that provides tools for FSC chassis configuration, application logic design and its compilation, visualization design and SCADA software configuration.
  2. The program uses client-server architecture. All project data is stored in a database on local or remote machine. The program connects to a server and works with a database.

The u7ide program allows many users to work with the application project at the same time. Every user has a login and a password and can change allowed part of the project.

The application project database keeps the history of all changes. To edit some project data, user should make a “check-out” operation for the item needed to be edited. If some other user has already checked out this item, the operation will fail. After finishing the editing process, user should perform a “check-in” operation to commit changes to the application project database. User can have authorized access to any previously committed data, but cannot change it.

After project building the program creates a set of files. Some of them are used by other software, and some of them contain hardware information and must be programmed to logic modules’ flash memory.

* 1. The u7ide program has following main functions:

– application project management (creating, editing and deleting);

– project users management;

– designing an application hardware configuration;

– application logic signals editing;

– application functional block library management;

– application logic designing;

– workflow schemes designing;

– diagnostic schemes designing;

– building the application project.

# Installing the program

* 1. System requirements
     1. Hardware and operating system requirements:

– x86 or x86-64 based workstation with at least 8 GB RAM memory;

– TCP/IP-based network;

– Microsoft Windows 7, Microsoft Windows 8.1, Microsoft Windows Server 2008, Linux OpenSuse 13.2, Linux Fedora 22.

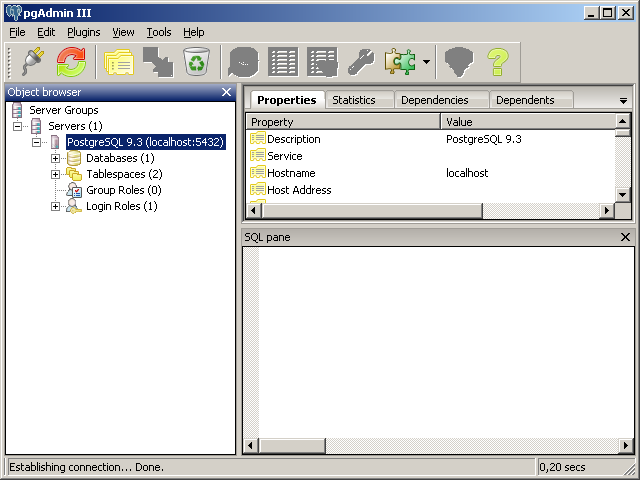
* + 1. The program requires following software and libraries to be installed:

– Qt Library (version 5.3.1) for client part;

– DLL module VFrame30.dll;

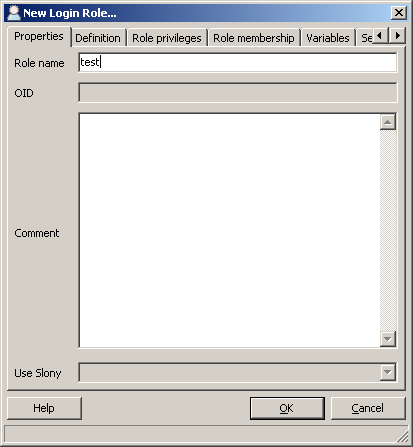
– PostgreSQL Database Management System (version 9.3.5) for server part.

* + 1. The program requires additional libraries path to be installed into PATH system variable. For example, in Windows-based OS add the following text to this variable: ";C:\Qt\Qt5.3.1\5.3\msvc2013\_64\_opengl\bin;C:\ PostgreSQL\9.3\bin".
  1. PostgreSQL Server setup
     1. Install and run PostgreSQL server.
     2. To configure the server, run the program called “pgAdmin III” and enter the superuser password supplied during server installation. The following screen will appear.



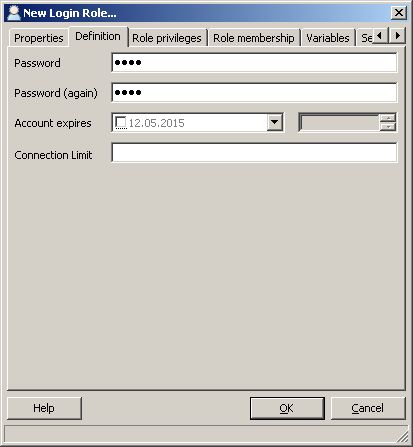
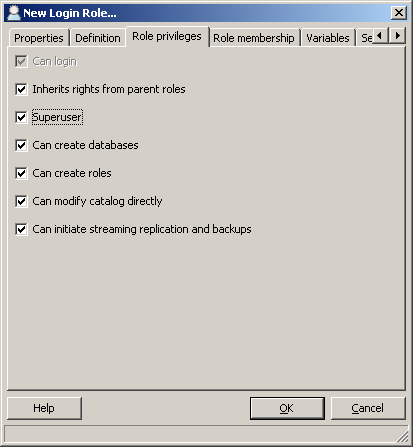
|  |  |
| --- | --- |
| Figure | 2.1 |

* + 1. Choose “Login Roles” item, select “New Login Role…” menu item and create a user with full access to the database. This login will be used by the u7ide program.



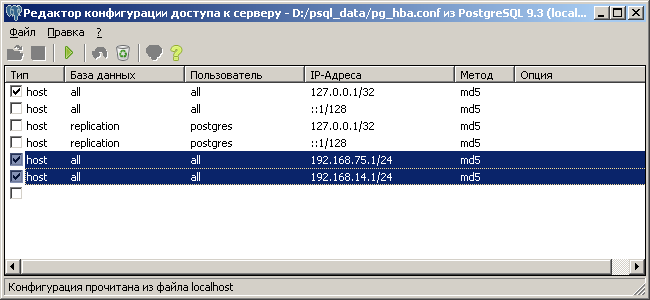
|  |  |
| --- | --- |
| Figure | 2.2 |

* + 1. Supply a password and full access rights for this user.

|  |  |
| --- | --- |
| Figure | 2.3 |

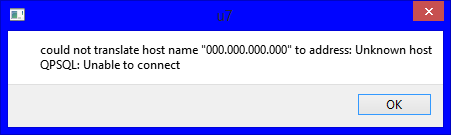
* + 1. Choose the “pg\_hba.conf” menu item in the “Tools” - “Server Configuration” menu and add IP address masks to allow remote hosts to connect to this server. For example, for IP range “192.168.75.1 – 192.168.75.254” write “192.168.75.1/24”.



|  |  |
| --- | --- |
| Figure | 2.4 |

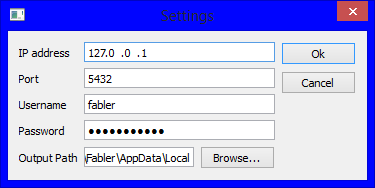
# Running and setting up the u7ide program

* 1. Starting the u7ide program
     1. After starting up, u7ide program connects to the server. If server address is not set or server is unavailable, the program shows an appropriate message (fig. 3.1).



|  |  |
| --- | --- |
| Figure | 3. |

To change the database server address, open the "Tools" menu in the main window and select the "Settings" item. Sample of "Settings" window is shown on figure 3.2.



|  |  |
| --- | --- |
| Figure | 3. |

The «Settings» window contains following items:

– IP address - server address to connect for;

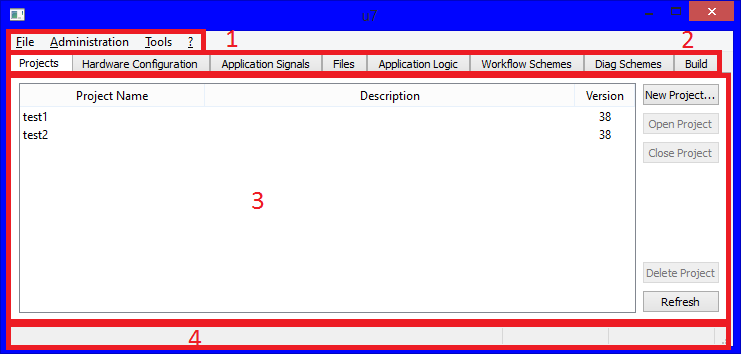
– Port - server port to connect for;

– Username - Server database user name;

– Password - User's database password;

– Output path - path, where program will store the build result.

* 1. Program interface
     1. The sample of program’s main window is shown on the figure 3.3.



|  |  |
| --- | --- |
| Figure | 3. |

Program's main window contains:

– main menu(1);

– tabs (2);

– program workspace(3);

– program status bar.

* + 1. Main menu contains following items:

– Files (contains the exit command);

– Administration (project logs and user administration tools);

– Tools (program’s different functions);

– ? (program’s help and additional debug commands).

