

OpenSCADA v. 0.7.0

(<http://oscada.org>)

November 22, 2010

Contents table

Introduction.....	10
Project targets.....	10
Policy of development. License.....	10
Scopes.....	10
Architecture.....	11
Functional characteristics and demands of OpenSCADA system.....	12
1. The employment area of system OpenSCADA.....	12
1.1. SCADA system's server.....	12
1.2. Station of the operator of technological process, the board of the dispatcher, the panel of monitoring, etc.:.....	14
1.3. The environment of execution of controllers (PLC):.....	14
2. Requirements for OpenSCADA.....	16
2.1. Execution.....	16
2.2. Building.....	17
OpenSCADA program description.....	19
1. Functions of the system.....	20
1.1. Modularity.....	20
1.2. Subsystems.....	21
1.3. PLC and other sources of dynamic data. A subsystem "Data acquisition".....	21
1.4. Databases. A subsystem of "Database".....	22
1.5. Archives. A subsystem "Archives".....	22
1.6. Communications. Subsystems "Transports" and "Transport protocols".....	23
1.7. Interfaces of the user. A subsystem "Interfaces of the user".....	24
1.8. Security of system. A subsystem "Security".....	25
1.9. Management of libraries of modules and modules. A subsystem "Management of modules".....	25
1.10. Unforeseen opportunities. A subsystem "Special".....	25
1.11. The user functions. Objective model and the environment of programming of system.....	25
2. SCADA systems and their structure.....	27
3. Ways of configuration and using of OpenSCADA system.....	29
3.1. Simple server connection.....	29
3.2. The duplicated server connection.....	30
3.3. The duplicated server connection on one server.....	30
3.4. Client access by means of the Web-interface. A place of the manager.....	31
3.5. The automated workplace (place of the manager/operator).....	31
3.6. Automated workplace with a server of acquisition and archiving on the single machine (a place of the operator, model...).....	32
3.7. The elementary mixed connection (model, demonstration, configurator...).....	33
3.8. The steady, allocated configuration.....	34
4. Configuration and adjustment of the system.....	36
4.1. "DB" subsystem.....	41
4.2. Subsystem "Security".....	46
4.3. Subsystem "Transports".....	49
4.4. Subsystem "Transport protocols".....	53
4.5. Subsystem "Data acquisition".....	54
4.6. Subsystem "Archives".....	66
4.7. Subsystem "User interfaces".....	77
4.8. Subsystem "Specials".....	78
4.9. Subsystem "Modules sheduler".....	79
4.10. Configuration file of the OpenSCADA and parameters of command-line OpenSCADA.....	

execution.....	80
5. System-wide API of user programming.....	90
5.1. System-wide user objects.....	90
Array object.....	90
XMLNodeObj object.....	91
5.2. System (SYS).....	91
5.3. Any object of OpenSCADA objects tree (SYS.*).....	92
5.4. "DB" subsystem (SYS.BD).....	92
5.5. Subsystem "DAQ" (SYS.DAQ).....	93
5.6. "Archives" subsystem (SYS.Archive).....	93
5.7. "Transports" subsystem (SYS.Transport).....	93
Data acquisition in OpenSCADA.....	94
1. Data acquisition methods.....	96
1.1. Simple synchronous acquisition mechanism.....	96
1.2. Simple asynchronous acquisition mechanism.....	97
1.3. Package acquisition mechanism.....	98
1.4. Passive acquisition mechanism.....	99
2. Virtual data sources.....	100
3. Logic level of data processing.....	102
4. Redundancy of the data sources.....	106
Quick start OpenSCADA.....	110
1. Terms, definitions and abbreviations.....	110
2. Installation and start.....	111
2.1. Installing OpenSCADA from packages.....	111
2.2. Installation from sources.....	112
3. Initial configuration and start.....	113
4. Working with Data Sources.....	117
4.1. Data inquiry of the TP device.....	117
4.2. TP data processing.....	125
4.3. Enabling the TP data archiving.....	132
5. The formation of visual presentation.....	135
5.1. Adding the template page in the project and linkage of the dynamics.....	135
5.2. The creation of the new frame, the mnemonic scheme.....	140
5.3. Creation of the new complex element.....	146
5.3.1. Creation the widget "Air cooler" on the basis of the primitive "Elementary figures".....	146
5.3.2. Creation the final complex widget "Cooler" on the basis of the primitive "Elements box".....	152
5.3.3. Adding the complex element to the mnemonic scheme.....	171
6. Recipes.....	173
Conclusion.....	173
Module of subsystem "Archives" <FSArch>.....	174
1. Message Archiver.....	174
1.1. File format of archive messages.....	177
1.2. Example of the archive of messages file.....	178
2. Values Archiver.....	179
2.1. File format of archive values.....	181
3. Efficiency.....	183
Module of subsystem "Archives" <DBArch>.....	184
1. Message Archiver.....	184
2. Values Archiver.....	185
3. Informational table of the archival tables.....	186
Module of the subsystem "DB" <DBF>.....	187
1. Operations over the database.....	187
2. Operations over the table.....	187

3. Operations over the contents of the table.....	187
4. Productivity of DB.....	188
Module of the subsystem “DB” <MySQL>.....	189
1. Operations over the database.....	189
2. Operations over the table.....	189
3. Operations over the contents of the table.....	190
4. Access rights.....	190
5. Productivity of DB.....	191
Module of the subsystem “DB” <SQLite>.....	192
1. Operations over the database.....	192
2. Operations over the table.....	192
3. Operations over the contents of the table.....	192
4. Access rights.....	193
5. Productivity of DB.....	193
Module of the subsystem “DB” <FireBird>.....	194
1. Operations over the database.....	194
2. Operations over the table.....	194
3. Operations over the contents of the table.....	194
4. Access rights.....	195
5. Productivity of DB.....	195
Module of the subsystem “DB” <PostgreSQL>.....	196
1. Operations over the database.....	196
2. Operations over the table.....	196
3. Operations over the contents of the table.....	197
4. Access rights.....	197
5. Productivity of DB.....	197
The module of subsystem “Data acquisition” <DiamondBoards>.....	198
1. Data controller of Diamond boards.....	199
2. Parameters of the Diamond controller.....	201
Links.....	202
The module of subsystem “Data acquisition” <System>.....	203
1. The controller of data.....	204
2. Parameters.....	205
The module of subsystem “Data acquisition” <BlockCalc>.....	207
1. The controller of the module.....	209
2. The block scheme of the controller.....	210
3. Parameters of the controller.....	213
4. Copying of the block schemes.....	214
The module of subsystem “Data acquisition” <JavaLikeCalc>.....	215
1. Java-like language.....	217
1.1. Elements of language.....	217
1.2. Operations of language.....	217
1.3. Embedded functions of language.....	218
1.4. Operators of the language.....	219
1.4.1. Conditional operators.....	219
1.4.2. Loops.....	219
1.4.3. Special characters of string variables.....	220
1.5. Object.....	220
1.6. Examples of programs on the language.....	222
2. Controller and its configuration.....	223
3. The parameter of the controller and its configuration.....	224
4. Libraries of functions of module.....	225
5. User functions of the module.....	225
The module of subsystem “Data acquisition” <LogicLev>.....	226
1. Data controller.....	227

2. Parameters.....	228
The module of subsystem “Data acquisition” <SNMP>.....	231
1. SNMP.....	231
1.1. MIB.....	231
1.2. Addressing.....	232
1.3. Interaction.....	232
1.4. Authorization.....	232
2. Module.....	233
2.1. Controller of data.....	233
2.2. Parameters.....	234
The module of subsystem “Data acquisition” <Siemens>.....	236
1. Communication controllers CIF.....	237
2. The controller of the data source.....	239
3. The parameters of the data source.....	240
4. Asynchronous recording mode.....	244
5. Comments.....	244
Links.....	244
The modules <ModBus> of subsystem “Data acquisition” and subsystem “Transport protocols”.....	245
1. General description of the ModBus protocol.....	245
1.1. Addressing.....	245
1.2. Standard codes of functions.....	246
2. Module of the implementation of the protocol.....	246
2.1. API functions of outgoing requests.....	246
2.2. Servicing of the requests for ModBus protocol.....	247
The mode of the node of the protocol “Data”.....	248
The mode of the node of the protocol “Gateway of the node”.....	251
The mode of the node of the protocol “Gateway of the network”.....	252
2.3 Report of the ModBus requests.....	253
3. Data acquisition module.....	254
3.1. Controller of data.....	254
3.2. Parameters.....	256
The module of subsystem “Data acquisition”<DCON>.....	258
1. General description of the protocol DCON.....	258
2. Module.....	259
2.1. Data controller.....	259
2.2. Parameters.....	260
3. Compatibility table of input/output modules of different manufacturers.....	261
The module of subsystem “Data acquisition” <ICP_DAS>.....	262
1. Data controller.....	263
2. Parameters.....	264
2.1 Module I-8017.....	265
2.2 Module I-8042.....	266
2.3 Module I-87019.....	266
2.4 Module I-87024.....	266
2.5 Module I-87057.....	266
3. LP-8x81 series controllers configuration.....	266
Links.....	266
The module of subsystem “Data acquisition” <DAQGate>.....	267
1. Controller of data.....	269
2. Parameters.....	270
The module of subsystem “Data acquisition”<SoundCard>.....	271
1. Controller of the data.....	272
2. Parameters.....	273
The <OPC-UA> module of “Data acquisition” and “Transport protocols” subsystems.....	275

1. OPC UA protocol.....	276
2. The module of the protocol implementation.....	277
2.1. Service the requests on the OPC UA protocol.....	277
3. Data acquisition module.....	279
3.1. Data controller.....	279
3.2. Parameters.....	281
4. Notes.....	282
Module <Sockets> of subsystem “Transports”.....	283
1. Incoming transports.....	284
2. Outgoing transports.....	286
Module <SSL> of subsystem “Transports”.....	288
1. Incoming transports.....	289
2. Outgoing transports.....	290
3. Certificates and keys.....	291
Module <Serial> of subsystem “Transports”.....	292
1. Incoming transports.....	293
2. Outgoing transports.....	295
Module <HTTP> of subsystem “Protocols”.....	297
1. Authentication.....	298
2. The modules of user WEB-interface.....	299
3. Outgoing requests function's API.....	299
Module <SelfSystem> of subsystem “Protocols”.....	301
1. The syntax of the protocol.....	301
2. The internal structure of an outgoing protocol.....	302
Module <UserProtocol> of subsystem “Protocols”.....	303
1. Part of the protocol for incoming requests.....	304
2. Part of the protocol for outgoing requests.....	305
The module <FLibComplex1> of the subsystem “Specials”.....	307
1. Alarm (alarm) <111>.....	307
2. Condition '<' (cond_lt) <239>.....	307
3. Condition '>' (cond_gt) <240>.....	307
4. Full condition (cond_full) <513>.....	308
5. Digital block (digitBlock) <252>.....	308
6. Division (div) <526>.....	308
7. Exponent (exp) <476>.....	308
8. Flow (flow) <235>.....	308
9. Iterator (increment) <181>.....	308
10. Lag (lag) <121>.....	309
11. Simple multiplication(mult) <259>.....	309
12. Multiplication + Division(multDiv) <468>.....	309
13. PID regulator (pid) <745>.....	309
14. Power (pow) <564>.....	310
15. Selection (select) <156>.....	310
16. Simple integrator (sum) <404>.....	310
17. Sum with the division (sum_div) <518>.....	310
18. Sum with the multiplication. (sum_mult) <483>.....	311
The module <FLibMath> of the subsystem “Specials” <FLibMath>.....	312
1. Functions.....	312
The module <FLibSYS> of the subsystem “Specials”.....	314
1. System-wide functions.....	314
1.1. Calling the console commands and operating system utilities (sysCall).....	314
1.2. SQL query (dbReqSQL).....	315
1.3. XML node (xmlNode).....	315
1.4. Request of the management interface (xmlCntrReq).....	315
1.5. Values archive (vArh).....	316

VArchObj object.....	316
1.6. Buffer of the values archive (vArhBuf).....	316
2. Functions for the astronomical time processing.....	317
2.1. Time string (tmFStr) <3047>.....	317
2.2. Full Date (tmDate) <973>.....	317
2.3. Absolute time (tmTime) <220>.....	317
2.4. Conversion the time from the symbolic representation to the time in seconds from the epoch of 1/1/1970 (tmStrPTime) <2600>.....	318
2.5. Planning of the time in the Cron format (tmCron).....	318
3. Functions of the messages processing.....	318
3.1. Messages request (messGet).....	318
3.2. Generation of the message (messPut).....	319
4. Functions of the strings processing.....	319
4.1. Getting the size of the string (strSize) <114>.....	319
4.2. Getting the part of the string (strSubstr) <413>.....	319
4.3. Insert of the on string to the another (strInsert) <1200>.....	319
4.4. Change the part of the string with the another one (strReplace) <531>.....	320
4.5. Parsing the string on separator (strParse) <537>.....	320
4.6. Path parsing (strParsePath) <300>.....	320
4.7. Path to the string with the separator (strPath2Sep).....	320
4.8. Coding of the string to HTML (strEnc2HTML).....	321
4.9. Encode text to bin (strEnc2Bin).....	321
4.10. Decode text from bin (strDec4Bin).....	321
4.11. Convert real to string (real2str).....	321
4.12. Convert integer to string (int2str).....	321
4.13. Convert the string to real (str2real).....	322
4.14. Convert the to integer (str2int).....	322
5. Functions for the real processing.....	322
5.1. Splitting the float to the words (floatSplitWord) <56>.....	322
5.2. Merging the float from words (floatMergeWord) <70>.....	322
The module <SystemTests> of the subsystem "Specials".....	323
1. Parameter (Param).....	324
2. XML parsing (XML).....	324
3. Messages (Mess).....	324
4. SO attaching (SOAttach).....	325
5. Attribute of the parameter (Val).....	325
6. DB test (DB).....	325
7. Transport (TrOut).....	326
8. Control system language (SysContrLang).....	326
9. Values buffer (ValBuf).....	326
10. Values archive (Archive).....	326
11. Base64 code (Base64Code).....	326
The module of subsystems "User Interfaces" <QTStarter>.....	327
The module <QTCfg> of subsystems "User Interfaces".....	329
1. Configuration.....	331
2. Basic elements.....	332
3. Commands.....	333
4. Lists.....	334
5. Tables.....	335
6. Images.....	336
The module <WebCfg> of subsystems "User Interfaces".....	337
1. Basic elements.....	339
2. Commands.....	340
3. Lists.....	340
4. Tables.....	341

5. Images.....	342
The module <WebCfgD> of subsystems "User Interfaces".....	343
1. Configuration.....	345
2. Basic elements.....	346
3. Commands.....	347
4. Lists.....	348
5. Tables.....	349
6. Images.....	350
7. Errors.....	351
The module <VCAEngine> of subsystems "User Interfaces".....	353
Introduction.....	353
1. Purpose.....	354
2. The configuration and the formation of interfaces of the VCA.....	355
3. Architecture.....	355
3.1. Frames and elements of visualization (widgets).....	357
3.2. Project.....	360
3.3. Styles.....	363
3.4. Events, their processing and the events' maps.....	365
3.5. Signaling.....	367
3.6. Rights management.....	368
3.7. Linkage with the dynamics.....	369
3.8. The primitives of the widget.....	374
3.8.1. Elementary graphic figures (ElFigure).....	377
3.8.2. Element of the form (FormEl).....	378
3.8.3. Text element (Text).....	380
3.8.4. Element of visualization of media materials (Media).....	381
3.8.5. Element of constructing diagrams/trends (Diagram).....	381
3.8.6. The element of building the protocols based on the archives of messages (Protocol).....	383
3.8.7. Element of formation of documentation(Document).....	383
3.8.8. Container (Box).....	386
3.9. Using the database to store the library of widgets and projects.....	387
3.10 API of the user programming and service interfaces of the OpenSCADA.....	389
3.10.1. API of the user programming.....	389
3.10.2. Service interfaces of the OpenSCADA.....	391
4. Configuring the module via the control interface of OpenSCADA.....	393
The module <Vision> of subsystems "User Interfaces".....	403
1. Purpose.....	404
2. Tool of the graphical formation of the VCA interface.....	405
2.1. Styles.....	414
2.2. Linkage with the dynamics.....	415
3. Execution of the VCA interfaces.....	417
4. Conception of basic elements (primitives).....	419
4.1. Elementary figure primitive (ElFigure).....	420
4.2. Text primitive (Text).....	421
4.3. Primitive of the form element (FormEl).....	422
4.4. Primitive of the displaying the media materials (Media).....	423
4.5. Primitive of the construction of diagrams/graphs (Diagram).....	424
4.6. Primitive of the protocol formation (Protocol).....	424
4.7. Primitive of the report formation (Document).....	425
4.8. Primitive of the box container (Box).....	426
5. The overall configuration of the module.....	426
The module <WebVision> of subsystems "User Interfaces".....	427
1. Purpose.....	428
2. Execution of the VCA interfaces.....	429

<u>3. Conception of basic elements (primitives).....</u>	<u>431</u>
<u>3.1. Elementary figure primitive (ElFigure).....</u>	<u>432</u>
<u>3.2. Text primitive (Text).....</u>	<u>432</u>
<u>3.3. Primitive of the form element (FormEl).....</u>	<u>433</u>
<u>3.4. Primitive of the displaying the media materials (Media).....</u>	<u>434</u>
<u>3.5. Primitive of the construction of diagrams/graphs (Diagram).....</u>	<u>435</u>
<u>3.6. Primitive of the protocol formation (Protocol).....</u>	<u>435</u>
<u>3.7. Primitive of the report formation(Document).....</u>	<u>436</u>
<u>3.8. Primitive of the box container (Box).....</u>	<u>437</u>
<u>4. The overall configuration of the module.....</u>	<u>437</u>
<u>Conclusion.....</u>	<u>437</u>
<u>The module <WebUser> of subsystems "User Interfaces".....</u>	<u>438</u>
<u>1. WEB - pages.....</u>	<u>440</u>

Introduction

OpenSCADA represents opened SCADA system constructed on principles of modules, multiplatform and scalability. (Supervisory Control And Data Acquisition) is the term which it is often used in sphere of automation of technological processes. The system OpenSCADA is intended for: acquisition, archiving, visualization of the information, delivery of operating influences, and also for other related operations, which are characteristic for full-function SCADA systems.

Project targets

The basic purposes which are pursued with the project, are:

- openness;
- reliability;
- flexibility;
- scalability;
- security;
- financial availability;
- giving of the convenient interface of management

Policy of development. License.

As policy of software realization of the given project principles of development are chosen. This policy will allow to involve in development, testing, distribution and using of the product the significant amount of developers, enthusiasts and other interested persons with the minimal financial expenses at the same time. The program is accessible on conditions of the GPL v2 license.

Scopes

The system OpenSCADA is intended for performance as SCADA systems of usual functions, and for use in adjacent areas of information technologies.

The system OpenSCADA can be used:

- on industrial targets as full-function SCADA system;
- in built in systems, as the execution environment (including PLC);
- for construction of various models (technological, chemical, physical, electric processes);
- on personal computers, servers and clusters for acquisition, processing, representation and archiving of the information about system and its environment.

As base (host) operational systems (OS) for the development and uses it is chosen the OS Linux which is POSIX compatible OS. Besides OS Linux is the optimum compromise in questions:

- safety;
- flexibility/scalability;
- availability;
- popularity and prevalence.

As the system OpenSCADA is developed on standard of POSIX OS, by principles of mutiplatform its adaptation on other OS will not make a problem.

Architecture

Heart of system is the modular kernel.

Depending on what modules are connected, the system can carry out both functions of various servers, and functions of clients of client-server architecture. Actually, the architecture of system allows to realize the distributed client-server systems of any complexity.

For achievement of high speed due to reduction of communications time, the architecture allows to unite functions of the distributed systems in one program.

Architecturally, the system OpenSCADA consists of subsystems:

- *The security subsystem.* Contains lists of users and groups of users, provides check of the rights of access to system elements, etc.
- *The modules DB subsystem.* Provides access to databases.
- *The modules transport subsystem.* Provides the communications with an environment by means of various communication interfaces.
- *The modules transport's protocol subsystem.* It is closely connected with a subsystem of transports and provides support of various reports of an exchange with external systems.
- *The modules DAQ subsystem.* Provides data acquisition from external sources: controllers, sensors, gauges, etc. Except for it the subsystem can give environment for a writing of generators of data (model, regulators...).
- *The modules archive subsystem.* Contains archives of two types: archives of messages and archives of values. An archivation way is defined by algorithm which is incorporated in the archivator's module.
- *The modules user interfaces subsystem.* Contains functions of the user interfaces.
- *The control modules subsystem.* Provides the control over modules.
- *The modules special subsystem.* Contains functions not entered in other subsystems.

Proceeding from a modules principle, the modular subsystems, which are specified above, can expand the functionality by connection of corresponding type of the modules.

The modular kernel of system OpenSCADA is designed in the form of static and shared libraries. It allows to build in functions of system existing programs, and also to create new programs on the basis of a modular kernel of OpenSCADA system.

However, the modular kernel is self-sufficient and can be used by means of the simple starting program.

Modules of system OpenSCADA are stored in dynamic libraries. Each dynamic library can contain set of modules of various type. Filling of dynamic libraries by modules is defined by functional connectivity of modules. Dynamic libraries suppose hot replacement that allows to make updating of modules during work. The method of storage of a code of modules in dynamic libraries is the core for system OpenSCADA as it is supported practically by all modern OS. It does not exclude an opportunity of development of other storage modules code methods.

Functional characteristics and demands of OpenSCADA system

1. The employment area of system OpenSCADA

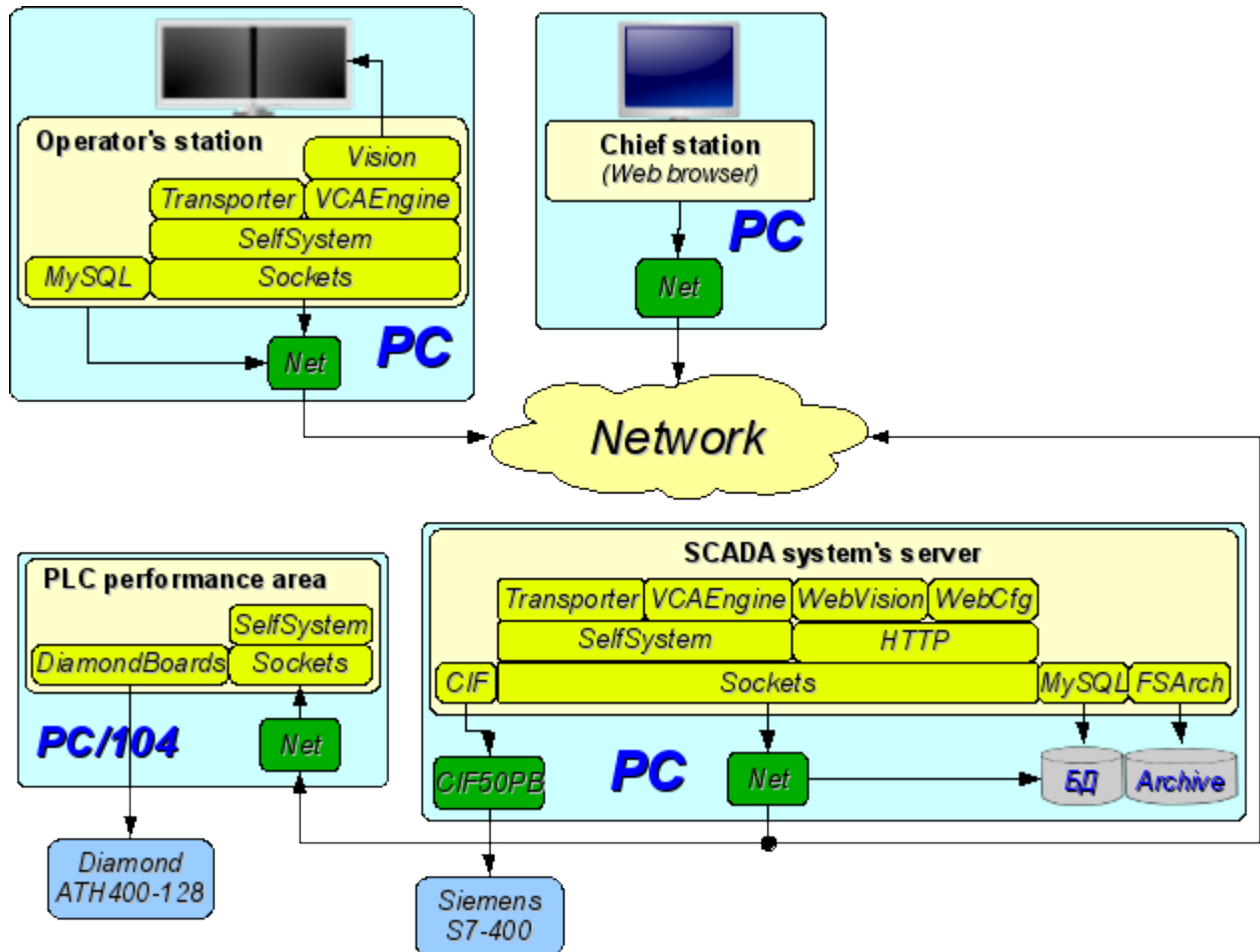


Fig. 1. OpenSCADA system's roles

1.1. SCADA system's server:

- The visual control and management by means of the interfaces:
 - Remote visualization server grounded on visualization and control area (VCA) engine [VCAEngine](#). The module **UI.Vision** local starting and connecting to the visualization server.
 - Remote WEB interface. By means of a Web-browser, the visualization module [WebVision](#) and the module of a kernel of visual control area [VCAEngine](#).
 - Simple remote Web-interfaces of user. By mean Web-browser and UI-module [WebUser](#).
- Data acquisition (DAQ) from sources:
 - Information about a platform (hardware-software) on which the server works. By means of the DAQ-module [System](#).
 - Data acquisition from sources which support protocol SNMP (Simple Network Management Protocol). By means of the DAQ-module [SNMP](#).
 - Data acquisition from controllers of firm Siemens of S7 series. By means of the DAQ-module [Siemens](#).
 - Data acquisition of industrial controllers under the protocol ModBus. By means of the DAQ-module [ModBus](#).

- Data acquisition of industrial controllers under the protocol DCON. By means of the DAQ-module [DCON](#).
 - Formation of derivative structures of parameters on the basis of templates of parameters and data from other sources. By means of the DAQ-module [LogicLev](#).
 - Data acquisition from other servers and PLC, based on OpenSCADA, possibly for duplication. By means of the DAQ-module [DAQGate](#).
 - Data acquisition from sound controller's input channels. By means of the DAQ-module [SoundCard](#).
 - Data acquisition from hardware of firm [ICP DAS](#). By means of the DAQ-module [ICP_DAS](#).
 - Data acquisition from sources which support protocol OPC-UA. By means of the DAQ-module [OPC UA \(RU\)](#).
 - Data acquisition from different sources, which have utilities for access to it data or it accessibly through simple special network protocols. Made by getting procedure writing on language of user programming by DAQ-module [JavaLikeCalc](#), and also transport-protocol-module [User Protocol](#).
 - Providing data to upper-level systems:
 - By means of interfaces:
 - Serial interface (RS232, RS485, Modem, ...), by helps of transport module [Serial](#).
 - IP-networks sockets and network levels protocols TCP, UDP and Unix, by helps of transport module [Sockets](#).
 - Security sockets layer (SSL), by helps of transport module [SSL](#).
 - By means of protocols:
 - Self OpenSCADA protocol, by helps of transport's protocol module [SelfSystem](#).
 - ModBUS family protocol (TCP, RTU and ASCII), by helps of transport's protocol module [ModBUS](#).
 - "OPC UA" protocol, by helps of transport's protocol module [OPC UA \(RU\)](#).
 - Simple special protocols, developed by users by helps of transport's protocol module [User Protocol](#).
 - Implementation of user calculations in languages:
 - Language of block schemes. By means of the DAQ-module [BlockCalc](#).
 - With the help of Java-like language of a high level. By means of the DAQ-module [JavaLikeCalc](#).
 - Archiving messages, conducting reports on various categories and levels by means of mechanisms:
 - Files in a XML-format or the flat text with packing the out-of-date archives. By means of the archiving module [FSArch](#).
 - In tables of archival DB. By means of the archiving module [DBArch](#).
 - In plans. On other server, it is possible to the allocated archiving server, based on OpenSCADA.
 - Archiving values of the collected data by means of mechanisms:
 - Files with double packing: consecutive and standard archiver gzip. By means of the archiving module [FSArch](#).
 - In tables of archival DB. By means of the archiving module [DBArch](#).
 - Configuration and management of a server through:
 - The WEB-interface. By means of a Web-browser and the UI-module [WebCfgD](#) and [WebCfg](#).
 - From the remote configuration station. By means of the UI-module at configuration station [QTCfg](#) and the interface of management OpenSCADA reflected in the protocol [SelfSystem](#).
 - Data storage of a server in a DB of types:
 - MySQL. By means of the DB-module [MySQL](#).
 - SQLite. By means of the DB-module [SQLite](#).
 - PostgreSQL. By means of the DB-module [PostgreSQL](#).
 - DBF. By means of the DB-module [DBF](#).
 - FireBird. By means of the DB-module [FireBird](#).
 - In plans. DB accessible on other server based on OpenSCADA.
 - In plans. LDAP.
-

1.2. Station of the operator of technological process, the board of the dispatcher, the panel of monitoring, etc.:

- The visual control and management by means of the interfaces:
 - The local (fast) interface based on QT library. By means of the visualization module [Vision](#) and the module of a kernel of the visual control area [VCAEngine](#) include ability of visualization from remote engine of VCA, visualization server.
 - Remote WEB interface. By means of a Web-browser, the visualization module [WebVision](#) and the module of a kernel of visual control area [VCAEngine](#).
 - Simple remote Web-interfaces of user. By mean Web-browser and UI-module [WebUser](#).
 - Data acquisition (DAQ) from sources:
 - Data acquisition from other servers and PLC, based on OpenSCADA, for data transportation and for duplication. By means of the DAQ-module [DAQGate](#).
 - Data acquisition from sources which support protocol SNMP (Simple Network Management Protocol). By means of the DAQ-module [SNMP](#).
 - Data acquisition from sources which support protocol OPC-UA. By means of the DAQ-module [OPC UA \(RU\)](#).
 - Implementation of the user calculations in languages:
 - Language of block schemes. By means of the DAQ-module [BlockCalc](#).
 - With the help of Java-like language of a high level. By means of the DAQ-module [JavaLikeCalc](#).
 - Archiving messages, conducting reports on various categories and levels by means of mechanisms:
 - Files in a XML-format or the flat text with packing the out-of-date archives. By means of the archiving module [FSArch](#).
 - In tables of archival DB. By means of the archiving module [DBArch](#).
 - In plans. On other server, it is possible to the allocated archiving server, based on OpenSCADA.
 - Configuration and management of station through:
 - The WEB-interface. By means of a Web-browser and the UI-module [WebCfgD](#) or [WebCfg](#).
 - The QT-interface. By means of the UI-module [QTCfg](#).
 - From the remote configuration station. By means of the UI-module at configuration station [QTCfg](#) and the interface of management OpenSCADA reflected in the protocol [SelfSystem](#).
 - Data storage of station in a DB of types:
 - MySQL. By means of the DB-module [MySQL](#).
 - SQLite. By means of the DB-module [SQLite](#).
 - PostgreSQL. By means of the DB-module [PostgreSQL](#).
 - DBF. By means of the DB-module [DBF](#).
 - FireBird. By means of the DB-module [FireBird](#).
 - In plans. DB accessible on other server based on OpenSCADA.
 - In plans. LDAP.
-

1.3. The environment of execution of controllers (PLC):

- Data acquisition (DAQ) from sources:
 - Cards of data acquisition of firm [Diamond Systems](#). By means of the DAQ-module [DiamondBoards](#).
 - Information on a platform (hardware-software) on which the server works. By means of the DAQ-module [System](#).
 - Data acquisition from sources which support protocol SNMP (Simple Network Management Protocol). By means of the DAQ-module [SNMP](#).
 - Data acquisition of industrial controllers under the protocol ModBus. By means of the DAQ-module [ModBus](#).
 - Data acquisition of industrial controllers under the protocol DCON. By means of the DAQ-module [DCON](#).
 - Formation of derivative structures of parameters on the basis of templates of parameters and data from other sources. By means of the DAQ-module [LogicLev](#).

- Data acquisition from other servers and PLC, based on OpenSCADA, possibly for duplication. By means of the DAQ-module [DAQGate](#).
- Data acquisition from sound controller's input channels. By means of the DAQ-module [SoundCard](#).
- Data acquisition from hardware of firm [ICP DAS](#). By means of the DAQ-module [ICP_DAS](#).
- Data acquisition from sources which support protocol OPC-UA. By means of the DAQ-module [OPC UA \(RU\)](#).
- Data acquisition from different sources, which have utilities for access to it data or it accessibly through simple special network protocols. Made by getting procedure writing on language of user programming by DAQ-module [JavaLikeCalc](#), and also transport-protocol-module [User Protocol](#).
- Providing data to upper-level systems:
 - By means of interfaces:
 - Serial interface (RS232, RS485, Modem, ...), by helps of transport module [Serial](#).
 - IP-networks sockets and network levels protocols TCP, UDP and Unix, by helps of transport module [Sockets](#).
 - Security sockets layer (SSL), by helps of transport module [SSL](#).
 - By means of protocols:
 - Self OpenSCADA protocol, by helps of transport's protocol module [SelfSystem](#).
 - ModBUS family protocol (TCP, RTU and ASCII), by helps of transport's protocol module [ModBUS](#).
 - "OPC UA" protocol, by helps of transport's protocol module [OPC UA \(RU\)](#).
 - Simple special protocols, developed by users by helps of transport's protocol module [User Protocol](#).
- Management, regulation and performance of other user calculations in languages:
 - Language of block schemes. By means of the DAQ-module [BlockCalc](#).
 - With the help of Java-like language of a high level. By means of the DAQ-module [JavaLikeCalc](#).
- Archiving messages, conducting reports on various categories and levels by means of mechanisms:
 - Files in a XML-format or the flat text with packing the out-of-date archives. By means of the archiving module [FSArch](#).
 - In tables of archival DB. By means of the archiving module [DBArch](#).
 - In plans. On other server, it is possible to the allocated archiving server, based on OpenSCADA.
- Archiving of values of the collected data by means of mechanisms:
 - Buffers in memory of the setting depth. By means of the built in archiving mechanism of the values of kernel OpenSCADA.
 - Files with double packing: consecutive and standard archiver gzip. By means of the archiving module [FSArch](#).
 - In tables of archival DB. By means of the archiving module [DBArch](#).
- Configuration and management PLC through:
 - The WEB-interface. By means of a Web-browser and the UI-module [WebCfgD](#) or [WebCfg](#).
 - From the remote configuration station. By means of the UI-module at configuration station [QTCfg](#) and the interface of management OpenSCADA reflected in the protocol [SelfSystem](#).
- Data storage PLC in a DB of types:
 - All data in a configuration file (fixed).
 - MySQL. By means of the DB-module [MySQL](#).
 - SQLite. By means of the DB-module [SQLite](#).
 - PostgreSQL. By means of the DB-module [PostgreSQL](#).
 - DBF. By means of the DB-module [DBF](#).
 - FireBird. By means of the DB-module [FireBird](#).
 - In plans. DB accessible on other server based on OpenSCADA.
 - In plans. LDAP.

2. Requirements for OpenSCADA

2.1. Execution

The demands to apparatus for OpenSCADA system execution at different roles viewed into table 1. The demands to programs for OpenSCADA system execution and its modules allow into table 2.

Table 1. The demands to apparatus for OpenSCADA system and its modules.

Role	Demands
SCADA system's server	CPU: x86_32 (more than i586) or x86_64, with frequency more 500 MHz MEM: 128 MB HDD: 10 GB include OS and place for archives
Station of the operator of technological process, the board of the dispatcher, the panel of monitoring, etc.	CPU: x86_32 (more than i586) or x86_64, with frequency more 1 GHz MEM: 512 MB HDD: 4 GB include OS without archives place
The environment of execution of controllers (PLC)	CPU: x86_32 (more than i586) or x86_64, with frequency more 133 MHz MEM: 32 MB HDD: 32 MB include OS without archives place.

Table 2. Dependences of performance of OpenSCADA system and its modules.


Component	Description
<i>Dependences of OpenSCADA system's kernel</i>	
OS Linux	The distribution kit of operating system Linux (ALTLinux, SuSELinux, Mandriva, ASPLinux, Fedora, Debian, Ubuntu ...)
"Standard libraries"	Standard set of libraries: linux-gate, libstdc++, libgcc_s, libc, libdl, librt, libcrypt, libm, libpthread. Certainly this already allow into installed distribution. Special demand is using native thread library NPTL, already used for all modern distributions of the Linux.
libgd	Graphic library GD version 2, it is desirable that it will be without XPM support (dependence on library of a X-server is excluded) and support of FontConfig.
libexpat	Library of XML-parser.
<i>DB.MySQL module</i>	
libMySQL	Library for access to MySQL DBMS.
<i>DB.SQLite module</i>	
libsqlite3	Library for access to built in DB SQLite version 3.
<i>DB.PostgreSQL module</i>	
libpq	Library for access to PostgreSQL DBMS version more 8.3.0.
<i>DB.FireBird module</i>	
FirebirdSS	FireBird DBMS version 2. Often is absent in distribution kits of Linux and demands individual loading from an official site (http://www.firebirdsql.org)!
<i>Transport.SSL module</i>	
libssl	Library for codifying OpenSSL.
<i>DAQ.SNMP module</i>	
libsnmp	Library for access to data of network devices under SNMP protocol.
<i>DAQ.System module</i>	
libsensors	Hardware sensors' library versions 2 and 3.
<i>DAQ.SoundCard module</i>	
libportaudio	Multiplatform library for access to sound controller version 19 and higher.
<i>DAQ.OPC-UA module</i>	

Component	Description
libssl	Library for codifying OpenSSL.
<i>Modules: UI.Vision, UI.WebVision, Special.FLibSYS</i>	
libfftw3	Library for fast Fourie transfer of signals.
<i>Modules: UI.QTStarter, UI.QTCfg, UI.Vision</i>	
libQT4(libQtCore, libQtGui)	Library for construction of user graphic interface QT version 4.3 and higher.

2.2. Building

Dependences of system OpenSCADA for building of the OpenSCADA kernel and its modules are tabulated bellow.

Table 3. Dependences of building of OpenSCADA system and its modules.

Component	Description
<i>The general requirements for building OpenSCADA</i>	
OS Linux	The distribution kit of operating system Linux (ALTLinux, SuSELinux, Mandriva, ASPLinux, Fedora, Debian, Ubuntu ...)
g++	The compiler of language C++ from a collection of compilers GCC, including library GLibC
autotools(automake, autoconf, libtool)	Tools for formation of building environment of OpenSCADA. They are necessary only in the case of changing building environment of OpenSCADA, for example for addition of the new module or change of the fixed parameters of building.
gettext	Group of utilities for preparation and compilations of translations of the interface of programs on various languages in conformity with internationalization standard I18N.
libgd(devel)	Graphic library GD version 2, a package for development, it is desirable that it will be without XPM support (dependence on library of a X-server is excluded) and support of FontConfig. It is used for construction of trends and other images in PNG format.
libexpat(devel)	Library of XML-parser, package for development. The interface of management of OpenSCADA and other components are constructed on the basis of language XML.
<i>DB.MySQL module</i>	
libMySQL(devel)	Library for access to MySQL DBMS, a package for development on language C.
<i>DB.SQLite module</i>	
libsqlite3(devel)	Library for access to built in DB SQLite version 3, a package for development.
<i>DB.PostgreSQL module</i>	
libpq	Library for access to PostgreSQL DBMS version more 8.3.0, a package for development.
<i>DB.FireBird module</i>	
FirebirdSS	FireBird DBMS version 2, a package for development. Often is absent in distribution kits of Linux and demands individual loading from an official site ( http://www.firebirdsql.org)!
<i>Transport.SSL module</i>	
libssl(devel)	Library for codifying OpenSSL, a package for development.
<i>DAQ.JavaLikeCalc module</i>	
bison	The program of generation of parsers on the basis of grammar of language.
<i>DAQ.SNMP module</i>	
libsnmp(devel)	Library for access to data of network devices under SNMP protocol, a package for development.

Component	Description
<i>DAQ.System module</i>	
libsensors(devel)	Hardware sensors' library versions 2 and 3, a package for development.
<i>DAQ.Siemens module</i>	
glibc-kernheaders	Linux-kernel headers by library GLibC.
<i>DAQ.SoundCard module</i>	
libportaudio(devel)	Multiplatform library for access to sound controller, a package for development version 19 and higher.
<i>DAQ.OPC-UA module</i>	
libssl(devel)	Library for codifying OpenSSL, a package for development.
<i>Modules: UI.Vision, UI.WebVision, Special.FLibSYS</i>	
libfftw3(devel)	Library for fast Fourie transfer of signals, package for development.
<i>Modules: UI.QTStarter, UI.QTCfg, UI.Vision</i>	
libQT4(devel)	Library for construction of user graphic interface QT version 4.3 and higher, package for development.

OpenSCADA program description

The given document is the description "open source" project of system called "OpenSCADA." OpenSCADA corresponds an open SCADA system constructed on principles of modularity, multiplatform and scalability.

As a policy of development of the given system the "open source" principles are chosen. The choice of the given policy is determined by necessity of creation of open, reliable and public SCADA system. The given policy allows to involve in development, testing, elaboration, distribution and use of a product the significant amount of developers, enthusiasts and other interested persons with minimization and distribution of financial expenses.

The OpenSCADA system is intended for acquisition, archiving, visualization of the information, delivery of operating influences, and also for other related operations characteristic for full-function SCADA systems. Owing to a high level of abstraction and modularity, the system can be used in many adjacent areas.

The OpenSCADA system can be applied:

- on industrial targets, as full-function SCADA system;
- in built in (embedded) systems, as an environment of performance, including inside PLC (programmed logic controllers);
- for construction of various models (technological, chemical, physical, electric processes);
- on personal computers, servers and clusters for acquisition, processing, representation and archiving of the information on system and its environment.

As basic (host) operational system, for development and use, the OS Linux is chosen, which is the optimum compromise in questions:

- reliability (vast majority of servers and clusters works on GNU/Linux);
- flexibility/scalability (in view of the openness and modularity allows to build decisions under any requirements);
- availability (owing to license GPL it is completely free system, and at high qualification of the user and free-of-charge);
- popularity, development, support, prevalence (the system is actively developed by set of enthusiasts, firms and official bodies from all over the world, it gets greater and greater support in the user and corporate market, it is actively implemented into the state structures of the various countries).

As far as the project is developed and realized by principles of multiplatformity, it does not make a problem to port it on other OS, that is planned in the future.

Heart of system is the modular kernel. And depending on that, what modules are connected, system can to act both in a role of various servers, and in a role of various clients, and also to combine these functions in one program. It allows to work in practice client-server architecture of SCADA system on the basis of the same components/modules, saving thus: machine memory, disk space, and also valuable time of programmers.

Server configurations of system are intended for acquisition, processing, delivery of influences, archiving, recording of the information from various sources, and also for granting of this information to clients (UI, GUI, TUI...). The modular architecture allows to expand functionality of a server without its restarting.

Client configurations can be built on the basis of various graphic libraries (GUI/TUI ToolKits), as using a kernel of the program and its modules (by addition to it the module of UI-user interface), and as the independent application, connecting the kernel of OpenSCADA as library.

The opportunity of a flexible configuration of system allows to build decisions under concrete requirements of reliability, functionality and the sizes of system.

1. Functions of the system.

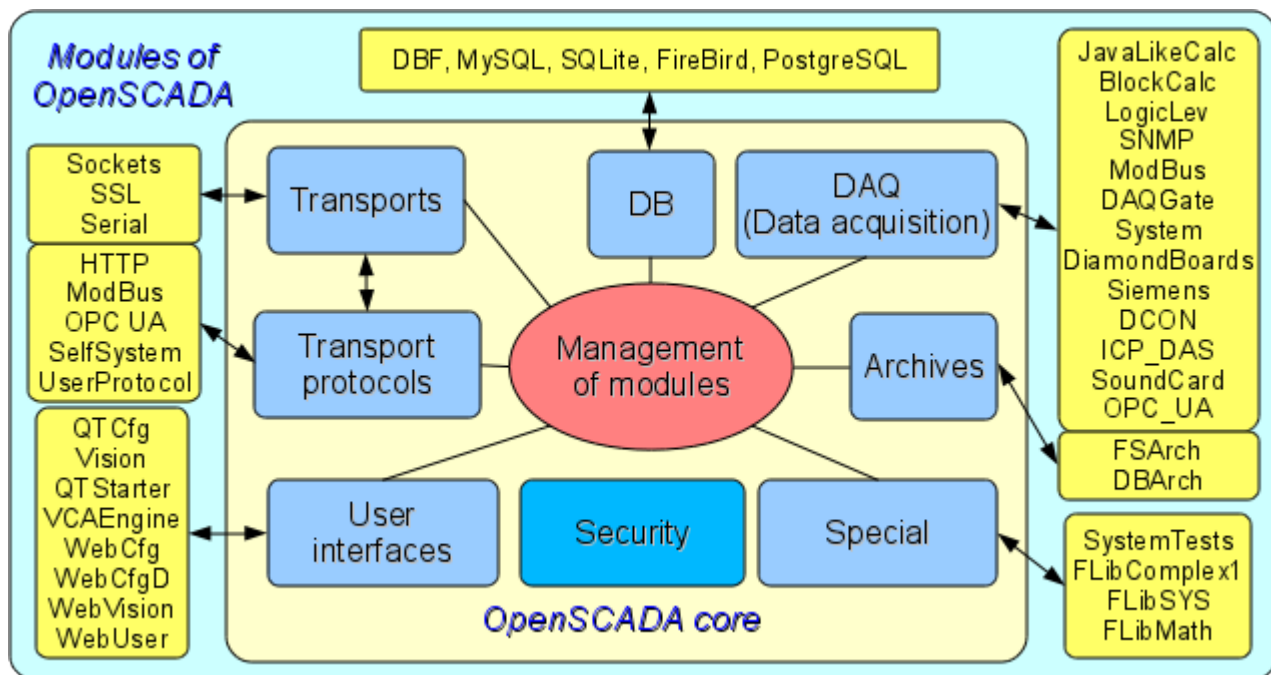


Fig. 1. The block scheme of OpenSCADA system

1.1. Modularity.

For giving flexibility and a high degree of scalability the OpenSCADA system is constructed by a modular principle. Close integration of modules with a kernel of system imposes the great responsibility on process of a writing of modules and enters an element of instability into the system, however owing to an opportunity of creation of the allocated configurations, this danger smooths out with preservation of a high degree of flexibility.

Modules of OpenSCADA system are stored in dynamic libraries. Each dynamic library can contain set of modules of various type. Filling of dynamic libraries by modules is determined by functional connectivity of modules. Dynamic libraries suppose hot replacement, that allows during functioning to update separate parts of system. The method of storage of a code of modules in dynamic libraries is the main for OpenSCADA as far as it is supported practically by all modern operational systems(OS). However it does not exclude an opportunity of development of other methods of storage of a code of modules.

On the basis of modules the following functional parts of OpenSCADA system are realized:

- databases;
- archives (messages and values);
- protocols of communication interfaces;
- communication interfaces, transports;
- sources of data and data acquisition;
- interfaces of the user (GUI, TUI, WebGUI, speech, signal...);
- the additional and special modules.

Management of modules is carried out by a subsystem "Management of modules". Functions of a subsystem are: connection, switching-off, updating of modules, and also other operations connected with modules and libraries of modules.

1.2. Subsystems.

Architecturally the OpenSCADA system shares on subsystems. Subsystems can be of two types: usual and modular. Modular subsystems possess the property of expansion by means of modules. Each modular subsystem can contain set of modular objects. For example the modular subsystem of "Database" contains modular objects of types of databases. The modular object is a root inside of the module.

In total the OpenSCADA system contains 9 subsystems from them 7 subsystems are modular. 9 subsystems of the OpenSCADA system are basic and are present at any configuration. To the list of 9 subsystems new subsystems by means of modules can be added. Subsystems of the OpenSCADA system:

- Archives (modular).
- Databases (modular).
- Safety.
- Interfaces of the user (modular).
- Management of modules.
- Data acquisition (modular).
- Transport protocols (modular).
- Special (modular).
- Transports (modular).

1.3. PLC and other sources of dynamic data. A subsystem "Data acquisition".

For support of sources of dynamic data, whether it be PLC-controllers, communication devices, virtual sources, etc., the subsystem "Data acquisition" is intended. Functions of this subsystem include granting the received data in the structured appearance and maintenance of management with these data, for example the updating of data.

The subsystem "Data acquisition" is modular and, as consequence, contains modular objects of types of sources of dynamic data. For example, for October 2007r, the OpenSCADA system supports following types of sources of data:

- Cards of data acquisition from "Diamond systems".
- Data acquisition from operational systems (OS).
- the Block calculator.
- the Calculator in Java-like language.
- the Transporter of data of a subsystem "Data acquisition" from one OpenSCADA station to another.
- Access to logic controllers by means of the protocol "ModBUS".
- Data acquisition from network devices by means of protocol SNMP.
- The source of data of a logic level of OpenSCADA system.
- Access to highly intellectual logic controllers by means of MPI protocol and communication processor CIF50PB of Hilscher GMBH firm.

Each type of a source is made in the form of the separate module which can be connected/disconnected. Each type of a source can contain separate sources (controllers).

Separately taken controller can contain the parameters of certain by the module types. For example parameters of analog type: the basic information which they gives the value of the integer or real type is. Structurally, the parameter represents the list of attributes which are contained by data. Attributes can be of four base types: symbolical string (text), integer, real and logic type.

Structures of controllers, parameters and their types are contained in the subsystem "Data acquisition", and objects of modules carry out their filling according to own specificity.

The source of dynamic data can be remotes, i.e. connected on the remote OpenSCADA system. For communication with such sources of data the transport type of controllers (Transporter) is used. Function of the given type of a source of data is reflection of sources of data of remote OpenSCADA stations on local station.

1.4. Databases. A subsystem of "Database"

For a data storage of system databases (DB) are everywhere used. With a view of systematization of access and management of databases in OpenSCADA system the subsystem "Database" is provided. For support of various DB/DBMS the subsystem is modular.

In a role of the modular objects, containing in a subsystem, type DB/DBMS acts, i.e. the module of a subsystem "Database", which practically contains realization of access to the certain type of a DB. For example modules: DBF, MySQL, SQLite.

The object of type DB/DBMS, in its turn, contains the list of objects of separated DB of the given type. And the object of a DB contains the list of objects of tables which are contained by data in the tabulated form.

Practically all the data of OpenSCADA system are stored in this or that DB. The toolkit of system allows to transfer easily the data from one type of a DB on another and as consequence provide an optimum selection of DB type under the concrete area of OpenSCADA system. Transfer of the information from one DB to another can be made by two ways. The first is a change of the address of a working DB and save of all system on it, the second is a direct copying the information between DB. Except for copying the function of direct editing of contents of tables of a DB is supported also.

For the organization of the centralized access of the allocated system to a uniform DB two ways are provided. The first is using of network DBMS, for example MySQL. The second way is using of transport type of a DB on local systems for access to one central DB (It is planned.). Function of a transport DB is transfer of queries to a DB on remote OpenSCADA system.

Data can be stored also in a configuration file of system. The mechanism of full reflection of structure of a DB on structure of a configuration file is realized. I.e. the standard configuration can be placed in a configuration file. An essence of such mechanism that by default for example at start without a DB, it is possible to describe the data of system in a configuration file. In the further, these data can be redefined in a DB. Besides for cases of impossibility of start of any DB generally, it is possible to store all data in a configuration file.

For access to databases the mechanism of registration of a DB is used. Registered DB in system are accessible to all subsystems of OpenSCADA system and can be used in their work. Owing to this mechanism it is possible to provide an allocation of data storage. For example, various libraries can be stored and extend independently, and connection of library will consist in simple registration of the necessary DB.

In the further, realization of duplication of a DB by linkage of the registered DB is planned. This mechanism will allow to increase considerably reliability of OpenSCADA system as a whole by reservation of the mechanism of a data storage. (It is planned.)

1.5. Archives. A subsystem "Archives".

Any SCADA system gives an opportunity of archiving the acquisition data, i.e. formation of history of change (dynamics) of processes. Archives, conditionally, it is possible to divide into two types: archives of messages and archives of values.

Feature of archives of messages is that the subject of archiving are, so-called, events. A characteristic attribute of event is time of occurrence of this event. Archives of messages, usually, are used for archiving messages in system, i.e. conducting logs and reports. Depending on a source, messages can be classified by various criteria. For example, it can be reports of emergencies, reports of actions of operators, reports of failures of connection, etc.

Feature of archives of values is their periodicity defined by the time interval between two adjacent values. Archives of values are applied for archiving of history of continuous processes. As far as process is continuous and it's archiving is possible only by introduction of conception of quantization of interrogation of values as differently we receive archives of the infinite sizes, in view of a continuity of the nature of process. Besides, practically, we can receive values with the period limited by sources of data. For example, qualitative enough sources of data, in the industry, data with frequency more 1kHz seldom allow to obtain.

And it without taking into account sensors having even less qualitative characteristics.

For the decision of tasks of archiving data flows in OpenSCADA system the subsystem "Archives" is provided. The subsystem "Archives" allows to conduct both: archives of messages and archives of values. The subsystem "Archives" is modular. The modular object containing in a subsystem "Archives" the type of the archiver acts. The type of the archiver defines the way of a data storage, i.e. storehouse (file system, DBMS, a network, etc.). Each module of a subsystem "Archives" can realize both: archiving of messages, and archiving of values. The subsystem "Archives" can contain set of the archives served by various modules of a subsystem.

The message in OpenSCADA system is characterized: by date, by level of importance, by category and the text of the message. Date of the message specifies for the period of creation of the message. The level of importance specifies a degree of importance of the message. The category determines the address or the conditional identifier of a source of the message. Usually, the category contains a full way to a source of the message in system. The text of the message, actually, also carries meaning content of the message.

During archiving messages are passed through the filter. The filter works on a level of importance and a category of the message. The level of the message in the filter specifies that it is necessary to pass messages with specified or higher level of importance. To filtering on a category templates are used, which define what messages are applied to pass. Each archiver contains own options of the filter. Consequently it is possible to create easily various specialized archivers for archive of messages. For example archivers of messages it is possible to dedicate on:

- logs for storage of the debugging information and other working information of a server;
- various reports (the report of actions of clients, the report of infringements and exceptions, the report of events...).

The archive of values in system OpenSCADA acts as an independent component which includes the buffer processable by archivers. Key parameter of archive of value is the source of data. In a role of a source of data attributes of parameters of OpenSCADA system and also other external sources of data (a passive mode) can act. Other sources of data can be: network archivers from remote OpenSCADA systems, the environment of programming of OpenSCADA system, etc.

Key component of archiving of values of continuous processes is the buffer of values. The buffer of values is intended for intermediate storage of a file of the values received with certain periodicity (quantum of time). The buffer of values is used as for direct storage of big arrays of values in archives of values, before direct "retire" on physical carriers, and for manipulations with the staff of values, i.e. in functions of range-accurate query of values and their placement in buffers of archives.

For the organization of the dedicated archivers, in the allocated systems it is possible to use transport type of the archiver (It is planned.). Function of transport type of the archiver is reflection of the remote central archiver on local system. As consequence, archivers of transport type carry out data transmission between local system and the archiver of the remote system, hiding from subsystems of local system the real nature of the archiver.

1.6. Communications. Subsystems "Transports" and "Transport protocols".

As far as the OpenSCADA system is pawned as is high-scaled system that support of communications should be flexible enough. For satisfaction of a high degree of flexibility, communications in OpenSCADA system are realized in subsystems "Transports" and "Transport protocols" which are modular.

The subsystem "Transports" is intended for an exchange of the not structured data between OpenSCADA system and external systems. In a role of external systems can act even remote OpenSCADA systems. Not structured data are understood as a file of symbols of the certain length. The modular object containing in a subsystem "Transports", the type of transport acts. The type of transport defines the mechanism of transfer of not structured data. For example it can be:

- sockets (TCP/UDP/UNIX);
- channels;
- shared memory.

The subsystem "Transports" includes support of input and output transports. Input transport is intended

for service of external queries and sending of answers. Output transport, on the contrary, is intended for sending messages and expectation of the answer. Consequently, input transport contains a configuration of the given station as server, and output transport contains a configuration of the remote server. The module of a subsystem "Transports" realizes support both: input and output transports.

The subsystem "Transport protocols" is intended for structurization of data received from a subsystem "Transports". As a matter of fact, the subsystem "Transport protocols" is continuation of a subsystem "Transports" and carries out functions of check of structure and integrity of the received data. So, for the indication of the protocol together with which transport should work, the special configuration field is provided. The modular object containing in a subsystem "Protocols" is the protocol. For example, transport protocols can be:

- HTTP (Hyper Text Transfer Protocol);
- SelfSystem (OpenSCADA the system protocol).

The full chain of connection can be written down as follows:

- the message is transferred in transport;
- transport transfers the message to the protocol, connected with it, by creation of new object of the protocol;
- the protocol checks integrity of data;
- if all data have come, transport must be informed about the termination of expectation of data and to transfer it the answer, differently to inform, that it is necessary to expect still;
- transport, having received {confirmation, sends the answer and delete object of the protocol;
- if confirmations are not present, the transport continues expectation of data, and in the case of their receipt transfers them to the saved object of the protocol.

Protocols for output transports are supported also. The output protocol incurs function of dialogue with transport and realization of features of the protocol. The internal side of access to the protocol is realized by data-flow way with own structure for each protocol module. Such mechanism allows to carry out transparent access to external system, by means of transport, simply specifying a name of the protocol by means of which to serve transfer.

Owing to standard API-access to transports of OpenSCADA system it is possible to change easily a way of data exchange not touching exchanging systems. For example, in the case of a local exchange it is possible to use faster transport on the basis of shared memory, and in the case of an exchange through the Internet and a local network to use TCP or UDP sockets.

1.7. Interfaces of the user. A subsystem "Interfaces of the user".

SCADA-systems as a class, assume presence of user interfaces. In OpenSCADA, for granting the user interfaces, the subsystem "The user interfaces" is provided. The user interface of OpenSCADA system is understood not only as the environment of visualization from which the end user should work, but also as everything, that concerns the user, for example:

- environments of visualization;
- configurators;
- alarming and signalling devices.

The subsystem "The user interfaces" is modular. As modular object of a subsystem the concrete interface of the user actually acts. Modularity of subsystem allows to create various interfaces of users on various GUI/TUI libraries and to use optimal of decisions in particularly taken case, for example, for environments of performance of programmed logic controllers it is possible to use configurators and visualizers on the basis of Web-technologies (WebCfg, WebUI), and in case of stationary workstations to use the same configurators and visualizers, but on the basis of libraries QT, GTK.

1.8. Security of system. A subsystem "Security".

The OpenSCADA system is the branched out system which consists of ten subsystems and can include set of modules. Consequently, granting of unlimited access by all to these resources is at least unsafe. Therefore, for differentiation of access in OpenSCADA system, the subsystem of "Security" is provided. The basic functions of a subsystem "Security" are:

- storage of registration records of users and groups of users;
- authentication of users;
- check of access rights of the user to this or that resource.

1.9. Management of libraries of modules and modules. A subsystem "Management of modules".

The OpenSCADA system is constructed by a modular principle that means presence of set of modules with which it is necessary to operate. For performance of function of management by modules of OpenSCADA system the subsystem "Management of modules" is provided. All modules, for the present moment are delivered in system by means of shared libraries (containers). Each container can contain set of modules of various type.

The subsystem "Management of modules" realizes the control over the status of containers and allows to carry out hot addition, removal and updating of containers and modules containing in them.

1.10. Unforeseen opportunities. A subsystem "Special".

Certainly, to provide all probable functions it is impossible, therefore in OpenSCADA system the subsystem "Special" is provided. The subsystem "Special" is modular and is intended for addition in OpenSCADA system unforeseen functions by modular expansion. For example, by means of a subsystem "Special" can be realized:

- tests of OpenSCADA system and its modules;
- libraries of functions of the user programming.

1.11. The user functions. Objective model and the environment of programming of system.

Any modern SCADA system should contain the mechanisms giving an opportunity to program at the user level, i.e. to contain the environment of programming. The OpenSCADA system contains such environment. By means of the environment of programming of OpenSCADA system it is possible to realize:

- Algorithms of management of technological processes.
- Large dynamic models of real time of technological, chemical, physical and other processes.
- Adaptive mechanisms of management on models.
- The user procedures of management by internal functions of system, its subsystems and modules.
- Flexible formations of structures of parameters at a level of the user, with the purpose of creation of parameters of non-standard structure and its filling on algorithm of the user.
- Auxiliary calculations.

The environment of programming of OpenSCADA system represents a complex of assets organizing the computing environment of the user. Into structure of a complex of assets are included:

- objective model of OpenSCADA system;
- modules of libraries of functions;
- computing controllers of a subsystem "Data acquisition" and other calculators.

Modules of libraries of functions give set of functions of the certain orientation expanding objective model of system. Libraries can be realized both: by the set of functions of the fixed type, and functions supposing free updating and addition.

Libraries of functions of the fixed type can be given by standard modules of system, organically

supplementing objective model. Functions of such libraries will represent the interface of access to assets of the module at a level of the user. For example, "The environment of visual data presentation" can give functions for delivery of various messages. Using these functions the user can realize interactive algorithms of communication with system.

Libraries of functions of free type give the environment of a writing of the user functions on one of programming languages. Within the limits of the module of libraries of functions mechanisms of creation of libraries of functions can be given. So, it is possible to create libraries of devices of technological processes, and in a consequence to use them by linkage. Various modules of libraries of functions can give realizations of various programming languages.

On the basis of the functions given by objective model, computing controllers are under construction. Computing controllers carry out linkage of functions with parameters of system and the mechanism of calculation.

2. SCADA systems and their structure.

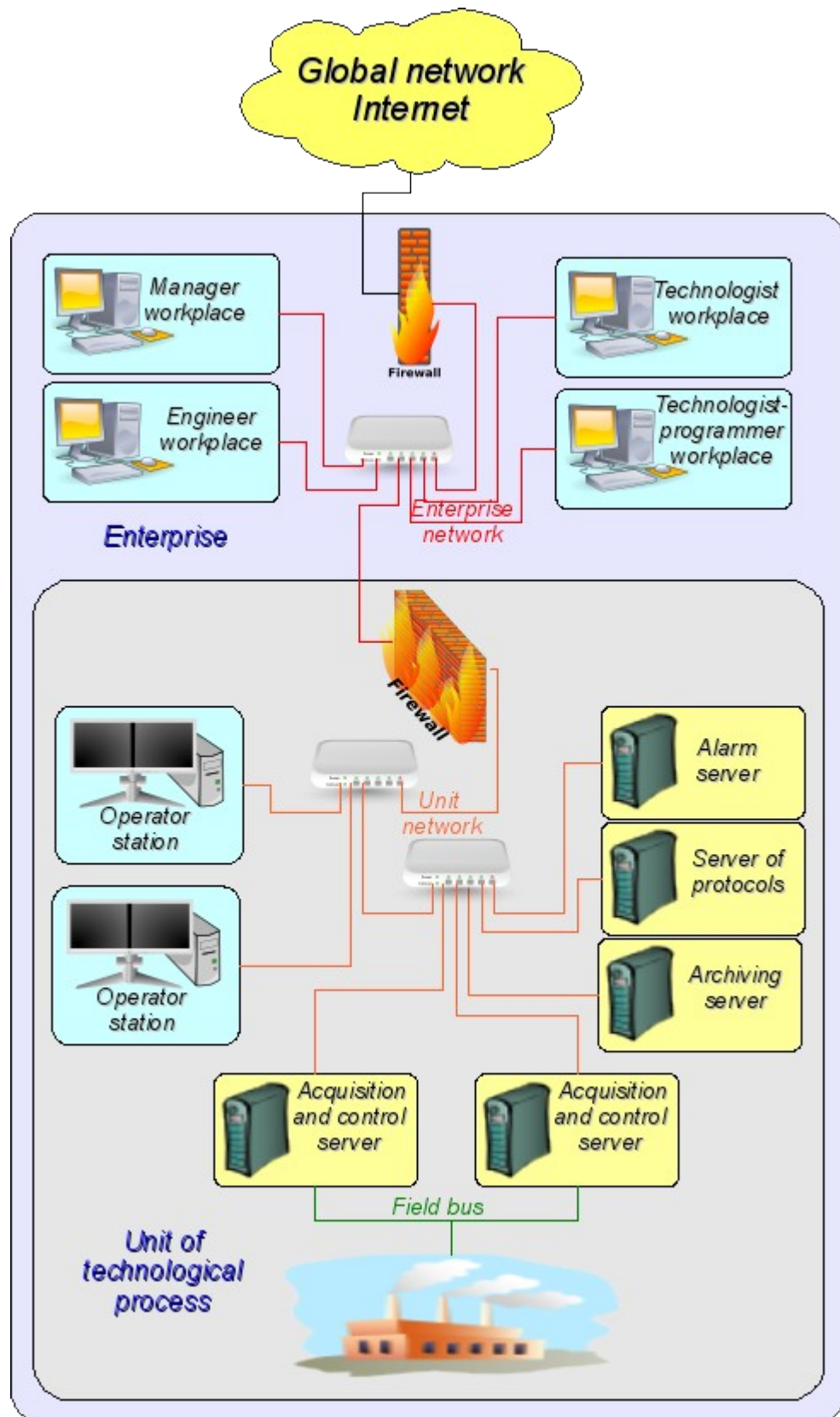


Fig. 2. SCADA-system.

SCADA (Supervisory Control And Data Acquisition), in a general view, have the allocated architecture like represented on fig. 2. Elements of SCADA systems, in sense of the software, carry out following functions:

The acquisition server: represents a task or group of tasks engaged in data acquisition from sources of data, or act in a role as a source of data. Into tasks of a server enters:

- reception and-or formation of data;
- data processing;
- service of queries about access to data;
- service of queries about updating of data.

The server of archiving: represents a task or group of tasks engaged in archiving of data. Into tasks of the server enters:

- archiving of data of SCADA-system;
- service of queries about access to contemporary records;
- import/export of archives.

The journaling server: represents a task or group of tasks engaged in archiving of messages. Into tasks of the server enters:

- archiving of messages of units of SCADA-system;
- service of queries about access to archival messages;
- import/export of archives.

The alarm server: represents a task or group of tasks carrying out functions of the server of recording concerning a narrow category of messages of the signal system.

The operator working place: represents constantly functioning GUI (Grafical User Interface) application executed in an one-monitor, multimonitor or panel mode and carrying out functions:

- granting of the user interface for the control over a condition of technological process;
- granting of an opportunity of formation of operating influences;
- granting of an opportunity of studying and the analysis of history of technological process;
- granting of toolkit for generation of the reporting documentation.

The engineer working place: represents GUI application used for configuration of SCADA system. Into tasks of the application enters:

- granting of toolkit for manipulation with system functions of system;
- granting of toolkit of a workplace of the operator;
- granting of toolkit for manipulation with architecture of SCADA system as a whole (distribution of functions between stations, creation, removal of stations...).

The chief working place: represents GUI application, as a rule, executed in an one-monitor mode and carrying out functions:

- granting of the user interface for the control over a condition of technological process;
- granting of toolkit for studying and the analysis of history of technological process as is direct from an active server, and on the basis of separate archives;
- granting of toolkit for generation of the reporting documentation.

The technologist working place: completely includes functions of a workplace of the operator plus model of technological process (without direct communication with technological process).

The work planner working place: completely includes functions of a workplace of the technologist plus toolkit for creation of models of technological processes.

3. Ways of configuration and using of OpenSCADA system.

3.1. Simple server connection.

In the elementary case the OpenSCADA system can be configured in a server mode (fig. 3.1) for acquisition and archiving of data. The given configuration allows to carry out following functions:

- interrogation of controllers;
- archiving of values of parameters;
- service of client queries about reception of various data of a server;
- granting of the configuration WEB-interface;
- the remote configuration from OpenSCADA system by means of the QT-interface or other local interface.
- secondary regulation (regulation in computing controllers);
- modeling, adjusting and supplementing calculations in computing controllers.

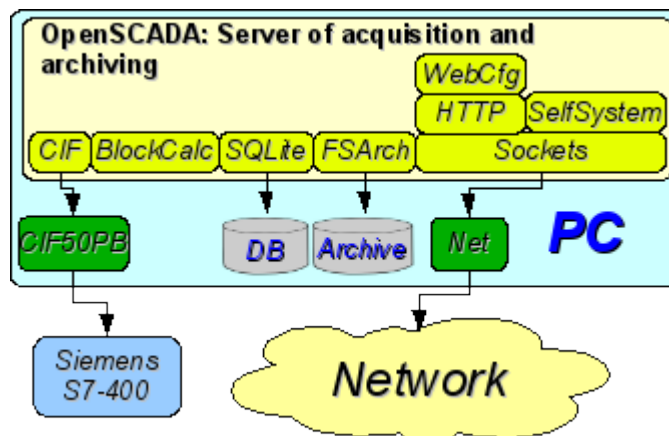


Fig. 3.1. Simple server connection.

3.2. The duplicated server connection.

For increasing of reliability and productivity the OpenSCADA system supposes plural reservation (fig. 3.2) at which controllers of one copy are reflected in other. At use of a similar configuration distribution of loading of interrogation/calculation at various stations is possible. The given configuration allows to carry out functions:

- interrogation of controllers;
- archiving of values of parameters;
- service of client queries about reception of various data of a server;
- reservation of parameters;
- reservation of archives;
- distribution of loading of interrogation on servers;
- granting of the configuration WEB-interface;
- secondary regulation (regulation in computing controllers);
- modeling, adjusting and supplementing calculations in computing controllers with an opportunity of distribution of loading on servers.

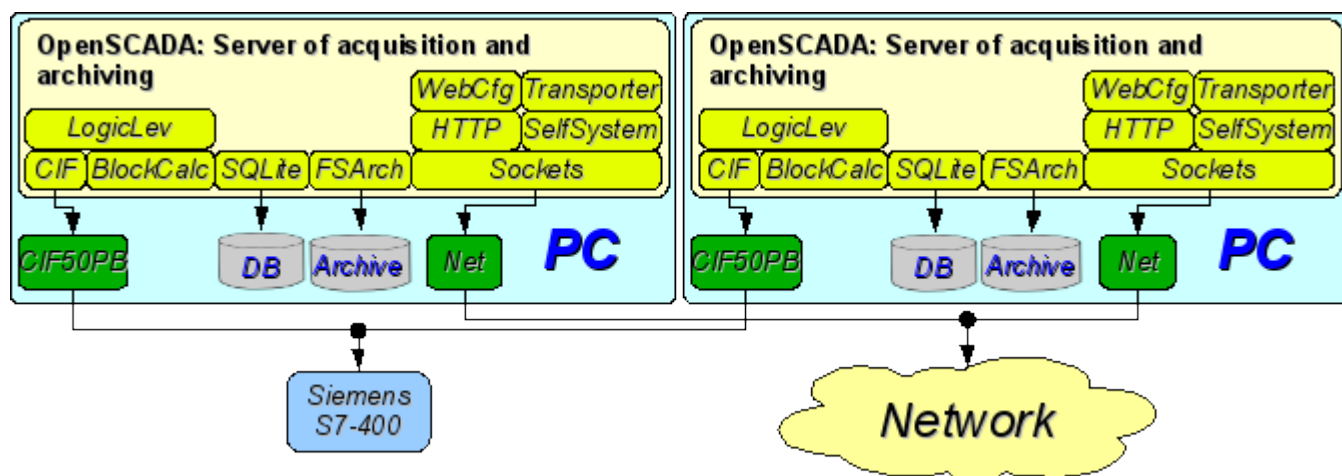
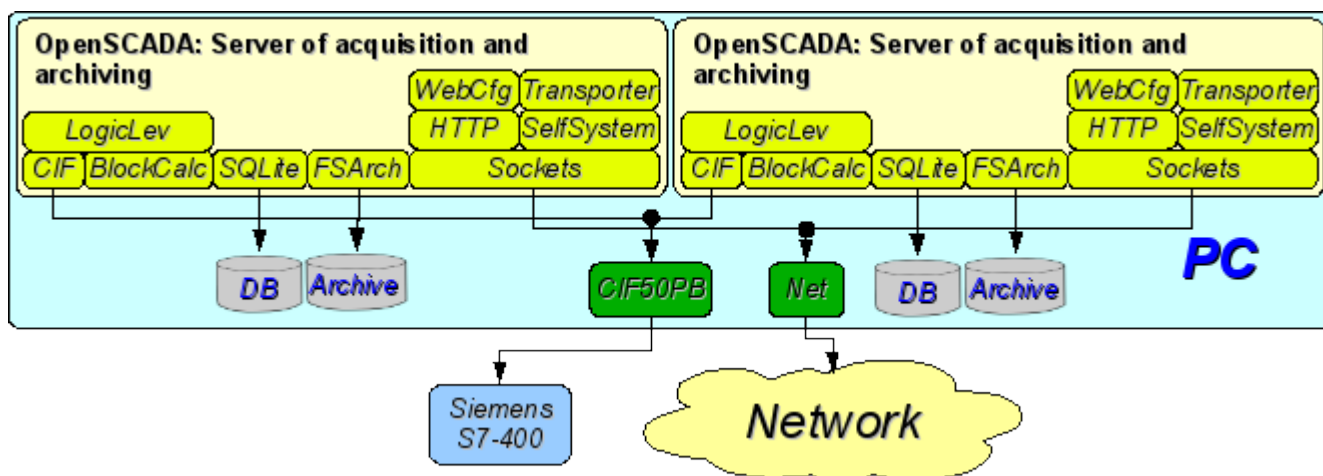


Рис. 3.2. The duplicated server connection.

3.3. The duplicated server connection on one server.

Special case of the duplicated connection is the duplicated connection within the limits of one server (fig. 3.3), that is start of several stations by one machine with a crossing of parameters. The purpose of the given configuration is increase of reliability and fault tolerance of system by reservation of software.



3.3. The duplicated server connection on one server.

Fig.

4.4. Client access by means of the Web-interface. A place of the manager.

For visualization of data containing on a server, the good decision is to use the user WEB-interface (fig. 3.4). The given decision allows to use a standard WEB-browser at the client side and therefore is the most flexible as it is not adhered to one platform, i.e. is multiplatform. However this decision has essential imperfections: low productivity and reliability. In this connection it is recommended to use the given method for visualization of noncritical data or data having a reserve highly reliable way of visualization. For example, the good decision will be using of this method at the heads of plants where always exists place(attendant position) with reliable way of visualization. The given configuration allows to carry out following functions:

- interrogation of a server for data acquisition of visualization and a configuration;
- visualization of data in a kind accessible to understanding;
- formation of protocols, reports;
- manipulation with parameters supposing change.

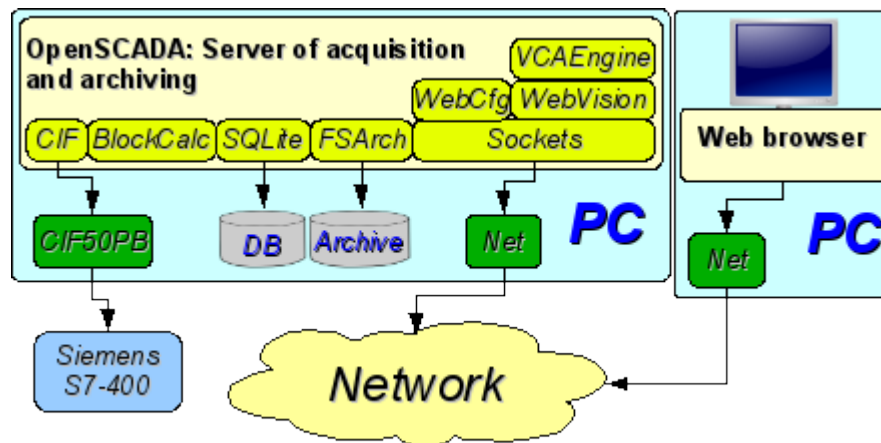


Fig. 3.4. Client access by means of the Web-interface. A place of the manager.

3.5. The automated workplace (place of the manager/operator).

For visualization of critical data, and also in case of if high quality and productivity is required, it is possible to use visualization on the basis of OpenSCADA system configured with the GUI module (fig. 3.5). The given configuration allows to carry out following functions:

- interrogation of a server for updating current values;
- visualization of the interrogated data in a kind accessible to understanding;
- formation of protocols and reports;
- manipulation with parameters supposing changes.

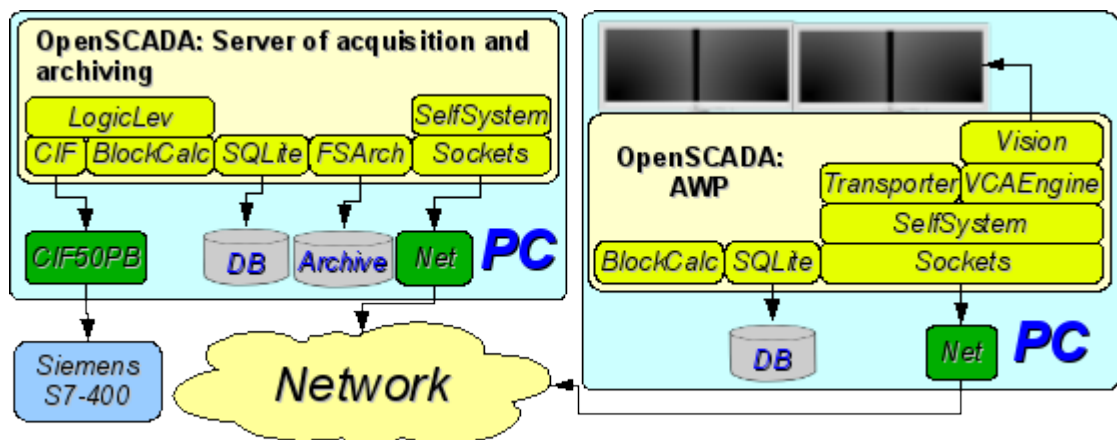


Fig. 3.5. The automated workplace (place of the manager/operator).

3.6. Automated workplace with a server of acquisition and archiving on the single machine (a place of the operator, model...).

The full-function client-server configuration on the single machine (fig. 3.6) can be used for increasing of reliability of system as a whole by start of the client and a server in different processes. The given configuration allows, without consequences for a server, to stop the client and to do with it various preventive works. It is recommended for use at stations of the operator by installation of two machines combining in itself the station of the operator and redundant server. The given configuration allows to carry out following functions:

- interrogation of controllers;
- service of client queries;
- visualization;
- delivery of operating influences;
- generation of protocols and reports;
- secondary regulation;
- modeling, adjusting and additional calculations in computing controllers;
- acquisition and visualization of the information on a personal computer, a server....

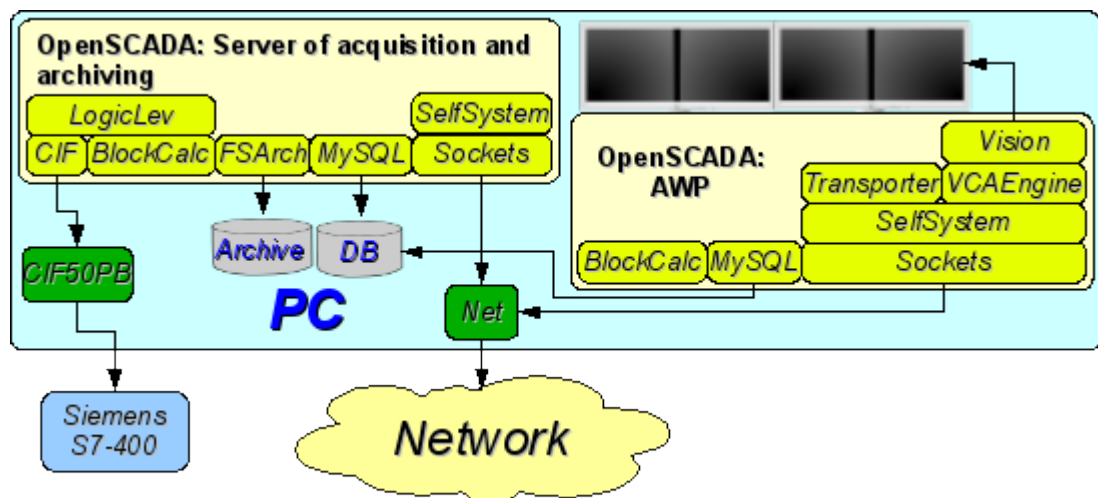


Fig. 3.6. Automated workplace with a server of acquisition and archiving on the single machine (a place of the operator, model...).

3.7. The elementary mixed connection (model, demonstration, configurator...).

The mixed connection combines functions of a server and the client (fig. 3.7). It can be used for test, demonstration functions, and also for granting models of technological processes as a unit. In this mode following functions can be carried out:

- interrogation of controllers;
- service of client inquiries;
- visualization;
- delivery of operating influences;
- generation of protocols and reports;
- secondary regulation;
- modeling, adjusting and supplementing calculations in computing controllers;
- acquisition and visualization of the current information on a personal computer, a server, model...;
- a configuration of databases, connections, etc.

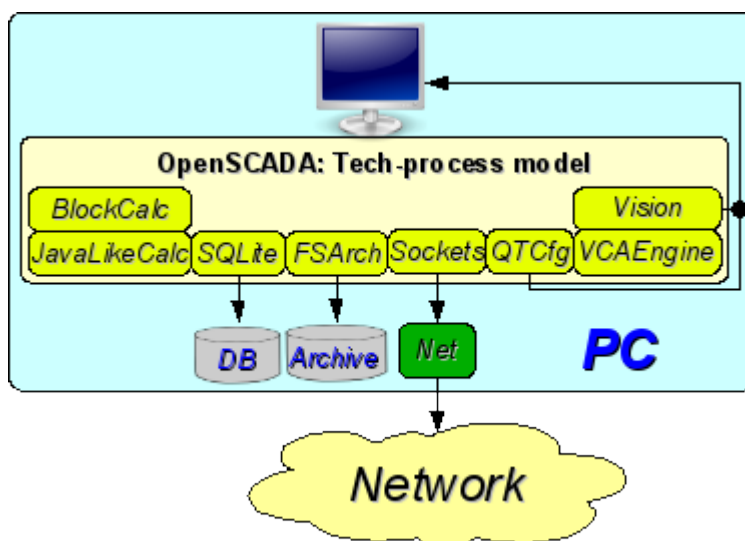


Fig. 3.7. The elementary mixed connection (model, demonstration, configurator...).

3.8. The steady, allocated configuration.

The given configuration is one of variants of steady/reliable connection (fig. 3.8). Stability is reached by distribution of functions on:

- to servers of interrogation;
- to the central server of archiving and service of client queries;
- to clients: automated workplaces and WEB-clients.

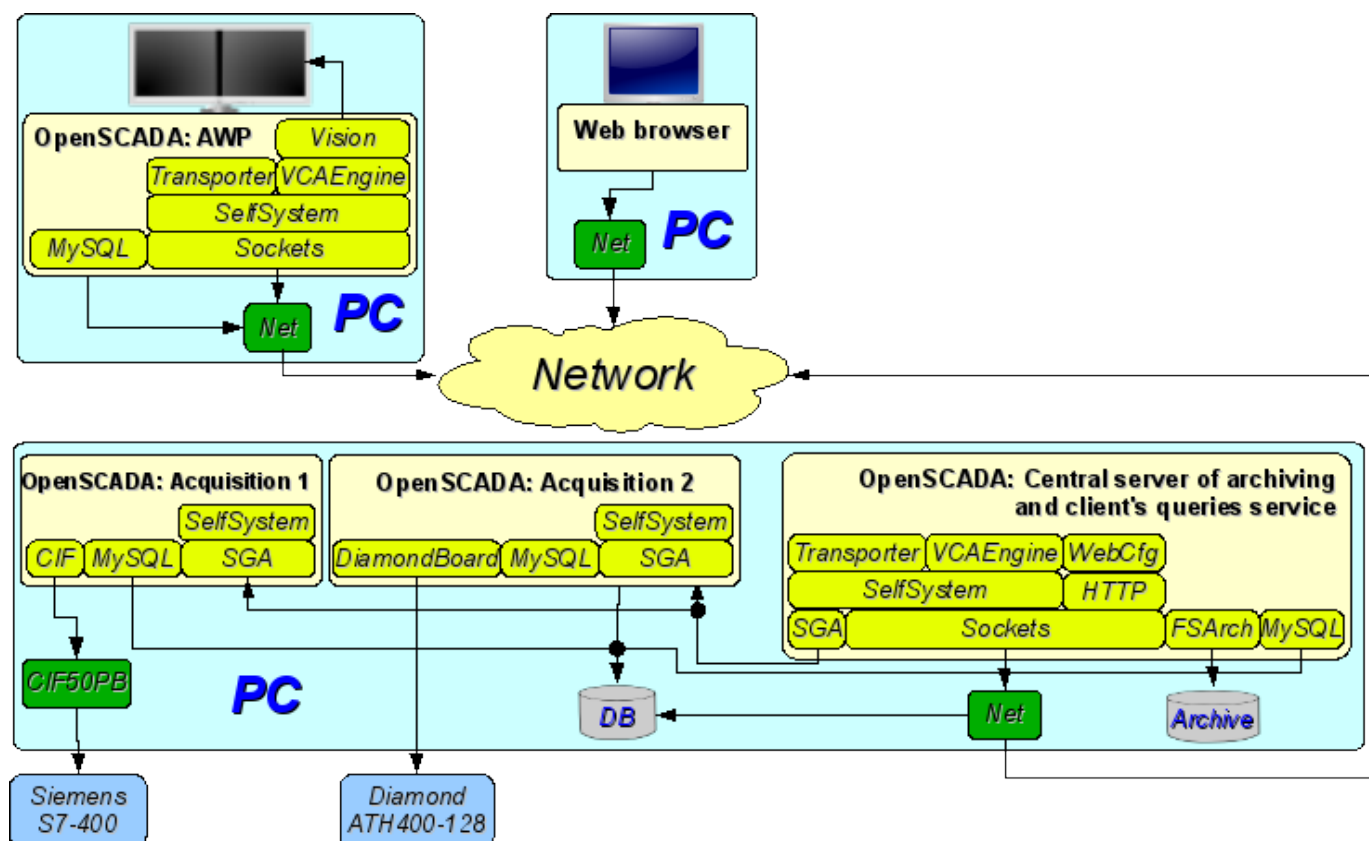


Fig. 3.8. The steady, allocated configuration.

The server of interrogation is configured on the basis of OpenSCADA system and represents the task (group of tasks) engaged with interrogation of the controller (group of controllers of the same type). The received values are accessible to the central server through any transport which support is added by connection of the corresponding module of transport. For decrease in frequency of interrogation and size of the network traffic the server of interrogation can be equipped with small archive of values. The configuration of a server of interrogation is stored in one of accessible DB.

The central server of archiving and service of client queries carries out function of the centralized acquisition and processing of parameters of servers of interrogation and their values. Access to servers of interrogation is carried out by means of one of accessible in OpenSCADA transports+protocols (for example it is SGA). For granting the uniform interface of access to parameters and controllers the module Transporter which reflects data of servers of interrogation on structure of local parameters is used.

For performance of internal calculations and the additional analysis of parameters computing controllers are used.

For versatile and deep archiving various modules of archives are used.

For access of clients to a server are used accessible for OpenSCADA network transports, for example it is Sockets, and transport protocols, for an example it is the protocol OpenSCADA "SelfSystem".

The configuration of the central server is stored in one of accessible DB (for example it is network DBMS MySQL).

For granting the user WEB-interface the module WebCfg by means of the transport protocol "HTTP" is used.

Various clients, among them automated workplaces and WEB-clients, are carried out on the separated machines in necessary quantity. The automated workplace is realized on the basis of OpenSCADA system. Its functions include interrogation of values of parameters from the central server and their visualization on the GUI interface(s). For reception of parameters in an automated workplace the module of reflection of the remote parameters Transporter, also, is used. For granting access to archives the module of archive of network type can be used. The configuration of an automated workplace can be stored in one of accessible DB (for example it is network DBMS MySQL, located on the machine of the central archiving server).

4. Configuration and adjustment of the system.

As it can be seen in the section above, OpenSCADA allows configuration for execution in various roles. Support of this possibility is provided by the developed mechanisms for configuration and storage of configuration data. This section contains a description of these mechanisms, designed to demonstrate the flexibility and diversity, thereby allowing to use OpenSCADA to 100%.

In describing the configuration mechanisms and methods of its storage in this section it will be focused the description of system-wide mechanisms. Features of the configuration of modules of subsystems of OpenSCADA are provided in their own module's documentation.

In OpenSCADA it is used the formalized approach to describing the configuration interfaces based on XML. In fact, features of the component's configuration are provided by the component itself, thereby running through the whole system, as the nervous system of the organism. In terms of OpenSCADA it is called the interface of control of OpenSCADA (Control interface). On the basis of the control interface the graphical interfaces of the user configuration are generated by means of modules of OpenSCADA. This approach has the following important advantages:

- Scalability. You can connect only the required configuration modules or use only the remote mechanisms.
- Excluding the need to update the configurators with the addition of new modules/functions, as well as the exclusion of "swelling" of the configurator, providing the support for all of history of now unnecessary and obsolete modules/functions.
- Simplicity of the creation of the graphical interfaces of configuration on the different basis owing to the clear formality.
- The possibility of dynamic configuration is available, ie configuration can be performed directly while the running of the system both locally and remotely, directly controlling the result.
- The simple and special extensibility of the configuration interface by adding the configuration fields on the control interface's description language only in the required components.

In OpenSCADA the three configuration modules on the different basis of visualization are provided. Lets observe them and their configuration options:

- Configuration module on the GUI library QT (<http://qt.nokia.com/products>) - [UI.QTCfg](#). Provides an advanced configuration interface, allowing to operate as a local station and the remote ones in the local and global networks, including secure connection.
- Configuration module based on the dynamic WEB-technologies (DHTML) - [UI.WebCfgD](#). Provides an advanced configuration interface, allowing to operate as a local server's station, and the remote stations in the local and global networks, including work on the secure connection. Client connection is provided through the usual Web-browser.
- Configuration module based on the static WEB-technologies (XHTML) - [UI.WebCfg](#). Provides an adequate configuration interface that allows to manage the local server's station via the usual Web-browser.

Configuration values, changed in the configurators, as well as most of the data are stored in databases (DB). Given the modularity of subsystems "DB", there can be different database. Moreover, there is the possibility of storing different OpenSCADA parts in different databases of the same type and in the database of different types as well.

In addition to the database configuration information may be contained in the OpenSCADA configuration file, and passed through the command line parameter's when you call OpenSCADA. However, the saving of configuration changes in the configurators is only available in the database. Standard name of the OpenSCADA configuration file is /etc/oscada.xml. The format of the configuration file and command line parameters we'll examine in the separate section.

Further examining of the OpenSCADA configuration will be based on the interface of the configurator UI.QTCfg, but the principles of work will be fully consistent with the rest of the configurators owing to the generality in the control interface of OpenSCADA.