* + 1. The tab panel contains following tabs:

– Projects (application project managing);

– Hardware configuration (presets, hardware and software configuration);

– Application signals (application signals management);

– Files (project files management for advanced tasks);

– Application Logic (application logic schemes designer);

– Workflow Schemes (visualization workflow schemes designer);

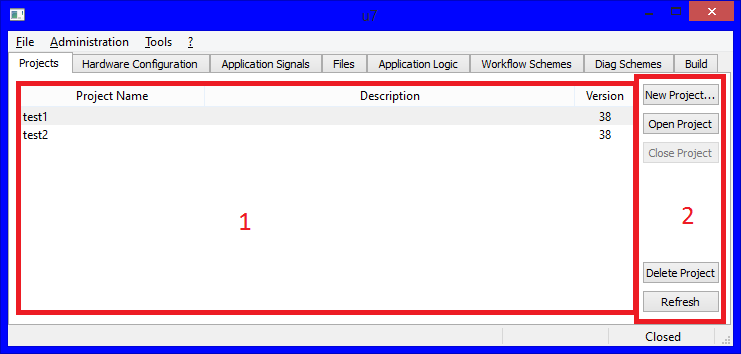
– Diag Schemes (diagnostics schemes designer).

– Build (project build).

* + 1. Workspace can show different information depend on selected tab.
    2. Status bar shows connection and other operations status.

# Application Project Management

* 1. Projects tab is shown on the figure 4.1.



|  |  |
| --- | --- |
| Figure | 4.1 |

Project panel contains:

– application project list (1);

– control panel(2).

The application project list has following columns:

– Project Name (name of the project);

– Description (project description);

– Version (application database project version).

Control Panel contains buttons to work with projects:

– “New Project…” (create a project);

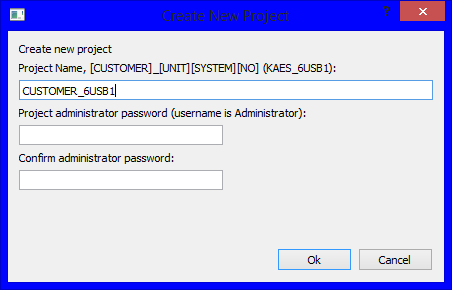
– “Open Project…” (open selected project);

– “Close Project” (close opened project);

– “Delete Project” (remove selected project from server);

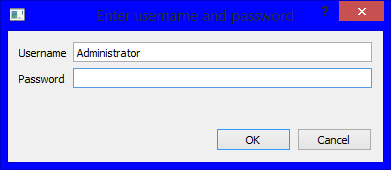
– “Refresh” (refresh the project list);

* 1. To create a new application project push “New Project…” button. The “Create New Project” window will appear on the screen (fig. 4.2). Fill the “Project Name” and “Project administrator password” (key-word to access the project) fields. Press “Ok” to accept changes, or “Cancel” to decline.

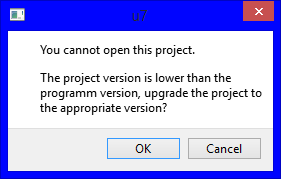


|  |  |
| --- | --- |
| Figure | 4. |

* 1. To open project press "Open Project…" button. Enter the login and password (fig. 4.3). If project's version is different from u7ide project database supported version, an upgrade project prompt message will appear (fig. 4.4). Press "OK" button to upgrade project before opening. The upgrade is possible only to higher version, version downgrade is impossible. It is impossible to open or upgrade the application database project, if its version is higher than version supported by u7ide program.

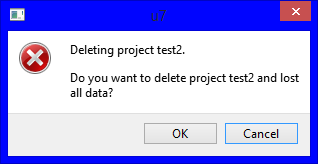


|  |  |
| --- | --- |
| Figure | 4. |



|  |  |
| --- | --- |
| Figure | 4.4 |

* 1. To close selected project press "Close Project" button. The project will be closed. If there is any unsaved data, the user will be prompted to save it.
  2. To remove the selected project, press "Delete Project" button. New dialog window will be shown (fig. 4.5). Press "OK" button to remove all project data from server, or "Cancel" to decline.



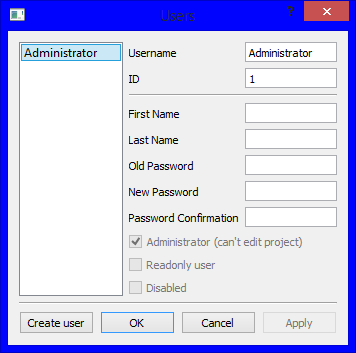
|  |  |
| --- | --- |
| Figure | 4. |

ATTENTION: all connections to this project from all users must be closed.

* 1. To refresh the list of projects press the "Refresh" button.
  2. To manage project users, select "Users" item from "Administration" in main menu.

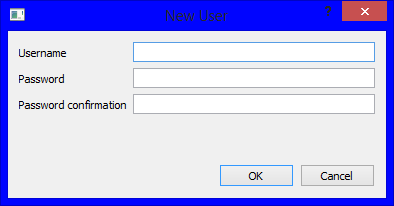
ATTENTION: user management is possible only in opened project.

A dialog window will be shown (fig. 4.6)



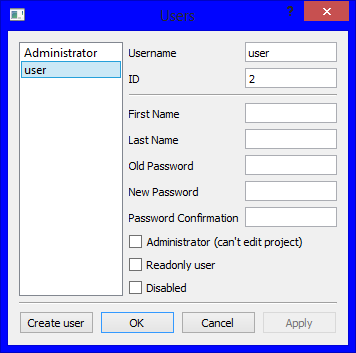
|  |  |
| --- | --- |
| Figure | 4.6 |

Here shown all project users. To create a new user press "Create user" button. A new user dialog window will be shown (fig. 4.7).



|  |  |
| --- | --- |
| Figure | 4.7 |

Fill the username and password. After pressing "OK" new user will be created, and user list in previous window will be updated (fig. 4.8).



|  |  |
| --- | --- |
| Figure | 4.8 |

User’s record has following properties:

– “First Name”;

– “Last Name”;

– password;

– access rights.

Access rights can be controlled by following options:

– Administrator (Read-Write administrator rights);

– Readonly user (Read-Only user);

– Disabled (Blocked user);

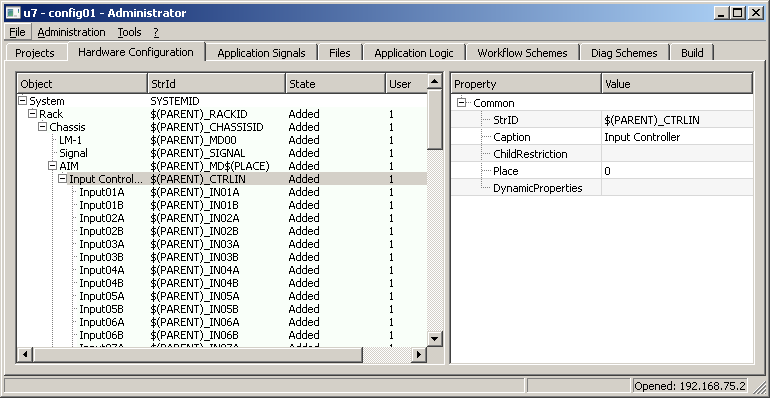
To accept changes press "Apply" button, or "Cancel" to discard them.

# Hardware configuration

* 1. Hardware configuration is a part of a project that describes the structure of the system. It contains different kinds of objects: system, rack, chassis, modules, controllers and signals and has hierarchical structure. A top-level object is always system that contains a set of child objects called racks. A rack contains child objects called chassis. A chassis contains a set of modules.

Every module can contain two types of child objects: controllers and signals. Controllers can contain only signals.

The “Hardware configuration” tab with sample configuration tree is shown on the figure 5.1.



|  |  |
| --- | --- |
| Figure | 5.1 |

Each hardware configuration object has a set of properties. Different types of objects may have same and different properties.

The most important common properties are:

– StrID – a string identifier of an object. Usually it has common structure: $(PARENT)\_ID, where $(PARENT) is the identifier of the parent object, and ID is an own identifier. When the project compiles, $(PARENT) macro is replaced by the real parent identifiers. $(PLACE) macro means an object place (Place property) and also is replaced by the real number.

– Caption – name of the object that contains a string description of the object;

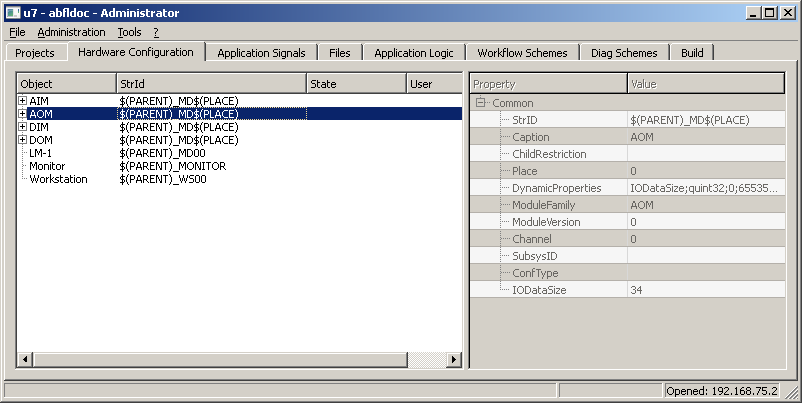
– Place – an object position. This property is important for racks, chassis, modules and signals. It determines a chassis number in a rack, a module’s position in a chassis, a number of a physical input or output for signal.

Some examples of identifiers (StrID) are shown in the following table. The first column is an identifier with a macro, and the second column is an identifier with macro replaced by parent id.

|  |  |  |
| --- | --- | --- |
| Id with a macro, entered in the hardware configurator | Id with expanded macro during project building | Object type |
| USB | USB | System ID |
| $(PARENT)\_RACK01 | USB\_RACK01  ($(PARENT) was replaced with USB) | Rack ID |
| $(PARENT)\_CHAS00 | USB\_RACK01\_CHAS00  ($(PARENT) was replaced with USB\_RACK01) | Chassis ID |
| $(PARENT)\_MD$(PLACE) | USB\_RACK01\_CHAS00\_MD00  ($(PARENT) was replaced with USB\_RACK01\_CHAS00,  ($(PLACE) was replaced with 00) | Module ID |
| $(PARENT)\_CTRLIN | USB\_RACK01\_CHAS00\_MD00\_CTRLIN  ($(PARENT) was replaced with USB\_RACK01\_CHAS00\_MD00) | Controller ID |
| $(PARENT)\_IN01 | USB\_RACK01\_CHAS00\_MD00\_CTRLIN\_IN01  ($(PARENT) was replaced  with USB\_RACK01\_CHAS00\_MD00\_CTRLIN) | Signal ID |

* 1. Hardware configuration database contains two parts: hardware configuration itself and a list of presets. User can switch between editing modes by right-clicking the mouse on the tree area and choosing the “Switch to Preset” or “Switch to Configurations” menu commands.

Preset editing window is shown on the figure 5.2.

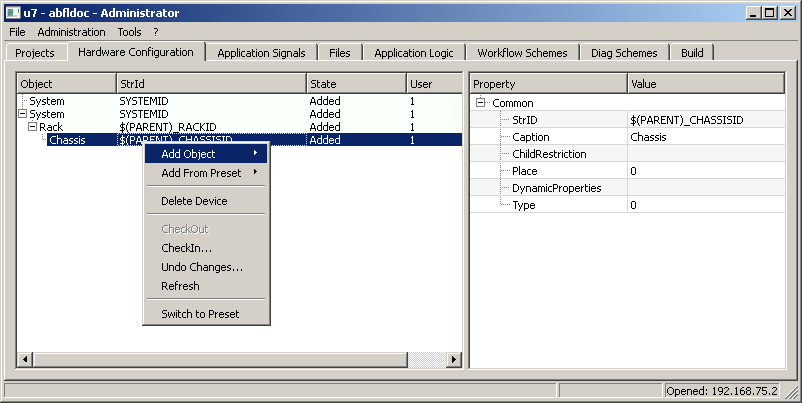


|  |  |
| --- | --- |
| Figure | 5.2 |

Preset is an object that can contain other child objects. For example, a module is a preset containing controllers and signals. Preset can be added to the hardware configuration, and all its child objects are also added automatically.

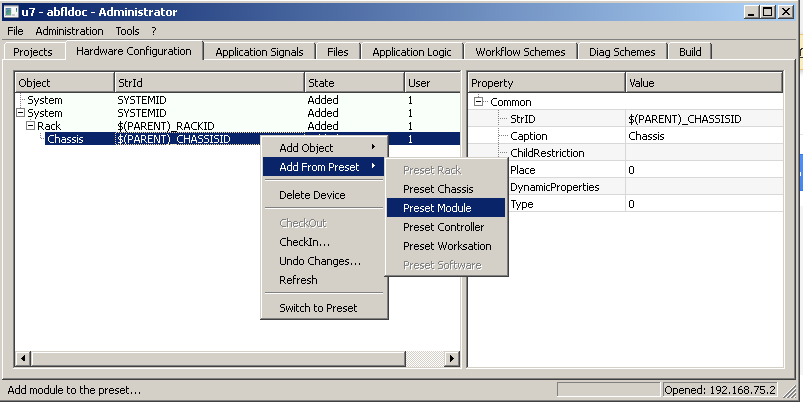
The hardware configuration may contain many instances of presets, for example, many same modules. If it is necessary to make changes in a module, these changes should be done in the preset library, after user command all changes will be applied to all instances of this module.

* 1. Editing process in both hardware configuration and presets editing modes is the same. On the left side of the “Hardware configuration” tab there is an object tree, and on the right side there is a properties window.
     1. To add an object to a tree, select the parent object and click the right mouse button on it, choose kind of an object in the “Add Object” submenu, as displayed on the figure 5.3. Note that objects should be added following the hierarchy: rack is a child of a system, chassis is a child of a rack and so on.

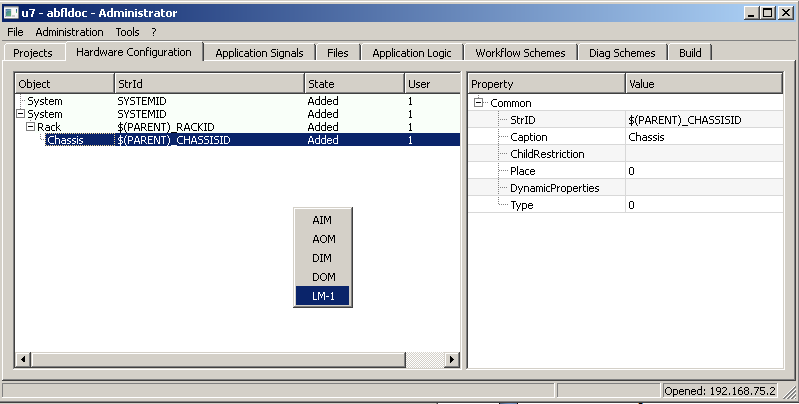


|  |  |
| --- | --- |
| Figure | 5.3 |

* + 1. To add a preset to hardware configuration select parent object and choose preset type in the “Add From Preset” submenu, as shown on a figure 5.4. Choose a preset from this menu, as shown on the figure 5.5.



|  |  |
| --- | --- |
| Figure | 5.4 |



|  |  |
| --- | --- |
| Figure | 5.5 |

* + 1. To delete an object, select it and choose the “Delete Object” command.
    2. Before editing an existing object, it must be checked out with the “CheckOut” menu command. After editing is finished, it must be checked it in by the “CheckIn” menu command. Thus, changes will be placed to the project database. To undo changes that were not checked in yet, use the “Undo” command.
    3. The “Refresh” command is used to reload hardware configuration data from the project database.
    4. To edit object properties, select it in the tree and make a check out operation if necessary. On the right side of the window the properties list will appear. To change a property, click on it in a table and enter a value in a “Value” column.
    5. System object has following properties:

|  |  |
| --- | --- |
| StrID | String identifier |
| Caption | Caption |
| ChildRestriction | Reserved |
| Place | Reserved |
| DynamicProperties | Dynamic properties list |

* + 1. Rack object has following properties:

|  |  |
| --- | --- |
| StrID | String identifier, may contain $(PARENT) macro |
| Caption | Caption |
| ChildRestriction | Reserved |
| Place | Reserved |
| DynamicProperties | Dynamic properties list |

* + 1. Chassis object has following properties:

|  |  |
| --- | --- |
| StrID | String identifier, may contain $(PARENT) macro |
| Caption | Caption |
| ChildRestriction | Reserved |
| Place | Reserved |
| DynamicProperties | Dynamic properties list |
| Type | Chassis type |

* + 1. Module object has following properties:

|  |  |
| --- | --- |
| StrID | String identifier |
| Caption | Caption |
| ChildRestriction | Reserved |
| Place | Module place in a chassis, 1-based. |
| DynamicProperties | Dynamic properties list |
| ModuleFamily | A family of a module, for example “AIM” or “LM-1” |
| ModuleVersion | Module software version |
| Channel | Number of a system channel, which this module belongs to. Used only in LM modules family. |
| SubsysID | Subsystem name |
| ConfType | Reserved |

NOTE – depending on the module type, other properties can exist.