We will start examining with the configuration of system parameters of OpenSCADA, which is located in the three tabs at the root page of the station:

- Tab "Station" contains basic information and configuration field of the station, Fig.4a. Here are the provided fields and comments on them:
 - *ID* - contains information about the station's identifier. It is specified by the command line parameter -Station. When loading it is sought the section in the configuration file appropriate to the station identifier, and if not detected, it uses the first available one.
 - *Station* - indicates the localized station's name.
 - *Program* - contains information on the program name. Usually it is OpenSCADA or name of solution based on OpenSCADA.
 - *Version* - contains the information on the current version of the programme.
 - *Host name* - contains the information on the name of the machine that runs the station.
 - *System user* - contains the information about the user on whose behalf the program is executed in the system (OS).
 - *Operation system* - contains the information about the name and version of operation system, operation system kernel on which the program is executed.
 - *Frequency (MHZ)* - contains the information about the frequency of the CPU, which runs the program. The value of frequency is checked every 10 seconds and allows you to monitor its change, for example, by the power management mechanisms.
 - *Realtime clock resolution (msec)* - contains information about the possibility or resolution of real-time clock of the operation system. It allows you to orient with the minimum interval of time of periodic tasks, for example, for task of data acquisition.
 - *Internal charset* - contains information about the charset in which text messages are stored within the program.
 - *Config file* - contains information about the configuration file used by the program. Set by the command-line parameter -Config.
 - *Work directory* - indicates the working directory of the station. It is used in relative addressing of the objects in the file system, for example, database files. It allows the user to save the modified system data to another database. The value of this field is not stored in the database, but can be changed only in the "WorkDB" section of the configuration file.
 - *Icons directory* - indicates the directory containing the program icons. If the configuration navigation tree have no icons, then you have incorrectly entered the value of this field.
 - *Modules directory* - indicates the directory of modules for OpenSCADA. If the value of this field is incorrect, then when at start you will not see any graphical interface, but the only information in the console on the correct running of the OpenSCADA kernel.
 - *Work DB* - indicates the working database (DB), namely, the database used to store basic data of the program. Changing of this field notes all objects as modified that allows you to save or to load station's data from the specified main database.
 - *Save system at exit* - points to the need to save the changed data at finishing.
 - *Save system period* - indicates the frequency in seconds with which to save the changed station's data.
 - *Language* - indicates the language of program's messages. Changing of this field is acceptable, but leads to a change of messages' language only for the interface and dynamic messages!
 - *Text variable's base language* - is used to activate the support of multilingual text variables by specifying a non-empty basic language. The value of the basic language is selected from the list of bi-character language code, usually only the current and the base language is in the list. Further for the text variables in the non basic language in the tables of the database it will be created the separate columns. Under the text variables the all text fields of configurator, which can be translated into another language are meant. Numbers and other symbolic values are not in their number and are not translated.
 - *Messages:* - section of the parameters' group that are processing by the work and messages of the stations:
 - *Least level:* - indicates the level of messages beginning from which they are to be processed. Messages below this level will be ignored. It is necessary, for example, to exclude from processing the debug messages of level 0.
 - *To syslog* - indicates the need of sending the message to the system logger, the mechanism of operation system for work with system messages and software. When

this option is enabled the possibility appears to manage and control the OpenSCADA messages by the mechanisms of OS.

- *To stdout* - indicates the using as a standard mechanism to display the message the output to the console. Disabling of this feature will eliminate the entire output in the console, unless you specify the following parameter.
- *To stderr* - indicated the using as a standard mechanism to display the message the error output, it is also usually sent to the console.
- *To archive* - indicated the need for output of the messages in the messages' archive of OpenSCADA. This option is usually enabled and its disabling leads to the actual disabling of the archiving at the station.
- Tab "Subsystems" tab contains the list of subsystems (Fig. 4b) and allows you to jump directly to them using the context menu.
- Tab "Tasks" contains the table with opened tasks by OpenSCADA components (Fig.4c). From table you can get several information about the tasks, and also set CPUs for tasks of multi-processors systems.
- Tab "Help" tab contains the brief help for that page, Fig. 4d. In this case, it is the available command line parameters and fields of configuration file for this page.

To modify the fields of this page it may be required the super user's rights. Get these rights you can by means of including your user into the superuser's group "root", or by entering the station from the superuser "root".

We must mention another one important point: the fields of the identifiers of all OpenSCADA objects are unacceptable for direct editing, because they are keys for storage of objects' data in the database. However, to change the object's identifier you can by the command of cutting and the further pasting of the object (Cut-> Paste) in the configurator.

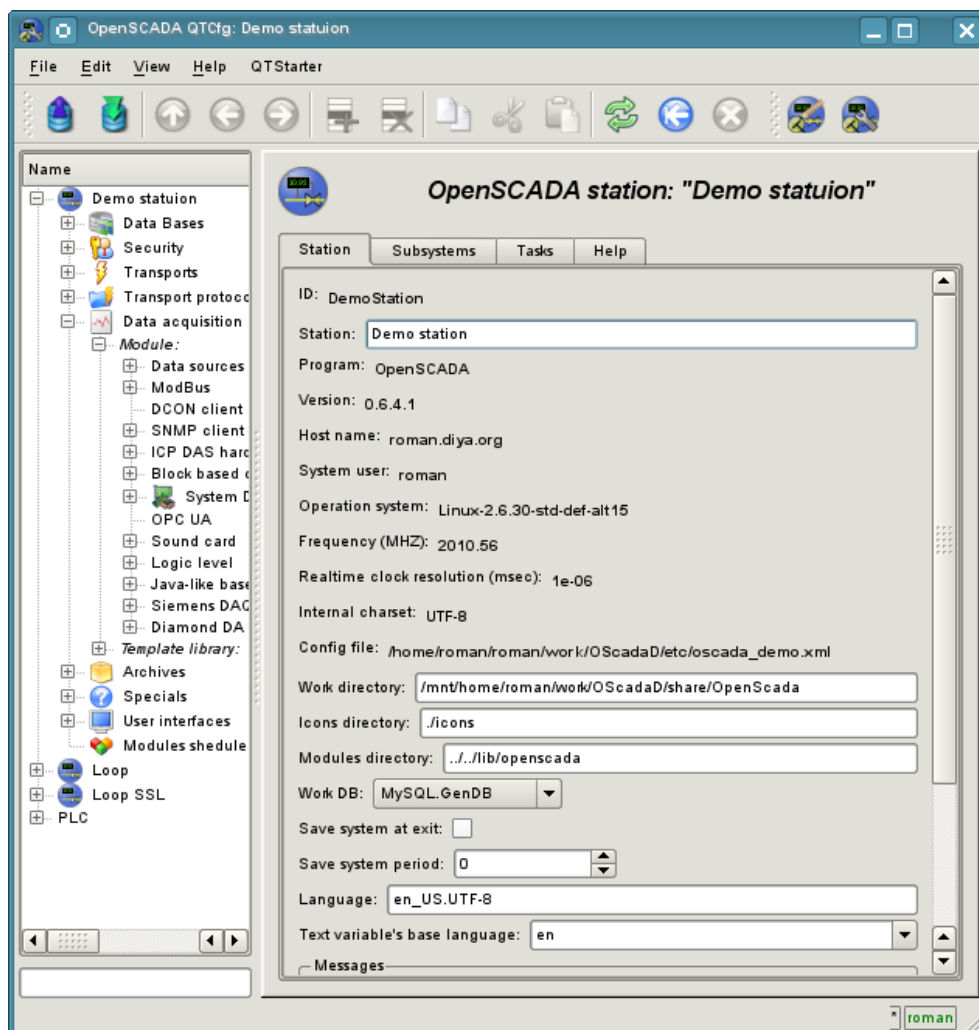


Fig. 4a. "Station" tab of the main page of the configuration of the station.

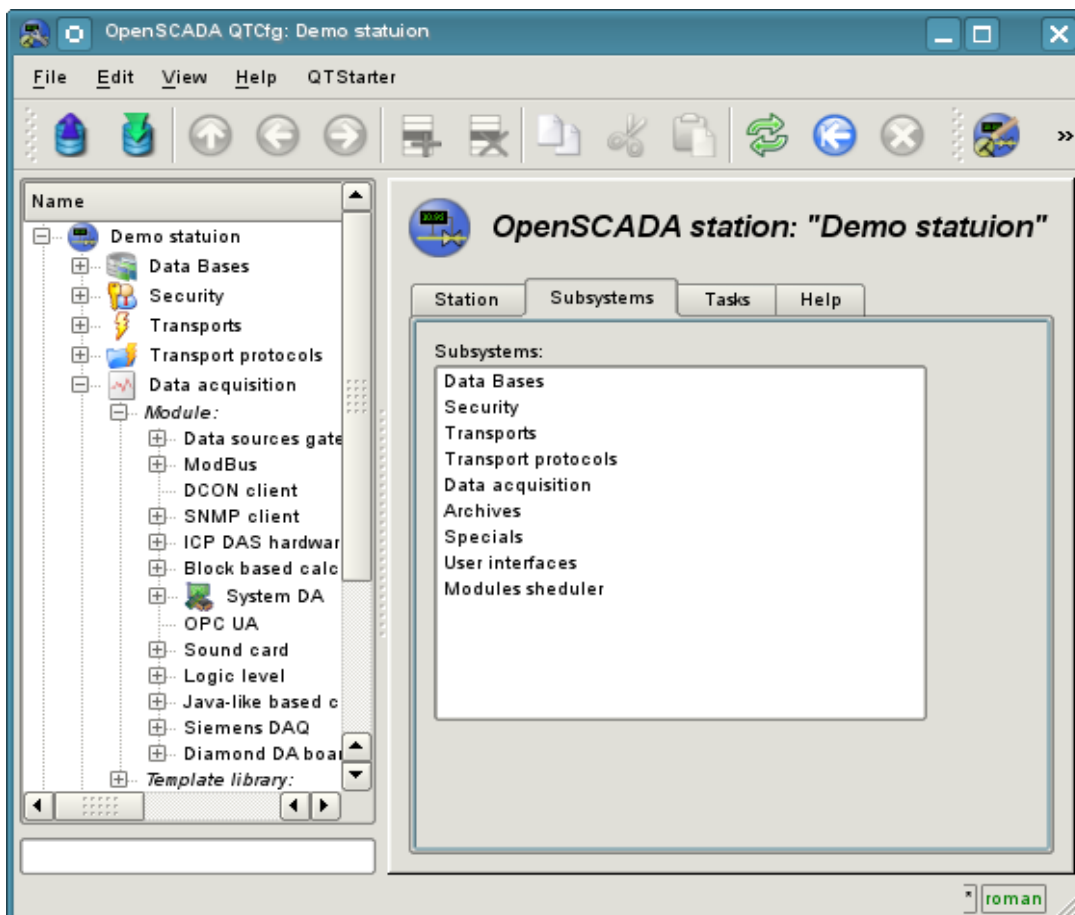


Fig 4b. "Subsystems" tab of the main page of the configuration of the station.

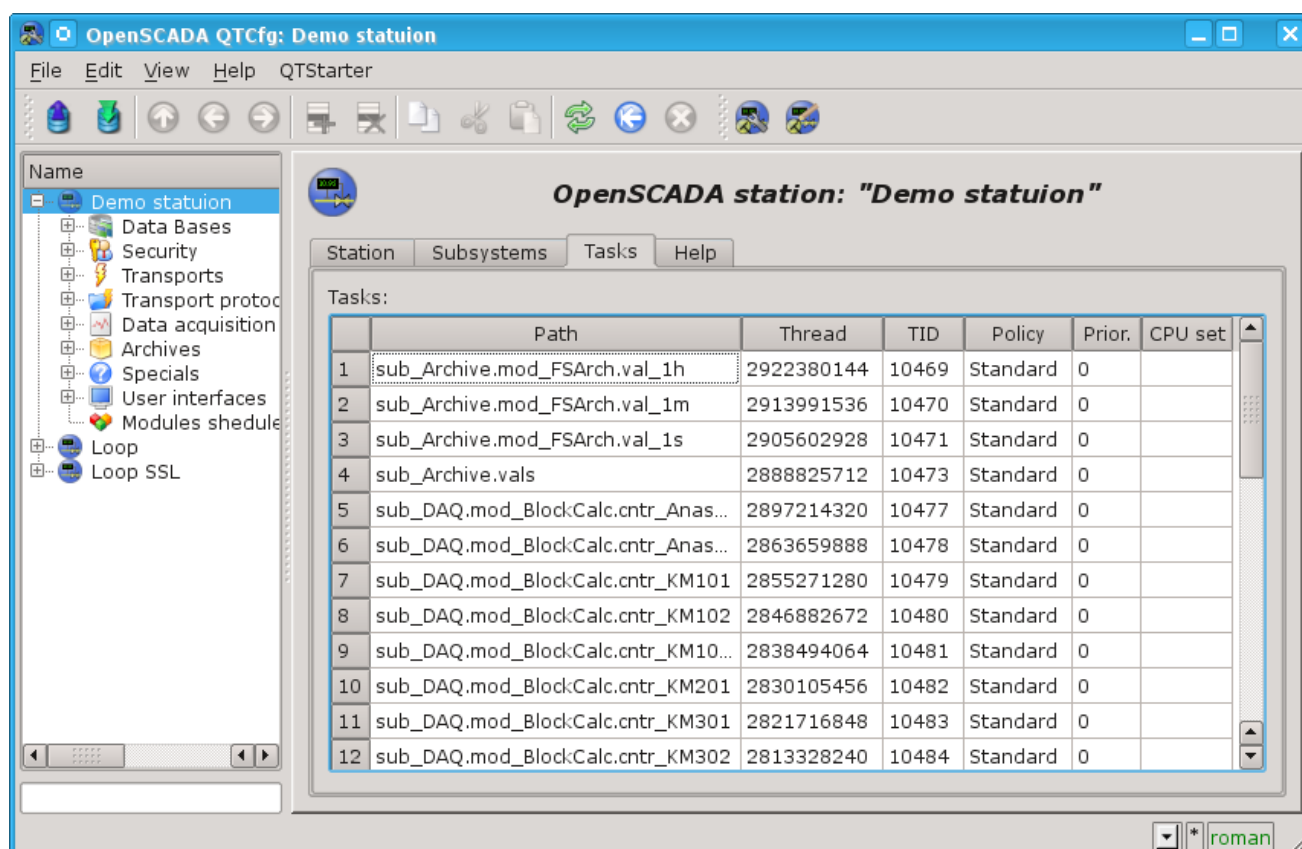


Fig 4c. "Tasks" tab of the main page of the configuration of the station.

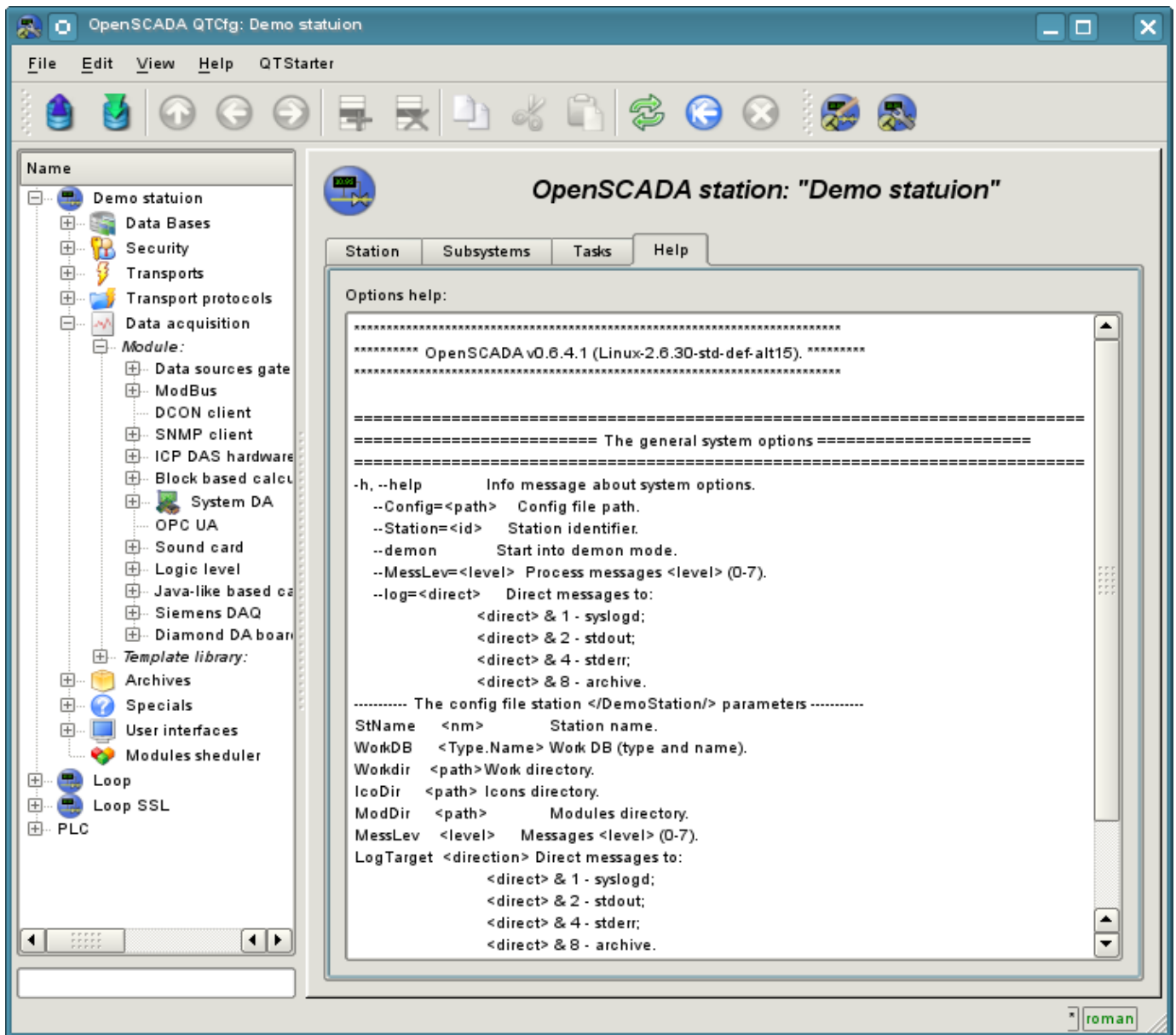


Fig. 4d. "Help" of the main page of the configuration of the station.

While examining the configuration pages of modular subsystems there will be described the general for all modules properties. However, it should be noted that each module can provide both: the additional tabs, and separate fields for the configuration of their own functioning for the pages, objects of which are inherited by modules. Information on the features and additions of modules can be found in separate documentation for each of them.

4.1. "DB" subsystem

The subsystem is the modular one and contains a hierarchy of objects depicted in Figure 4.1a. To configure the subsystem the root page of the subsystem "DB" containing the tabs "Modules" and "Help" is provided. Tab "Modules" (Fig. 4.1b) contains the list of modules in subsystem "DB", available at the station. Tab "Help" tab contains a brief help for this page.

To modify the page's fields of this subsystem it may be required the super user's rights or the inclusion of your user to the "DB" group.

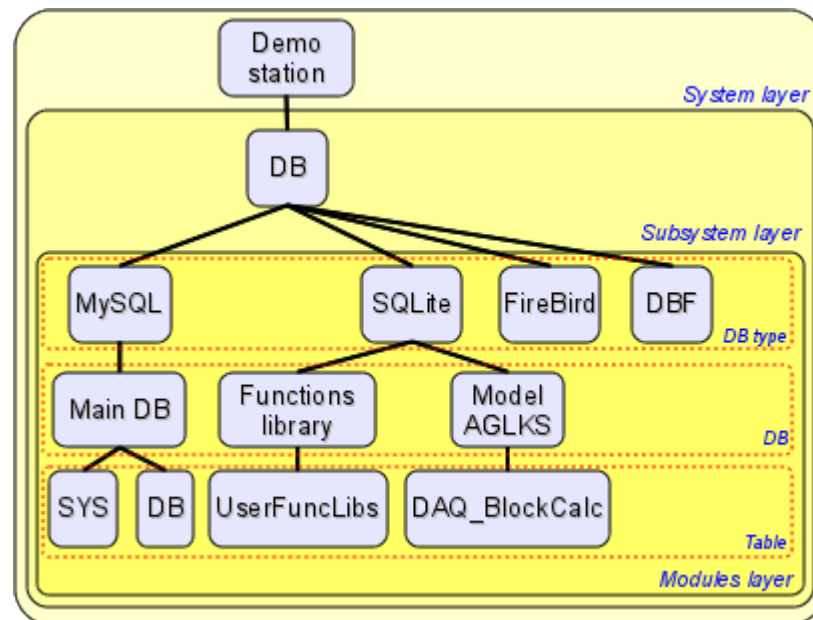


Fig. 4.1a. The hierarchical structure of "DB" subsystem.

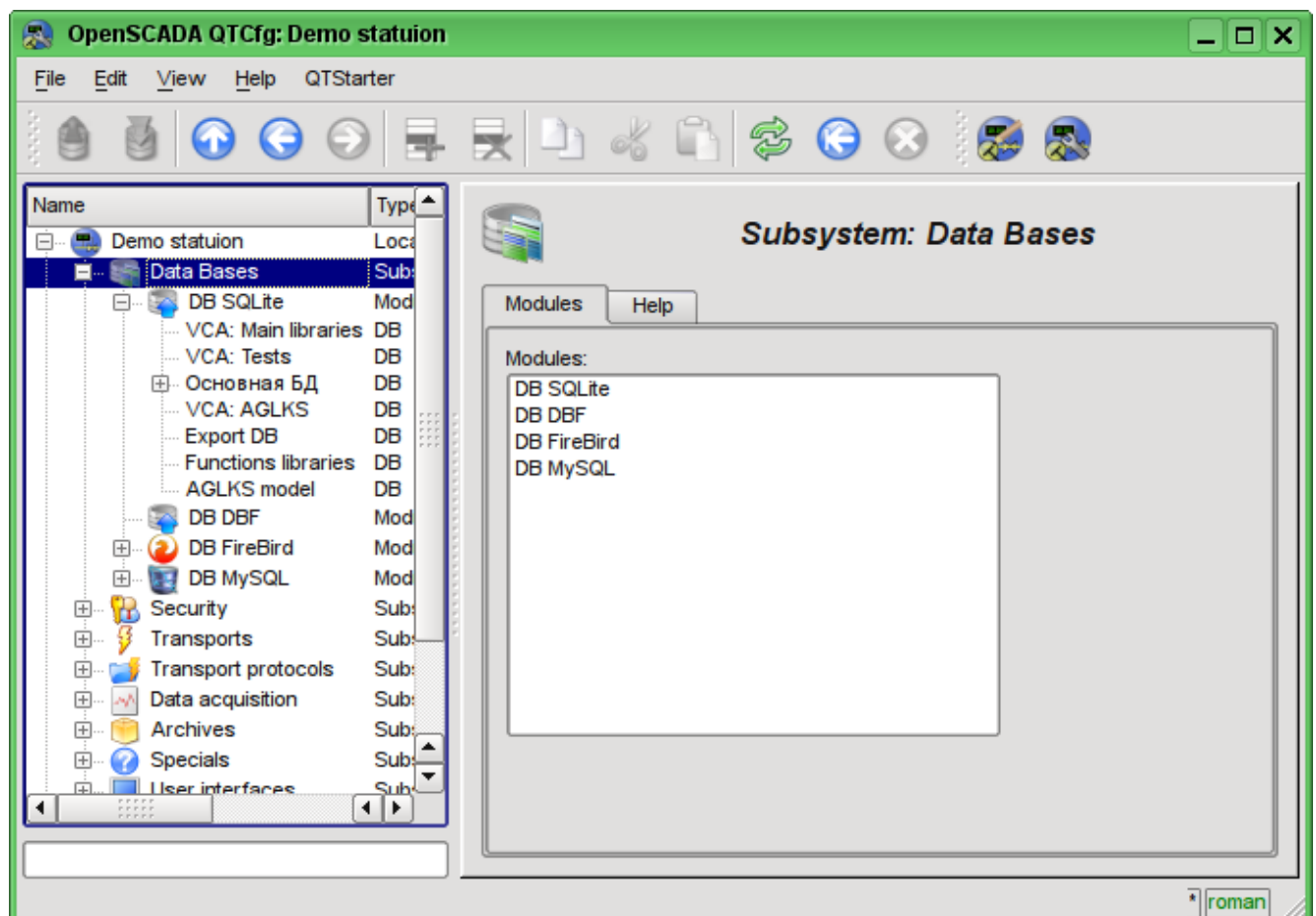


Fig. 4.1b. Tab "Modules" tab of the root page of "DB" subsystem.

Each module of the "DB" subsystem provides the configuration page with the following tabs: "DB" and "Help". "DB" tab (Fig. 4.1c) contains the list of databases registered in the module and the flag of the sign of full deleting of the database when making the delete command. In the context menu of the databases' list the user is provided with an opportunity to add, delete and move to the desired database. The "Help" tab contains information about the module of the "DB" subsystem (Fig.4.1d):

- *Module* - module's identifier.
- *Name* - module's name.
- *Type* - module's type, subsystem's identifier, which contains the module.
- *Source* - shared library - the source of the module.
- *Version* - module's version.
- *Author* - module's author.
- *Description* - module's short description.
- *License* - license agreement of module's distribution.

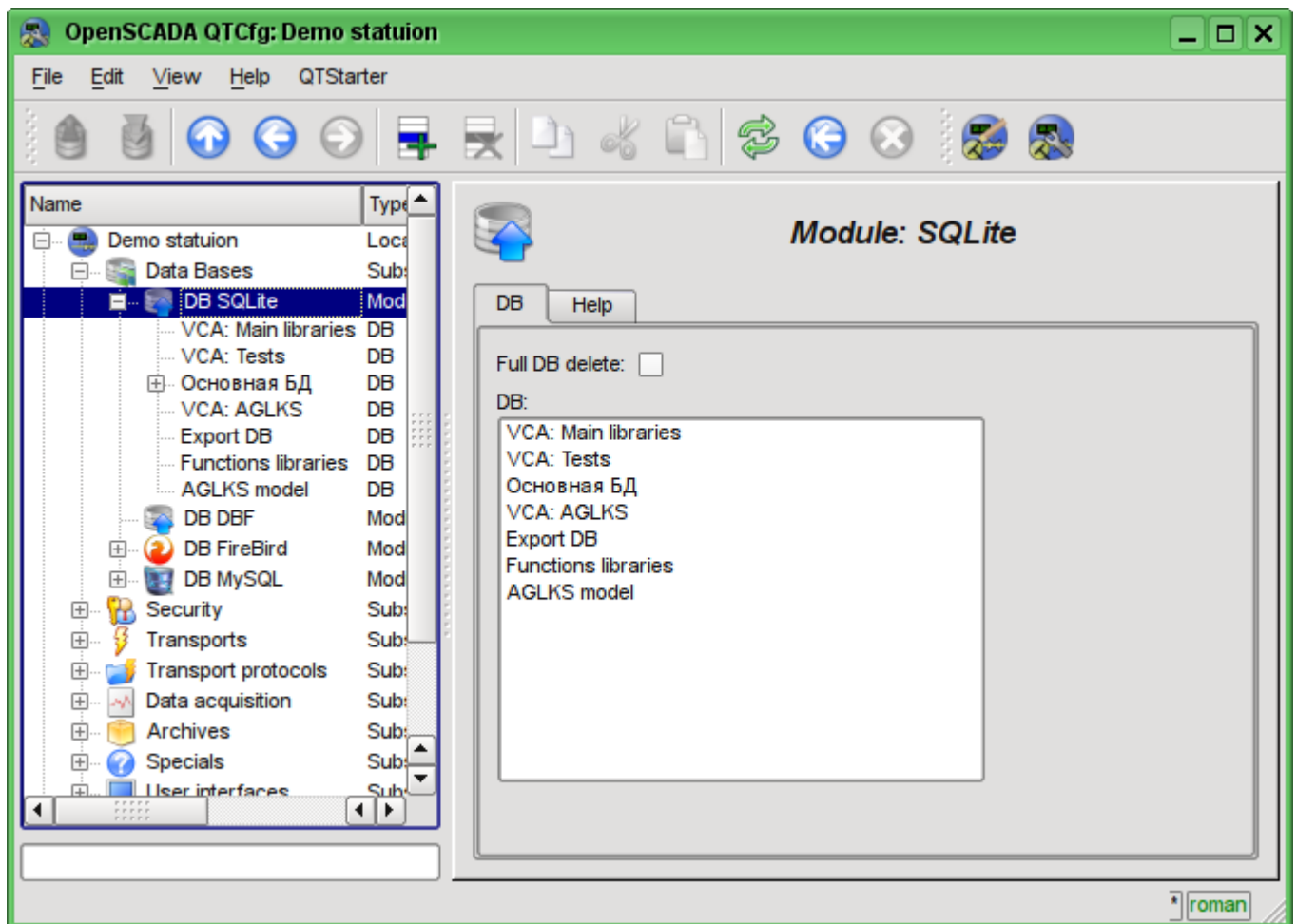


Fig. 4.1c. "DB" tab of the module of "DB" subsystem.

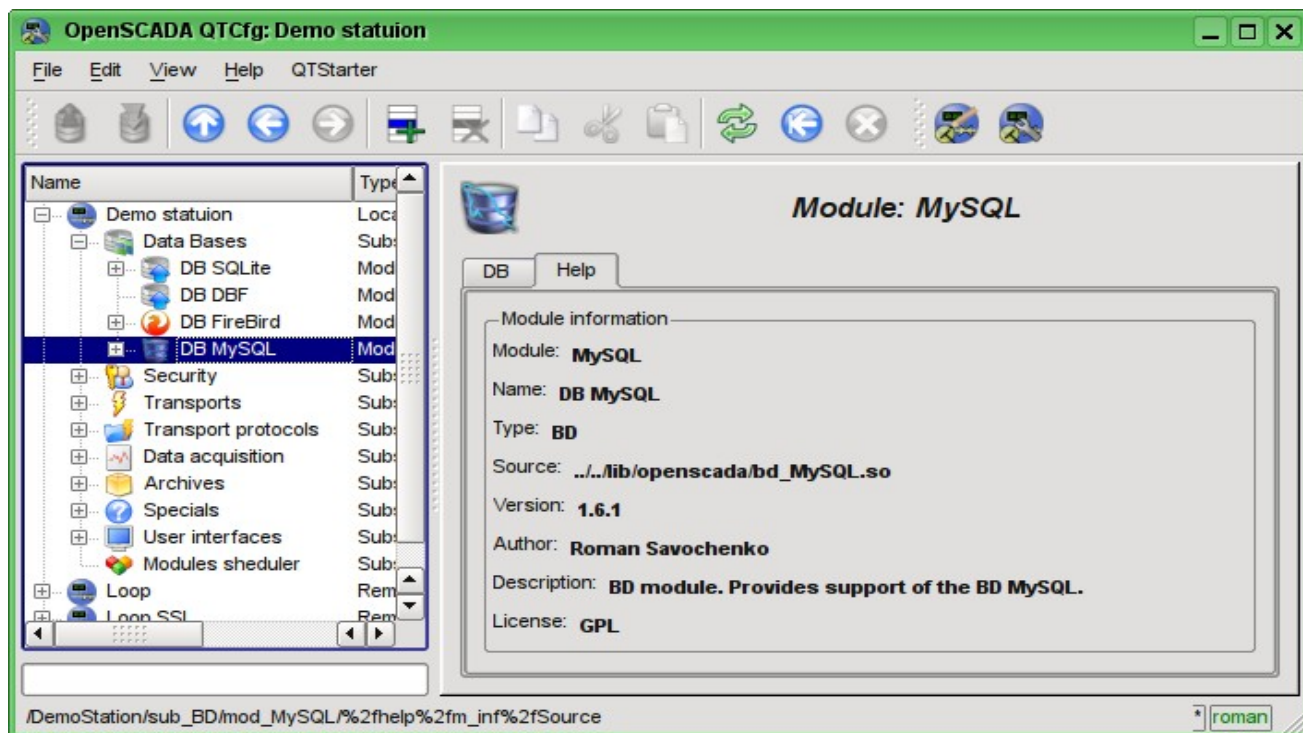


Fig. 4.1d. "Help" tab of the module of the "DB" subsystem.

Each database contains its own configuration page with the tabs "Data base" and "Tables". Besides the basic operations you can copy the contents of the DB by means of the standard function for the copying the objects in the configurator. The copying operation the DB contents involves the copying of the original database to the destination database, and the contents of the destination database is not cleared before the copy operation. Copying the contents of database is made only when the both databases are enabled, otherwise it will run a simple copy of the object of the database.

Tab "Data base" (Fig.4.1e) contains the main configuration options of the DB as follows:

- Section "State" - contains the properties which characterize the DB status:
 - *Enable* - DB status "Enable".
 - *Accessible tables* - list of tables that are in the database. Context menu of the property gives the opportunity to physically remove the tables from the database.
 - *Load system from this DB* - command to make load from this database. Can be used when transferring data in the database between stations. For example, you can save the section of one station in the export database, physically to move the DB to another station and connect it in this subsystem, and call this command.
- Section "Config" - contains the following configuration fields:
 - *ID* - contains the information on the DB identifier.
 - *Name* - specifies the DB name.
 - *Description* - short description of the DB and it's appointment.
 - *Address* - DB address in the specific for the database type (module) in the format. Format Description of the DB address recording format is usually available in the tooltip for this field.
 - *Code page* - indicates the code page, in which the text values of database are stored and provided. The value of the code page of database in conjunction with the internal code page of the station is used for clear transcoding of the text message while exchange between the station and the database.
 - *To enable* - indicates the state "Enable", in which to set the DB when start.

Tab "Tables" (Fig.4.1f) contains the list of the opened pages. In normal mode of the program operation this tab is empty, because after the completion of working with tables the program closes them. The presence of opened tables tells that the program is now working with tables or tables are opened by the user to examine their contents. In the context menu of list of opened tables you can open the table for study (the command "Add"), close the opened page (the command "Delete") and proceed to examination of the contents of the table.

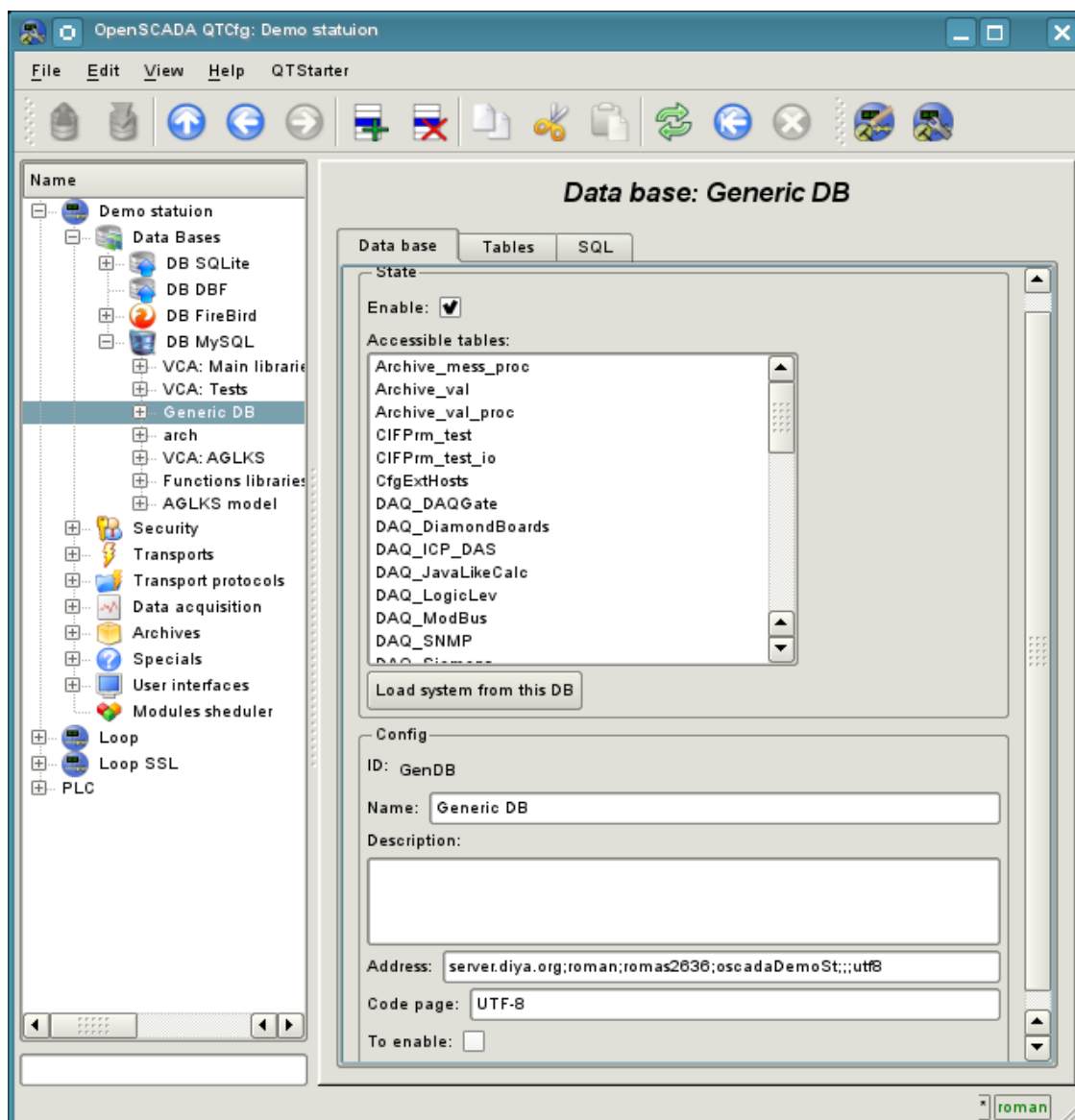


Fig. 4.1e. Tab "Data base" of the DB of module of subsystem "DB".

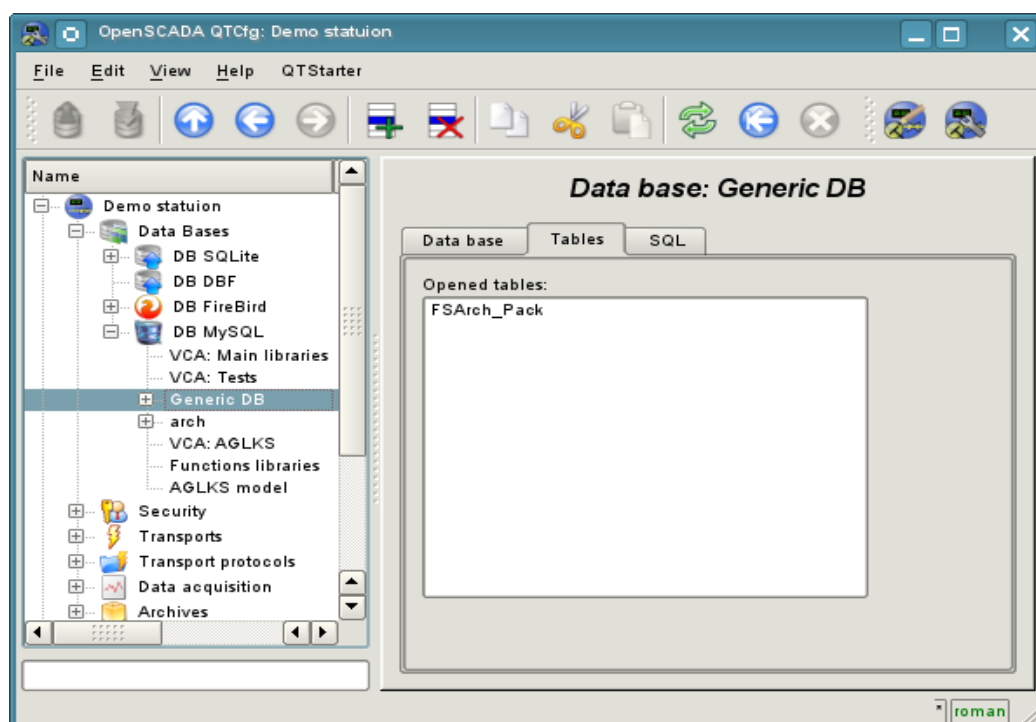


Fig. 4.1f. Tab "Tables" of the DB of module of subsystem "DB".

Page of the examination of the contents of the table contains only one tab, "Table". Tab "Table" (Figure 4.1g) contains the field of the name of the table and the table with the contents. Table of contents provides the following functions:

- table's cells content redaction;
- addition of the line;
- deleting of the line.

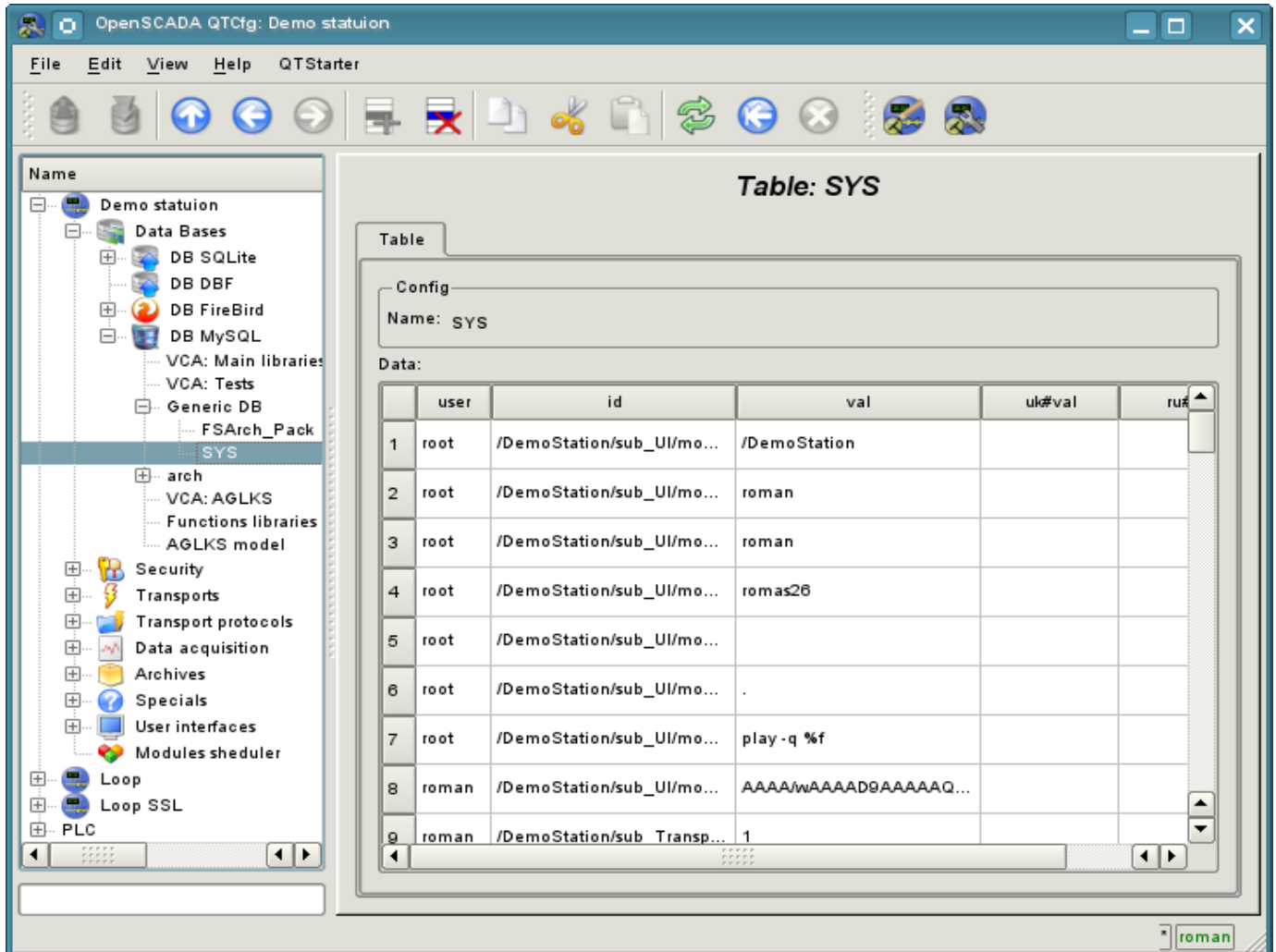


Fig. 4.1g. Tab "Table" of the DB table of the module of the subsystem "DB".

4.2. Subsystem "Security"

The subsystem is not modular one. To configure the subsystem the root page of the subsystem "Security" is provided, which contains the tab "Users and Groups" and "Help". Tab "Users and Groups" (Figure 4.2a) contains the list of users and users' groups. Users in the group "Security" and with the rights of the privileged user can add, delete the user or group of users. All other users can go to the page the user or the users' group. Tab "Help" contains the brief help for this page.

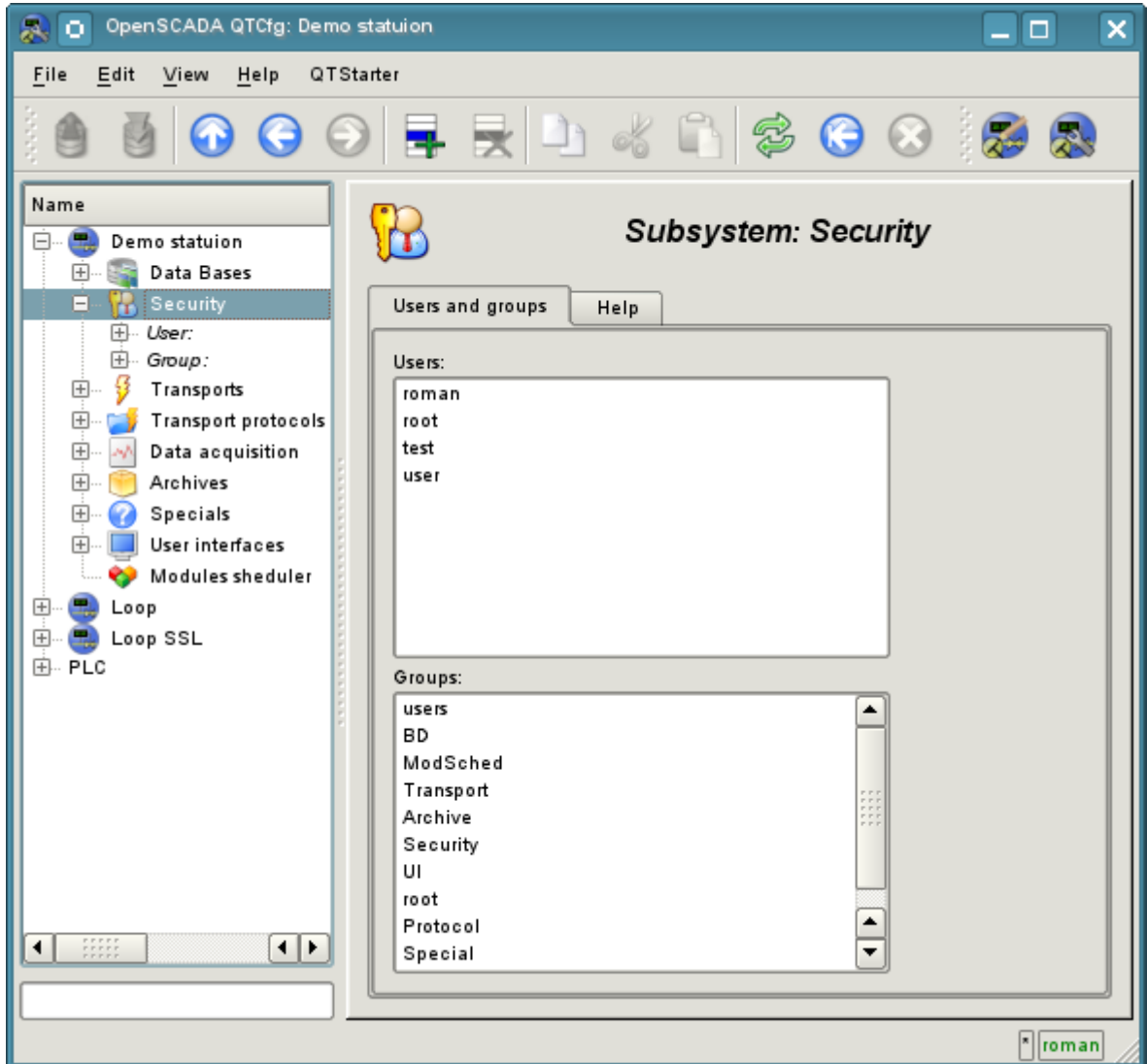


Fig. 4.2a. Tab "Users and Groups" of the root page of the subsystem "Security".