* + 1. Controller object has following properties:

|  |  |
| --- | --- |
| StrID | String identifier, may contain $(PARENT) macro |
| Caption | Caption |
| ChildRestriction | Reserved |
| Place | Reserved |
| DynamicProperties | Dynamic properties list |

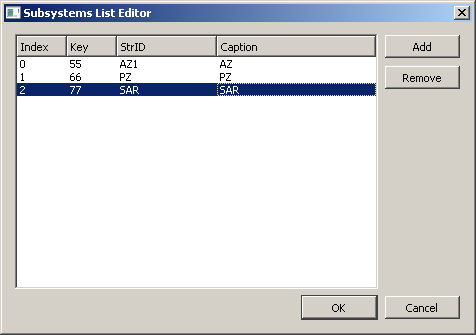
* + 1. Signal object has following properties:

|  |  |
| --- | --- |
| StrID | String identifier, may contain $(PARENT) macro |
| Caption | Caption |
| ChildRestriction | Reserved |
| Place | Index of the signal, may be used in some families of modules |
| DynamicProperties | Dynamic properties list |
| Type | Signal type: analog or discrete |
| Function | Signal function: input, output, validity, diagnostic etc. |
| ByteOrder | Bytes order: big endian or little endian |
| Format | Data format: signed, unsigned etc. |
| Size | Size, in bits, of a signal |
| ValidityOffset | Validity offset in a data packet |
| ValidityBit | Validity bit number in a data packet |
| ValueOffset | Value offset in a data packet |
| ValueBit | Value bit number in a data packet (always 0 for analog signals, because they have whole word or byte) |

NOTE – depending on the signal type, other properties can exist.

* 1. After adding a module to the hardware configuration it is necessary to set Place property. For LM is also necessary to set an existing SubsysID property. Without this, build errors will appear.
  2. SubsysID is an identifier, which shows the subsystem key for an LM module. It can be a number with range from 1 to 63. This number is hardwired on the chassis and determines module’s LM function. A set of subsystems must be created before building the project.

To edit subsystems set choose command “Subsystem List Editor” in “Tools” menu. Subsystems list editor window is shown on a figure 5.6. To add and remove items, use “Add” and “Remove” buttons. To edit values in “Key”, “SubsysID” and “Caption” columns, double click left mouse button on them and enter the data.

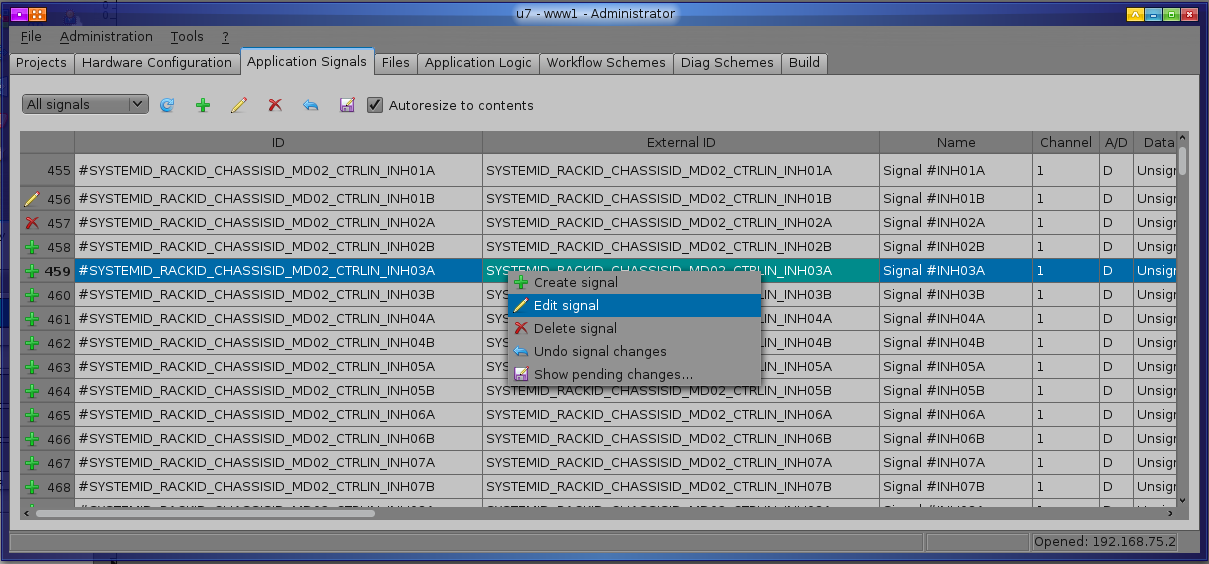


|  |  |
| --- | --- |
| Figure | 5.6 |

# Application logic signal editor. Adding, editing and deleting

* 1. Common description

Application signals describe signal parameters used in the application logic schemes. The signals correspond to application Input, Output of modules or intermediate values in the calculations. Application signals tab is shown in Figure

****

For correct the association signal and input or output module, user should set the «Device ID» signal property equal to «Str ID» property input unit. These signals can be used for drawing schemes in the application logic designer. Display panel has a combobox to select the type of signal (analog / discrete) for a more convenient signal presentation. While editing a signal all changes are saved in a preparation state and they can be easily canceled until committing the changes (command "Checkin" used for that). After checkin a new signal state is available to other users of the database. For all uncommitted changes there is icon in the left header column drawn as icon of uncommitted operation.

* 1. Application signals operations

As shown in figure 6.1 signals can be carried out as:

* Create signal
* Edit signal
* Delete signal
* Undo signal changes
* Checkin signal changes

These operations are available via the context menu and on the panel in the form of icons. Also when user double-clicks on any field signal the signal properties editing dialog opens. In addition to the basic operations from the context menu on the panel there is an icon, when clicked, the list of signals will be reread from the database. This operation is useful to carry out in case another user has committed the changes and current user want to get them.

* + 1. Create signal

When user selects this option, adding settings window opens, as shown in figure



To create multiple signals user should specify device identifier (optional), type of signal, the number of channels and the number of signals. After user clicks OK, signal properties dialog appears, described in paragraph 6.3. After user clicks OK in the signal properties dialog specified amount of signals will be generated. For a uniqueness signal ID is appended by the signal number (if the number of signals is greater than 1) and channel number (if the number of channels is greater than 1) and signal name is appended by name of the device identifier (if it was specified) and the signal number (if the number of signals is greater than 1).

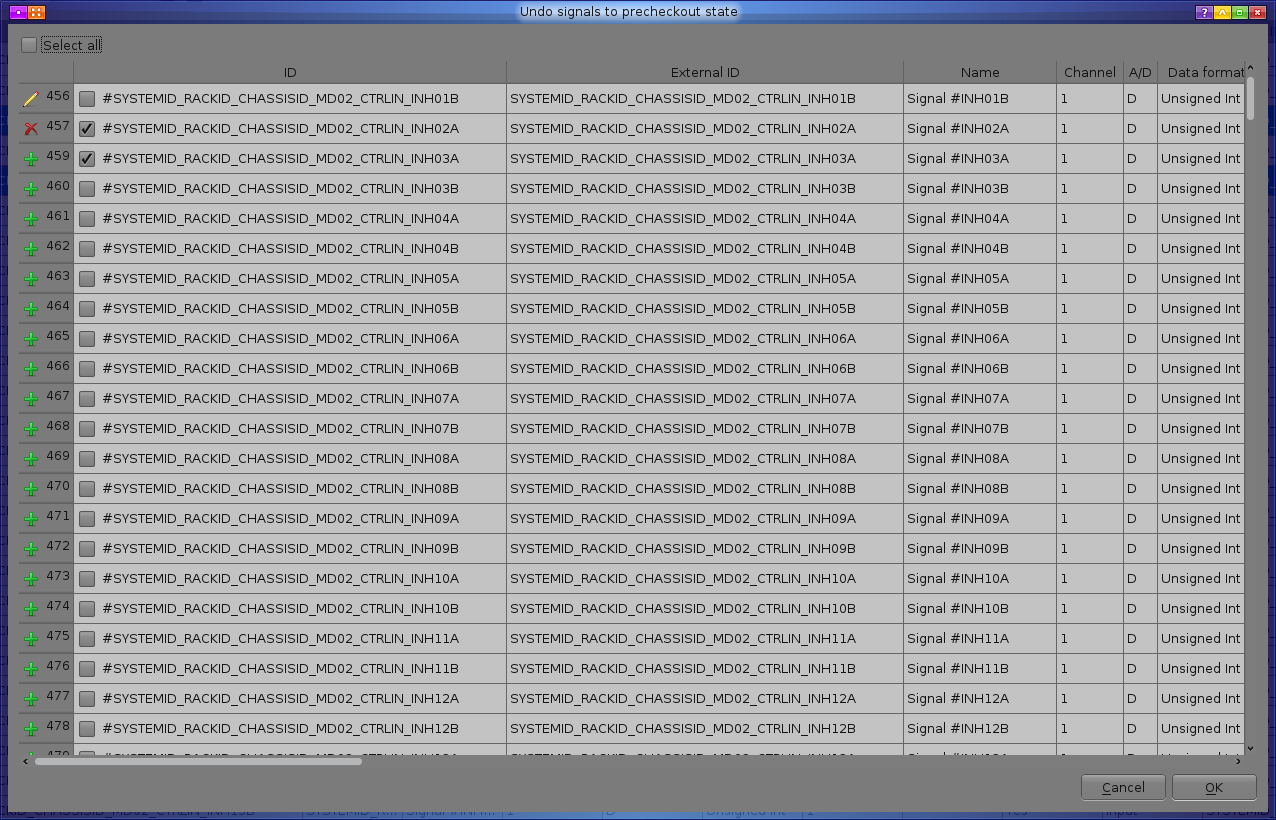
* + 1. Edit signal.

When user double-clicks on any field signal or when user selects "Edit Signal" signal properties dialog opens. When user clicks OK the properties dialog will automatically mark the signal as a signal having uncommitted changes. In the future it will be necessary to commit or undo the changes.

* + 1. Delete signal

If the signal has been added, without committing changes - it will be deleted (will disappear) in all other cases it will be marked as deleted for a commit. Removal applicable to the group of signals, all selected signals will be processed.

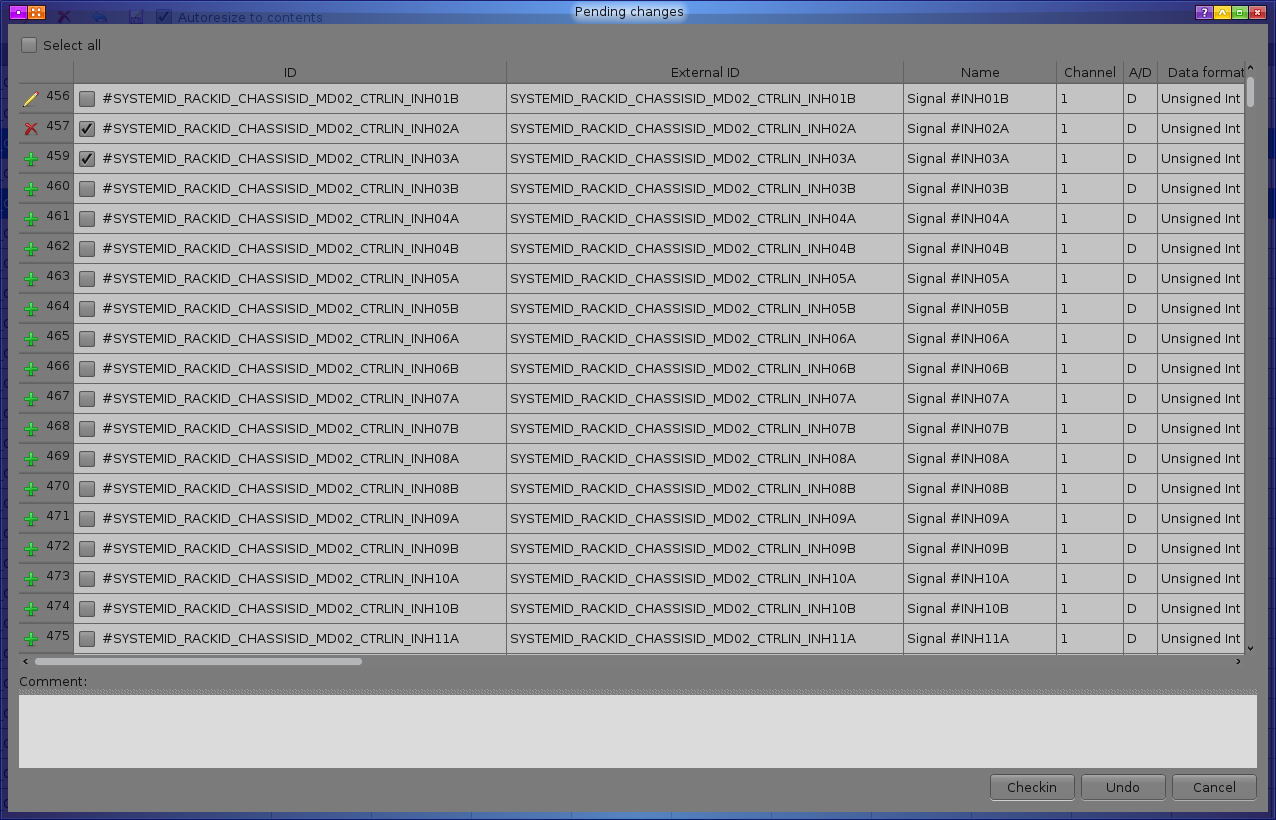
* + 1. Undo signal changes



To support undo and transactional commit capabilities all pending signals are in a temporary state of waiting commit after processing. If user applies operation "Cancel changes" to such signals, the signals become in the original state. When user selects "Cancel changes" window opens with a list of signals with uncommitted changes as shown in Figure, in which these changes can be undone. For convenience, the signals, that have been selected before undo operation, are also checked in the undo window. If user wants to set or unset all the signal checkboxes, he can use check "Select All." After user clicks OK, the signals return to the last recorded status (or disappear if there were just added).

* + 1. Checkin signal changes

After the user has made sure that all the changes coordinated and can provide access to another, user need to select the operation "Checkin signal changes." After selecting the operation "Pending changes" window appears as in figure 6.4. For convenience, the signals, that have been selected before undo operation, are also checked in the checkin window. For successful commit user must mark at least one signal, write comment about done work and push Checkin button. If user want to undo some changes he could push Undo button. Undo signal changes window appears after that.



* 1. Signal propeties dialog



Signal properties dialog allows to edit application signal parameters. Signal automatically checks out on any field change (gets pending changes). Fields meaning is:

|  |  |  |
| --- | --- | --- |
| Field name | Field description | Analog only/ and Discrete |
| ID | Unique signal name used to describe signal role for application logic | A and D |
| External ID | Signal name used to identify signal role for production processes. It could be sensor name, from which data were | A and D |
| Name | Full description | A and D |
| Low ADC, high ADC | Integer range used in logic calculation and transfer to upper server applications | A |
| Low limit, high limit | Floating point range used for physical values in reports | A |
| Unit | Units for physical values in reports | A |
| Adjustment | Displacement value that correct current physical value | A |
| Drop limit, excess limit | Normal physical limits beyond which signals marked as behind measures. | A |
| Unbalance limit | Max difference between channels of same signal | A |
| Output range mode | Configuration option (limits) in ADC element in module | A |
| Sensor type | Type of sensor that generates signal, used for metrology tests. | A and D |
| Calculated | Is signal received from modules (false) or generated by software server (true) | A and D |
| Normal state | Safety state value | A and D |
| Decimal places | Required display precision in reports | A |
| Aperture | Signal value movement step after which value is stored in archive | A |
| Input-output type | Is signal input or output | A and D |
| Byte order | Order of byte sequence of each value |  |
| Device ID | Signal name used to make relation with hardware module inputs/outputs | A and D |

# AFB library

* 1. AFB (Application Functional Block) is XML-based description of units which are used in Logic Module to run Application Logic.

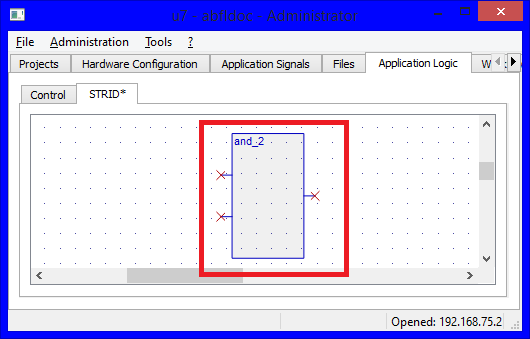
A library of AFBs is created automatically when a new application project is created. AFB description example is shown on fig. 7.1. In most cases user should not edit this description.



|  |  |
| --- | --- |
| Figure | 7.1 |

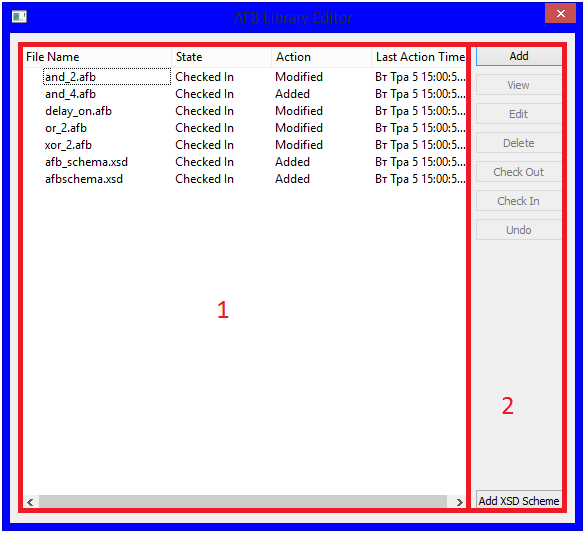
AFBs can be added to application logic schemes and then converted to logic module’s commands at project build time.

Example of AFB on application logic scheme is shown on fig. 7.2.



|  |  |
| --- | --- |
| Figure | 7.2 |

* 1. AFB element editor. Creating, editing and removing AFBs.
     1. To open AFBs editor select "AFBL Editor..." from "Tools" in main menu. An editor window will be shown (fig. 7.3).



|  |  |
| --- | --- |
| Figure | 7.3 |

* + 1. Editor window contains list of AFBs descriptions and a control panel. List contains following columns:

– AFB name ("File Name");

– AFB status ("State");

– Last action with AFB ("Action");

– AFB last action time (“Last action time”).