To configure the user it is provided the page containing only the tab "User" (Fig.4.2b). Tab contains the configuration data of the user's profile, which can be changed by the user itself, the user of the "Security" group or the privileged user:

- *Name* - information about the name (identifier) of the user.
- *Full name* - specifies the full name of the user.
- *User picture* - specifies the user's picture. Picture can be loaded and saved.
- *User DB* - DB address for the user's data storage.
- *Password* - the field to change the user's password. It always displays "*****".
- *Groups* - the table with a list of user groups of the station and with the sign of identity of the user to the groups.

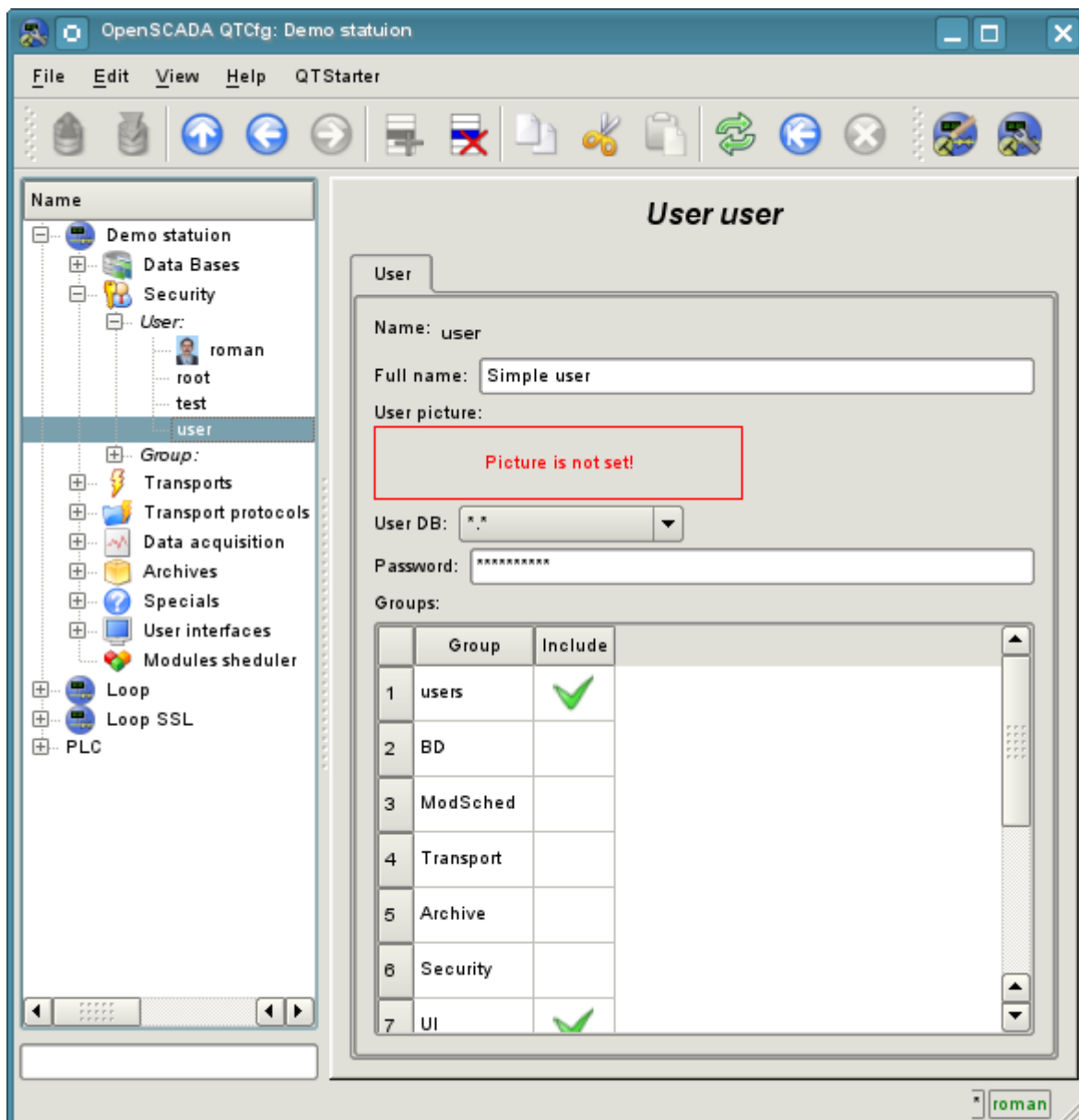


Fig. 4.2b. The tab "User" of the user's page of "Security" subsystem.

To configure the user's group it is provided the page containing only the tab "Group" (Fig.4.2c). Tab contains the configuration data of the group's profile, which can be changed only by the privileged use:

- *Name* - information about the name (identifier) of the user's group.
- *Full name* - specifies the full name of the user's group.
- *User group DB* - DB address for the user group's data storage.
- *Users* - list of users included in this group. With the context menu of the list you can add or remove the user in the group.

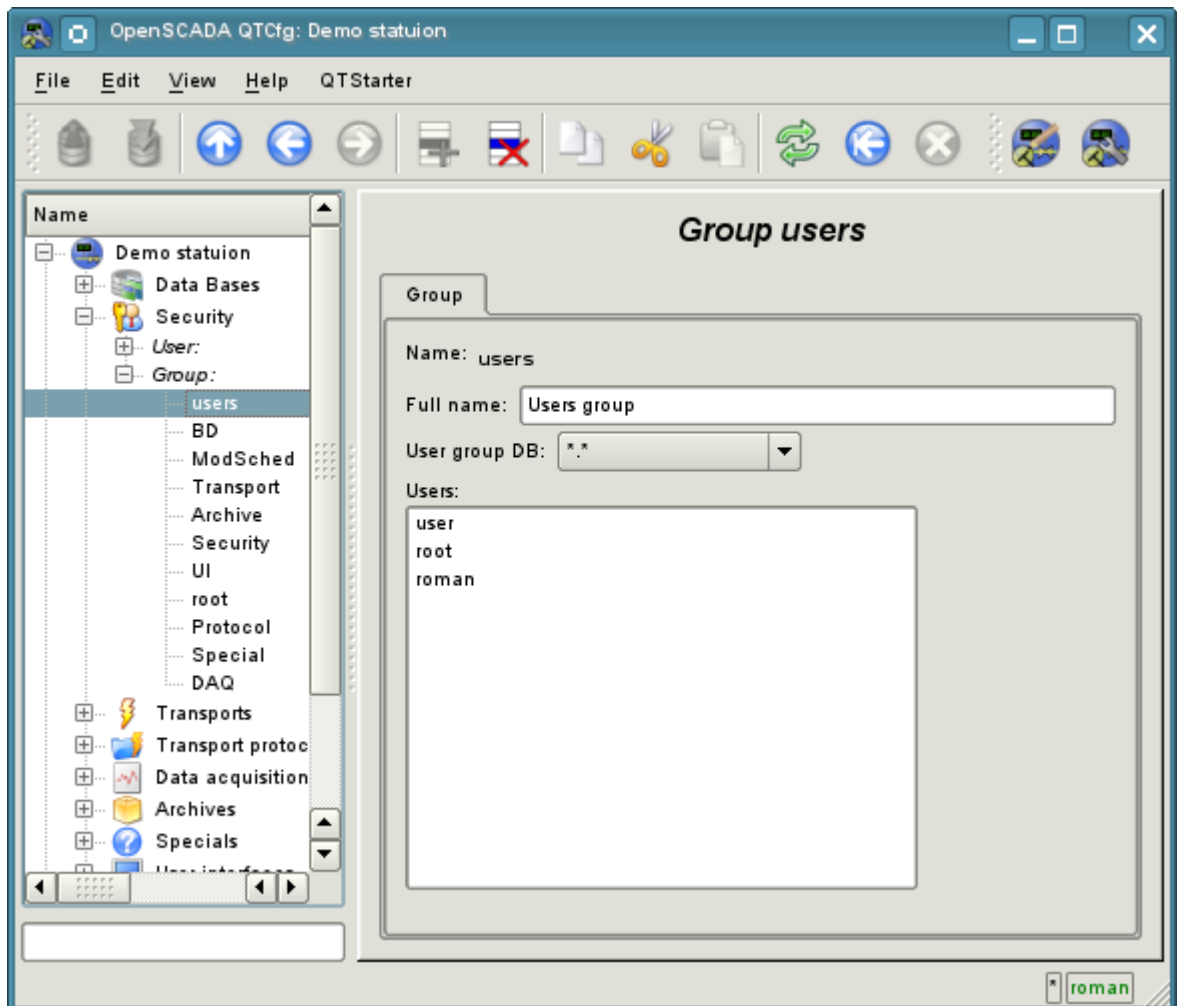


Fig. 4.2c. The tab "Group" of the user's group page of "Security" subsystem.

4.3. Subsystem "Transports"

The subsystem is the modular one and contains the hierarchy of objects shown in Figure 4.3a. To configure the subsystem it is provided the root page of the subsystem "Transports", containing the tabs "Subsystem", "Modules" and "Help".

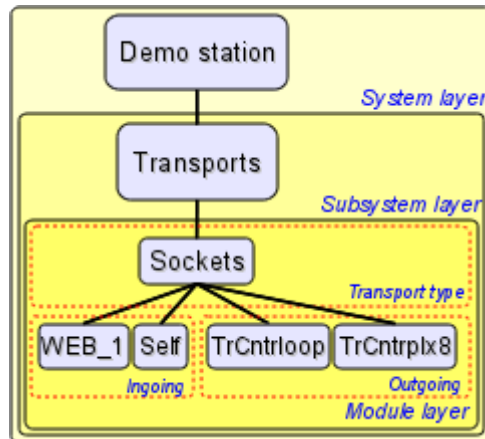


Fig. 4.3a. The hierarchical structure of subsystems "Transports".

The tab "Subsystem" (Figure 4.3b) contains the configuration table of the external stations for a given OpenSCADA. External stations can be the system's and the user's ones that is selected by the appropriate option. System's external stations are available only to the super user and are used by the components of the system purpose, for example, the mechanism of the horizontal redundancy and module [DAQ.DAQGate](#). User's external stations are tied to the user who created them, and thus the list of user's external stations is individual for each user. User's external stations are used by the components of graphical interface, for example, [UI.QTCfg](#), [UI.WebCfgD](#) and [UI.Vision](#). In the table of the external stations it is possible to add and delete records about the station, as well as their modification. Each station contains the following fields:

- *Id* - identifier of the external station.
- *Name* - the name of the external host.
- *Transport* - the combobox of the subsystem's module "Transports" for the using of it when access to the external station.
- *Address* - address of the external station if the format, specific to the chosen in the previous field of the module of the subsystem "Transports".
- *User* - the name/identifier of the user of the external station on behalf of whom to perform the connection.
- *Password* - password of the user of the external station.

Tab "Modules" tab (fig. 4.1b) contains the list of modules in subsystem "Transports" and is identical for all modular subsystems. Tab "Help" contains a brief help for this page.

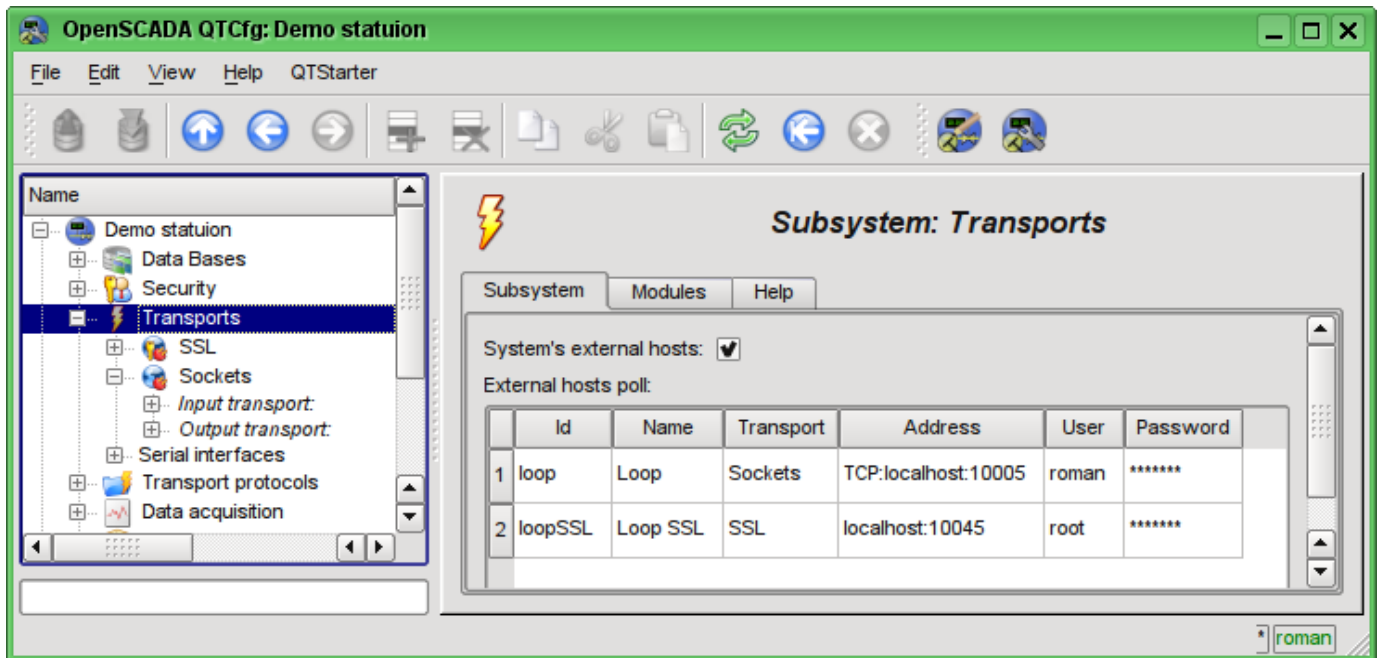


Fig. 4.3b. Tab "Subsystem" of the root page of subsystem "Transports".

Each module of the subsystem "Transports" provides the configuration page with the tabs "Transports" and "Help". The tab "Transports" (Fig.4.3c) contains the list of incoming and outgoing transports registered in the module. The context menu of lists of transports provides the user with the possibility to add, delete and move to the desired transport. On the "Help" tab it is provided the information about the module of subsystem "Transports" (Fig. 4.1d), whose structure is identical for all modules.

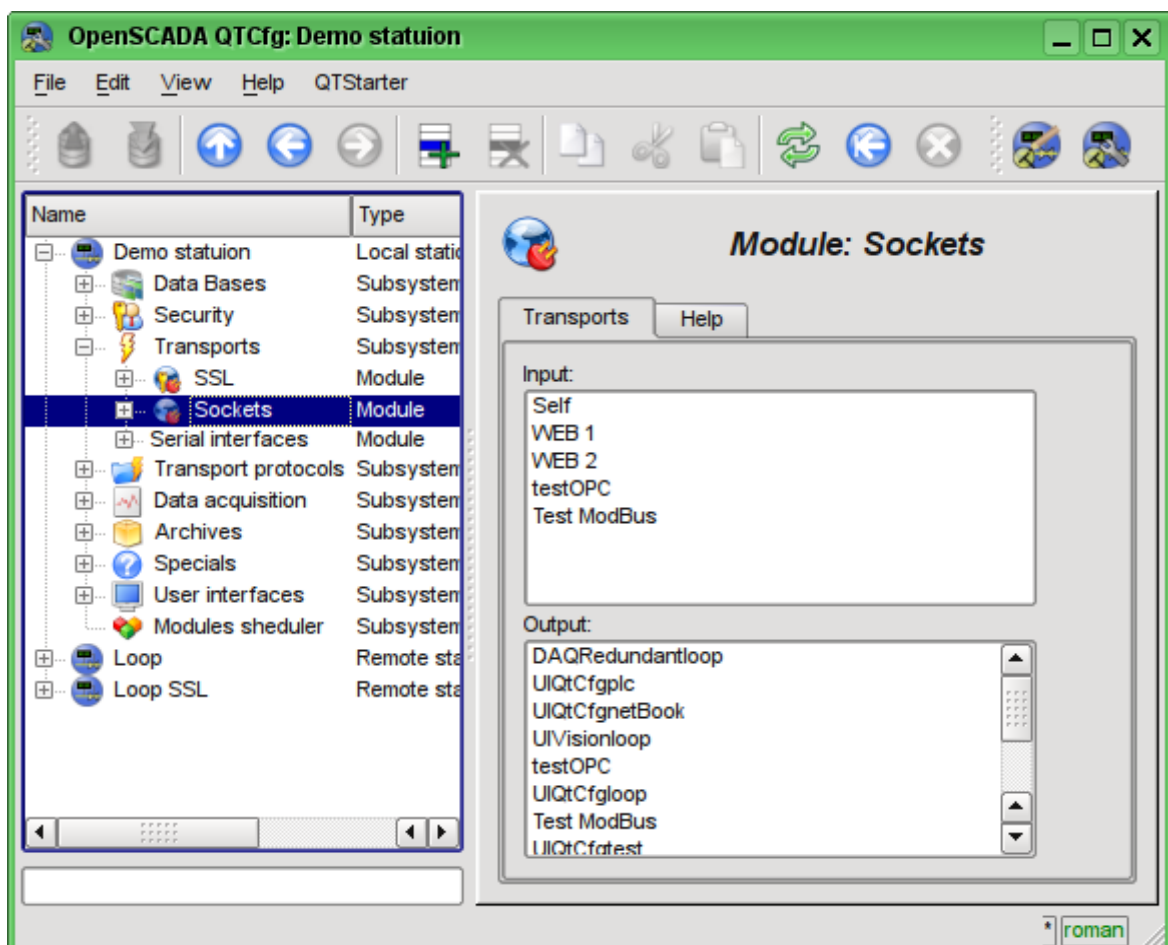


Fig. 4.3c. The tab "Transports" of the module of subsystem "Transports".

Each transport contains its own configuration page with one tab "Transport". This tab contains the basic settings of transport. Incoming transport (fig.4.3d) includes:

- Section "State" - contains the settings that characterize the state of the transport:
 - *Status* - information on the current transport's status and statistics of its work.
 - *Running* - state of the transport "Running".
 - *Transport DB* - DB address to store the transport's data.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the transport's identifier.
 - *Name* - specifies the transport's name.
 - *Description* - brief description of the transport and its appointment.
 - *Address* - transport's address in the specific for the type of transport (module) format. Description of the record format addresses transport, as a rule, is available in the tooltip for this field.
 - *Transport protocol* - indicates the transport protocol module (subsystem "Transport protocols") that should work in conjunction with the input transport. I.e. the received unstructured data this module will sent to the structuring and processing to the specified module of the transport protocol.
 - *To start* - indicates the status of "Running", in which to transfer the transport at startup.

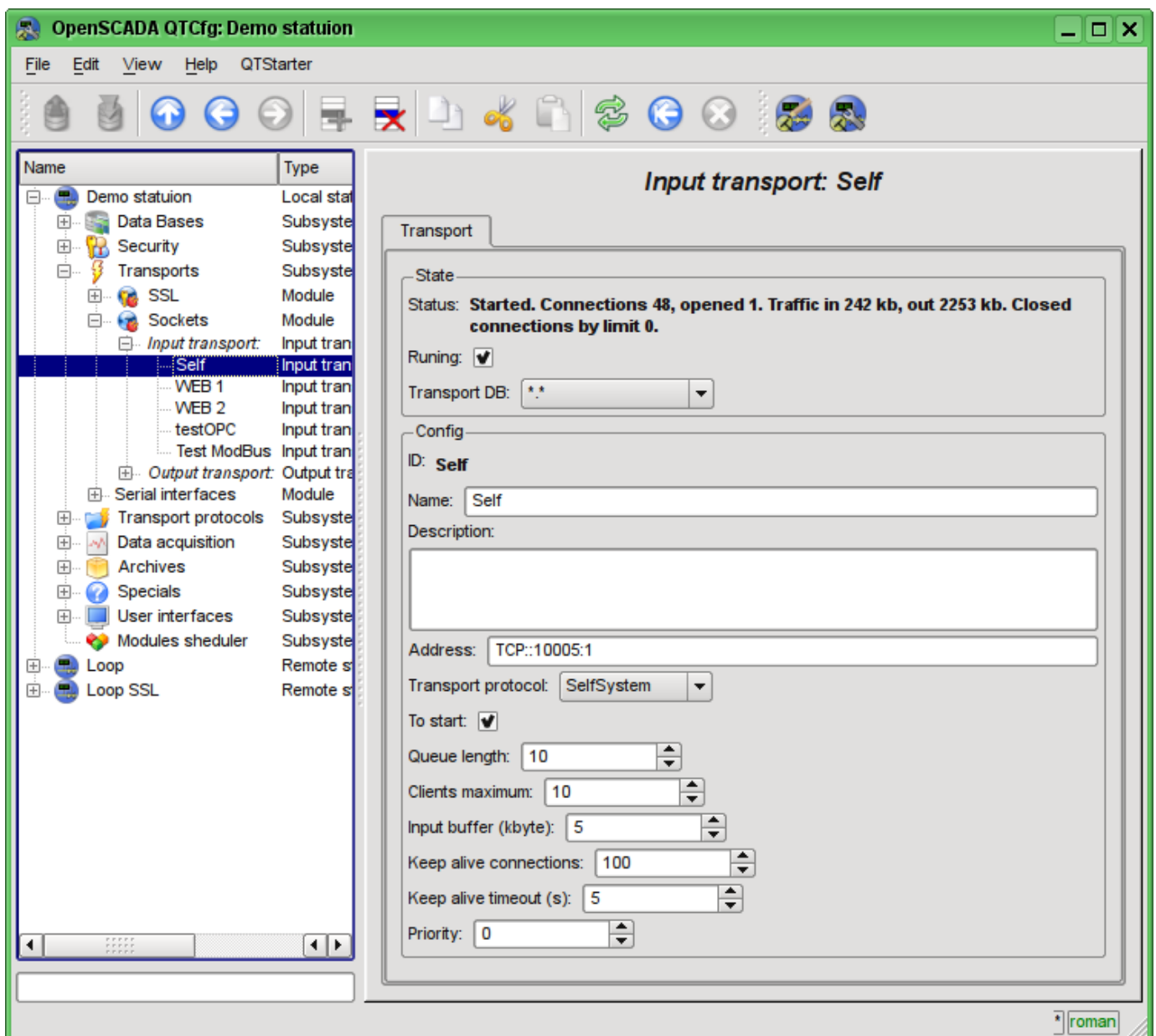


Fig. 4.3d. Tab "Transport" of the page of incoming transport of module of subsystem "Transports".

Outgoing transport (Fig. 4.3e) contains:

- Section "State" - contains the settings that characterize the state of the transport:
 - *Status* - information on the current transport's status and statistics of its work.
 - *Running* - state of the transport "Running".
 - *Transport DB* - DB address to store the transport's data.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the transport's identifier.
 - *Name* - specifies the transport's name.
 - *Description* - brief description of the transport and its appointment.
 - *Address* - transport's address in the specific for the type of transport (module) format. Description of the record format addresses transport, as a rule, is available in the tooltip for this field.
 - *To start* - indicates the status of "Running", in which to transfer the transport at startup.

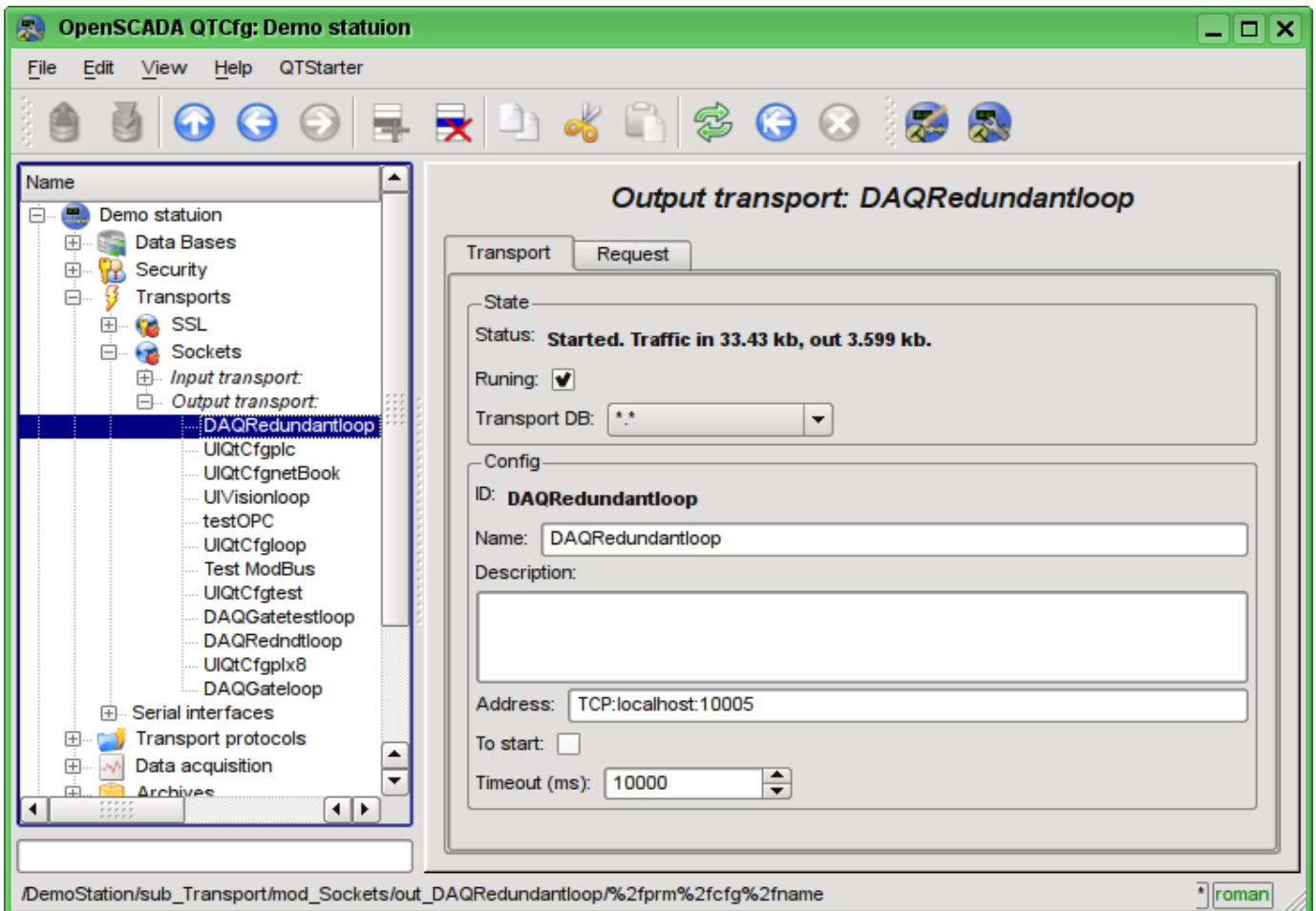


Fig. 4.3e. Tab "Transport" of the page of outgoing transport of module of subsystem "Transports".

Outgoing transport, in addition, provides the tab for forming the user request via this transport (Fig.4.3f). The tab is provided for setting communication, as well as for debugging the protocols and includes:

- *Time (ms)* - information about the time taken for request and receiving the answer.
- *Mode* - indicates the regime of data from the following list: "Text" and "Binary", in which the request will be formed and the answer will be provided. In binary mode data is recorded in pairs of numbers in hex, ie bytes, separated by spaces.
- *Send* - command to send a request.
- *Request* - contains the request in the selected mode of data representing.
- *Answer* - provides the answer in the selected mode of data representing.

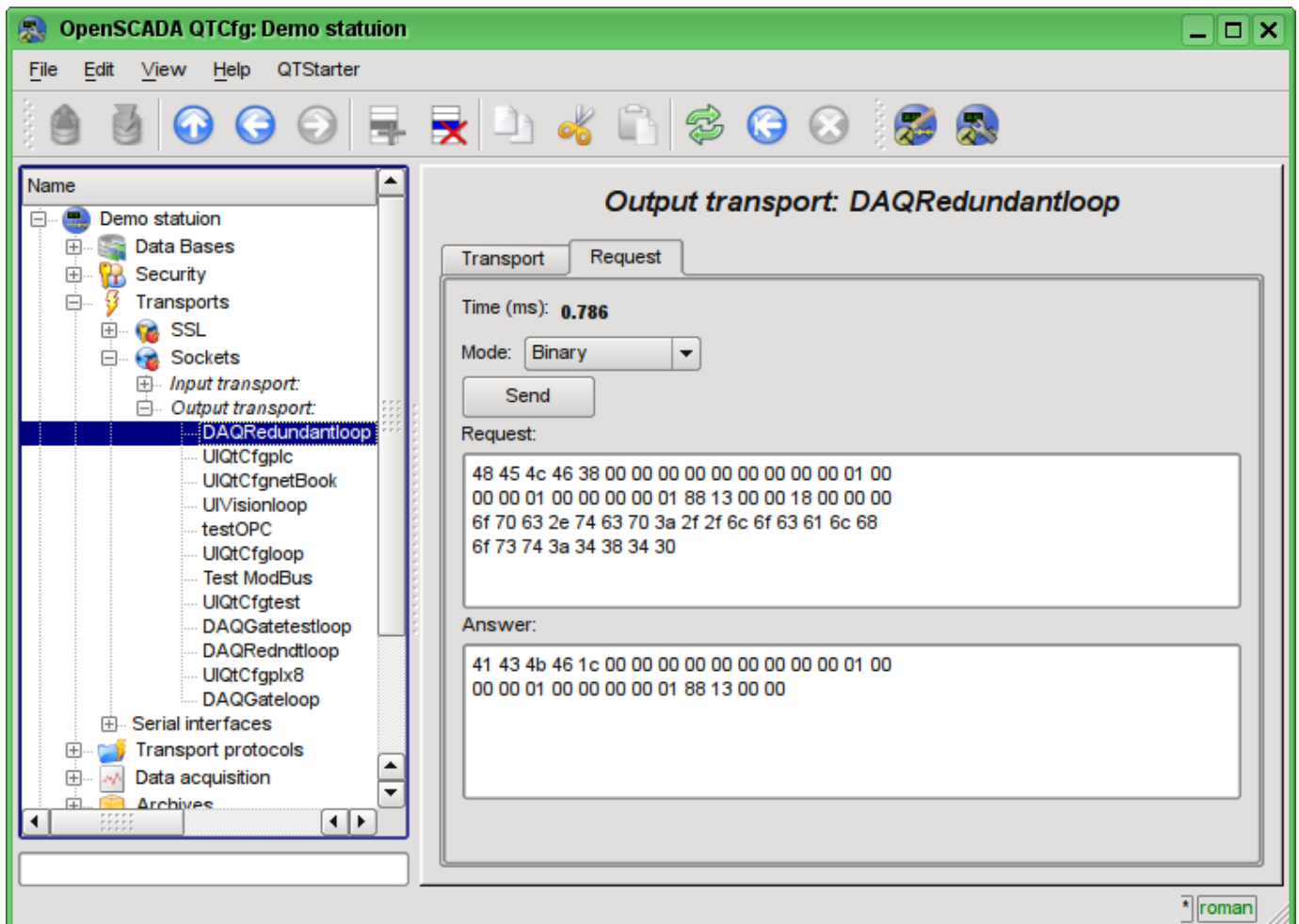


Fig. 4.3f. The tab "Request" of the page of outgoing transport of module of subsystem "Transports".

4.4. Subsystem "Transport protocols"

The subsystem is modular. To configure the subsystem the root page of the subsystem "Transport Protocols" is provided, it contains the following tabs: "Modules" and "Help". The tab "Modules" (Fig. 4.1b) contains the list of modules in subsystem "Transport Protocols" and is identical for all modular subsystems. The tab "Help" contains a brief help for this page.

Each module of subsystem "Transport Protocols" provides configuration page with the only one tab - "Help". On the tab "Help" there is the information on the module of subsystem "Transport Protocols" (Fig. 4.1d), which structure is identical for all modules.

4.5. Subsystem "Data acquisition"

The subsystem is modular and contains the hierarchy of objects depicted in Fig.4.5a. To configure the subsystem the root page of subsystem "Data acquisition" is provided, which contains the tabs "Template libraries", "Modules" and "Help".

To obtain access to modify the objects of this subsystem the user of the group "DAQ" or the rights of the privileged user are required.

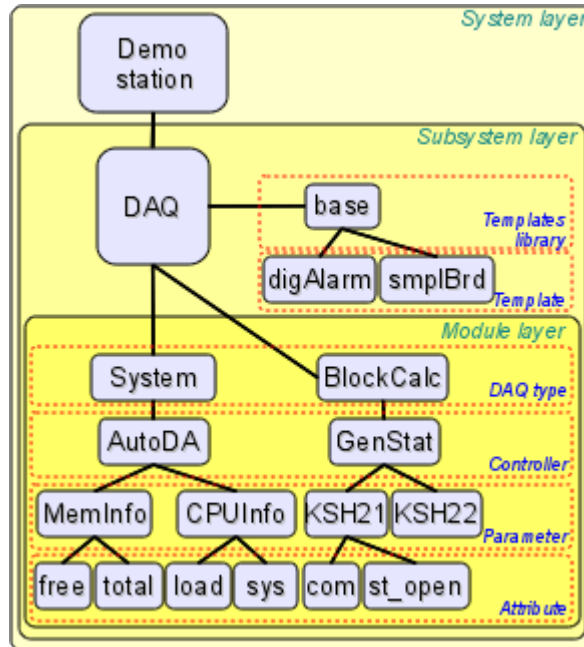


Fig. 4.5a. The hierarchical structure of subsystem "Data acquisition".

Tab "Redundancy" (Fig. 4.5b) contains the configuration of redundancy of data sources of subsystem "Data acquisition" of the station with the following settings:

- *Status* - contains information on redundancy scheme, this is usually the time spent on the execution of one cycle of the task of reserve processing.
- *Station level* - indicates the level of the station in an arrangement (0-255).
- *Redundant task period (s)* - indicates the frequency of execution of redundancy task in seconds (1-255).
- *Restore connection timeout (s)* - indicates over the which period of time to attempt to reconnect with the lost redundant station in seconds (0-255).
- *Restore data depth time (hours)* - indicates the maximum depth of archival data to restore from the archive of the remote station when start up in hours (0-12).
- *Stations* - contains the table with information about the redundant stations. Stations can be added and removed via contextual menu. Id of the added stations is to be chosen from the list of available OpenSCADA system stations. The table provides the following information about the station:
 - *ID* - ID of the system OpenSCADA station, should be changed after the addition by choosing from the list of available ones;
 - *Name* - name of the system OpenSCADA station;
 - *Live* - sign of the connection with the redundant station;
 - *Level* - level of the remote station in the redundancy scheme;
 - *Counter* - requests' counter to the redundant station or waiting time in the case of the absence of connection;
 - *Run* - the list of available controllers with the sign (+) of the local execution on the remote station.
- *Go to remote stations list configuration* - command to go to the configuration page of the remote OpenSCADA stations in the subsystem "Transports".
- *Controllers* - contains the table with the list of controllers, available for redundancy, and their current status:

- *Controller* - full controller's ID;
- *Name* - controller's name;
- *Started* - the sign of the controller's execution on the local station;
- *Redundant* - redundancy mode of the controller can be changed from the list of: "Off", "Asymmetric" and "Symmetric";
- *Preferable run* - configuration of the preferred execution at the specified station can be changed; reserved values: <High Level> - execution at the station with the highest level, <Low Level> - execution at the station with the lowest level, <Optimal> - the choice for the execution of the least loaded station.
- *Remoted* - sign indicating the execution of the controller on the remote station and the transfer of the local one to the mode of data synchronization from the remote station.

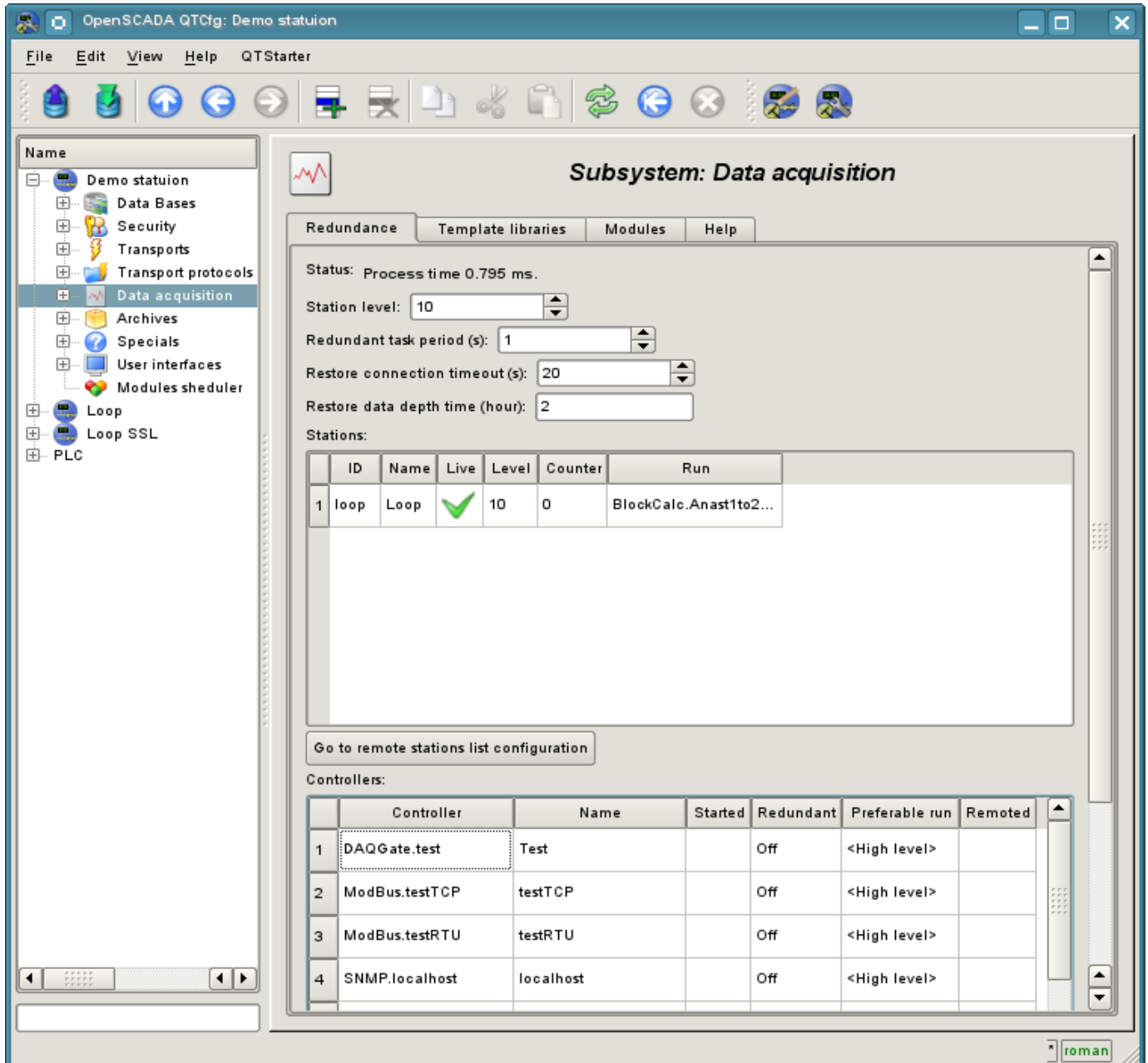


Fig. 4.5b. Tab "Redundancy" tab of subsystem "Data acquisition".

The tab "Template libraries" (Fig.4.5c) contains the list of libraries of templates for the parameters of this subsystem. In the context menu of the list of template libraries the user can add, delete and move to the desired library. The tab "Modules" (Fig. 4.1b) contains the list of modules in the subsystem "Transports" and is identical for all modular subsystems. The tab "Help" contains the brief help for this page.

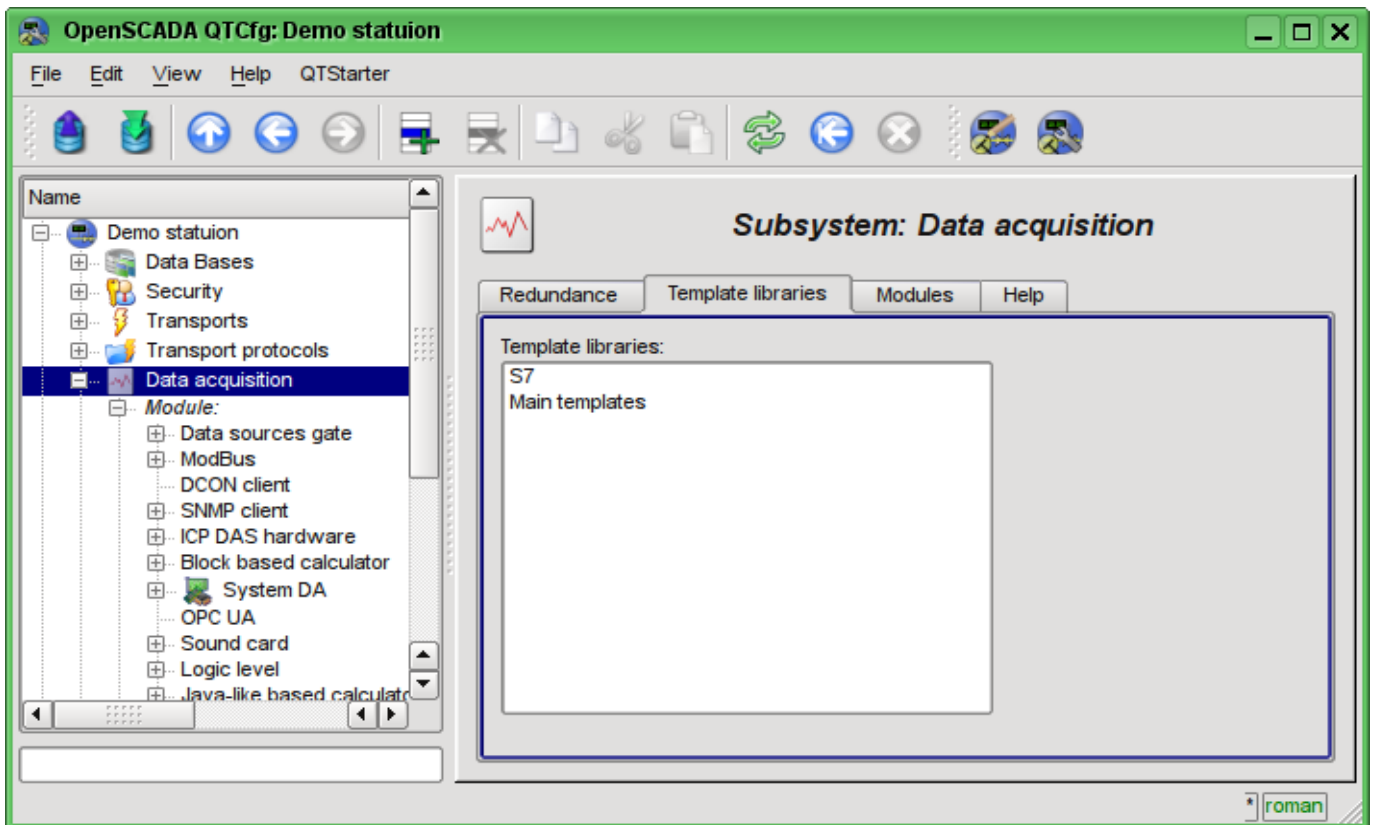


Fig. 4.5c. The tab "Template libraries" of the subsystem "Data acquisition".

Each template library of subsystem "Data acquisition" provides the configuration page with the tabs "Library" and "Parameter templates". Tab "Library" (fig. 4.5d) contains the basic settings of the library:

- Section "State" - contains properties that characterize the state of the library:
 - *Accessing* - state of library "Accessing".
 - *Library DB* - address of the database for data storage of the library and templates.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the ID of the library.
 - *Name* - specifies the name of the library.
 - *Description* - short description of the library and its purpose.

Tab "Parameter templates" (Fig.4.5e) contains the list of templates in the library. In the context menu of the list the user can add, delete and move to the desired template.