* + 1. Control panel includes following buttons:

– "Add" - add new AFB;

– "View" - open selected AFB for viewing;

– "Edit" - open selected AFB for editing;

ATTENTION: “Edit” button is available only if AFB is checked out for editing.

– "Delete" - remove selected AFB;

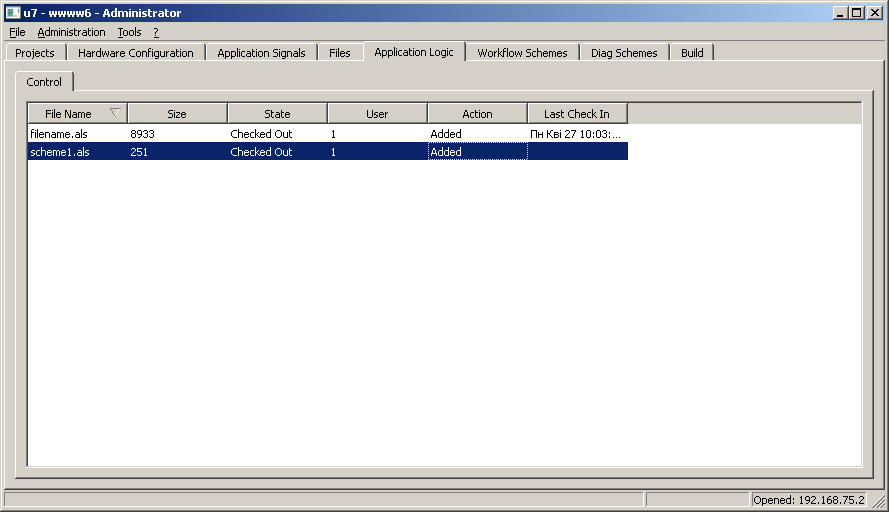
– "Check Out" - prepare selected AFB for editing;

– "Check In" – commit changes to the project database;

– "Undo" - cancel pending changes.

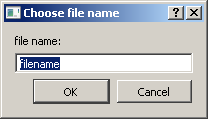
# Application Logic Editor

* 1. To edit application logic, switch to the “Application Logic” tab. This tab contains a list of files with application logic schemes, as shown on the figure 8.1.

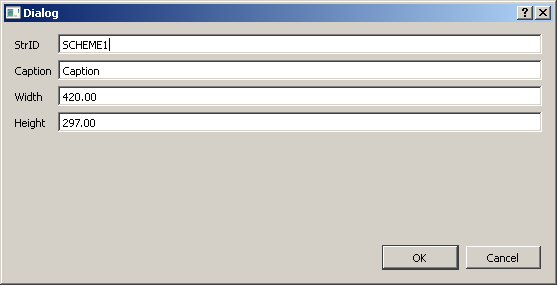


|  |  |
| --- | --- |
| Figure | 8.1 |

* + 1. To add a new scheme, press the right mouse button on the list and choose the “Add File…” item. Enter the file name in the dialog shown on the figure 8.2. Enter scheme parameters in the next dialog shown on the figure 8.3. A new scheme will be created and it will be in checked out state and available for editing.



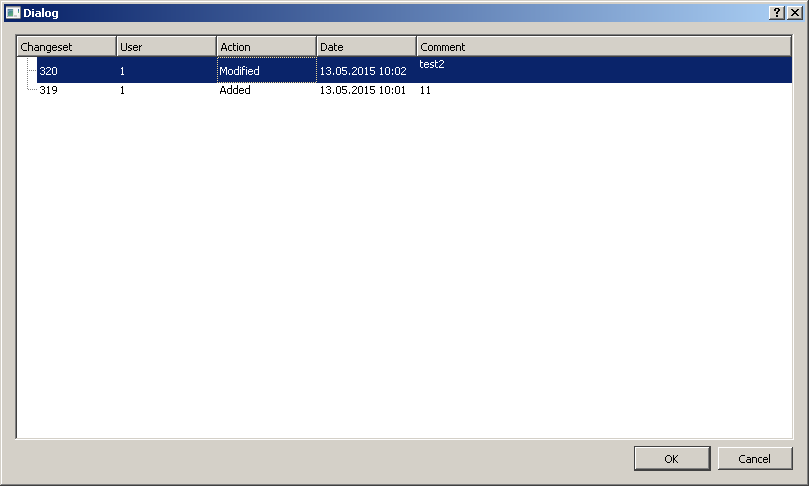
|  |  |
| --- | --- |
| Figure | 8.2 |



|  |  |
| --- | --- |
| Figure | 8.3 |

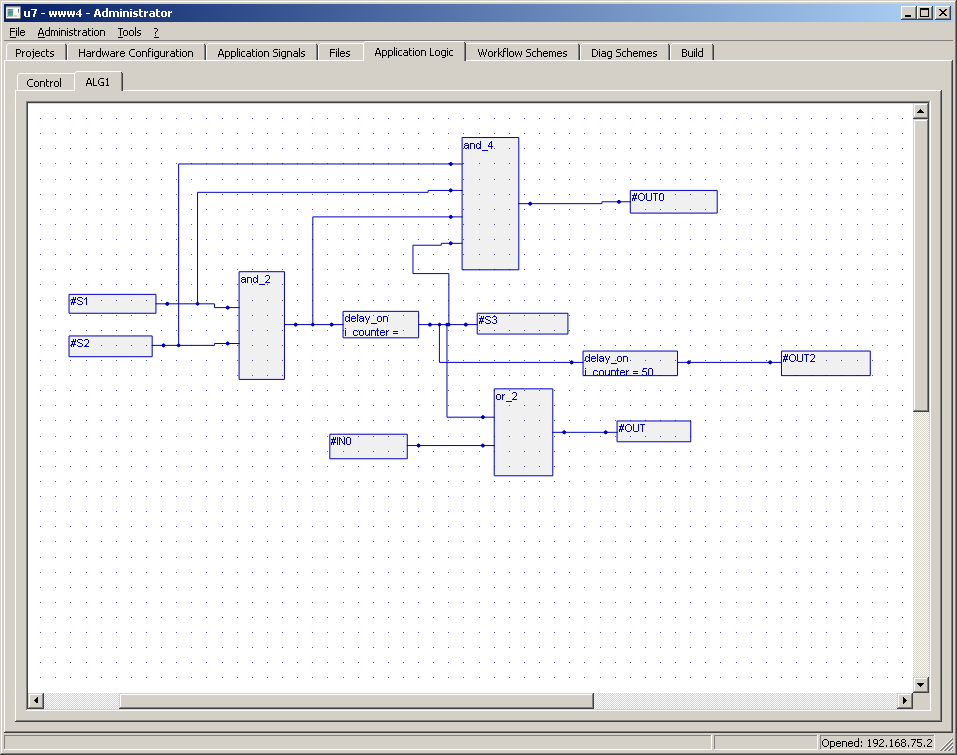
To edit a scheme, it must be checked out before. To check out a scheme, click the right mouse button on a file name and select “Check Out…” command. To commit changes to project database choose “Check In…” command. To undo changes, use the “Undo Changes…” command.

* + 1. To delete an existing file, select “Delete File…” command.
    2. To view file contents, click the right mouse button on a file and select “View…” command. A window with the file check-in history will appear, shown on a figure 8.4. Choose a file changeset to view and press OK.



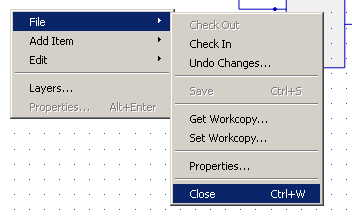
|  |  |
| --- | --- |
| Figure | 8.4 |

* + 1. To edit an existing scheme, click the right mouse button on the file name in a list and choose “Open…”. Only checked out files can be edited.
  1. When a scheme file is opened for viewing or editing, a new tab with file contents is created. In view mode, no changes can be made to the file. An application logic editing tab is shown on the figure 8.5.



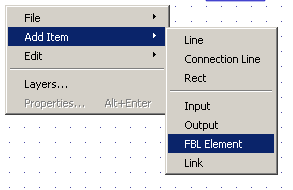
|  |  |
| --- | --- |
| Figure | 8.5 |

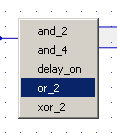
* + 1. To save changes, click the right mouse button on the white space and choose the “File” submenu, then choose the “Save…” command.
    2. To close a tab, choose the “Close” command in “File” submenu.