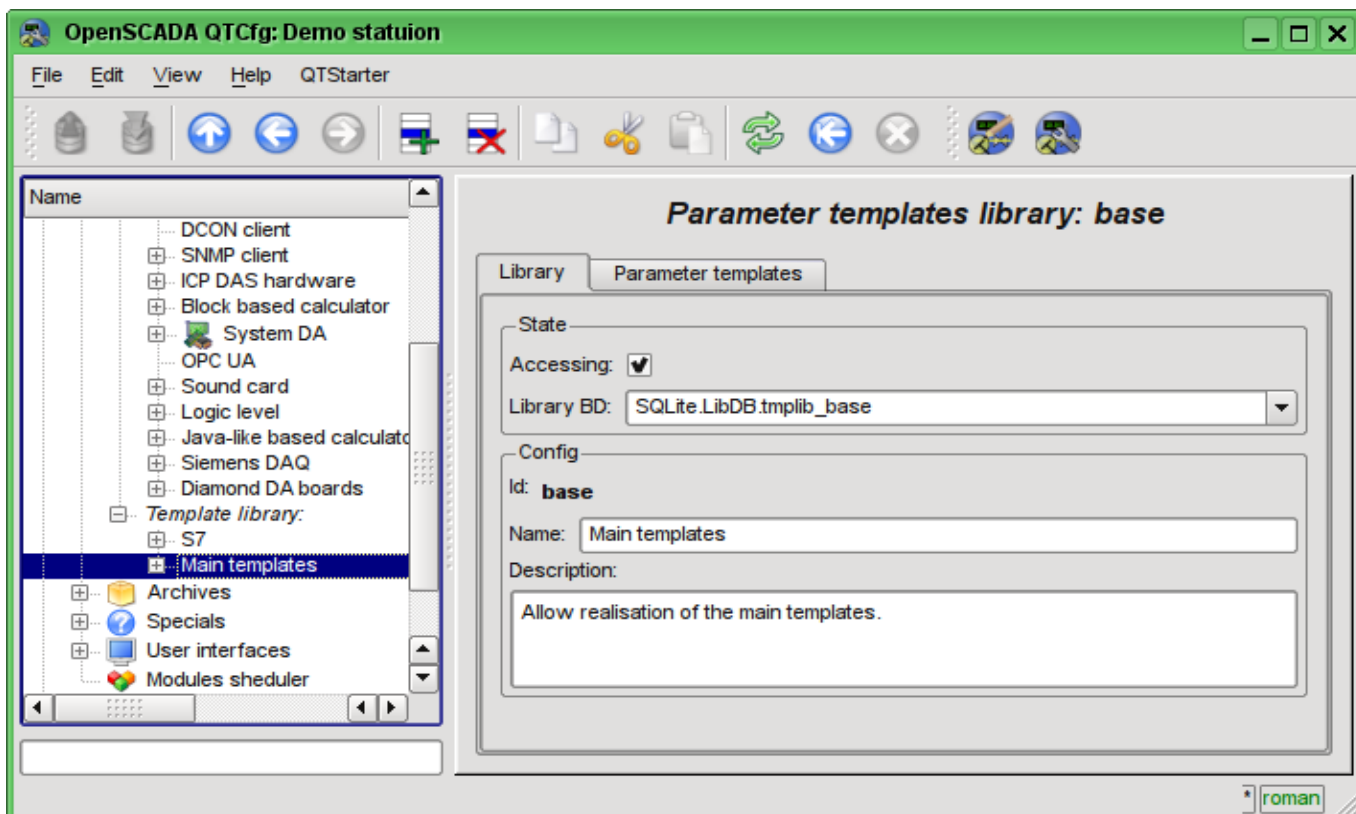


Fig. 4.5d. The main tab of configuration of template library of subsystem "Data acquisition".

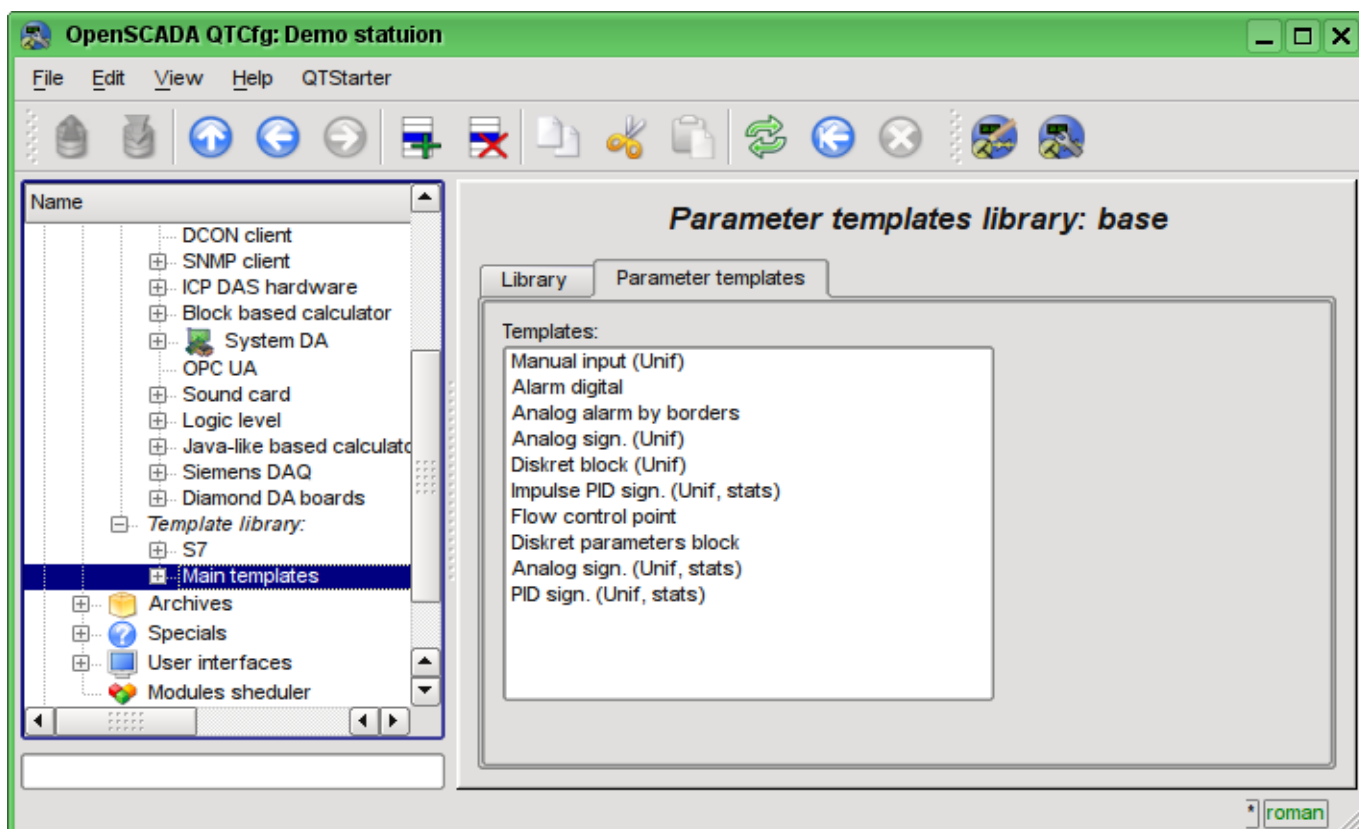


Fig. 4.5e. The tab of the list of templates in the template library of subsystem "Data acquisition".

Each template of the template library provides the configuration page with the tabs "Template" and "IO". The tab "Template" (Figure 4.5f) contains the basic settings of the template:

- Section "State" - contains properties that characterize the state of the template:
 - *Accessing* - state of template "Accessing".
 - *Used* - counter of the template's using. Allows you to determine whether the template is used and, consequently, the ability to edit the template.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the ID of the template.
 - *Name* - specifies the name of the template.
 - *Description* - short description of the template and its purpose.

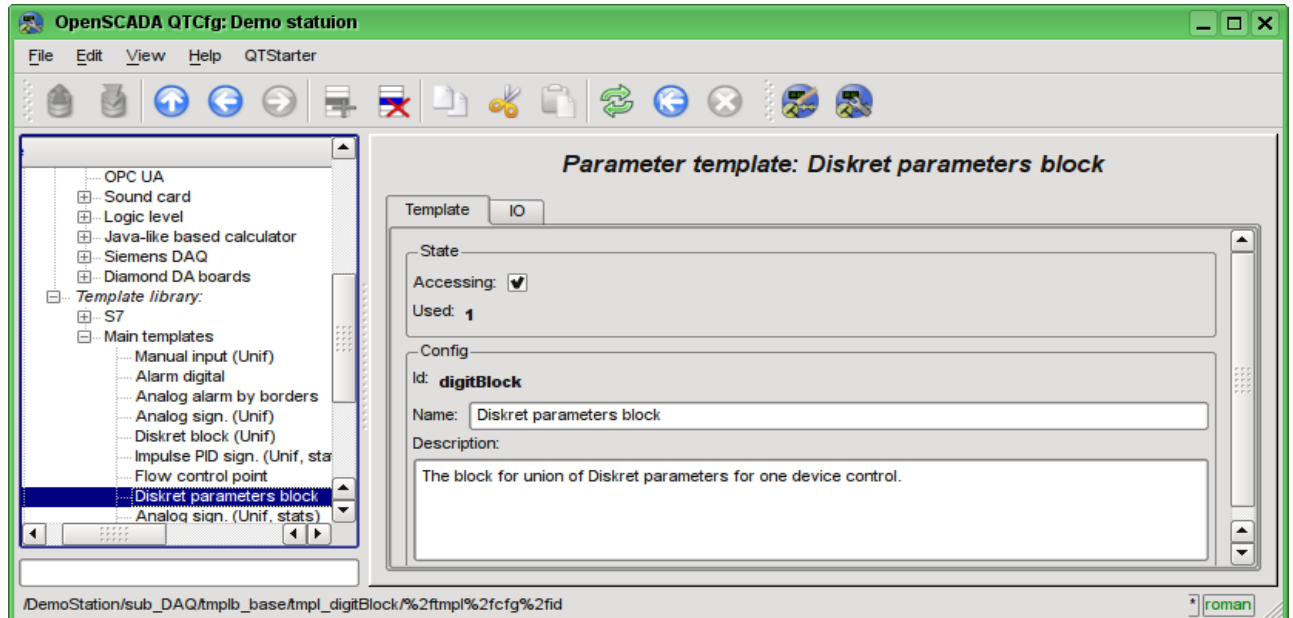


Fig. 4.5f. The main configuration tab of the parameters template of subsystem "Data acquisition".

The tab "IO" (Fig.4.5g) contains the configuration of attributes (IO) of templates and the program of template on the one of languages of the user programming of OpenSCADA, for example, DAQ.JavaLikeCalc.JavaScript. To the table of attributes of template user can, through the context menu, add, insert, delete, move up or down the record of attribute, as well as edit the attribute's fields:

- *Id* - ID of the attribute.
- *Name* - the name of the attribute.
- *Type* - select the value's type of the attribute from the following: "Real", "Integer", "Boolean", "String".
- *Mode* - select the mode of the attribute: "Input", "Output".
- *Attribute* - mode of the parameter's attribute, implemented based on a template from the list: "No attribute", "Read Only", "Full access". For the attributes of a template, in which this field is set, it will be created an appropriate attribute in the controller's parameter of this subsystem.
- *Configure* - configuration mode of the attribute in the configuration tab of a template of the controller's parameter of this subsystem from the list: "Constant", "Public constant", "Link". In "Public constant" and "Link" modes tab in the configuration tab of the template will be added these attributes to set the constant or specify an external link of the parameter.
- *Value* - attribute's default value or template of the links to access by the link. The format of the link's template depends on the component that uses it. Usually for the module [DAQ.LogicLev](#) the link's template is written the following way: **{Parameter}{attribute}**. Field **{Parameter}** - specifies the parameter's name as the attribute's container. Attributes with the equal value **{Parameter}** will be grouped and will be appointed only by the indication of attributes' container, and individual attributes will be associated with the attributes of the container in accordance with the field **{attribute}**.

The syntax of the language of the template's program you can see in the documentation of the module, providing an interpreter of the chosen language. For example, a typical user programming language of OpenSCADA - [DAQ.JavaLikeCalc](#)

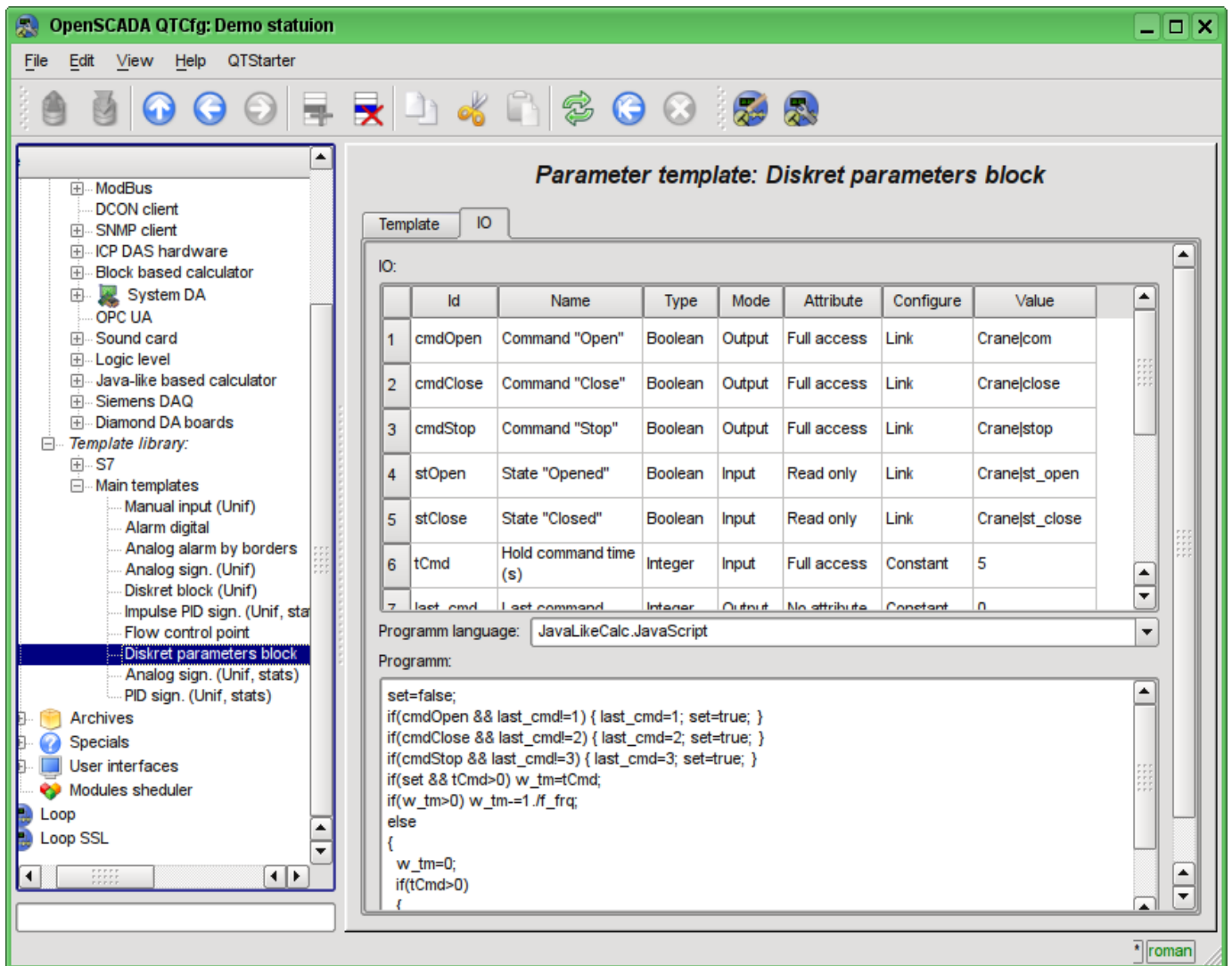


Fig. 4.5g. The configuration tab of the attributes and template's program of subsystem "Data acquisition".

Each module of the subsystem "Data acquisition" provides the configuration page with the tabs "Controllers" and "Help". The tab "Controllers" (Fig.4.5h) contains the list of controllers, registered in the module. In the context menu user can add, delete and move to the desired controller. The tab "Help" provides information about the module of the subsystem "Data acquisition" (Fig. 4.1d), which structure is identical for all modules.

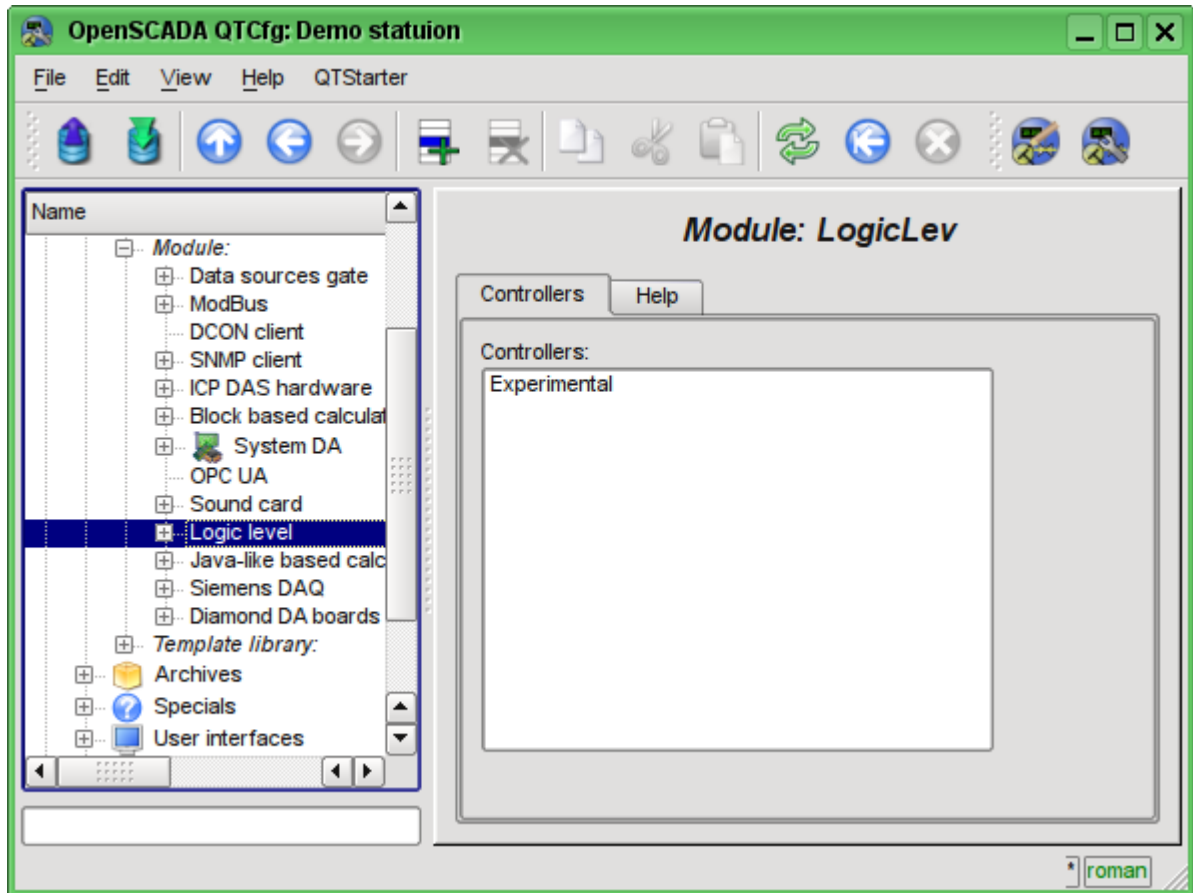


Fig. 4.5h. The tab "Controllers" of the module of the subsystem "Data acquisition".

Each controller contains its own configuration page with the tabs "Controller" and "Parameters".

The tab "Controller" (Fig.4.5i) contains the basic settings. The structure of these settings may differ slightly from one module of this subsystem to another, as you can find in the own documentation of modules. As an example, lets examine the settings of the controller in the module of the controller of logic [DAQ.LogicLev](#):

- Section "State" - contains the properties, which characterize the state of the controller:
 - *Status* - specifies the controller's status. In our case, the controller is running and the computation time is 0.394644 milliseconds.
 - *Enable* - the state of the controller "Enable". When enabled, the controller provides the possibility of creating the parameters and their configuration.
 - *Run* - the state of the controller "Run". The running controller performs the physical data acquisition and/or includes mechanisms for access to these data.
 - *Controller DB* - the address of the database for data storage of the controller and its parameters.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the controller's identifier.
 - *Name* - specifies the controller's name.
 - *Description* - brief description of the controller and its purpose.
 - *To enable* - indicates the status of "Enable" in which to transfer the controller at startup.
 - *To start* - indicates the status of "Run" in which to transfer the controller at startup.
 - *Redundant* - includes the controller in the scheme of the horizontal redundancy of data acquisition of the subsystem "Data acquisition" and points the redundancy mode: "Asymmetric" and "Symmetric".

- *Preferable run* - indicates preference for the execution of the controller at the station in the redundancy scheme.
- *Parameters table* - table name in which to save the parameters (the objects of the parameters of data acquisition) of the controller.
- *Request data period (ms)* - periodicity of the data acquisition task. In this example it is the periodicity of the template's calculation.
- *Request task priority* - sets the priority of data acquisition of this controller. It is used when scheduling the operating system tasks. In the case of execution of the station as the superuser "root", this field includes the planning of the controller's task in real time and with the specified priority.

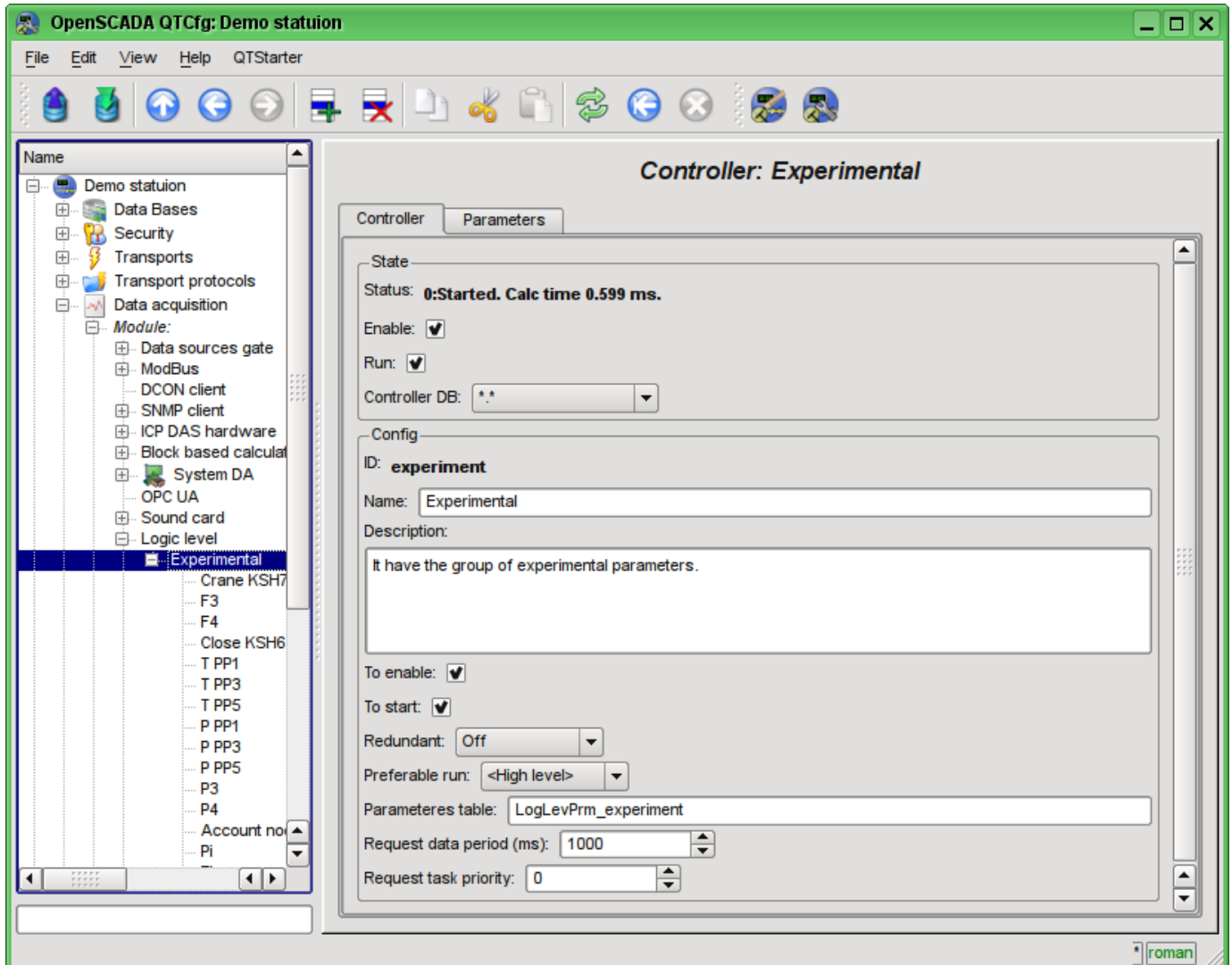


Fig. 4.5i. The main configuration tab of the controller of subsystem "Data acquisition".

"Parameters" tab (Fig.4.5j) contains a list of parameters in the controller, as well as information on the total number and the number of enabled parameters. In the context menu user can add, delete and move to the desired parameter.

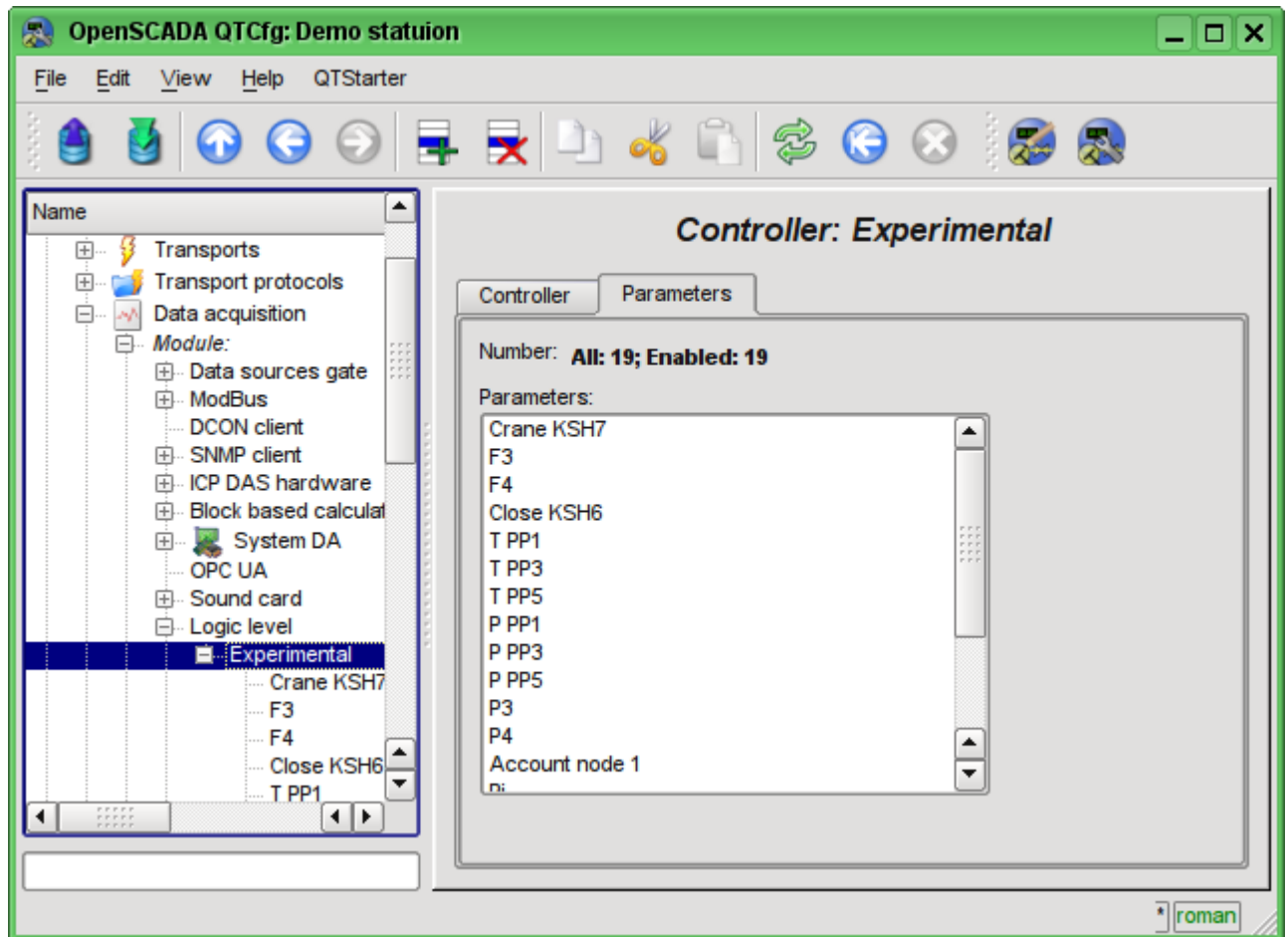


Fig. 4.5j. "Parameters" tab of the configuration page of the controller of subsystem "Data acquisition".

Parameters of the controllers of subsystem "Data acquisition" provides the configuration page with the tabs "Parameters", "Attributes", "Archiving" and "Template config". The tab "Template config" is not standard, but it is present only in the modules of subsystem "Data acquisition", which implement the mechanisms of working under the template in the context of the data source, which they are served. In this review this tab is included for logical completeness of the review of the configuration of templates of parameters of subsystem "Data acquisition" and as the final stage - using.

The tab "Parameter" (Fig.4.5k) contains the main settings:

- Section "State" - contains the properties, which characterize the state of the parameter:
 - *Type* - information on the parameter's type.
 - *Enable* - the state of the parameter "Enable". Enabled parameter is used by the controller for data acquisition.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the parameter's identifier.
 - *Name* - specifies the parameter's name.
 - *Description* - brief description of the parameter and its purpose.
 - *To enable* - indicates the status of "Run" in which to transfer the parameter at startup.
 - *Mode* - contains two fields: directly the mode and its configuration. In the case of the parameter of the controller of the module of this type it is the mode of working "under the template" and the address of the previously discussed template.

The tab "Attributes" (Fig.4.5l) contains the parameter's attributes and their values in accordance with the configuration of the used template and calculation of its program.

The "Archiving" tab (Fig.4.5m) contains the table with the attributes of a parameter in the columns and the archivers in rows. The user can set the archiving for the desired attribute with the required archiver

simply by changing the cell at the intersection.

The "Template config" tab (Figure 4.5n) contains the configuration fields in accordance with the template. In this example it is the group link on the external parameter. This link can be set simply by pointing the way to the parameter if the flag "Only attributes are to be shown" is not set, or to set the addresses of the attributes separately in the case if the flag is set.

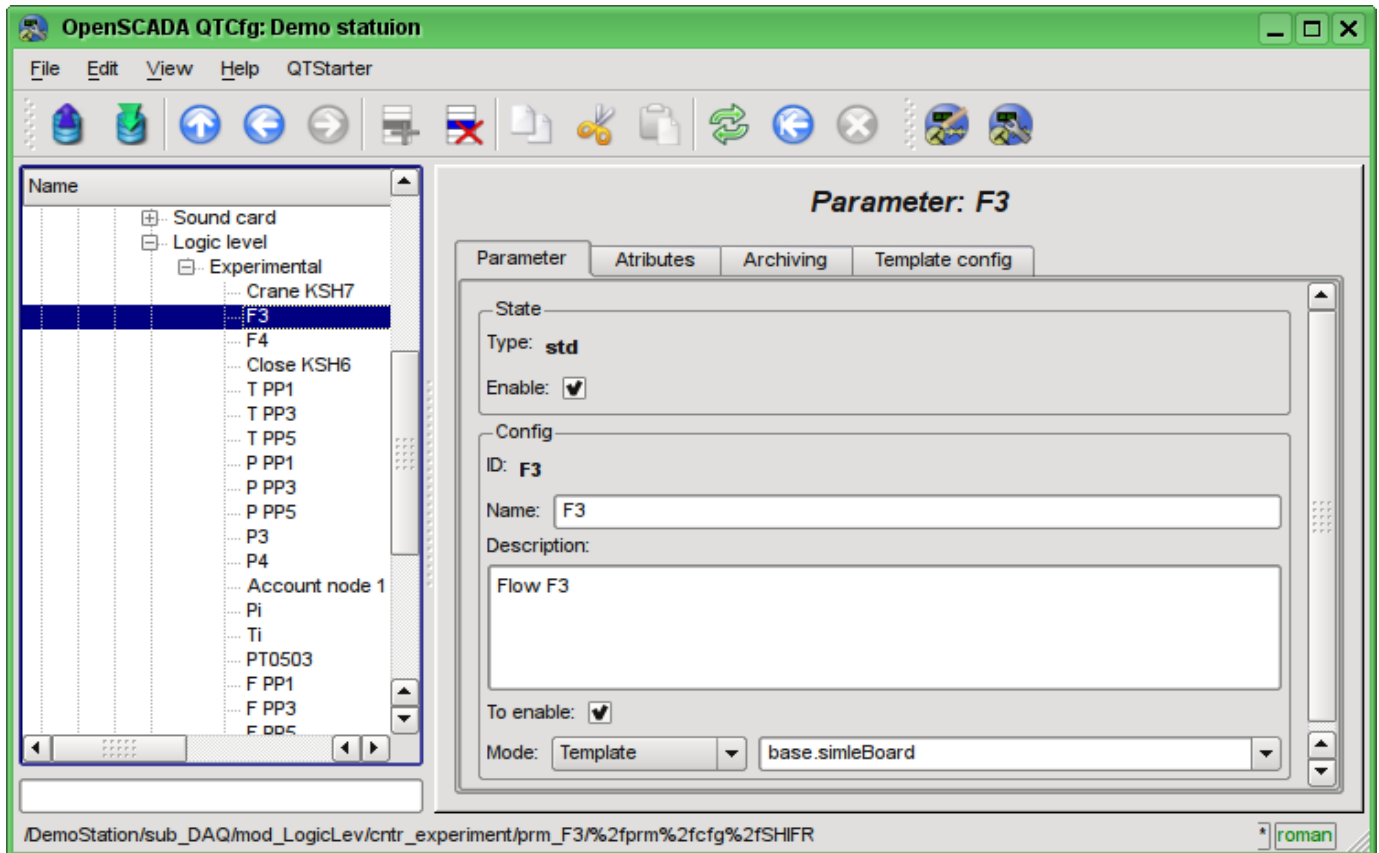


Fig. 4.5k. The main configuration tab of the parameter of the controller of subsystem "Data acquisition".

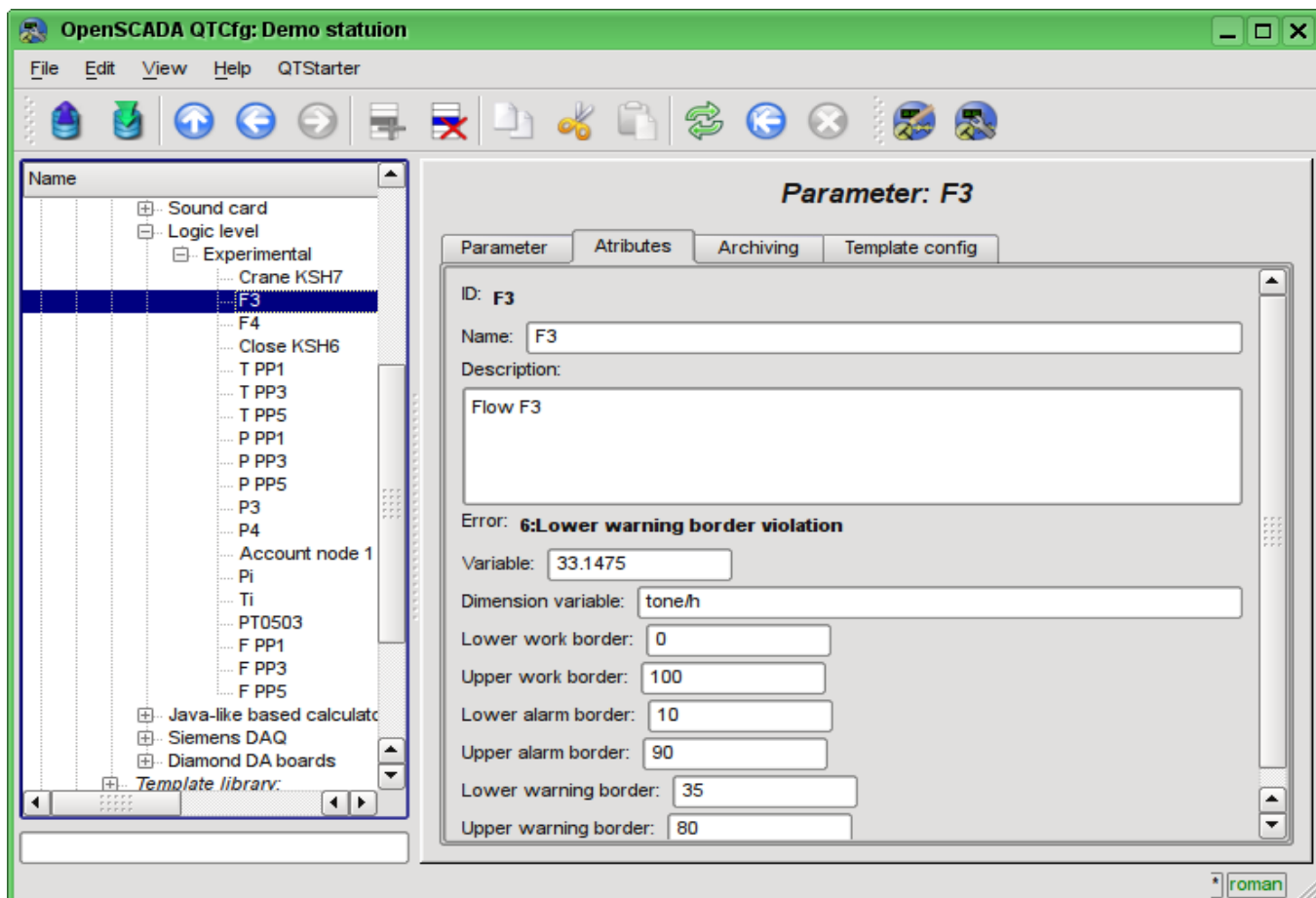


Fig. 4.5l. The "Attributes" tab of the parameter of the controller of subsystem "Data acquisition".

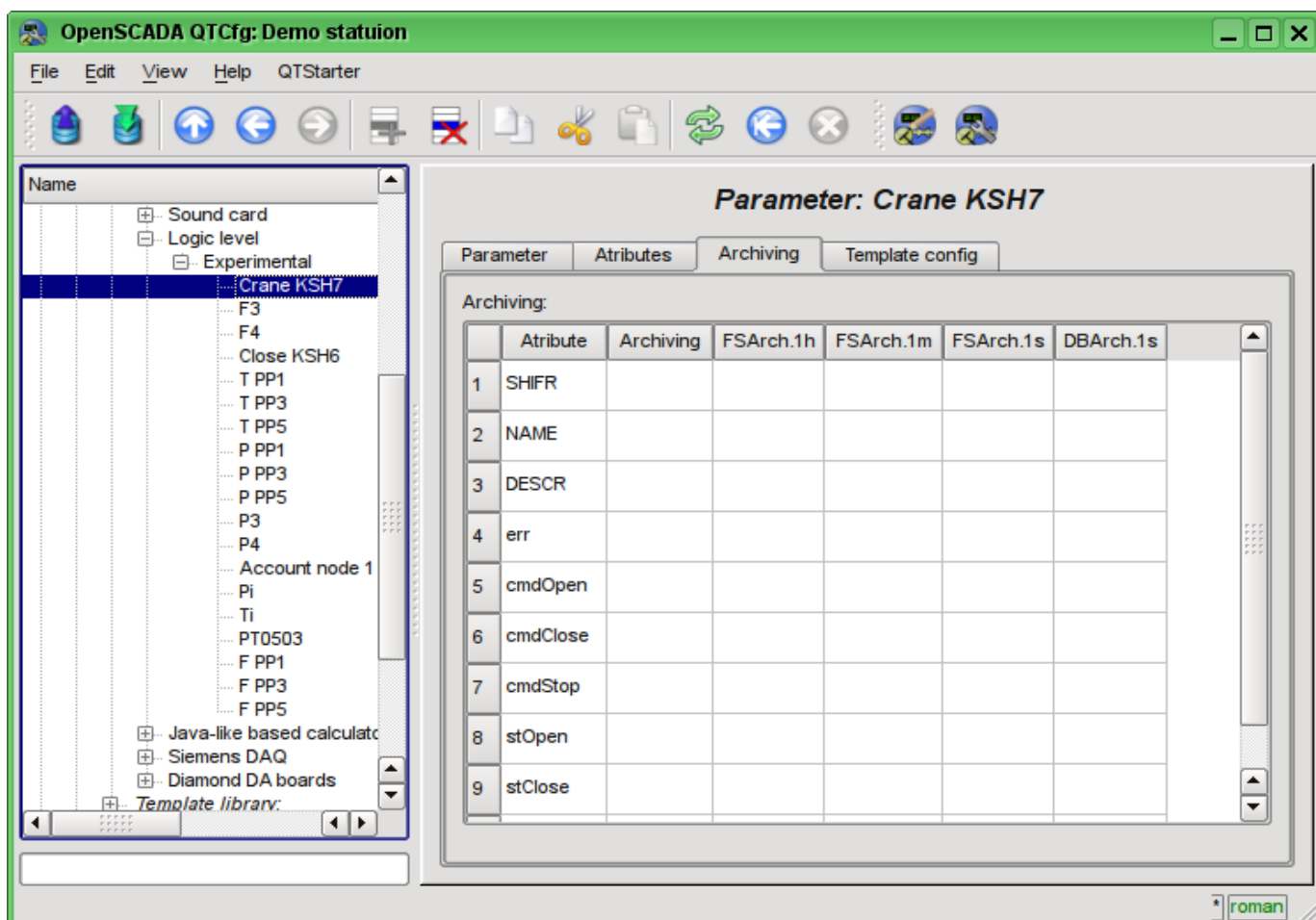


Fig. 4.5m. The "Archiving" tab of the parameter of the controller of subsystem "Data acquisition".

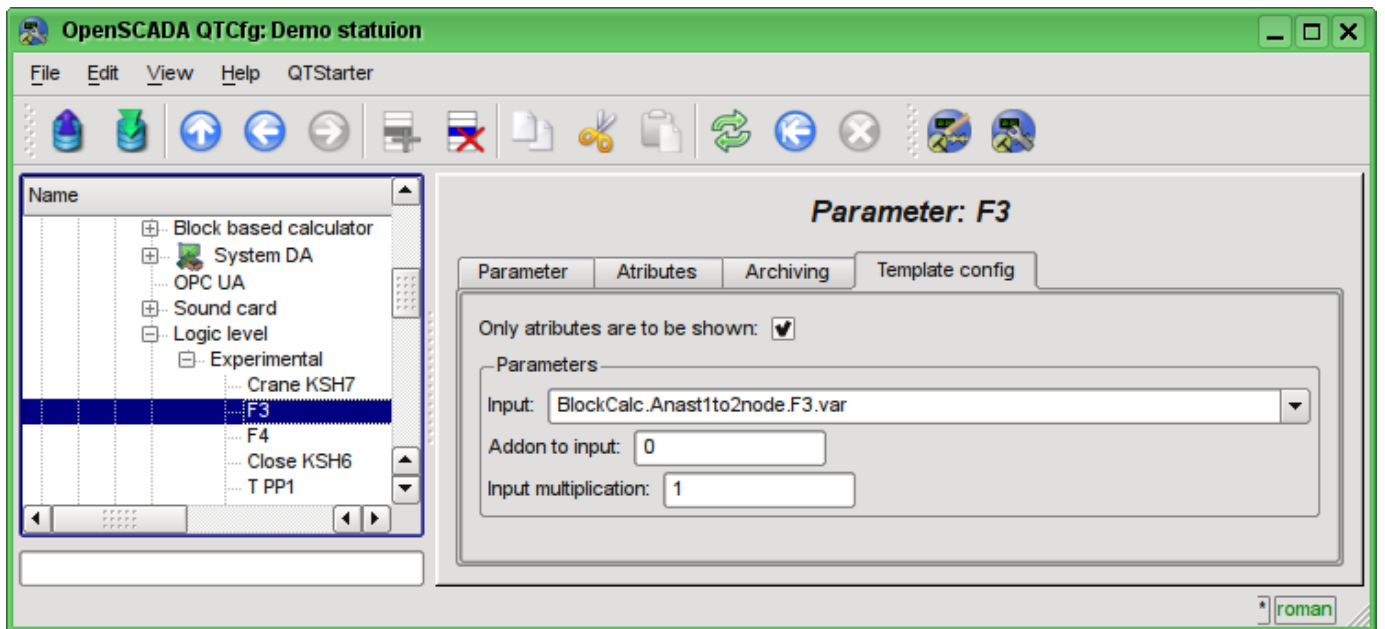


Fig. 4.5n. The "Template config" tab of the parameter of the controller of subsystem "Data acquisition".

4.6. Subsystem "Archives"

The subsystem is modular and contains the hierarchy of objects depicted in Fig.4.6a. To configure the subsystem the root page of the subsystem "Archives" is provided, it contains tabs "Messages archive", "Value archives", "Modules" and "Help".

To gain the access to modify the objects of this subsystem the user of the group "Archive" or the privileged user rights are required.

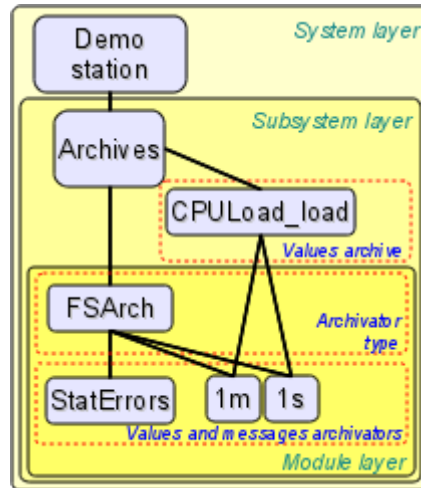


Fig. 4.6a. The hierarchical structure of subsystem "Archives"

The "Messages archive" tab (Fig.4.6b) contains the configuration of messages archive and the request form of messages from the archive.

Configuration of the messages archive is represented by the fields:

- *Maximum requested messages* - indicates the global limit on the maximum number of messages processed by the request.
- *Messages buffer size* - indicates the dimension of the area of memory reserved for the interim buffer of messages. Messages from the buffer are requested for viewing and archived with the messages archivers.
- *Archiving period (s)* - the periodicity with which the archivers select messages from the buffer for their archiving.

The messages request form contains the configuration fields of the request and the table of results. Configuration fields of the request are:

- *Time* - specifies the request time.
- *Size (s)* - specifies the size and the depth of the request in seconds.
- *Category pattern* - specifies the category of the requested messages. In the category you can specify the elements of a sample of the template, namely, the characters '*' - for any string and '?' - for any character.
- *Level* - indicates the minimum level of messages, ie request will be processed for messages with a level more than or equal to the specified one.
- *Archivator* - indicates the messages archiver, for which the request is to be processed. If the value is missing, the request will be processed for the buffers and all archivers. If <buffer> is specified, then the request will be processed only for the messages buffer.

The result table contains rows of messages with the following columns:

- *Time* - message's time.
- *Category* - message's category.
- *Level* - message's level.
- *Message* - message's text.

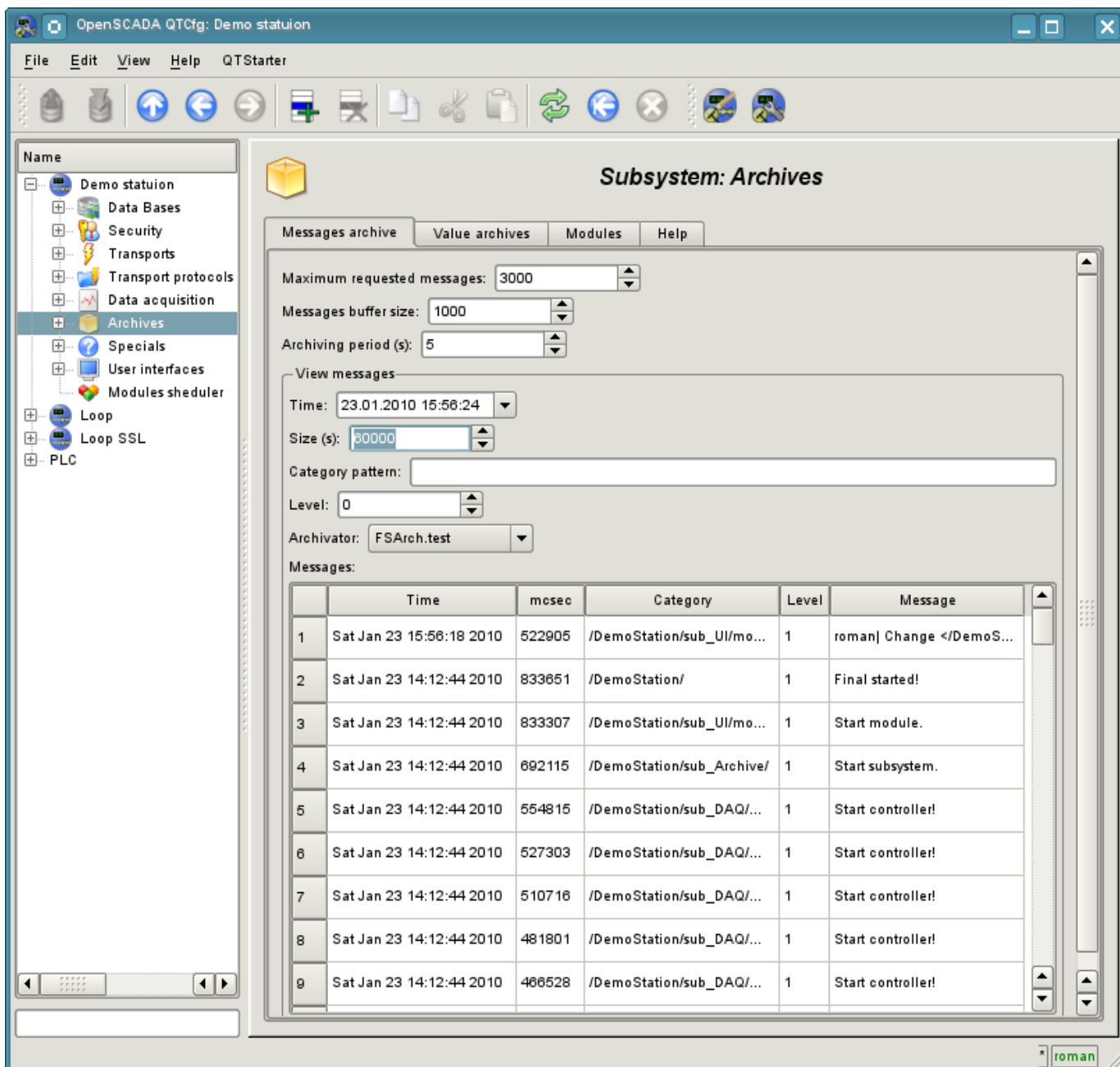


Fig. 4.6b. The "Messages archive" tab of the subsystem "Archives".

Tab "Value archives" (Fig.4.6c) contains the general configuration of value's archiving and the list of archives of values. In the context menu of the list of values the user has the opportunity to add, delete and move to the desired archive. The general configuration of archiving is represented by the fields:

- *Get data period (ms)* - indicates the periodicity of the active archiving task. In fact, the highest level of detail or the minimum period of active archives is determined by this value.
- *Get data task priority level* - sets the priority of task of active archiving. It is used when scheduling the operating system tasks. In the case of execution of the station with the rights of the superuser "root" this field includes scheduling of the archiving task in real time and with the specified priority.

The "Modules" tab (Fig. 4.1b) contains a list of modules in subsystem "Archives" and is identical for all modular subsystems. The "Help" tab contains the brief help for this page.

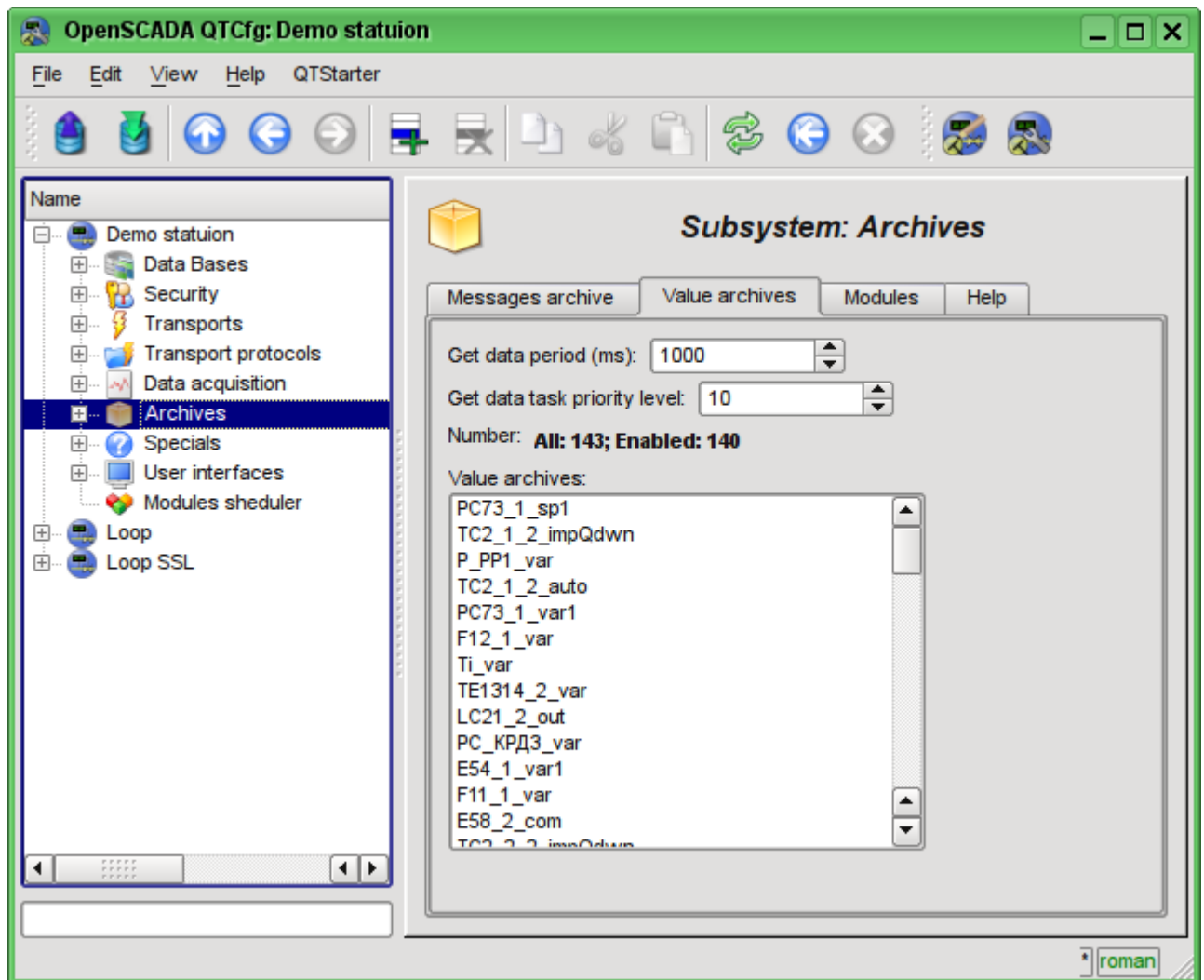


Fig. 4.6c. The "Value archives" tab of the subsystem "Archives".

Archive of values of subsystem "Archives" provides the configuration page with the tabs "Archive", "Archivators" and "Values".

Tab "Archive" (Fig.4.6d) contains the basic settings of the archive:

- Section "State" - contains the properties, which characterize the state of the archive:
 - *Running* - the state of the parameter "Running". Running archive collects data in the buffer and is served by the archivators.
 - *Archive DB* - database address for storing the archive's data.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the archive's identifier.
 - *Name* - specifies the archive's name.
 - *Description* - brief description of the archive and its purpose.
 - *To start* - indicates the state "Running" in which to transfer the archive at startup.

- *Value type* - indicates the type of values which are stored in the archive from the list: "Boolean", "Integer", "Real" и "String".
- *Source* - indicates the type and address of the source. Type of source is indicated from the list: "Passive", "Passive param. attribute" or "Active param. attribute". Passive archive does not have an associated source of values, the data to the such archive the source transfers by itself. Types with the attribute of the parameter in the address field indicate the parameter of the subsystem "Data acquisition" as the source. Passive attribute of the parameter sends data to the archive by itself with its own period of data acquisition. Active attribute of the parameter is queried by the archiving task of this subsystem.
- *Buffer period (s)* - indicates the periodicity of values in the archive's buffer.
- *Buffer size (items)* - indicates the dimensionality and depth of the archive's buffer. The dimensionality is usually set in terms of 60 sec of the periodicity of the archiving task with the reserve.
- *Buffer hard time griding* - indicates the mode of the buffer. The hard grid mode involves the memory reservation for each value, but without the timestamp. This mode eliminates the possibility of packaging the adjacently-identical values, but also saves on storage of the timestamp. Otherwise, the buffer operates in the mode of storage the value and timestamp and supports the packaging of adjacently-identical values.
- *Buffer high time resolution* - indicates the possibility of storing values at intervals up to 1 microsecond, differently the values can be stored at intervals up to 1 second.

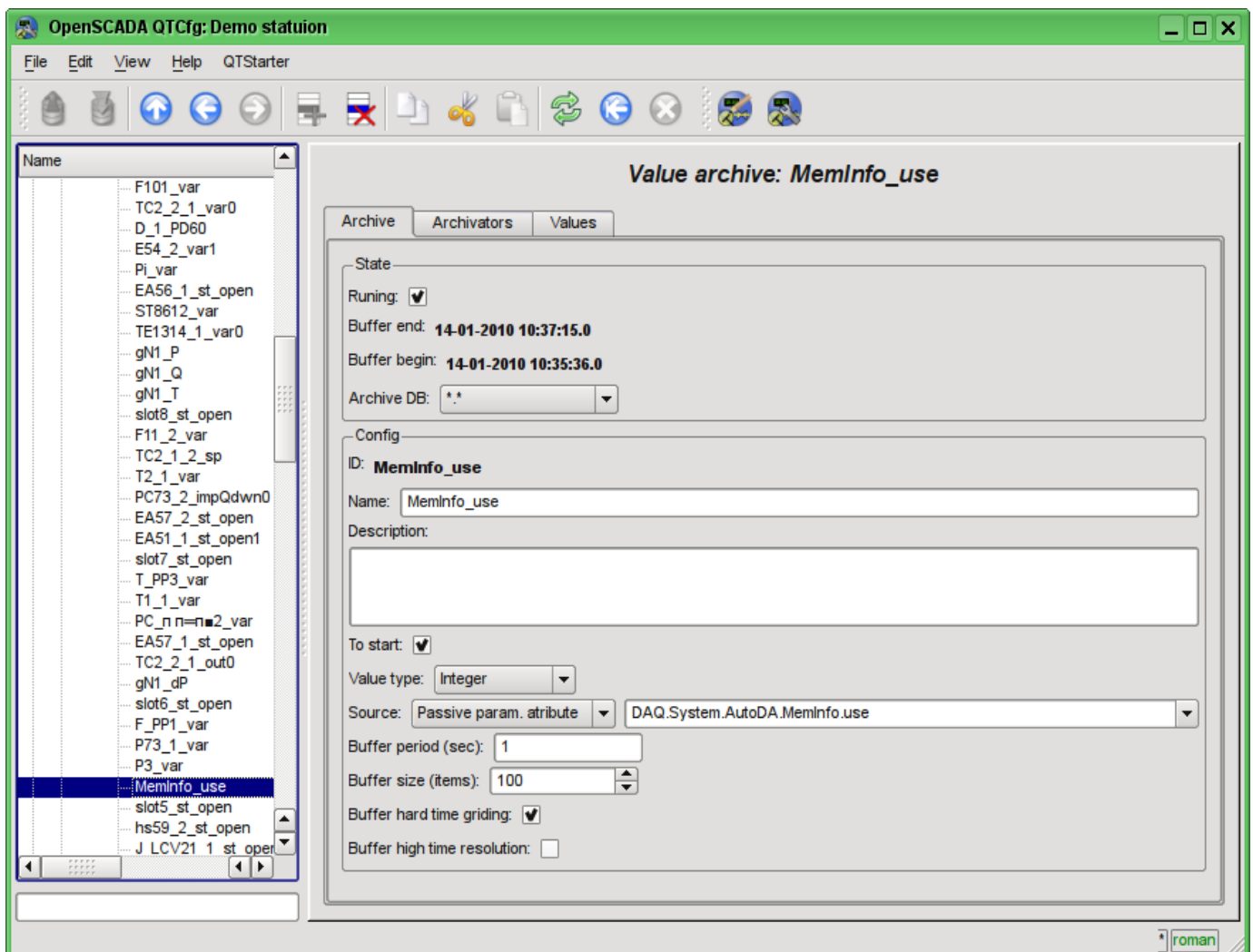


Fig. 4.6d. The main configuration tab of the values' archive of subsystem "Archives".

Tab Archivators' (Fig.4.6e) contains the table with the configuration of the processing of the archive by the available archivers. Lines are available archivers, and the columns are the following parameters:

- *Archivator* - information on the archiver's address.
- *Start* - information on the archiver's state "Started".
- *Process* - sign of the processing this archive be the archiver. The field is available for modification by the user.
- *Period (s)* - information on the periodicity of the archiver.
- *Begin* - date of the archive data beginning in the archiver.
- *End* - date of the archive data ending in the archiver.

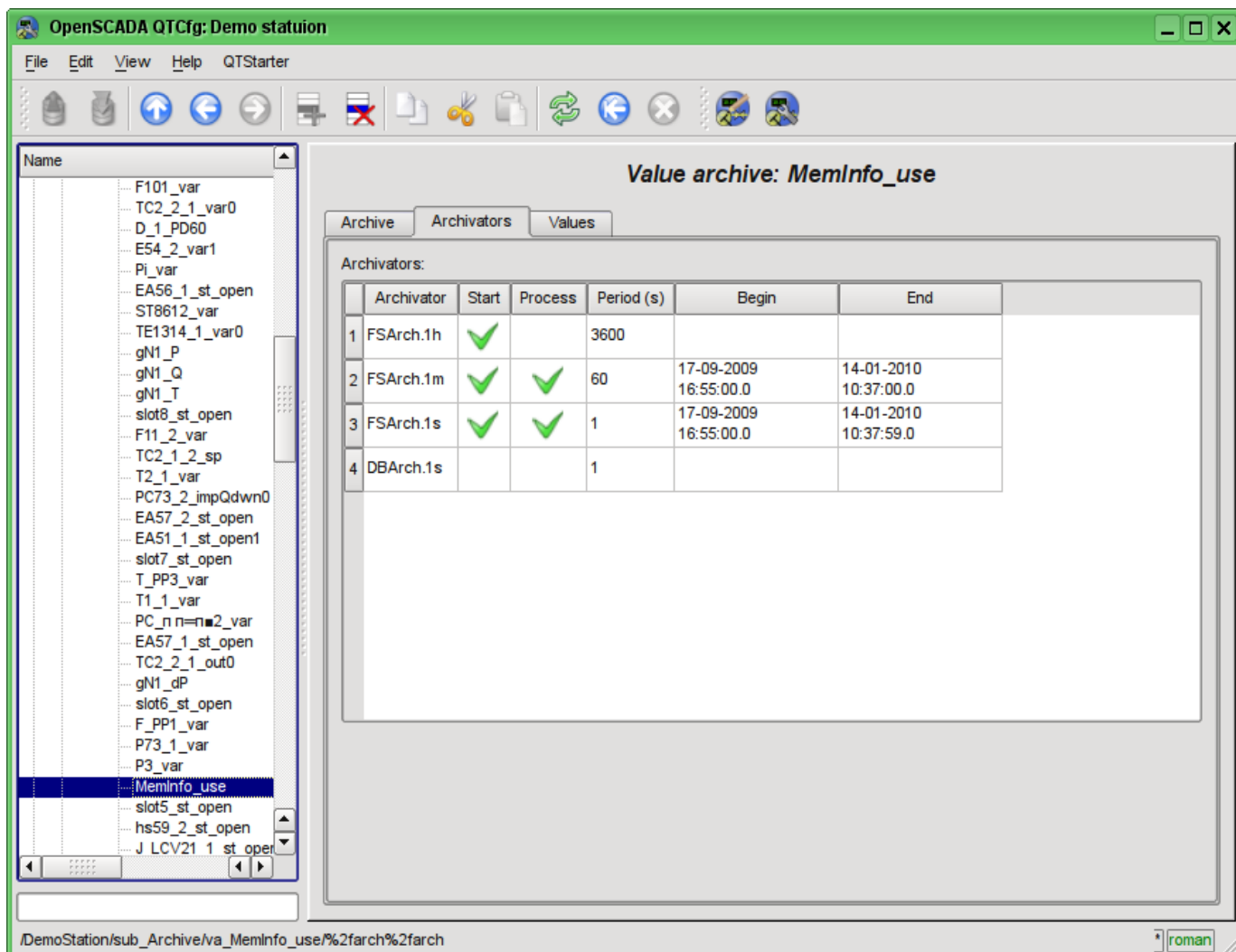


Fig. 4.6e. The "Archivators" tab of the values archive of subsystem "Archives".

Tab "Values" (Fig.4.6f) contains the values request in the archive and the result as a table of values or image of the trend. Values request contains the fields:

- *Time* - indicates the time of request. It contains two fields: the field of date + time and microseconds.
- *Size (s)* - specifies the size or depth of the request in seconds.
- *Archivator* - indicates values archiver for which the request is to be processed. If the value is missing, the request will be processed for the buffer and for all archivers. If the <buffer> is specified, then the request will be processed only for the archive's buffer.
- *Show trend* - indicates the necessity for presentation of the archive's data in the form of a graph (trend), otherwise the result is presented in a table that contains only time and value. In the case of installation of this field the schedule is formed and displayed, in addition additional configuration fields of the image settings are appeared:
 - *Picture size* - indicates the width and height of the generated image in pixels.
 - *Value scale* - indicates the lower and upper limit of the scale of value. If both values are set to 0 or equal, then the scale will be determined automatically depending on the values.

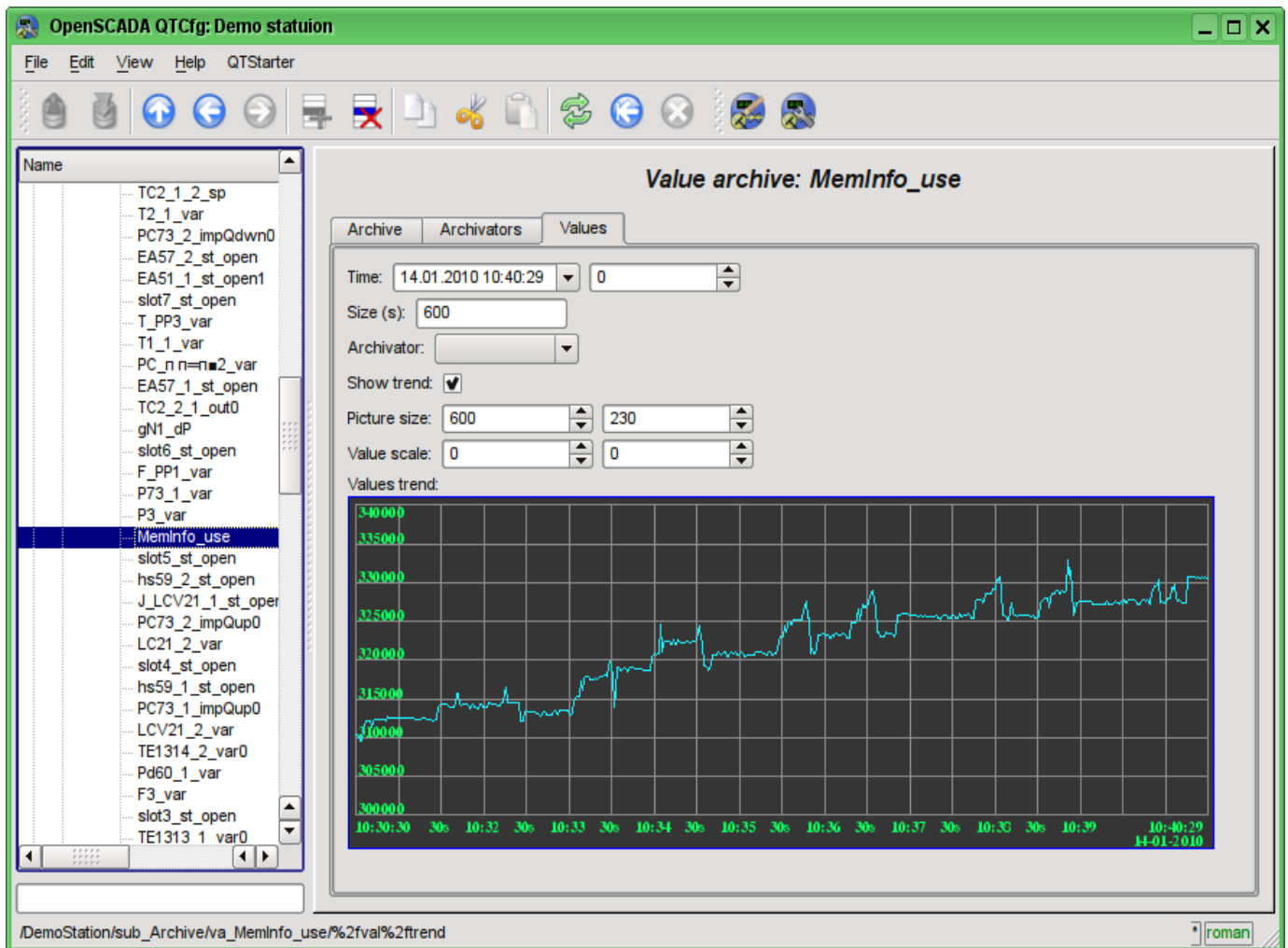


Fig. 4.6f. The "Values" tab of the values archive of subsystem "Archives".

Each module of the "Archives" subsystem provides configuration page with the tabs "Archivators" and "Help". The "Archivators" tab (Fig.4.6g) contains a list of messages and values archivers registered in the module. The context menu of the list provides user with possibility to add, delete and move to the desired controller. The "Help" tab contains information about the module of subsystem "Archives" (Fig. 4.1d), whose structure is identical for all modules.

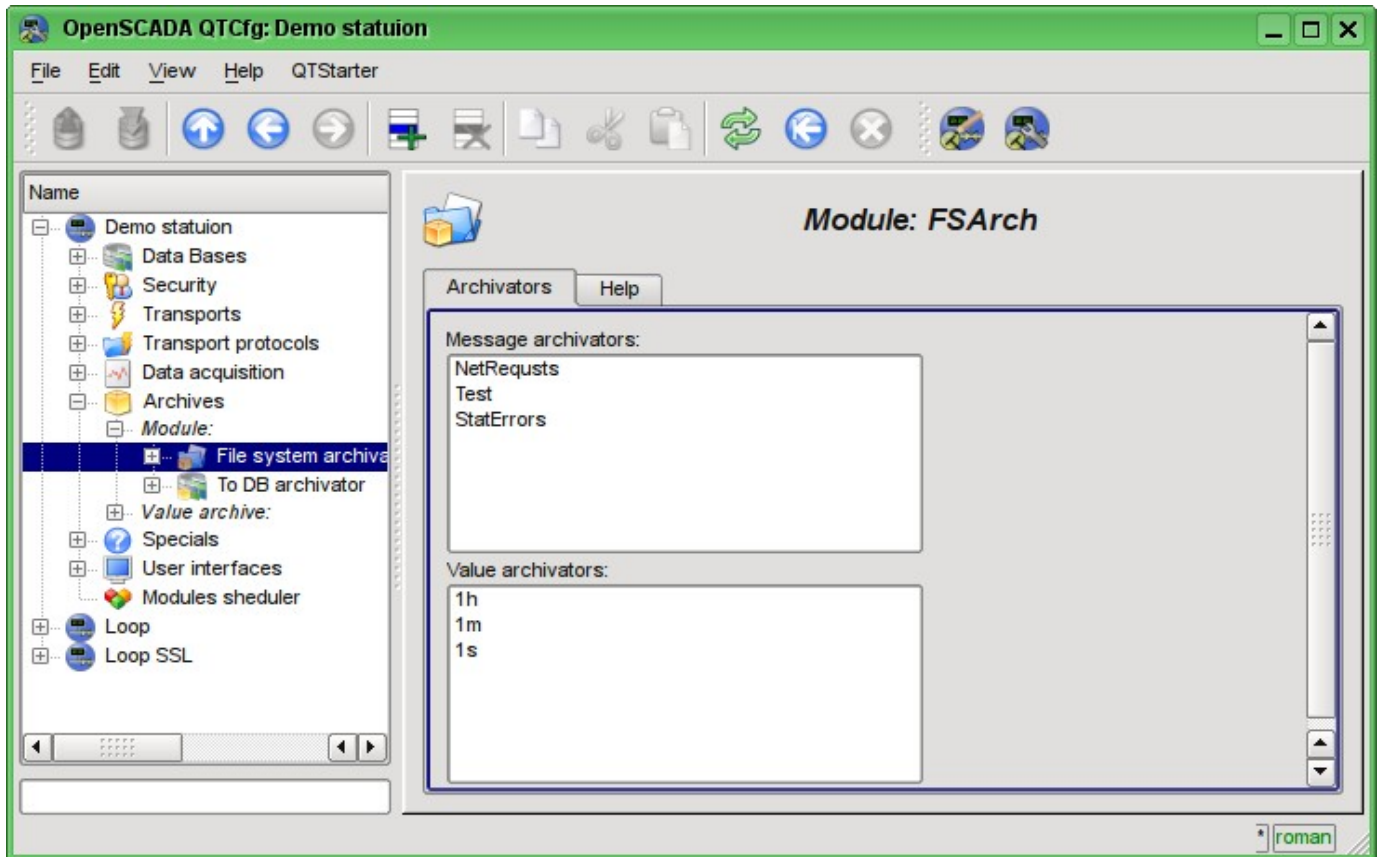


Fig. 4.6g. The "Archivators" tab of the module of subsystem "Archives".

Messages archivers contains their own configuration page with tabs "Archivator" and "Messages".

The "Archivator" tab (Fig.4.6h) contains the basic settings. The structure of these settings may differ slightly from one module of this subsystem to another as you can find in the own documentation of modules. As an example we shall examine the settings of the messages archiver from the module of the archive on the file system [Arch.FSArch](#) Settings:

- Section "State" - contains the properties, high characterize the archivers' state:
 - *Running* - archivers' state "Running". The running archiver processes the messages archive buffer and puts his data in its repository, but also it processes requests for access to data in the repository.
 - *Archivator DB* - database address for storing the archiver's data.
 - *End* - date + time of the last data in the archiver's repository.
 - *Begin* - date + time of the first data in the archiver's repository.
 - *Archivator files size (kB)* - information about the total size of the archiver's files with the data.
 - *Archiving time (ms)* - time spent on the archiving of messages archive data.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the archiver's identifier.
 - *Name* - indicates the archiver's name.
 - *Description* - brief description of the archiver and its purpose.
 - *Address* - address of the storage in the specific for the type of archiver (module) format. Format description usually available in the tooltip for this field. In the example it is the relative path to the storage directory.
 - *Message level* - indicates the level of archiver's messages. Messages with a level greater than or equal to the specified one are processed by the archiver.

- *Message categories* - list of categories, separated by ';', of messages. Messages matched with the templates of categories will be processed by the archiver. In the category you can specify the elements of a sample of the template, namely, the characters '*' - for any string and '?' - for any character.
- *To start* - indicates the status "Running", in which to transfer archiver at startup.
- Section "Additional options" - specialized section for module about the contents of which you can read in the documentation on the module.

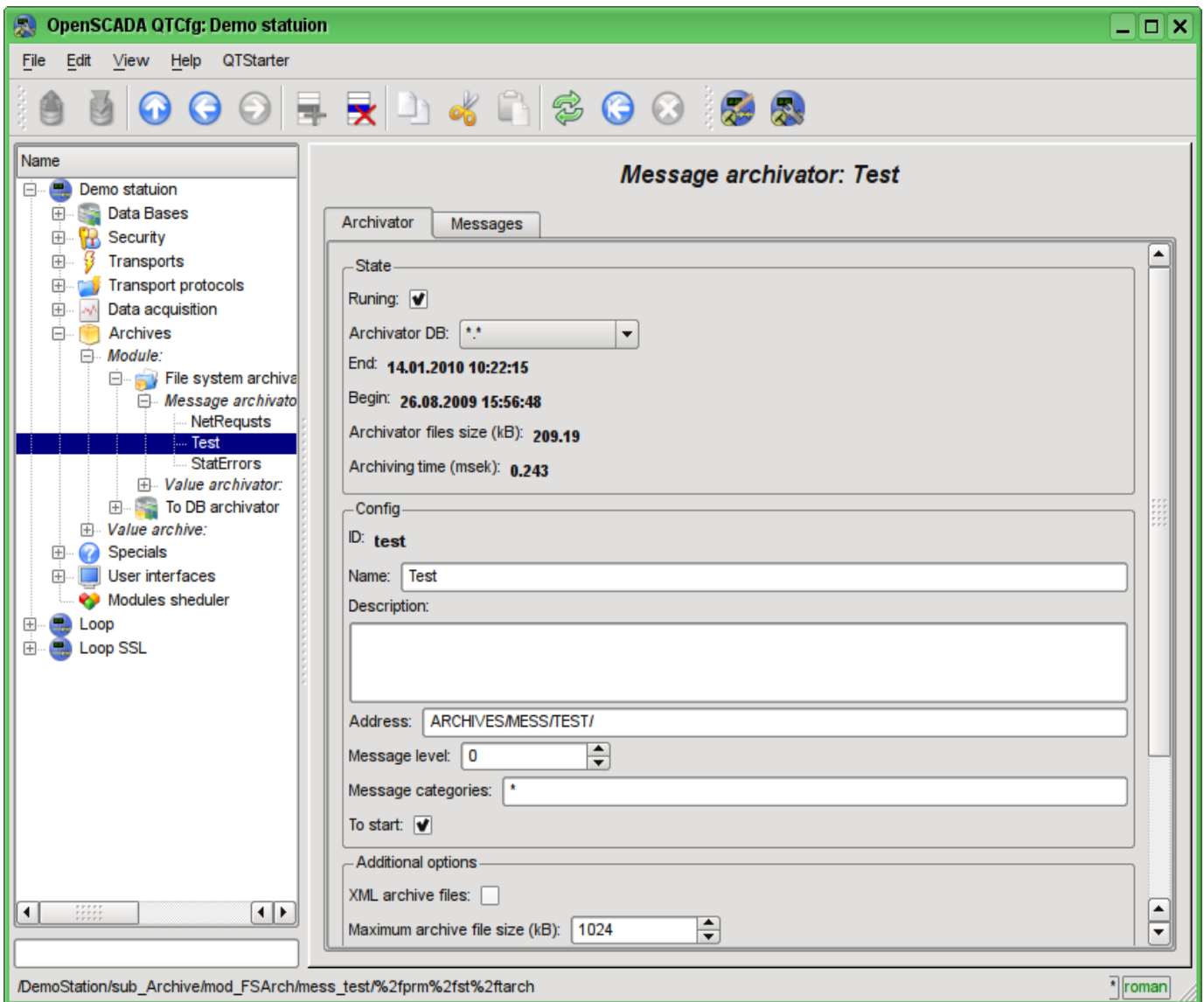


Fig. 4.6h. The main tab of the messages archiver configuration of subsystem "Archives".

The "Messages" tab (Fig.4.6i) contains the form of the messages request from the archive of the archiver:

- *Time* - indicates the time of the request.
- *Size (s)* - indicates the size and depth of the request in seconds.
- *Category pattern* - indicates the category of the requested messages. In the category you can specify the elements of a sample of the template, namely, the characters '*' - for any string and '?' - for any character.
- *Level* - indicates a minimum level of messages, ie the request will be processed for messages with the level greater or equal to the specified one.

The result table contains messages rows with the following columns:

- *Time* - message time.
- *Category* - message category.
- *Level* - message level.
- *Message* - message text.

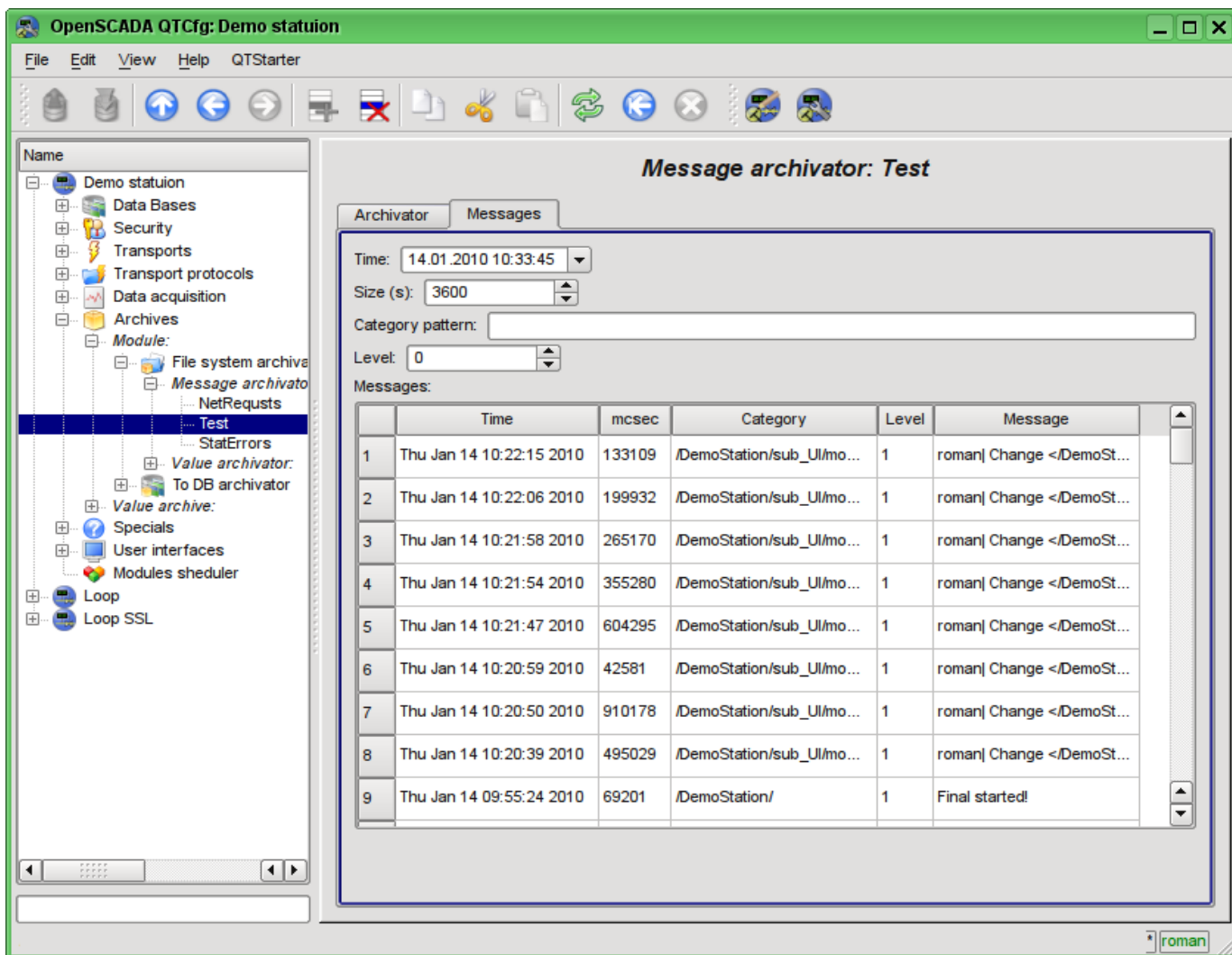


Fig. 4.6i. Tab of the messages request "Messages" of the messages archiver of subsystem "Archives".

Values archivers contains their own configuration page with tabs "Archivator" and "Archives".

The "Archivator" tab (Fig.4.6j) contains the basic settings. The structure of these settings may differ slightly from one module of this subsystem to another as you can find in the own documentation of modules. As an example we shall examine the settings of the messages archiver from the module of the archive on the file system [Arch.FSArch](#) Settings:

- Section "State" - contains the properties, which characterize the archivers' state:
 - *Running* - archivers' state "Running". The running archiver processes the messages archive buffer and puts his data in its repository, but also it processes requests for access to data in the repository. запросы на доступ к данным в хранилище.
 - *Archiving time (ms)* - information about the time spent on archiving data of the archives buffers. Periodicity of archiving is set in the field "Period archiving" in the section "Config" of the tab.
 - *Archivator DB* - database address for storing the archiver's data.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the archiver's identifier.
 - *Name* - indicates the archiver's name.
 - *Description* - brief description of the archiver and its purpose.
 - *Value period (s)* - indicates the periodicity of values that are contained in the archiver's repository.
 - *Period archiving (s)* - indicates the periodicity of the archives buffers data archiving task. The dimension of the archives buffers in the time expression must not be less, and preferably somewhat greater than the periodicity of the archiving task.
 - *Address* - address of the storage in the specific for the type of archiver (module) format. Format description usually available in the tooltip for this field. In the example it is the

relative path to the storage directory.

- *To start* - indicates the status "Running", in which to transfer archiver at startup.
- Section "Additional options" - specialized section for module about the contents of which you can read in the documentation on the module.

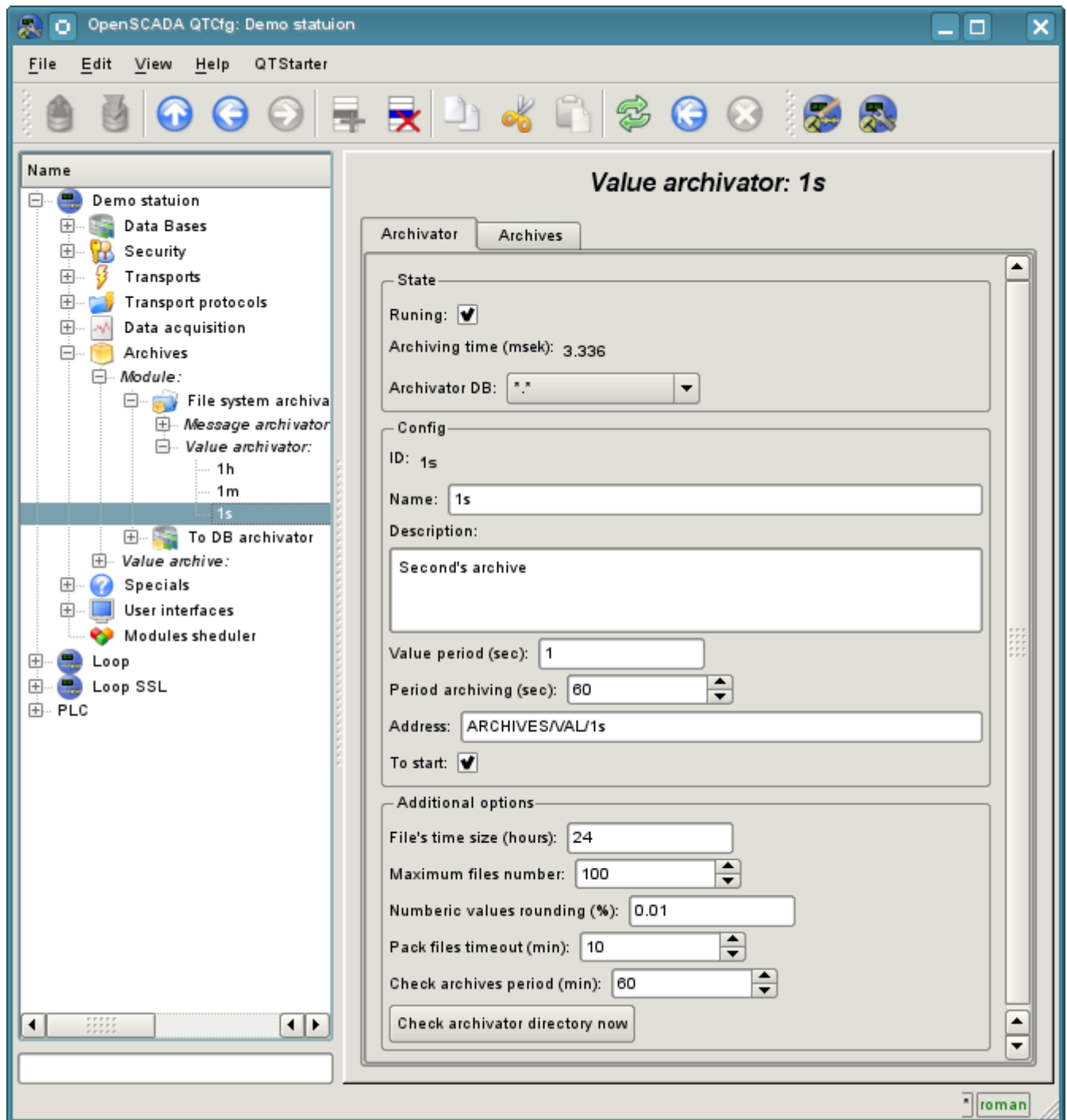
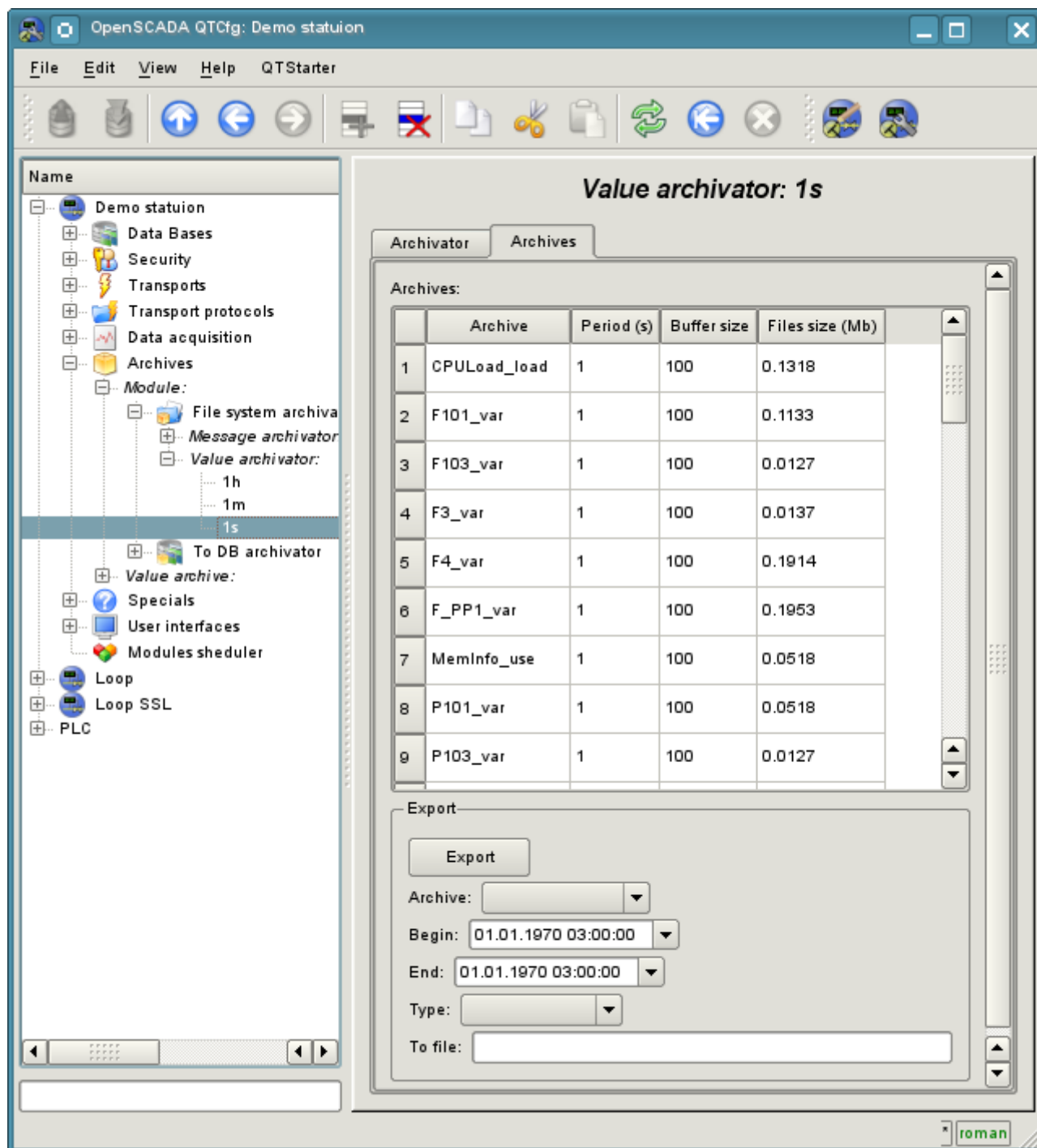


Fig. 4.6j. The main tab of the values archiver configuration of subsystem "Archives".