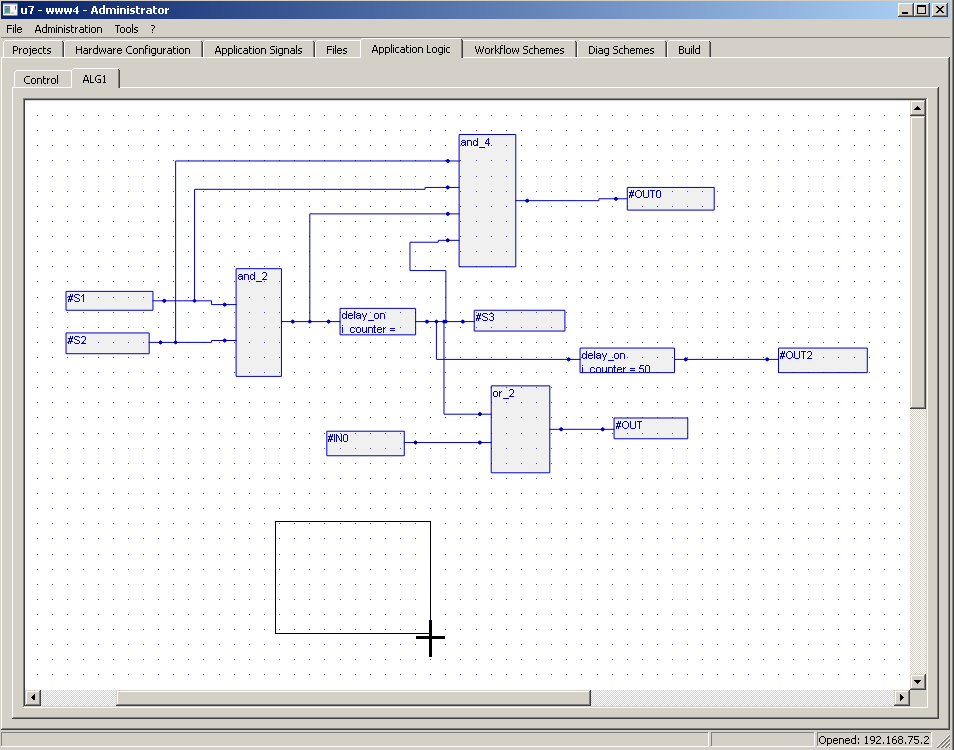
|  |  |
| --- | --- |
| Figure | 8.6 |

* + 1. To add a new item to the scheme, click the right mouse button on the white space and choose the “Add Item” submenu. A list of available items will appear as shown on the figure 8.7. If the “FBL Element” item is chosen, an additional menu will appear to choose the type of an element, as shown on the figure. Select the necessary element in the menu, press the left mouse button on the scheme white space, move the mouse and release the left button, as shown on the figure 8.8. A newly created item will appear on the scheme.



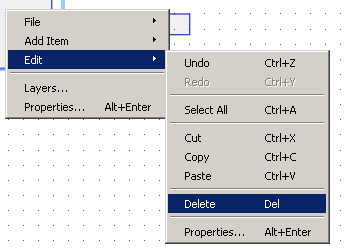


|  |  |
| --- | --- |
| Figure | 8.7 |



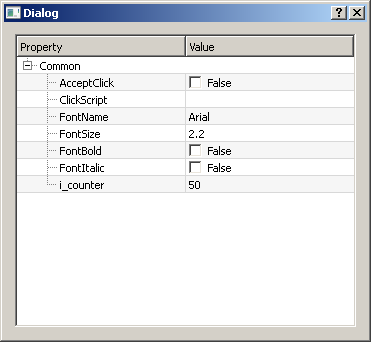
|  |  |
| --- | --- |
| Figure | 8.8 |

* + 1. To move an item, press the left mouse button on it and move the mouse. Then release the button.
    2. For item editing commands, select “Edit” submenu in the scheme popup-menu. “Edit” submenu contains “Undo”, “Redo”, “Select All”, “Delete” and clipboard commands, as shown on the figure 8.9.



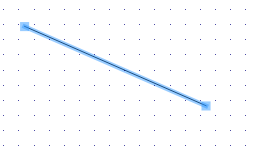
|  |  |
| --- | --- |
| Figure | 8.9 |

* + 1. To edit the item’s properties, select an item and choose the “Properties” command in the “Edit” submenu. The properties dialog will appear, as shown on the figure 8.10.



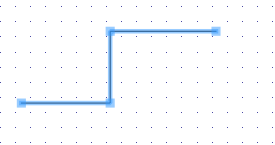
|  |  |
| --- | --- |
| Figure | 8.10 |

* 1. Application scheme can contain static and dynamic items.
  2. Static scheme items are used to display static scheme elements like lines, rectangles, static text etc.
     1. A “Line” element is a simple line, as shown on the figure 8.11.



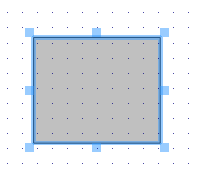
|  |  |
| --- | --- |
| Figure | 8.11 |

* + 1. A “Connection Line” is a line contains a few sections. To add a connection line to the scheme, select the “Connection Line” command in the menu. Then press the left mouse button and move the mouse. To make a connection point, hold the left mouse button and click the right mouse button. Then move the mouse again. After adding all connections, release the left mouse button. A connection line is shown on the figure 8.12.



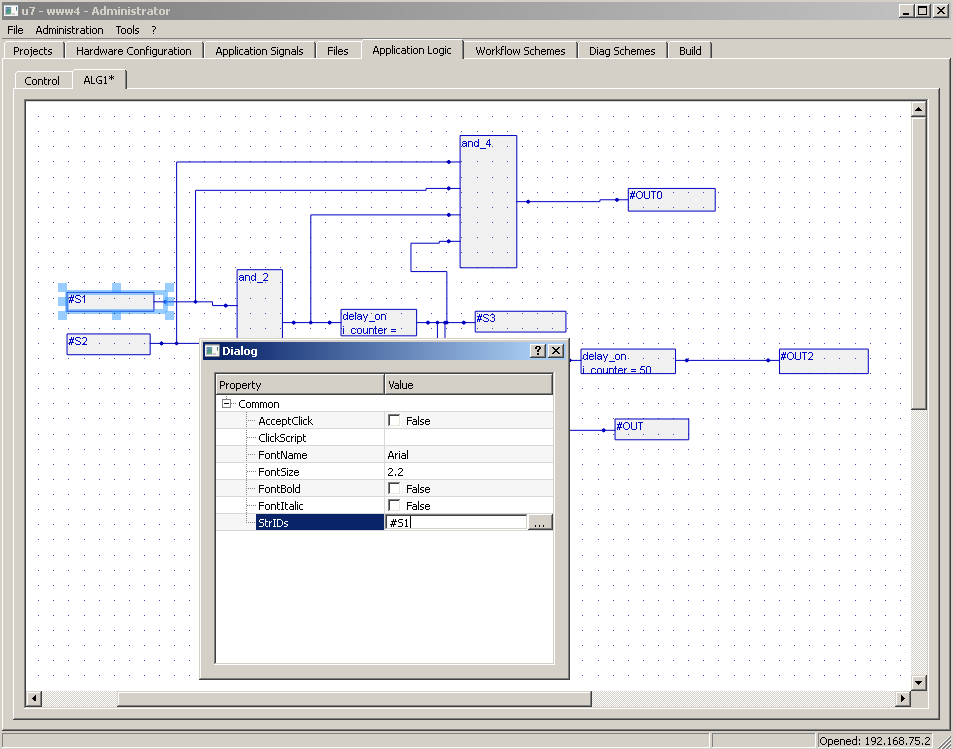
|  |  |
| --- | --- |
| Figure | 8.12 |

* + 1. A “Rectangle” item is a simple rectangle, shown on the figure 8.13.