The "Archives" tab (Fig.4.6k) contains a table with information about the archives being processed by the archiver. In the rows the table contains archives, and in the columns - the following information:

- *Archive* - archive's name.
- *Period (s)* - archive's periodicity in seconds.
- *Buffer size* - buffer's dimension in units.
- *Files size (Mb)* - specific to the module Arch.FSArch field with information about the total size of the files of the archiver's storage for the archive.

In the case of the module Arch.FSArch in this tab you can find the form of export the archiver's data.



4.6k. The "Archives" tab of the values archiver of subsystem "Archives".

4.7. Subsystem "User interfaces"

The subsystem is modular. To configure the subsystem the root page of the subsystem "User Interfaces" is provided, it contains the tabs "Modules" and "Help". The "Modules" tab (Fig. 4.1b) contains a list of modules of subsystem and it is identical for all modular subsystems. The "Help" tab contains a brief help for this page.

Each module of the subsystem "User Interfaces" provides configuration page with the tabs "User Interface" and "Help". The "User Interface" tab (Fig.4.7a) provides the parameter for monitoring the "Running" status of the module, as well as the configuration sections specialized for the modules of this subsystem. On the "Help" tab there is an information about the module of the subsystem "User Interfaces" (Fig. 4.1d), which structure is identical for all modules.

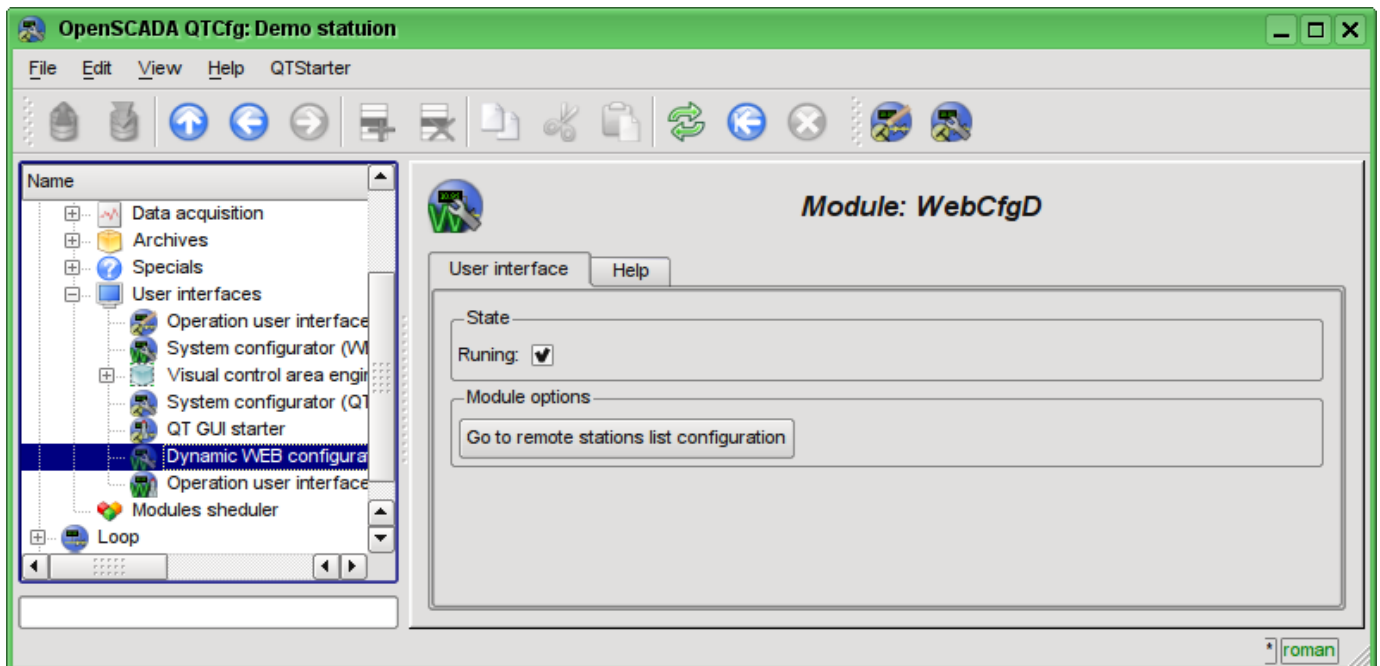


Fig. 4.7a. The "User Interface" tab of the module of subsystem "User Interfaces".

4.8. Subsystem "Specials"

The subsystem is modular. To configure the subsystem the root page of the subsystem "User Interfaces" is provided, it contains the tabs "Modules" and "Help". The "Modules" tab (Fig. 4.1b) contains a list of modules of subsystem and it is identical for all modular subsystems. The "Help" tab contains a brief help for this page.

Each module of the subsystem "Specials" provides configuration page with the tabs "Special" and "Help". The "Special" tab (Fig.4.8a) provides the parameter for monitoring the "Running" status of the module, as well as the configuration sections specialized for the modules of this subsystem. On the "Help" tab there is an information about the module of the subsystem "Specials" (Fig. 4.1d), which structure is identical for all modules.

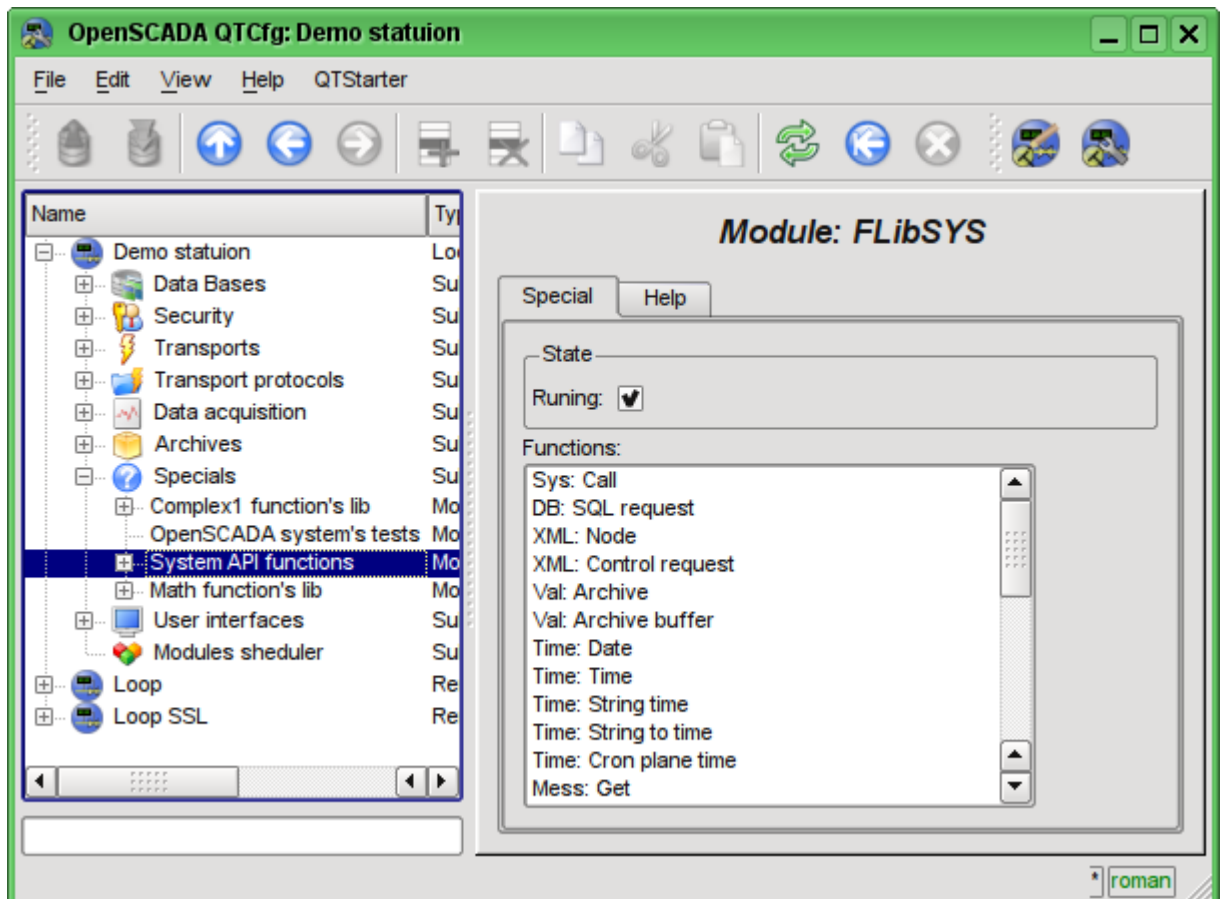


Fig. 4.8a. The "Special" tab of the module of subsystems "Specials".

4.9. Subsystem "Modules sheduler"

The subsystem is not modular. To configure the subsystem the subsystem's page "Modules sheduler" is provided, it contains tabs "Subsystem" and "Help". The "Subsystem" tab (Fig.4.9a) contains the basic settings of the subsystem. The "Help" tab contains a brief help for this page. The structure of the tab "Subsystem":

- *Path to shared libs (modules)* - information about the location of the directory with the modules of the OpenSCADA system. It is set by the parameter `<ModDir>` of the station, of the configuration file.
- *Allowed modules* - information about the list, separated by ';', of modules that are authorized for automatic connection and renewal. The value of '*' is used to resolve all the modules. It is set by the parameter `<ModAllow>` of the section of subsystem, `sub_ModSched`, of the station of the configuration file.
- *Denied modules* - information about the list, separated by ';' of modules that are denied for automatically connection and updating. It is set by the parameter `<ModDeny>` of the section of subsystem "sub_ModSched" of station of configuration file. List of denied modules has higher priority than allowed.
- *Check modules period (sec)* - indicates the periodicity of testing modules on the fact of their updating. Modules that are allowed for automatically connection and updating will be automatically updated.
- *Check modules now* - command to check the modules on the fact of their updating. Modules that are allowed for automatically connection and updating will be automatically updated.
- *Shared libs (modules)* - table with the list of shared libraries with the modules detected by OpenSCADA. Rows are modules, and in the columns there is an information about them:
 - *Path* - information on the full path to the shared library.
 - *Time* - information about the time the of last modification of a shared library.
 - *Modules* - information about the list of modules in a shared library.
 - *Enable* - state "Enable" of the shared library. Privileged users are provided with an opportunity to manually enable/disable the shared libraries by changing this field.

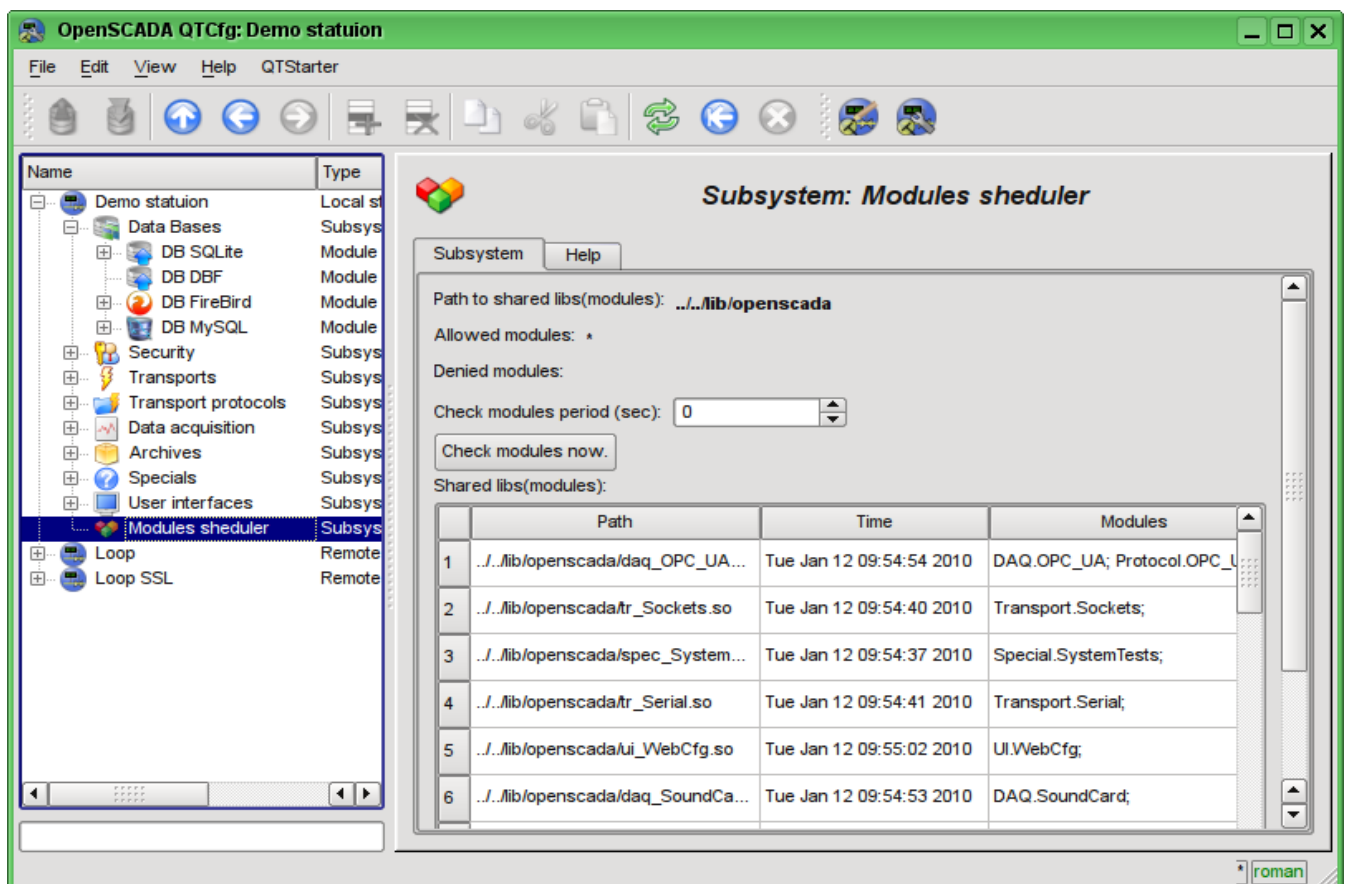


Fig. 4.9a. The main configuration tab of subsystem "Modules sheduler".

4.10. Configuration file of the OpenSCADA and parameters of command-line OpenSCADA execution.

Configuration file of the OpenSCADA system is provided to store the system and general configuration of OpenSCADA-station. Only in the configuration file and through the command-line options you can specify the part of the key system parameters of the station, so familiarity with the structure of the configuration file is necessary for professionals who make solutions based on OpenSCADA.

The configuration file of the OpenSCADA system can be called somehow, but the oscada.xml name and derived from it are accepted. The configuration file is usually indicated when you start the station by the command-line option `--Config=/home/roman/roman/work/OScadaD/etc/oscada_demo.xml`. For the convenience of the calling the startup scripts of the station are created with the correct configuration file, for example script (openscada_demo) of the demo station execution:

```
#!/bin/sh
openscada --Config=/etc/oscada_demo.xml $@
```

If the configuration file is not specified then the standard configuration file: /etc/oscada.xml is used.

Structure of the configuration file based on the extensible markup language XML. Therefore the strict adherence to the rules of XML syntax is required. An example of the configuration file of the OpenSCADA, with configuration nodes of most of the OpenASCADA components, is given below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<OpenSCADA>
  <!-- This is the OpenSCADA configuration file. -->
  <station id="DemoStation">
    <!-- Discribe internal parameter for station. Station this only OpenSCADA programm. -->
    <prm id="StName">Demo station</prm>
    <prm id="StName_ru">Демо станция</prm>
    <prm id="StName_uk">Демо станція</prm>
    <prm id="WorkDB">SQLite.GenDB</prm>
    <prm id="Workdir">~/openscada</prm>
    <prm id="IcoDir">./icons</prm>
    <prm id="ModDir">/usr/lib/openscada</prm>
    <prm id="LogTarget">10</prm>
    <prm id="MessLev">0</prm>
    <prm id="Lang2CodeBase">en</prm>
    <prm id="SaveAtExit">0</prm>
    <prm id="SavePeriod">0</prm>

    <node id="sub_BD">
      <prm id="SYSStPref">0</prm>
      <tbl id="DB">
        <fld ID="GenDB" TYPE="SQLite" NAME="Generic DB" NAME_ru="Основная БД"
          NAME_uk="Основна БД" ADDR="./DEMO/DemoSt.db" CODEPAGE="UTF-8"/>
      </tbl>
    </node>

    <node id="sub_Security">
      <!--
      <tbl id="Security_user">
        <fld
          NAME="root"
          DESCR="Super user"
          DESCR_ru="Супер пользователь"
          DESCR_uk="Супер користувач"
          PASS="openscada"/>
        <fld
          NAME="user"
          DESCR="System user"
          DESCR_ru="Системный пользователь"
          DESCR_uk="Системний користувач"
          PASS="" />
      </tbl>
      <tbl id="Security_grp">
        <fld
          NAME="root"
          DESCR="Super users groups"
          DESCR_ru="Группа суперпользователей"
          DESCR_uk="Група суперкористувачів"
          USERS="root;user" />
      </tbl>-->
    </node>
```

```

<node id="sub_ModSched">
  <prm id="ModAllow">*</prm>
  <prm id="ModDeny"></prm>
  <prm id="ChkPer">0</prm>
</node>

<node id="sub_Transport">
  <!--
  <tbl id="Transport_in">
    <fld
      ID="WEB_1"
      MODULE="Sockets"
      NAME="Generic WEB interface"
      NAME_ru="Основной WEB интерфейс"
      NAME_uk="Основний WEB інтерфейс"
      DESCRIPT="Generic transport for WEB interface."
      DESCRIPT_ru="Основной транспорт для WEB интерфейса."
      DESCRIPT_uk="Основний транспорт для WEB інтерфейсу."
      ADDR="TCP::10002:0"
      PROT="HTTP"
      START="1"/>
    <fld
      ID="WEB_2"
      MODULE="Sockets"
      NAME="Reserve WEB interface"
      NAME_ru="Резервный WEB интерфейс"
      NAME_uk="Резервний WEB інтерфейс"
      DESCRIPT="Reserve transport for WEB interface."
      DESCRIPT_ru="Резервный транспорт для WEB интерфейса."
      DESCRIPT_uk="Резервний транспорт для WEB інтерфейсу."
      ADDR="TCP::10004:0"
      PROT="HTTP"
      START="1"/>
  </tbl>
  <tbl id="Transport_out">
    <fld
      ID="testModBus"
      MODULE="Sockets"
      NAME="Test ModBus"
      NAME_ru="Тест ModBus"
      NAME_uk="Тест ModBus"
      DESCRIPT="Data exchange by protocol ModBus test."
      DESCRIPT_ru="Тест обмена по протоколу ModBus."
      DESCRIPT_uk="Тест обміну за протоколом ModBus."
      ADDR="TCP:localhost:10502"
      START="1"/>
  </tbl>-->
</node>

<node id="sub_DAQ">
  <!--
  <tbl id="tmplib">
    <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
      DESCR="" DESCR_ru="" DESCR_uk="" DB="tmplib_test2"/>
  </tbl>
  <tbl id="tmplib_test2">
    <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
      DESCR="" DESCR_ru="" DESCR_uk="" DB="test2"
      PROGRAM="JavaLikeCalc.JavaScript&#010;cnt=5*i"/>
  </tbl>
  <tbl id="tmplib_test2_io">
    <fld TEMPL_ID="test2" ID="i" NAME="I" NAME_ru="I" NAME_uk="I"
      TYPE="4" FLAGS="160" VALUE="" POS="0"/>
    <fld TEMPL_ID="test2" ID="cnt" NAME="Cnt" NAME_ru="Cnt" NAME_uk="Cnt"
      TYPE="4" FLAGS="32" VALUE="" POS="0"/>
  </tbl>-->

  <node id="mod_LogicLev">
    <!--
    <tbl id="DAQ">
      <fld
        ID="test2"
        NAME="Test 2"
        NAME_ru="Тест 2"
        NAME_uk="Тест 2"
        DESCR=""
        DESCR_ru=""
        DESCR_uk=""
        ENABLE="1"
        START="1"
        PRM_BD="test2prm"
        PERIOD="1000"

```

```

        PRIOR="0"/>
    </tbl>
    <tbl id="test2prm">
        <fld SHIFR="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
            DESCR="" DESCR_ru="" DESCR_uk="" EN="1" MODE="2"
            PRM="test2.test2"/>
    </tbl>-->
</node>

<node id="mod_System">
    <!--
    <tbl id="DAQ">
        <fld
            ID="DataOS"
            NAME="Data OS"
            NAME_ru="Данные ОС"
            NAME_uk="Дані ОС"
            DESCR="Data of services and subsystems OS."
            DESCR_ru="Данные сервисов и подсистем ОС."
            DESCR_uk="Дані сервісів та підсистем ОС."
            ENABLE="1"
            START="1"
            AUTO_FILL="0"
            PRM_BD="DataOSprm"
            PERIOD="1000" PRIOR="0"/>
    </tbl>
    <tbl id="DataOSprm">
        <fld SHIFR="CPU" NAME="CPU load" NAME_ru="Нагрузка CPU"
            NAME_uk="Навантаження CPU" DESCR="" DESCR_ru="" DESCR_uk=""
            EN="1" TYPE="CPU" SUBT="gen"/>
        <fld SHIFR="MEM" NAME="Memory" NAME_ru="Память" NAME_uk="Пам\ 'ять"
            DESCR="" DESCR_ru="" DESCR_uk="" EN="1" TYPE="MEM"/>
    </tbl> -->
</node>

<node id="mod_DiamondBoards">
    <!--
    <tbl id="DAQ">
        <fld ID="Athena" NAME="Athena board" NAME_ru="Плата Athena"
            NAME_uk="Плата Athena" DESCR="" DESCR_ru="" DESCR_uk=""
            ENABLE="1" START="0" BOARD="25" PRM_BD_A="AthenaAnPrm"
            PRM_BD_D="AthenaDigPrm" ADDR="640" INT="5" DIO_CFG="0"
            ADMODE="0" ADRANGE="0" ADPOLAR="0" ADGAIN="0"
            ADCONVRATE="1000"/>
    </tbl>
    <tbl id="AthenaAnPrm">
        <fld SHIFR="ai0" NAME="AI 0" NAME_ru="AI 0" NAME_uk="AI 0"
            DESCR="" DESCR_ru="" DESCR_uk=""
            EN="0" TYPE="0" CNL="0" GAIN="0"/>
    </tbl>
    <tbl id="AthenaDigPrm">
        <fld SHIFR="di0" NAME="DI 0" NAME_ru="DI 0" NAME_uk="DI 0"
            DESCR="" DESCR_ru="" DESCR_uk=""
            EN="0" TYPE="0" PORT="0" CNL="0"/>
    </tbl> -->
</node>

<node id="mod_BlockCalc">
    <!--
    <tbl id="DAQ">
        <fld ID="Model" NAME="Model" NAME_ru="Модель" NAME_uk="Модель"
            DESCR="" DESCR_ru="" DESCR_uk="" ENABLE="1" START="1"
            PRM_BD="Model_prm" BLOCK_SH="Model_blcks"
            PERIOD="1000" PRIOR="0" PER_DB="0" ITER="1"/>
    </tbl>
    <tbl id="Model_blcks">
        <fld ID="Klap" NAME="Klapan" NAME_ru="Клапан" NAME_uk="Клапан"
            DESCR="" DESCR_ru="" DESCR_uk=""
            FUNC="DAQ.JavaLikeCalc.lib_techApp.klap" EN="1" PROC="1"/>
    </tbl>
    <tbl id="Model_blcks_io">
        <fld BLK_ID="Klap" ID="l_kl1" TLNK="0" LNK="" VAL="50"/>
        <fld BLK_ID="Klap" ID="l_kl2" TLNK="0" LNK="" VAL="20"/>
    </tbl>
    <tbl id="Model_prm">
        <fld SHIFR="l_kl" NAME="Klap lev" NAME_ru="Полож. клапана"
            NAME_uk="Полож. клапана" DESCR="" DESCR_ru="" DESCR_uk=""
            EN="1" BLK="Klap" IO="l_kl1"/>
    </tbl> -->
</node>

<node id="mod_JavaLikeCalc">

```



```

<!--
<tbl id="DAQ">
    <fld ID="CalcTest" NAME="Calc Test" NAME_ru="Тест вычисл."
        NAME_uk="Тест обчисл." DESCR="" DESCR_ru="" DESCR_uk=""
        ENABLE="1" START="1" PRM_BD="Cal FUNC="TemplFunc.d_alarm"
        PERIOD="1000" PRIOR="0" PER_DB="0" ITER="1"/>
</tbl>
<tbl id="CalcTest_val">
    <fld ID="in" VAL="0"/>
    <fld ID="alarm" VAL=""/>
    <fld ID="alarm_md" VAL="1"/>
    <fld ID="alarm_mess" VAL="Error present."/>
</tbl>
<tbl id="CalcTest_prm">
    <fld SHIFR="alarm" NAME="Alarm" NAME_ru="Авария" NAME_uk="Аварія"
        DESCR="" DESCR_ru="" DESCR_uk="" EN="1" FLD="alarm"/>
</tbl>
<tbl id="lib">
    <fld ID="TemplFunc" NAME="" NAME_ru="" NAME_uk="" DESCR="" DESCR_ru=""
        DESCR_uk="" DB="lib_TemplFunc"/>
</tbl>
<tbl id="lib_TemplFunc">
    <fld ID="d_alarm" NAME="Digit alarm" NAME_ru="Авария по дискр."
        NAME_uk="Аварія за дискр" DESCR=""
        FORMULA="alarm=(in==alarm_md)?&quot;1:&quot;
            +alarm_mess:&quot;0:&quot;;"/>
</tbl>
<tbl id="lib_TemplFunc_io">
    <fld F_ID="d_alarm" ID="in" NAME="Input" NAME_ru="Вход" NAME_uk="Вхід"
        TYPE="3" MODE="0" DEF="" HIDE="0" POS="0"/>
    <fld F_ID="d_alarm" ID="alarm" NAME="Alarm" NAME_ru="Авария"
        NAME_uk="Аварія" TYPE="0" MODE="1" DEF="" HIDE="0" POS="1"/>
    <fld F_ID="d_alarm" ID="alarm_md" NAME="Alarm mode"
        NAME_ru="Режим аварии" NAME_uk="Режим аварії" TYPE="3"
        MODE="0" DEF="" HIDE="0" POS="2"/>
    <fld F_ID="d_alarm" ID="alarm_mess" NAME="Alarm message"
        NAME_ru="Сообщ. аварии" NAME_uk="Повід. аварії" TYPE="0"
        MODE="0" DEF="" HIDE="0" POS="3"/>
</tbl>-->
</node>

<node id="mod_Siemens">
<!--
<tbl id="DAQ">
    <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" ENABLE="1" START="1"
        PRM_BD="test2prm" PERIOD="1000" PRIOR="0" CIF_DEV="0" ADDR="5"
        ASYNC_WR="0"/>
</tbl>
<tbl id="test2prm">
    <fld SHIFR="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" EN="1" TMPL="S7.ai_man"/>
</tbl>-->
</node>

<node id="mod_SNMP">
<!--
<tbl id="DAQ">
    <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" ENABLE="1" START="1"
        PRM_BD="test2prm" PERIOD="1000" PRIOR="0" ADDR="localhost"
        COMM="public" PATTR_LIM="20"/>
</tbl>
<tbl id="test2prm">
    <fld SHIFR="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" EN="1" OID_LS="system"/>
</tbl>-->
</node>

<node id="mod_ModBus">
<!--
<tbl id="DAQ">
    <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" ENABLE="1" START="1"
        PRM_BD="test2prm" PERIOD="1000" PRIOR="0" TRANSP="Sockets"
        ADDR="exlar.diya.org" NODE="1"/>
</tbl>
<tbl id="test2prm">
    <fld SHIFR="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk=""
        EN="1" ATTR_LS="321:0:tst:Test"/>
</tbl>-->

```

```

</node>

<node id="mod_Transporter">
  <!--
  <tbl id="DAQ">
    <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
      DESCR="" DESCR_ru="" DESCR_uk="" ENABLE="1" START="1"
      PRM_BD="test2prm" PERIOD="1000" PRIOR="0" SYNCPER="60"
      STATIONS="loop" CNTRPRM="System.AutoDA"/>
    </tbl>-->
  </node>
</node>

<node id="sub_Archive">
  <prm id="MessBufSize">1000</prm>
  <prm id="MessPeriod">5</prm>
  <prm id="ValPeriod">1000</prm>
  <prm id="ValPriority">10</prm>
  <!--
  <tbl id="Archive_mess_proc">
    <fld
      ID="StatErrors"
      MODUL="FSArch"
      NAME="Errors"
      NAME_ru="Ошибки"
      NAME_uk="Помилки"
      DESCR="Local errors\ ' archive"
      DESCR_ru="Архив локальных ошибок"
      DESCR_uk="Архів локальних помилок"
      START="1"
      CATEG="/DemoStation*"
      LEVEL="4"
      ADDR="ARCHIVES/MESS/stError/"
      FSArchMSize="300"
      FSArchNFiles="10"
      FSArchTmSize="30"
      FSArchXML="1"
      FSArchPackTm="10"
      FSArchTm="60"/>
    </fld>
    <fld
      ID="NetRequests"
      MODUL="FSArch"
      NAME="Net requests"
      NAME_ru="Сетевые запросы"
      NAME_uk="Мережеві запити"
      DESCR="Requests to server through transport Sockets."
      DESCR_ru="Запросы к серверу через транспорт Sockets."
      DESCR_uk="Запити до сервера через транспорт Sockets."
      START="1"
      CATEG="/DemoStation/Transport/Sockets*"
      LEVEL="1"
      ADDR="ARCHIVES/MESS/Net/"
      FSArchMSize="300"
      FSArchNFiles="10"
      FSArchTmSize="30"
      FSArchXML="1"
      FSArchPackTm="10"
      FSArchTm="60"/>
    </tbl>
  <tbl id="Archive_val_proc">
    <fld
      ID="1h"
      MODUL="FSArch"
      NAME="1hour"
      NAME_ru="1час"
      NAME_uk="1год"
      DESCR="Averaging for hour"
      DESCR_ru="Усреднение за час"
      DESCR_uk="Усереднення за годину"
      START="1"
      ADDR="ARCHIVES/VAL/1h/"
      V_PER="360"
      A_PER="60"
      FSArchTmSize="8640"
      FSArchNFiles="10"
      FSArchRound="0.1"
      FSArchPackTm="10"
      FSArchTm="60"/>
    </tbl>
  <tbl id="Archive_val">
    <fld
      ID="test1"

```

```

NAME="Test 1"
NAME_ru="Тест 1"
NAME_uk="Тест 1"
DESCR="Test 1"
DESCR_ru="Тест 1"
DESCR_uk="Тест 1"
START="1"
VTYPE="1"
BPER="1"
BSIZE="200"
BHGRD="1"
BHRES="0"
SrcMode="0"
Source=""
ArchS="" />
</tbl>-->
</node>

<node id="sub_Protocol">
</node>

<node id="sub_UI">
  <node id="mod_QTStarter">
    <prm id="StartMod">QTCfg</prm>
  </node>
  <node id="mod_WebCfg">
    <prm id="SessTimeLife">20</prm>
  </node>
  <node id="mod_VCAEngine">
    <!--
    <tbl id="LIB">
      <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" DB_TBL="wlib_test2" ICO=""
        USER="root" GRP="UI" PERMIT="436"/>
    </tbl>
    <tbl id="wlib_test2">
      <fld ID="test2" ICO="" PARENT="/wlb_originals/wdg_Box" PROC=""
        PROC_ru="" PROC_uk="" PROC_PER="-1" USER="root" GRP="UI"
        PERMIT="436"/>
    </tbl>
    <tbl id="wlib_test2_io">
      <fld IDW="test2" ID="name" IO_VAL="Test 2" IO_VAL_ru="Тест 2"
        IO_VAL_uk="Тест 2" SELF_FLG="" CFG_TMPL="" CFG_TMPL_ru=""
        CFG_TMPL_uk="" CFG_VAL="" />
      <fld IDW="test2" ID="dscr" IO_VAL="Test module 2"
        IO_VAL_ru="Тест модуля 2" IO_VAL_uk="Тест модуля 2"
        SELF_FLG="" CFG_TMPL="" CFG_TMPL_ru="" CFG_TMPL_uk=""
        CFG_VAL="" />
    </tbl>
    <tbl id="PRJ">
      <fld ID="test2" NAME="Test 2" NAME_ru="Тест 2" NAME_uk="Тест 2"
        DESCR="" DESCR_ru="" DESCR_uk="" DB_TBL="prj_test2" ICO=""
        USER="root" GRP="UI" PER </tbl> <tbl id="prj_test2">
      <fld OWNER="/test2" ID="pg1" ICO="" PARENT="/wlb_originals/wdg_Box"
        PROC="" PROC_ru="" PROC_uk="" PROC_PER="-1" USER="root"
        GRP="UI" PERMIT="436" FLGS="1"/>
      <fld OWNER="/test2/pg1" ID="pg2" ICO=""
        PARENT="/wlb_originals/wdg_Box" PROC="" PROC_ru="" PROC_uk=""
        PROC_PER="-1" USER="root" GRP="UI" PERMIT="436" FLGS="0"/>
    </tbl>
    <tbl id="prj_test2_incl">
      <fld IDW="/prj_test2/pg_pg1" ID="wdg1"
        PARENT="/wlb_originals/wdg_Box"/>
    </tbl>-->
  </node>
</node>

<node id="sub_Special">
  <node id="mod_SystemTests">
    <prm id="PARAM" on="0" per="5" name="LogicLev.experiment.F3"/>
    <prm id="XML" on="0" per="10" file="/etc/oscada.xml"/> <prm id="MESS" on="0"
      per="10" categ="" arhtor="DBArch.test3"/>
    <prm id="SOAttDet" on="0" per="20" name="../../lib/openscada/daq_LogicLev.so"
      full="1"/>
    <prm id="Val" on="0" per="1" name="LogicLev.experiment.F3.var" arch_len="5"
      arch_per="1000000"/>
    <prm id="Val" on="0" per="1" name="System.AutoDA.CPULoad.load" arch_len="10"
      arch_per="1000000"/>
    <prm id="BD" on="0" per="10" type="MySQL"
      bd="server.diya.org;roman;123456;oscadaTest"
      table="test" size="1000"/>
    <prm id="BD" on="0" per="10" type="DBF" bd="./DATA/DBF" table="test.dbf"
      size="1000"/>
  </node>
</node>

```

```

        <prm id="BD" on="0" per="10" type="SQLite" bd="./DATA/test.db" table="test"
            size="1000"/>
        <prm id="BD" on="0" per="10" type="FireBird"
            bd="server.diya.org:/var/tmp/test.fdb;roman;123456"
            table="test" size="1000"/>
        <prm id="TrOut" on="0" per="1" addr="TCP:127.0.0.1:10001" type="Sockets"
            req="time"/>
        <prm id="TrOut" on="0" per="1" addr="UDP:127.0.0.1:10001" type="Sockets"
            req="time"/>
        <prm id="TrOut" on="0" per="1" addr="UNIX:./oscada" type="Sockets"
            req="time"/>
        <prm id="TrOut" on="0" per="1" addr="UDP:127.0.0.1:daytime" type="Sockets"
            req="time"/>
        <prm id="Func" on="0" per="10"/> <prm id="SysContrLang" on="0" per="10"
            path="/Archive/FSArch/mess_StatErrors/%2fprm%2fst"/>
        <prm id="ValBuf" on="0" per="5"/> <prm id="Archive" on="0" per="30"
            arch="test1" period="1000000"/>
        <prm id="Base64Code" on="0" per="10"/>
    </node>
</station>
</OpenSCADA>

```

Lets examine in details the structure of the configuration file. A configuration file can contain a configuration of several stations in the sections `<station id="DemoStation"/>`. To attribute set the identifier of the station. Using one or another section of the station at startup is specified by the command-line option `--Station=DemoStation`. Section of the station directly contains parameters of the station and subsystems' sections. Configuration options of the section are written in the form `<prm id="StName">Demo station</prm>`. Where in the attribute `<id>` the ID of the attribute is specified, and in the tag's body the value of parameter "Demo station" is specified. The list of available options and their description for the station and all other sections can be obtained from the console by calling OpenSCADA with parameter `--help` or in the "Help" tabs of the pages of the components of the configuration files of OpenSCADA (Fig.4.10a).

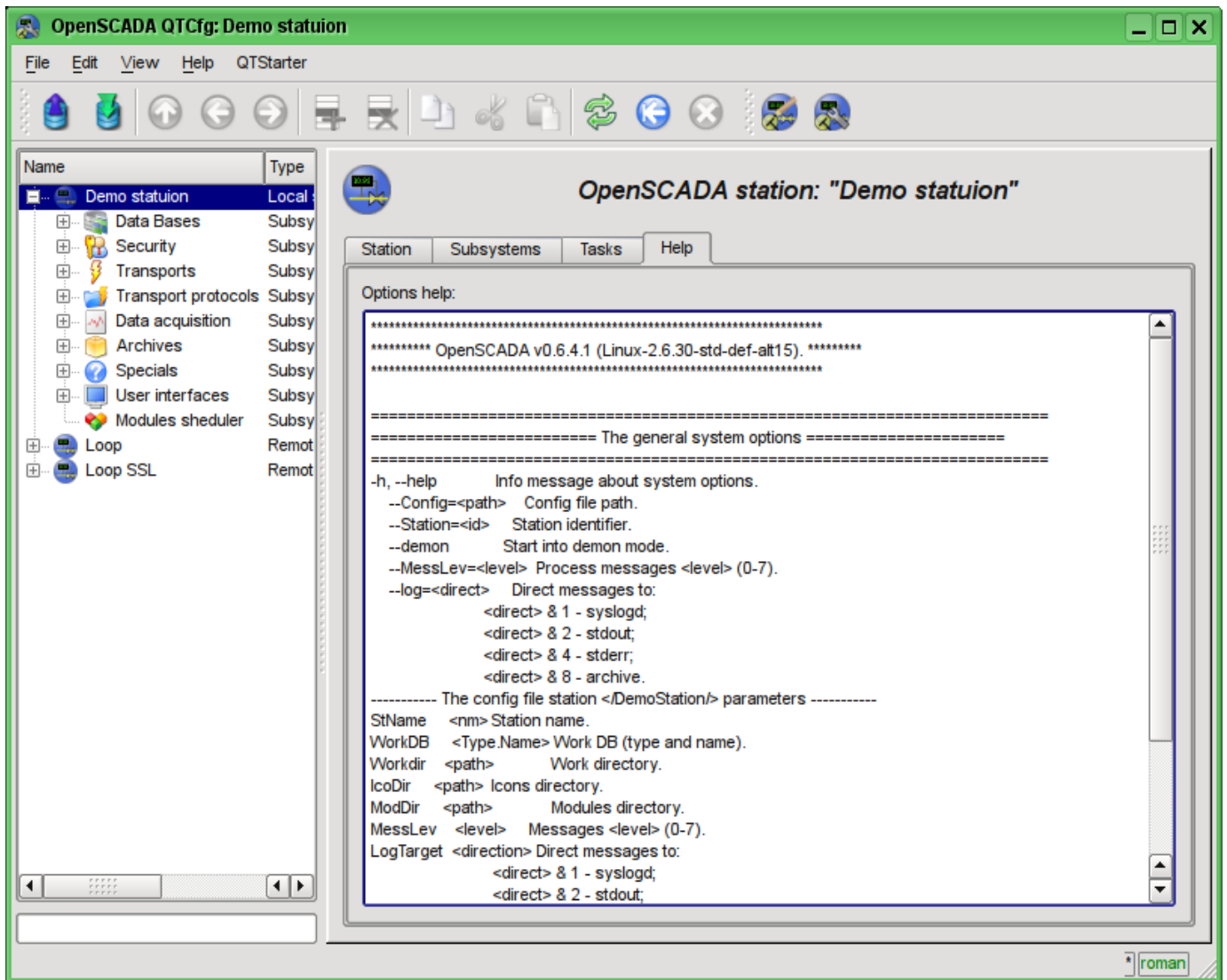


Fig. 4.10a. The "Help" tab of the OpenSCADA component.

The result of the command: # ./openscada_demo --help

```
*****
***** OpenSCADA v0.6.4.1 (Linux-2.6.30-std-def-alt15). *****
*****
=====
===== The general system options =====
=====
-h, --help           Info message about system options.
--Config=<path>      Config file path.
--Station=<id>       Station identifier.
--demon             Start into demon mode.
--MessLev=<level>    Process messages <level> (0-7).
--log=<direct>       Direct messages to:
                    <direct> & 1 - syslogd;
                    <direct> & 2 - stdout;
                    <direct> & 4 - stderr;
                    <direct> & 8 - archive.

----- The config file station </EmptySt/> parameters -----
StName    <nm>       Station name.
WorkDB     <Type.Name> Work DB (type and name).
WorkDir    <path>     Work directory.
IcoDir     <path>     Icons directory.
ModDir     <path>     Modules directory.
MessLev    <level>    Messages <level> (0-7).
LogTarget  <direction> Direct messages to:
                    <direct> & 1 - syslogd;
                    <direct> & 2 - stdout;
                    <direct> & 4 - stderr;
                    <direct> & 8 - archive.

Lang2CodeBase <lang> Base language for variable texts translation, two symbols code.
SaveAtExit <true>   Save system at exit.
SavePeriod <sec>    Save system period.

===== Subsystem "Module sheduler" options =====
--ModPath=<path>     Modules <path> (/var/os/modules/).
----- Parameters of section </DemoStation/sub_ModSched/> in config file -----
ModPath    <path>     Path to shared libraries(modules).
ModAllow   <list>     List of shared libraries allowed for automatic loading, attaching and starting
                    (bd_DBF.so;daq_JavaLikeCalc.so). Use '*' value for allow all modules.
ModDeny    <list>     List of shared libraries deny for automatic loading, attaching and starting
                    (bd_DBF.so;daq_JavaLikeCalc.so).
ChkPer     <sec>      Period of checking at new shared libraries(modules).

===== Subsystem "DB" options =====
----- The config file station </DemoStation/sub_BD/> parameters -----
SYSStPref  <l>       Use station id prefix into generic (SYS) table.

===== Subsystem "Security" options =====

===== Subsystem "Transports" options =====

===== Subsystem "Transport protocols" options =====

===== The module <Protocol:HTTP> options =====
----- Parameters of the module section </DemoStation/sub_Protocol/mod_HTTP/> in config file -----
AuthTime   <min>     Life time of the authentication, minutes (default 10).

===== Subsystem "Data acquisition" options =====
----- Parameters of section </DemoStation/sub_DAQ/> in config file -----
RdStLevel  <lev>     The curent station redundant level.
RdTaskPer  <s>       The redundant task call period.
RdRestConnTm <s>     Restore connection timeout to dead reserve stations.
RdRestDtTm <hour>    Restore data archive depth from a reserve station after deadline.
RdStList   <list>    Redundant stations list, separated symbol ';' (st1;st2).

===== Subsystem "Archives" options =====
----- Parameters of section </DemoStation/sub_Archive/> in config file -----
MessBufSize <items>   Messages buffer size.
MessPeriod  <sec>     Message arhiving period.
ValPeriod   <msec>    Values arhiving period.
ValPriority  <level>   Values task priority level.
MaxReqMess  <items>   Maximum request messages.
MaxReqVals  <items>   Maximum request values.

===== Subsystem "Special" options =====

===== The module <Special:SystemTests> options =====
----- Parameters of the module section </DemoStation/sub_Special/mod_SystemTests/> in config file -----
-----
All tests main options:
    id          test's id;
```

```

on          on test's flag;
per         repeat period (sek).
*** Test's options ***
1) Param    DAQ parameters test. Make read a parameter's attributes and config fields.
  1:name     DAQ parameter address
2) XML      XML file parsing test. Parse and show selected file structure.
  1:file     XML file
3) Mess Messages archive test. Periodic read new messages from archive, for selected archivor.
  1:arhtor   Archivor
  2:categ    Messages category pattern
  3:depth    Messages depth (s)
4) SOAttach Attach/detach module test.
  1:name     Path to module
  2:mode     Mode (1-attach;-1-detach;0-change)
  3:full     Full attach(to start)
5) Val      Parameter attribute's value test.
Periodic make gathering for last value of selected attribute, and also gathering from archive for selected
depth.
  1:name     Parameter attribute path
  2:arch_len Archive value getting depth (s)
  3:arch_per Archive value getting period (us)
6) DB       Full database test. Make:
  - make/open DB;
  - make/open table;
  - make multiply records for determined structure;
  - modify multiply records;
  - get and check values for multiply records;
  - modify record and table structure;
  - remove multiply records;
  - close/remove table;
  - close/remove DB.
  1:type     DB type
  2:addr     DB address
  3:table    DB table
  4:size     Records number
7) TrOut    Output and/or input transports test.
Make test for output transport by send the request to selected input transport.
  1:addr     Address
  2:type     Transport module
  3:req      Request text
8) SysContrLang System control language test.
Make request to language elements by full path set.
Full path to language element have view </Archive/%2fbd%2fm_per>.
Full path contained two included path.
First </d_Archive/> is path to the node of the control tree.
Second </bd/m_per> is path to concrete node's element.
  1:path     Path to language element
9) ValBuf   Value buffer tests.
Contain 13 tests for all aspects of value buffer (subsystem "Archives").
10) Archive Value archive allocation tests.
Contain 7(8) tests for value archivor for check to correct working the consecutive pack mechanism.
  1:arch     Value archive
  2:period   Values period (us)
11) Base64Code Mime Base64 encoding algorithm tests.

===== Subsystem "User interfaces" options =====
===== The module <UI:Vision> options =====
----- Parameters of the module section </DemoStation/sub_UI/mod_Vision/> in config file -----
StartUser   <user>    No password requested start user.
RunPrjs     <list>    Run projects list on the module start.
RunTimeUpdt <mode>    RunTime update mode (0 - all widgets periodic adaptive update, 1 - update only
changed widgets).
VCAstation  <id>      VCA station id ('.' - local).