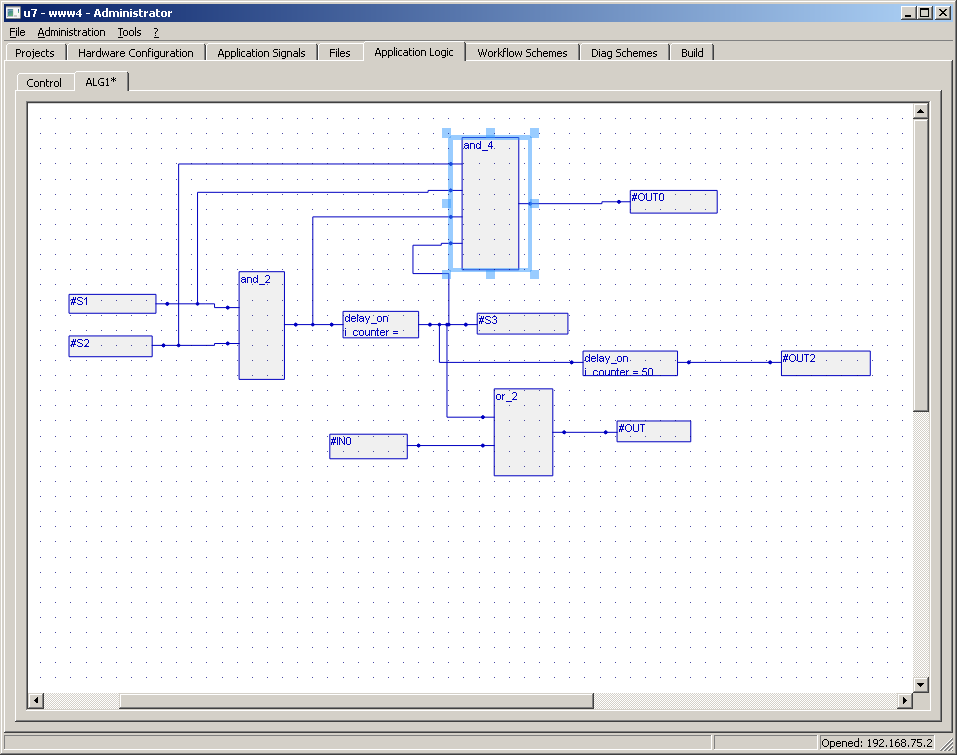
|  |  |
| --- | --- |
| Figure | 8.13 |

* 1. Dynamic items describes application logic itself.
     1. “Input” item is selected on the figure 8.14. It contains input signal of the application logic scheme. The “StrIDs” property conatins a signal identifier this item belongs to.
     2. “Output” item is similar to “Input” item. It contains output signal of the scheme. “StrIDs” property contains a signal identifier this item belongs to.



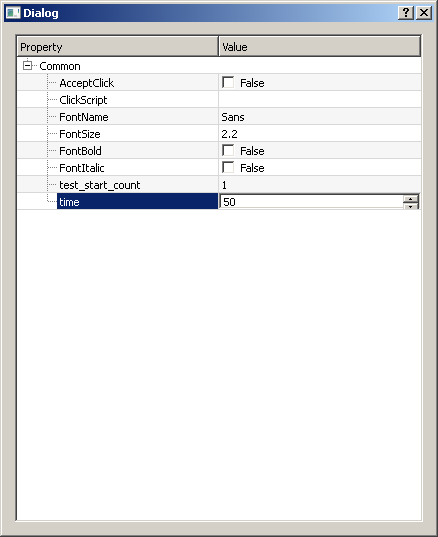
|  |  |
| --- | --- |
| Figure | 8.14 |

* + 1. AFB item describes application logic functions. They are described in AFB library and are implemented in the hardware. Most AFB elements have inputs and outputs, as shown on the figure 8.15.



|  |  |
| --- | --- |
| Figure | 8.15 |

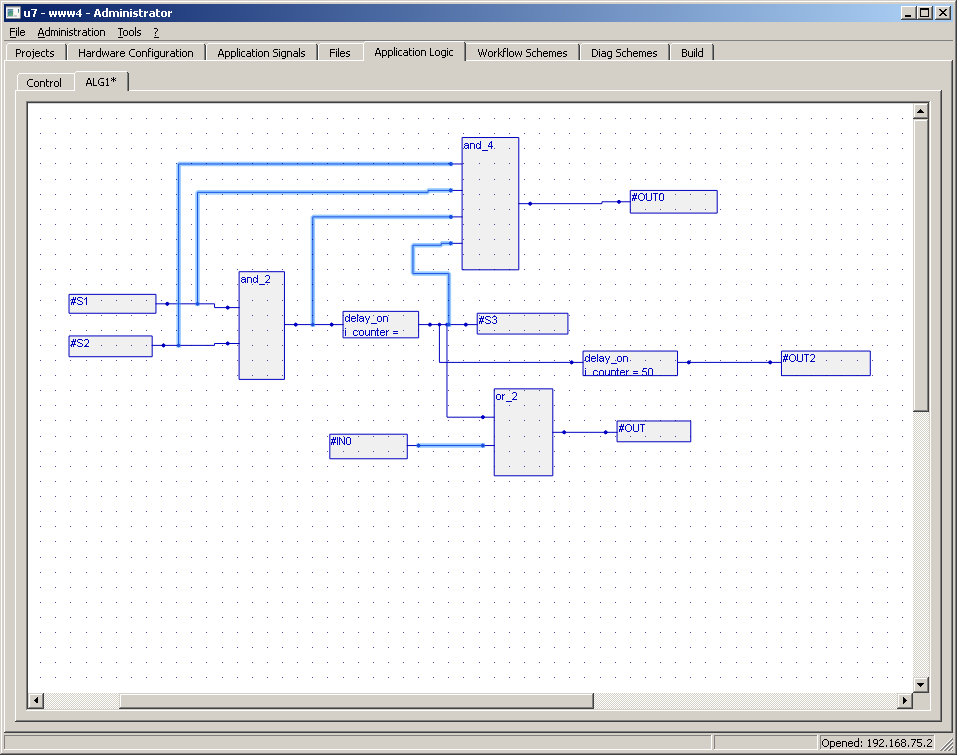
AFB items may also contain parameters that user can change. For example, a “Delay” item contains “time” property that user can change to control the delay time. An example is shown on the figure 8.16.



|  |  |
| --- | --- |
| Figure | 8.16 |

* + 1. “Link” item is an item that connects inputs, outputs and AFB items. An example is shown on the figure 8.17.

NOTE – to build a project, all pins must be connected.



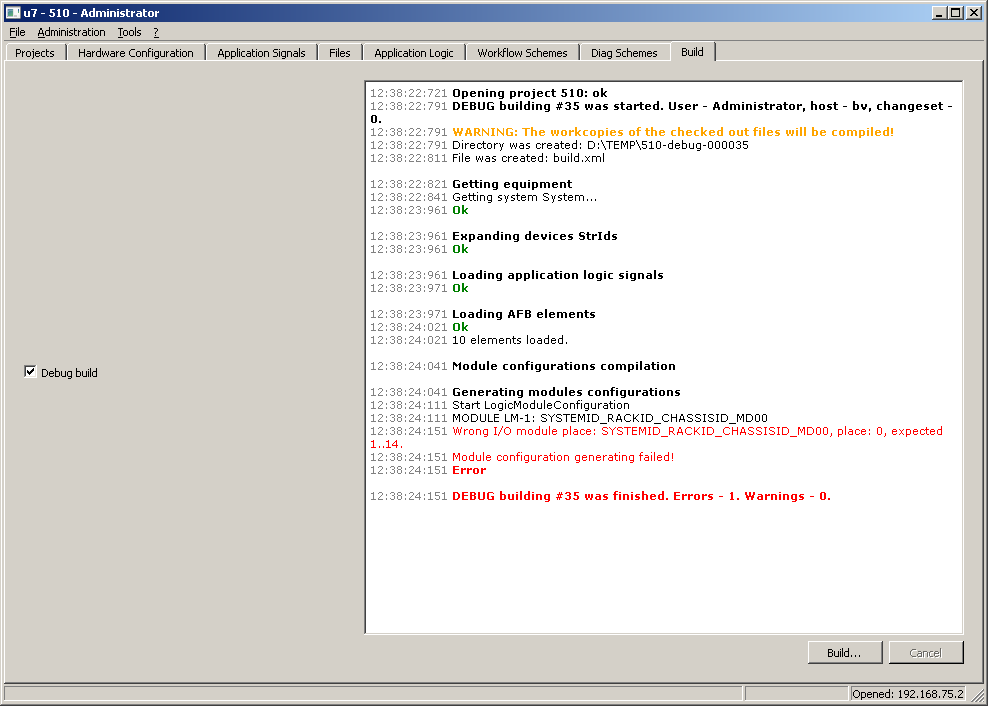
|  |  |
| --- | --- |
| Figure | 8.17 |

# Редактор технологических видеокадров (Workflow schemes)

# Редактор диагностических видеокадров (Diag schemes)

# Project building

* 1. To generate a set of files for hardware and SCADA software a project build process must be done. To build a project, switch to the “Build” tab and press “Build” button. The build tab is shown on the figure 11.1.



|  |  |
| --- | --- |
| Figure | 11.1 |

* 1. During the build process, diagnostic messages are shown in the log window. Warning messages are displayed with orange color, error messages are displayed with red color. At the end of the build process the total number of errors and warnings are displayed.
  2. Error messages are critical issues and must be fixed to finish the build process successfully. Warning messages allow the build process to complete but some limitations may take place.
  3. Build output files are stored in the output folder chosen in the “Settings” window. For every build, a new directory is created. It has the following name format: “ProjectName – debug/release – buildno”, for example “KAES\_6USB1-debug-000029”.

In this directory, following files and subfolders are created:

|  |  |
| --- | --- |
| build.xml | Contains an xml document with all files and their checksums list |
| DataAcquisionService/equipment.xml | Xml document with hardware hierarchy |
| DataAcquisionService/applicationSignals.xml | Xml document with Unit list and Signal list |
| <SysName>/<ModuleName>.alb | Application logic file for a module |
| <SysName>/<ModuleName>.mcb | Hardware configuration for a module |
|  |  |
|  |  |
|  |  |