===== The module <UI:VCAEngine> options =====
--VCADBCclearForce Force clear VCA DB from data of API 1.

===== The module <UI:QTCfg> options =====
----- Parameters of the module section </DemoStation/sub_UI/mod_QTCfg/> in config file -----
StartPath   <path>    Configurator start path.
StartUser   <user>    No password requested start user.

===== The module <UI:QTStarter> options =====
----- Parameters of the module section </DemoStation/sub_UI/mod_QTStarter/> in config file -----
StartMod    <moduls>  Start modules list (sep - ';').

===== The module <UI:WebVision> options =====
----- Parameters of the module section </DemoStation/sub_UI/mod_WebVision/> in config file -----
SessTimeLife <time>   Time of the session life, minutes (default 10).

```


Sections of subsystem (`<node id="sub_DAQ" />`) contains parameters of subsystem, sections of modules and sections of tables of reflections of the data of databases in the configuration file. Sections of modules (`<node id="mod_DiamondBoards" />`) contain the individual parameters of modules and sections of tables of reflection of the data of databases in the configuration file.

Sections of the tables of reflection of the data of databases are provided for placement in the configuration file records of DB tables for the OpenSCADA components. Lets examine the table of incoming transports "Transport_in" of subsystem transports (`<node id="sub_Transport">`) from the example of configuration file above. The table contains two records with fields: ID, MODULE, NAME, DESCRPT, ADDR, PROT, START. After booting with this section and in general without the DB in the subsystem "Transports" of the "Sockets" module you'll see two input transports. Formats of the table's structures of the main components are included in the demo configuration files. For the details of the database's structure you should read the relevant documentation of modules.

5. System-wide API of user programming.

User programming API is the tree of OpenSCADA objects, every object of which can provide own list of properties and functions. Properties and functions of objects can be used by the user in procedures on the languages of user programming of OpenSCADA. The entry point for access to the objects of system OpenSCADA from user programming language JavaLikeCalc is the reserved word "SYS" of the root OpenSCADA object. For example, to access the function of outgoing transport you should write: **SYS.Transport.Serial.out_ModBus.messIO(mess);**.

API of the objects provided by the modules is described in the own documentation of the module.

5.1. System-wide user objects.

Abstract object is an associative container of properties and functions. Properties can contain the data of four basic types and other objects. Access to the properties of an object is usually made by recording the names of properties through a point to the object `<obj.prop>`, as well as by entering the property name in brackets `<obj["prop"]>`. It is obvious that the first mechanism is static, while the second lets you to specify the name of the property through a variable. The basic definition of the object does not contain functions. Copying of an object actually makes reference to the original object. When you delete an object the reduce of the reference counter is made, and when the reference counter is equal to the zero object is removed physically.

Different components can redefine the basic object with special properties and functions. The standard extension of the object is an array "Array".

Array object

Peculiarity of the array is that it works with the properties like with the indexes, and complete their naming if senseless, and hence the mechanism of addressing is available only by the conclusion of the index in square brackets `<arr[l]>`. Array stores the properties in its own container of one-dimensional array.

Array provides the special property "length" to get the array size `<var = arr.length;>`. Also array provides the following functions:

- `string join(string sep = " "), string toString(string sep = " "), string valueOf(string sep = " ")` - Returns the string with the array elements separated by `<sep>` or the character ','.
- `Array concat(Array arr);` - Adds to the initial array the elements of the `<arr>` array. Returns the initial array with changes.
- `int push(ELTp var, ...);` - Places the element(s) `<var>` to the end of the array, as to the stack. Returns the new array size.
- `ELTp pop();` - Deleting of the last element of the array and return of its value, as from the stack.
- `Array reverse();` - Changing the order of the elements of the array. Returns the initial array with changes.
- `ELTp shift();` - The shift of the array to the top. The first element is removed and its value is

returned.

- *int unshift(ElTp var, ...);* - Shift element(s) *<var>* to the array. The first element to the 0, second to the 1 and so on.
- *Array slice(int beg, int end);* - Returns an array fragment from *<beg>* to *<end>*. If the value of beginning or end is negative, then the count is made from the end of the array. If the end is not specified, then the end is the end of the array.
- *Array splice(int beg, int remN, ElTp val1, ElTp val2, ...);* - Inserts, deletes or replaces the elements of the array. Returns the initial array with the changes. Firstly it is made the removing of elements from the position *<beg>* and in the quantity of *<remN>*, and then the values *<val1>* are inserted and so on, beginning from the position *<beg>*.
- *Array sort();* - Sort array elements in lexicographical order.

XMLNodeObj object

Functions:

- *string name()* - The name of the node, XML-tag.
- *string text()* - The text of the node, contents of the XML-tag.
- *string attr(string id)* - The value of the node's attribute *<id>*.
- *XMLNodeObj setName(string vl)* - Setting of the node's name to *<vl>*. Returns the current node.
- *XMLNodeObj setText(string vl)* - Setting of the node's text to *<vl>*. Returns the current node.
- *XMLNodeObj setAttr(string id, string vl)* - Setting the attribute *<id>* to the value *<vl>*. Returns the current node.
- *int childSize()* - Quantity of the embedded nodes.
- *XMLNodeObj childAdd(ElTp no = XMLNodeObj)* - Addition of the object *<no>* as the embedded one. *<no>* may be the direct object-result of the function *xmlNode()*, and the string with the name of the new tag. Returns the embedded node.
- *XMLNodeObj childIns(int id, ElTp no = XMLNodeObj)* - Insert of the object *<no>* as the embedded one to the position *<id>*. *<no>* may be the direct object-result of the function *xmlNode()*, and the string with the name of the new tag. Returns the embedded node.
- *XMLNodeObj childDel(int id)* - Deleting the embedded node from the position *<id>*. Returns the current node.
- *XMLNodeObj childGet(int id)* - Getting the embedded node in the position *<id>*.
- *string load(string str, bool file = false)* - Loading the XML from the string *<str>* or from the file with the path in *<str>* if the *<file>* "true".
- *string save(int opt = 0, string path = "")* - Saving the XML tree to the string or to the file *<path>* with the formatting parameter *<opt>*. Returns the XML text or the error code. The following formatting options *<opt>* are provided:
 - 0x01 - interrupt the string before the opening tag;
 - 0x02 - interrupt the string after the opening tag;
 - 0x04 - interrupt the string after a closing tag;
 - 0x08 - interrupt the string after the text;
 - 0x10 - interrupt the string after the instruction;
 - 0x1E - interrupt the string after all.

5.2. System (SYS)

Object functions:

- *string system(string cmd, bool noPipe = false);* - calls the console commands *<cmd>* of OS returning the result by the channel. If *<noPipe>* is set the return code is returned the the execution of the programs in the background ("sleep 5 &") is possible. The function offers great opportunities to the OpenSCADA user by calling any system software, utilities and scripts, as well as by way of access to the huge volume of system data. For example the command "ls-l" returns the detailed contents of the working directory.
- *int message(string cat, int level, string mess);* - formation of the system message *<mess>* with the category *<cat>*, level *<level>*. The negative value of the level forms the alarms (Alarm).
- *int messDebug(string cat, string mess); int messInfo(string cat, string mess); int*

messNote(string cat, string mess); int messWarning(string cat, string mess); int messErr(string cat, string mess); int messCrit(string cat, string mess); int messAlert(string cat, string mess); int messEmerg(string cat, string mess); -formation of the system message *<mess>* with the category *<cat>* and the appropriate level.

- *XMLNodeObj XMLNode(string name = "");* - creation of the XML node object with the name *<name>*.
- *string cntrReq(XMLNodeObj req, string stat = "");* - request of the control interface to the system via XML. The usual request is written as *<get path="/OPath/%2felem"/>*. If the station is indicated to the request to the external station is made.
- *int time(int usec);* - returns the absolute time in seconds from the epoch of 1/1/1970 and in microseconds, if *<usec>* is specified.
- *int localtime(int fullsec, int sec, int min, int hour, int mday, int month, int year, int wday, int yday, int isdst);* - returns the full date in seconds (sec), minutes (min), hours (hour), days of the month (mday), month (month), year (year), days in the week (wday), days in the year (yday) and sign of summer time (isdst), based on the absolute time in seconds *<fullsec>* from the epoch 1.1.1970.
- *string strftime(int sec, string form = "%Y-%m-%d %H:%M:%S");* - Converts an absolute time *<sec>* to the string of the desired format *<form>*. Record of the format corresponds to the POSIX-function *strftime*.
- *int strptime(int str, string form = "%Y-%m-%d %H:%M:%S");* - Returns the time in seconds from the epoch of 1/1/1970, based on the string record of time *<str>*, in accordance with the specified template *<form>*. For example the template "%Y-%m-%d %H:%M:%S" corresponds with the time "2006-08-08 11:21:55". Description of the template's format can be obtained from the documentation on POSIX-function "strptime".
- *int cron(string cronreq, int base = 0);* - returns the time, planned in the format of the standard Cron *<cronreq>*, beginning from basic time *<base>* or from the current, if the basic is not specified.
- *string strFromCharCode(int char1, int char2, int char3, ...);* - String creation from symbol's codes char1, char2 ... charN.

5.3. Any object of OpenSCADA objects tree (SYS.*)

Object functions:

- *TArrayObj nodeList(string grp = "", string path = "");* - Get child nodes list for group *<grp>* and node from path *<path>*. If *<grp>* empty then return nodes for all groups.
- *TCntrNodeObj nodeAt(string path, string sep="");* - Attach to node *<path>* into OpenSCADA objects tree. If a separator set into *<sep>* then path process as separated string.

5.4. "DB" subsystem (SYS.BD)

DB object functions (SYS.BD["TypeDB"]["DB"]):

- *Array SQLReq(string req);* - Formation of the SQL-request to the DB.

Example:

```
DBTbl=SYS.BD.MySQL.GenDB.SQLReq("SELECT * from DB;");
for( var i_rw = 0; i_rw < DBTbl.length; i_rw++ )
{
    var rec = "";
    for( var i_fld = 0; i_fld < DBTbl[i_rw].length; i_fld++ )
        rec += DBTbl[i_rw][i_fld]+"\\t";
    SYS.messDebug("TEST DB","Row "+i_rw+": "+rec);
}
```

5.5. Subsystem "DAQ" (SYS.DAQ)

Functions of object of attribute of controller's parameter (SYS.DAQ["Modul"]["Controller"] ["Parameter"] ["Attribute"]):

- *ElTp get(int tm = 0, int utm = 0, bool sys = false);* - get attribute value at time *<tm:utm>* and system access flag *<sys>*.
- *bool set(ElTp val, int tm = 0, int utm = 0, bool sys = false);* - write value *<val>* to attribute with time label *<tm:utm>* and system access flag *<sys>*.

5.6. "Archives" subsystem (SYS.Archive)

Functions of the subsystem's object:

- *Area messGet(int btm, int etm, string cat = "", int lev = 0, string arch = "");* - request of the system messages for the time from *<btm>* to *<etm>* for the category *<cat>*, level *<lev>* and archiver *<arch>*.

5.7. "Transports" subsystem (SYS.Transport)

Functions of the outgoing transport object (SYS.Transport["Modul"] ["OutTransp"]):

- *string messIO(string mess, real timeOut = 1000);* - sending the message *<mess>* through the transport with the waiting timeout *<timeOut>*.
- *int messIO(XMLNodeObj req, string prt);* - sending the request *<req>* to the protocol *<prt>* for the implementation of a connection session through the transport by means of protocol.

Data acquisition in OpenSCADA

Data acquisition of the SCADA (Supervisory Control and Data Acquisition)-system is its integral part, which get data from sources of different type. The nature of data, which operates SCADA, is characterized by signals of basic value's types (integer, real, boolean and string). The signals vary over time and has their history, life. In the theory of technological processes (TP) under the signal it is meant the value of TP sensor in the ADC code, "raw" signal or in the real value. Signals can be combined into groups, which are often called parameters. For example, the developed data sources can provide the structures of parameters with the predefined set of related signals. In addition to the direct data acquisition in the function of this mechanism is also included the transfer of actions to control devices of TP; usually it is a gate valve, pumps and control valves. Taken together, this process is known as computer-process interface (CPI).

Sources of data are characterized by their great variety, which can be divided into three groups.

- Sources of "raw" data, providing the ADC code or levels of discrete signals, and also the sources which include simple processing. Usually, it is the modules of the allocated CPI or the simplest industrial programmable logic controllers (PLCs).
- Powerful industrial PLCs, which have significant computing power and the possibility of formation of complex parameters with different structure.
- Local or related data sources. For example, the CPI as expansion cards, and also the data of the hardware and software environment in which the system operates.

The variety of data sources has created a wide range of mechanisms to access them. Local data sources are different in application programming interface (API), and network sources, in their turn, in transport and protocol interaction level. In general, this has led to the fact that the addition of support for a new data source requires the creation of interface module or driver. Taking into account the great variety of sources, it is extremely expensive and actually impossible to cover the entire spectrum of the market of these devices. The situation is somewhat simplified with the network source due to the presence of the number of standard and free interaction protocols, but many sources still use their own protocols: private, commercial or protocols, tied to private mechanisms of the limited range of commercial operating systems (OS).

In terms of OpenSCADA system the following objects to serve the data acquisition mechanism are provided:

- Attribute - object of reflection of the signal data, it includes the current value with the type of signal and the history of changes of value;
- Parameter - object of the attributes' (signals') group with the structure corresponding to the characteristics of the separate data source;
- Controller - object of the separate data device. Typically, this is a separate CPI module or the devices of industrial PLC.

To account the features of different data acquisition devices, as well as the different mechanisms of interaction in the OpenSCADA the modular subsystem "Data acquisition" is provided. The module of the subsystem is the driver for interfacing with a data source of specific type. Each module can contain a configuration of several devices of this type in the form of "Controller" objects of OpenSCADA. The general scheme of objects of "Data acquisition" subsystem is shown in Figure 1.

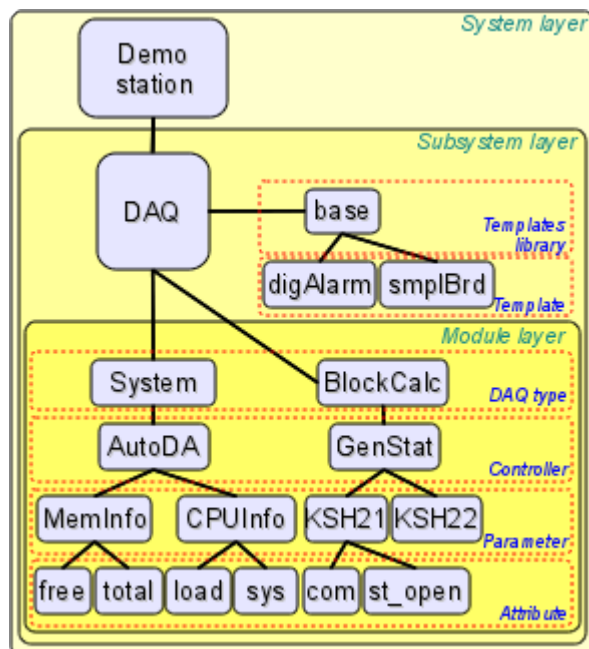


Fig. 1. The subsystem's "Data acquisition" scheme.

1. Data acquisition methods

Taking into account variety of the data sources, and also the ways of their possible interaction data acquisition methods can be divided to simple synchronous, simple asynchronous, package and passive ones.

To the examination of the mechanisms below the following objects will be involved:

- ObjectSCADA - any object of the SCADA-system, applying for the signal value, for example, archives and visualizers;
- DAQParamAttribute - attribute of the parameter of subsystem "Data acquisition" which is an intermediary for access to the value of the signal of data source;
- DAQParamAttributeArch - attribute's archive object;
- HardwarePLC - data source object, for example, modules of the allocated CPI or industrial PLC.

1.1. Simple synchronous acquisition mechanism

The mechanism is characterized by requests to the data source synchronously with the request to the attribute of parameter (Fig. 2). This mechanism is usually used when working with local sources of data, characterized by low latency, ie delay in response to the request. With this method you can get actual data directly with the request, but the time of the request of object will include the time for transportation and processing of the request by the data source.

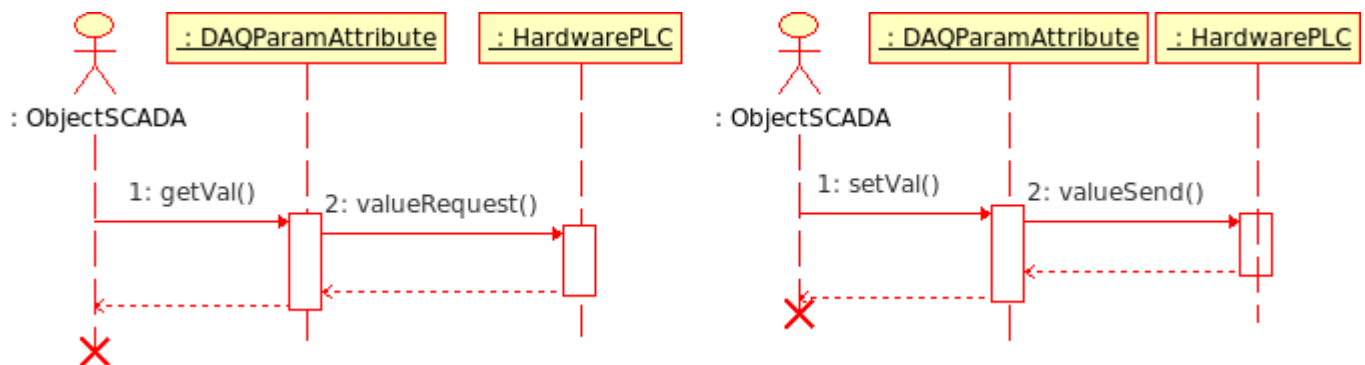


Fig. 2. Diagram of the sequence of interaction with the synchronous requests.

In accordance with the diagram above, we obtain the following sequence of requests for data acquisition and their transfer:

- object of the SCADA-system sends the value request to the object of attribute of the parameter `DAQParamAttribute::getVal()`;
- object of the attribute of parameter, receiving the request, sends it to the data source `HardwarePLC::valueRequest()`;
- source of data after processing the request returns the result;
- object of the attribute of parameter, receiving the result, returns its to the SCADA-system object.

In OpenSCADA this mechanism is implemented by the following modules of subsystem "Data acquisition".

- [*ModBus*](#) - module of access to data of the sources through the family of ModBus protocols. In the module the synchronous mode for recording data is implemented.
- [*DiamondBoards*](#) - module of the data access to the PC/104 card of Diamond Systems company. PC/104 boards are available on the ISA-bus, hence are local and available relatively quickly. When data acquisition is made not by interruption the access to the values of the ADC is synchronous. Recording mode of the DAC values always works synchronously.
- [*DAQGate*](#) - module of the reflection of the controller's objects of the remote OpenSCADA-stations on the local one. In the module the synchronous mode for recording data is implemented.
- [*BlockCalc*](#) - calculator in the language of block diagrams. The source of data for it is the custom block diagram. Attributes of parameters of the module synchronously address the inputs/outputs of the blocks of block scheme.
- [*JavaLikeCalc*](#) - calculator on the Java-like high level language. The source of data it supports is the user program on the Java-like language. Attributes of the parameter of module synchronously

address the inputs/outputs of the user computing function.

- [LogicLev](#) - module of the logic-level parameters of data acquisition, see more about it in section 2. The source of data for this module are the other parameters of subsystem "Data acquisition" and the execution context of the parameters' template. Attributes of the parameters of module synchronously address the attributes of other parameters in the reflective mode of parameters of subsystem "Data acquisition", or the inputs/outputs of the execution context of the template when work under the template.

1.2. Simple asynchronous acquisition mechanism

The mechanism is characterized by requests to the data source, regardless of the request to the attribute of parameter (Fig. 3). Usually, requests to the source of the data are made periodically in the own inquiry task of the single controller and with the blocks of few signals. This request to the parameter's attribute returns the value obtained from the last connection session with the data source. This mechanism is usually used when working with remote (network) data sources, characterized by high latency, ie delay in the response to the request.

With this method it is possible to optimize the time resource spent on one signal, and thereby increase the maximum number of requested signals during the time interval of the inquiry.

As an example, lets examine an industrial PLC Siemens S7-315 during requesting him on the bus Profibus (1,5 Mbit/s). The average processing time of the MPI-request of this controller is 30 ms. If you use a synchronous mechanism for each signal, ie one request for each signal, then in one second we can get something about 33 signals. And if you apply an asynchronous mechanism, ie in the MPI-package to receive up to 220 bytes or 110 signals of integer type of 16-bit, then we can for one second get up to the 3630 signals. As you can see, the effectiveness of asynchronous mechanism in this case is 110 times, namely, the maximum capacity of MPI-package.

The disadvantage of asynchronous mechanism is that the request of the value of attribute of the parameter returns not actual at the time of request value, but value of the last session of the inquiry of the controller. However, taking into account that the source of data can be updated at intervals of ADC hardware limitations, and the sensors themselves may have certain restrictions on the reaction rate, the using of an asynchronous acquisition mechanism could have a serious grounds.

Application of asynchronous mechanism for recording the values to the PLC is a fairly rare fact, because recording of values usually involves impact of the operator on the TP. Operator on the fact rarely makes adjustments to the process, therefore, the recording can be performed synchronously. However, there are situations, such as managing of the TP by the regulator on SCADA-system, acting as a runtime of PLC.

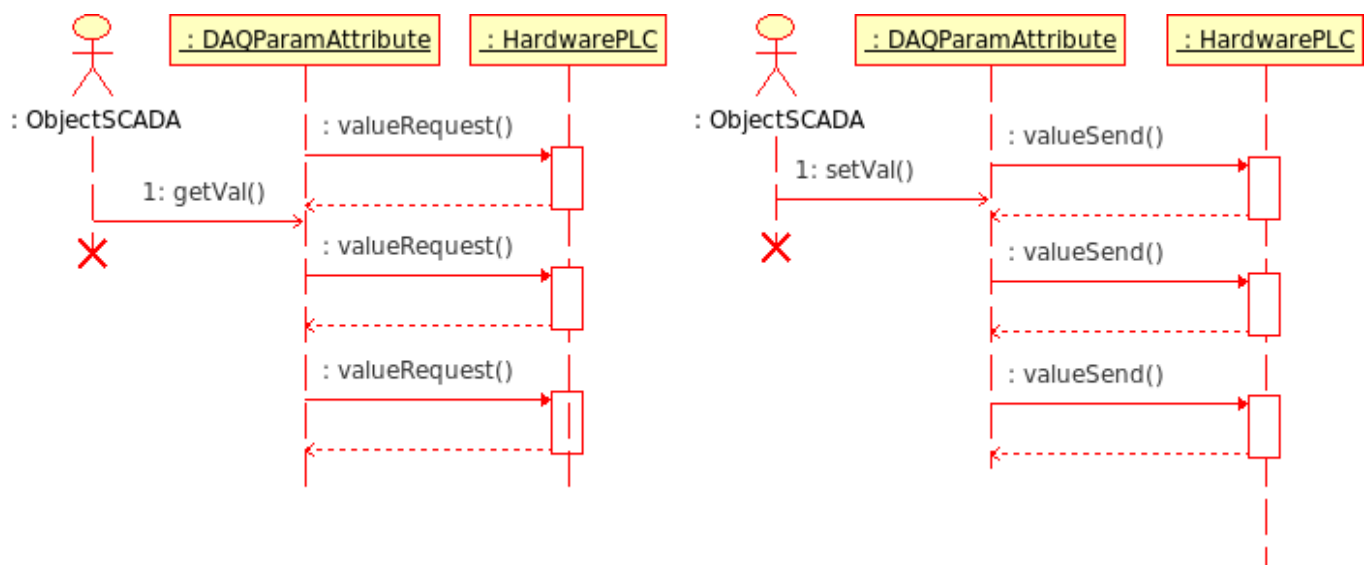


Fig. 3. Diagram of interaction sequence with asynchronous requests.

In accordance with the diagram above, we obtain the following picture:

- object of the attribute of parameter (or the parent object of the controller) performs the periodic

requests `HardwarePLC::valueRequest()` to get the value of a signal or group of signals;

- received signal values stored in the objects of parameter's attributes locally;
- an object of SCADA-system sends the value request to the object of parameter's attribute `DAQParamAttribute::getVal()` and gets locally saved value of the previous session of the inquiry of data source.

In OpenSCADA this mechanism is implemented by the following modules of subsystem "Data acquisition".

- [*Siemens*](#) - module of access to the data of Siemens controllers of S7 series. In this module an asynchronous mode is implemented as for reading data and for recording (optional) to the PLC.
- [*ModBus*](#) - module of access to data sources through the family of ModBus protocols. In the module an asynchronous mode of reading data is implemented.
- [*SNMP*](#) - module of access to the data of the network devices through the Simple Network Management Protocol. In the module an asynchronous mode of reading data is implemented.
- [*System*](#) - module of access to the data of the execution area of OpenSCADA. In the module an asynchronous mode of reading data is implemented.
- [*DAQGate*](#) - module of the reflection of controller's objects of the remote OpenSCADA-stations on the local one. In the module an asynchronous mode of reading data is implemented.

1.3. Package acquisition mechanism

Package data acquisition mechanism is characterized by the acquisition of data for each signal by the packet that includes the history of its changes. Ie per one session of data inquiry we obtain multiple values of history of the signal. Package mechanism works in conjunction with synchronous and asynchronous mechanisms.

In the case of working with the synchronous mechanism the actual transfer of the archive of data source for operational work in the system is done (Fig. 2). As the simple synchronous mechanism, it is desirable to apply only to low-latency data sources or to the sources whose work is a session type, for example, in the commercial account to read the values of the counters.

When working in conjunction with an asynchronous mechanism the history of the received signals is usually placed directly in the archives (Fig. 4), and the current value of the parameter's attribute is set to last value of the package. This combination is effective during the acquisition of the fast data or during the synchronization of the archives after the loss of connection to the remote data source.

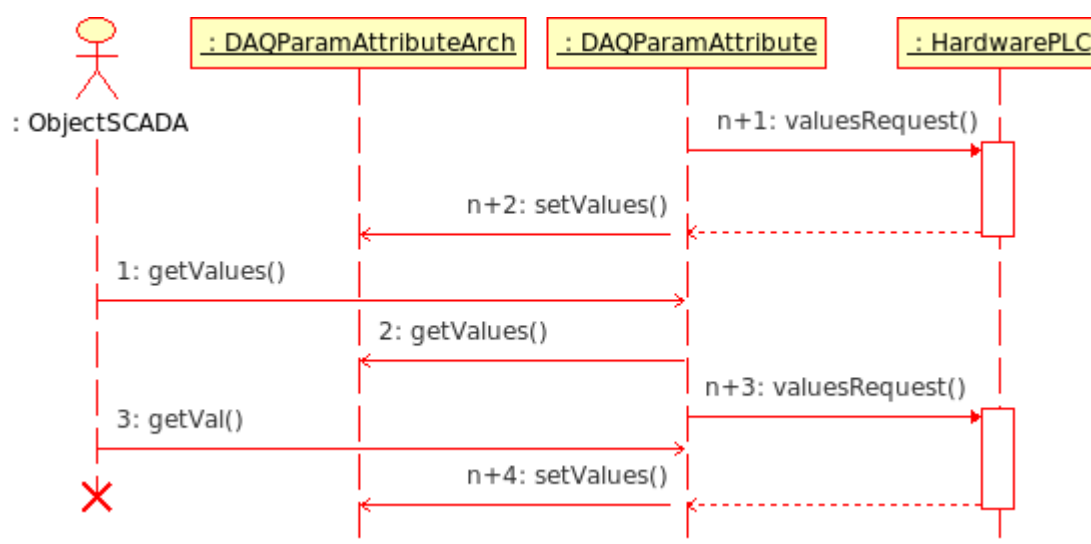


Fig. 4. Diagram of interaction sequence with the asynchronous requests of the package mechanism.

In accordance with the diagram above, we obtain the following behavior of the package mechanism for asynchronous requests:

- object of the attribute of parameter (or the parent object of the controller) performs the periodic requests `HardwarePLC::valueRequest()` to get the value's packages of a signal or group of signals;
- received value's packages of signal are placed in the archive by the request

DAQParamAttributeArch::setValues(), and the last value of the packages is located in the objects of parameters' attributes;

- object of SCADA-system sends the request of the archive's fragment to the object of parameter's attribute DAQParamAttribute::getValues(), and he relays the request to the archive DAQParamAttributeArch::getValues(). As the result the fragment of the archive, available after the previous session of the inquiry of data source, is returned;
- object of the SCADA-system sends the request of the last value of the object of parameter's attribute DAQParamAttribute::getVal() and gets the locally saved value of the previous session of the inquiry of data source.

In OpenSCADA this mechanism is implemented by the following modules of subsystem "Data acquisition".

- [*DiamondBoards*](#) - module for data access of PC/104 cards of Diamond Systems company. PC/104 cards are available on the ISA-bus, hence, are local and available relatively quickly. When data acquisition is done through interruption the expectation of the packets of fast (up to 200 kHz) in one second (up to 200,000 values in the package) is made and the subsequent placing of packets data in the archives of the DAQ parameters' attributes.
- [*DAQGate*](#) - module of reflection of controller's objects of remote OpenSCADA-stations on the local one. The synchronous and asynchronous packet mode of reflection of the archives of remote OpenSCADA-stations is provided.

1.4. Passive acquisition mechanism

The feature of the passive data acquisition mechanism is the initiative of the providing data in the SCADA-system from the data source. This mechanism is quite rare, but can occur in certain conditions or restrictions of the possibility of using the direct data acquisition mechanisms, Fig. 5. An example of such a situation can be the geographically allocated systems of data acquisition through mobile networks GPRS/EDGE. In such networks, empowering the individual client nodes with the real IP-address or the formation of a corporate wireless network can be rather expensive, and therefore more accessible is an initiative of the data transfer session from client dynamic IP-addresses to the one real IP-address of the SCADA-system server. Nevertheless it is possible to work through the network DBMS of the dealer.

Impacts of the modification are transmitted to the source of data at the time of data transfer session by the source.

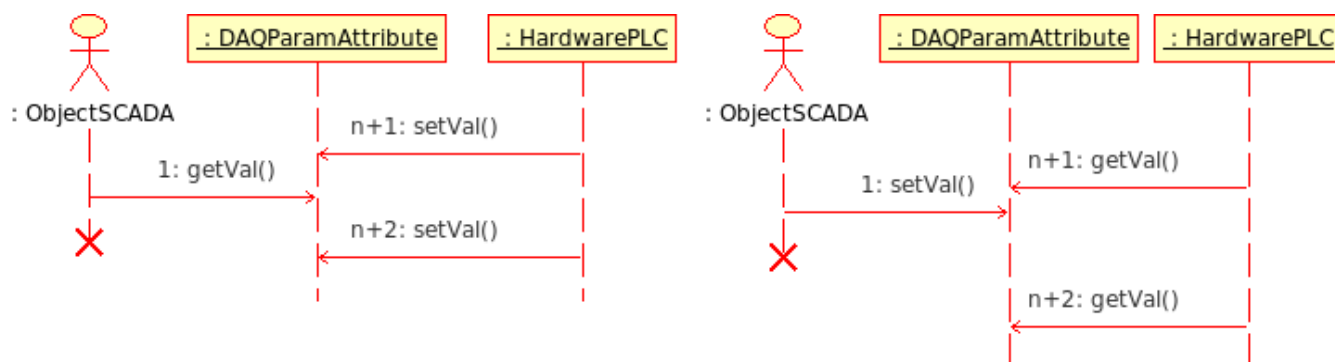


Fig. 5. Diagram of interaction sequence with the passive working mode.

In accordance with the diagram above, we obtain the following behavior of the passive mechanism:

- data source object carries out periodic connection sessions with the object of the parameter's attribute DAQParamAttributeArch::setVal() to transfer its own data and receive influence commands;
- object of the SCADA-system sends the request to the last value of the object of parameter's attribute DAQParamAttribute::getVal() and gets the locally stored value of the previous connection session of the data source.

In OpenSCADA this mechanism has not been yet used, but in principle there is the possibility of its realization in the system.

2. Virtual data sources

In addition to physical data acquisition the function of the virtual data acquisition is also important. Virtual data are the data obtained inside the system both independently and on the basis of physical data. Practically the formation mechanisms of virtual data are implemented in conjunction with the mechanism of user computing. Among the industrial controllers and SCADA-systems the different programming languages are used. In the case of controllers such languages can be for example low-level languages (assemblers), but in recent years the high-level languages (C, Pascal and others) are increasingly used, as well as the formal languages of IEC 61131-3 (sequential function chart SFC, function block diagrams FBD, LD relay circuits and text ST, IL). In the case of SCADA-systems computations are often provided with the help of high-level programming languages and formal languages.

In the OpenSCADA system the programming interfaces and virtual data sources on the basis of different languages in separate modules of a subsystem "Data acquisition" can be implemented. At the time of version 0.6.3.2 the available modules of virtual calculators are:

- Calculator on Java-like language: [JavaLikeCalc](#);
- Block calculator: [BlockCalc](#).

At the OpenSCADA kernel the mechanism for user-defined functions or API of user programming is integrated. User functions can be provided by any object of the system, including modules in accordance with their functionality, thus providing the user with the set of functions for the control of one or another object. User API functions can be either static, ie implementing the fixed functionality of an individual object, and the dynamic ones, ie formed by the user for the desired task in the language of the user high-level programming.

Module [JavaLikeCalc](#) provides the system with the mechanism to create dynamic user-defined functions and libraries for Java-like language. Description of functions for Java-like language is to tie up the parameters of the function by the algorithm. In addition, the module has the functions of the direct calculations by creating a computer controllers with the associated computational function. Module provides the mechanism to precompile the context-dependent functions that are used to embed the user algorithms directly in the context of the various components of OpenSCADA. For example, the mechanism of the parameters' templates of subsystem "Data acquisition" and the visual control engine (VCA).

Module [BlockCalc](#) provides the OpenSCADA system with the mechanism for creating user calculations. Mechanism of calculations based on the formal language of block diagrams (functional blocks). Languages of block programming based on the concept of block diagrams (functional blocks). And depending on the nature of the block, block scheme can be: logic circuits, relay logic circuits, a model of technological process and others. The essence of the block scheme is that it contains the list of blocks and links between them. From a formal point of view the block - is an element (function), which has inputs, outputs and an algorithm for computing. Based on the concept of programming area block - is a frame of values associated with the object of function. Inputs and outputs of blocks are to be connected to get the whole block scheme.

With the purpose of filling user programming API with user functions the following specialized modules of static user programming API functions are created:

- Library of function for the compatibility with SCADA Complex1: [FLibComplex1](#);
- Library of standard mathematical functions: [FLibMath](#);
- Library of System API functions: [FLibSYS](#).

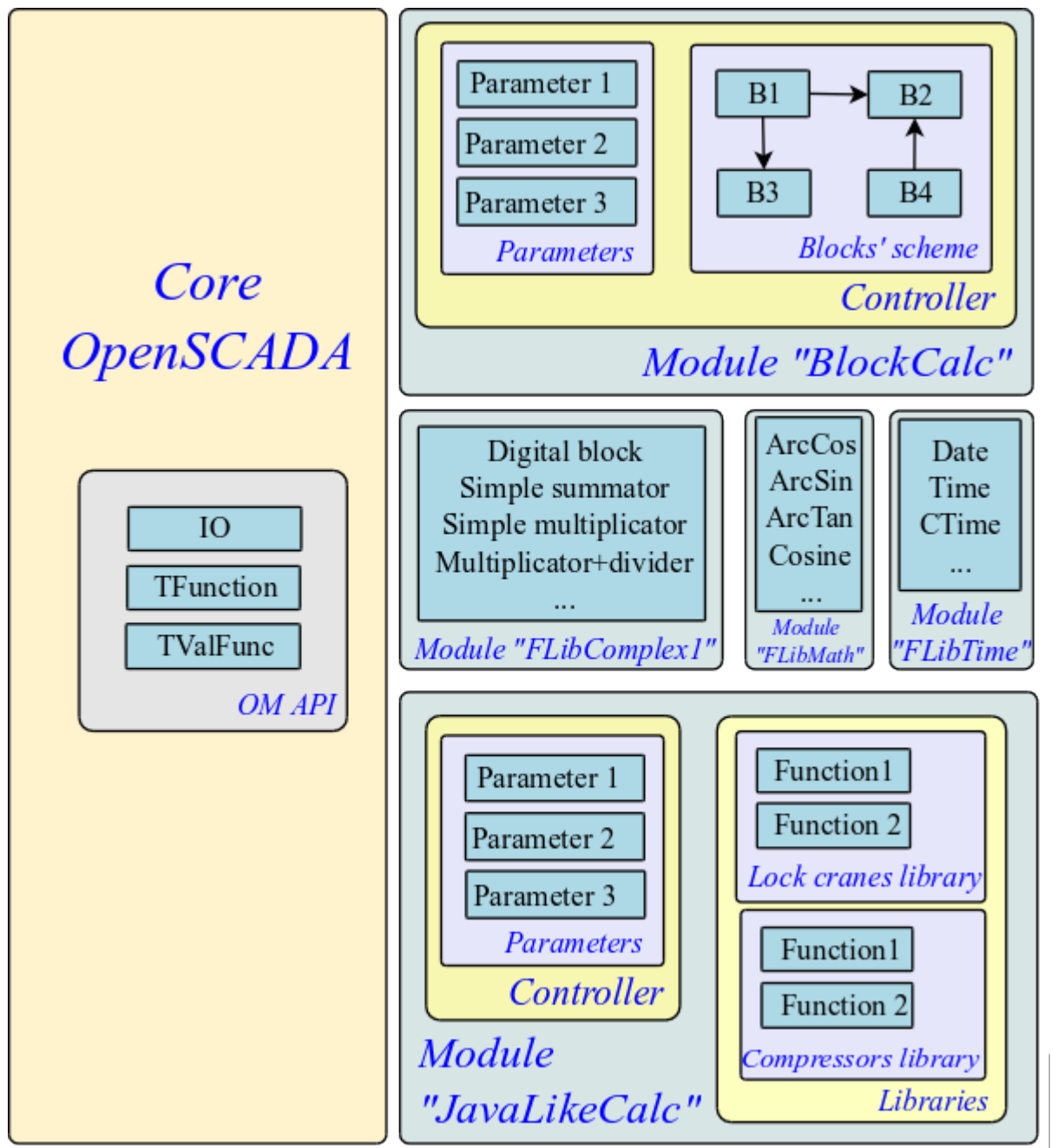


Fig. 6. The overall structure of the components of the programming area

3. Logic level of data processing

Above we talked that type of data source can vary from a "raw" to the complex. The "raw" means the source that provides only the basic signal (integer, real, boolean, string, ...) separately. Under the complex it is meant the source that groups the signals and in the parameter of subsystem "Data acquisition" it provides the attributes of an additional purpose, covering practically all diagnostic tasks, ie the parameter is the complete object, which do not need any additions.

Taking into account this variation, the situation may occur, when the information in the object of data source controller's parameter, is insufficient to describe the real TP object in general and the derived object of a higher level of abstraction is needed. The solution of this situation is the formation of complementary parameters, which is not obvious and confusing. The better solution is to use layer, so-called "Logic level", serving for the flexible formation of parameters, containers of signals with the necessary structure, and which has post-processing.

Functionally "Logic level" is intended to provide the OpenSCADA system with mechanism of free formation of parameters' objects, containers of signals of the necessary structure.

Operating appointment of the "Logic level" is:

- expansion of the scope of the OpenSCADA system by increasing the flexibility of description of parameter's objects of subsystem "Data acquisition";
- reduction of labor costs for the creation of complex automated systems.

The conception of "Logic level" based on the parameters' templates for which in the subsystem "Data acquisition" it is provided the container of the templates libraries (Fig. 1). Each library contains templates of parameters that can be used by the modules of "Data acquisition" subsystem for the implementation of parameters based on templates. The modules of OpenSCADA, which use the templates in their work, are:

- [LogicLev](#) - module of the implementation of the classical conception of "Logic level".
- [Siemens](#) - data acquisition module for Siemens controllers Series S7. Taking into account the high flexibility and functionality of this controllers, which allows you to create complex data types of different structure, all the parameters of this module work on templates.

General mechanism of the "Logic level" on the example of the [LogicLev](#) module is shown in Fig. 7.

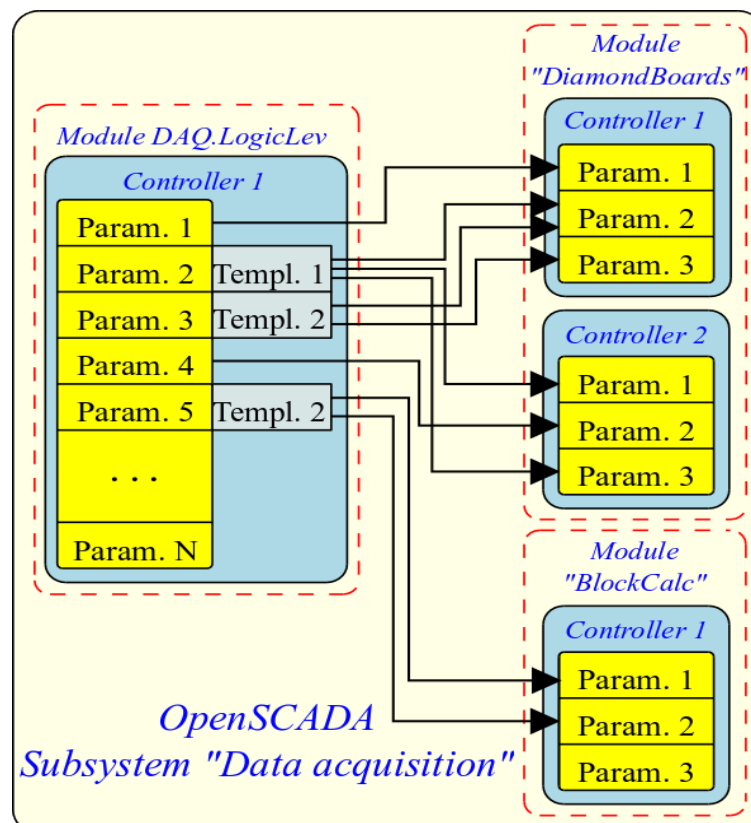


Fig. 7. The mechanism of the "Logic level" on the example of [LogicLev](#) module.

On the figure you can see that the parameters of the logic level controller function as reflections of other parameters of "Data acquisition" subsystem (on the example of parameters 1 and 4) and the free formation of parameters based on templates 1, 2 and other parameters of "Data acquisition" subsystem (on the example of the parameters 2, 3 and 5).

Structure of the parameters with the template in their basis has the structure shown in Fig. 8.

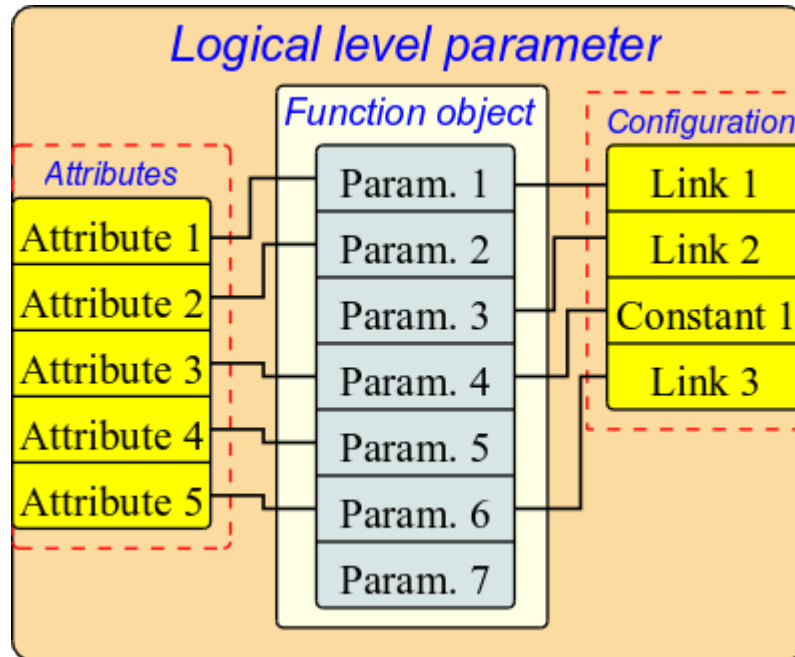


Fig. 8. Structure of the parameters, with a template in its basis.

As can be seen from the structure, the logic level parameter consists of the function object, attributes and configuration of the template. The function object is an instance of the execution of the template's function with the set of inputs/outputs and the computation program of the template on the language of user programming, usually it is the Java-like programming language of the module [DAQ.JavaLikeCalc](#). But the template may be generally without the program, providing only the structure of transfer the inputs/outputs. Attributes in the structure represent the list of attributes of the result parameter in accordance with the template. Configuration in the structure provides the configuration of the template's properties and its external links.

The logic of the work of logic-level parameters can be written as follows:

- Parameter connects with the template from which we obtain the structure of attributes in accordance with the template's function.
- At the moment of linking the parameter with the function the linkage of an object of the parameter's function instance with the function of the template.
- Further, in accordance with the template of function, the structure of links is formed. Based on the structure of links the form of linkage the parameter is formed and the user sets the links .
- When you access the attributes of the obtained parameter the check for the presence of a direct link is done. In the case of a direct link presence the request is routed by this link, otherwise the value is taken from an object of the parameter's function instance.
- At this moment the template's function calculation works using the the object of the parameters' function. However, before the calculation the reading of the values by the links is made, and after calculation the results are recorded by these links.

Parameters' template in general provides the following:

- structure of I/O of the template's function;
- signs of the configuration and linkage of the template (constant, link);
- preliminary values of the configuration of constants and templates of links' configuration;
- signs of the attributes of the resulting parameter of the logic level types: not attribute, an attribute with full access, attribute with read-only access;
- mechanism for calculating the I/O of the templates' function using the user programming language of OpenSCADA.

Fig. 9 shows image of the configuration tab of the parameters' template of subsystems "Data acquisition" as the table with the configuration of inputs/outputs and the text of the program of user programming.

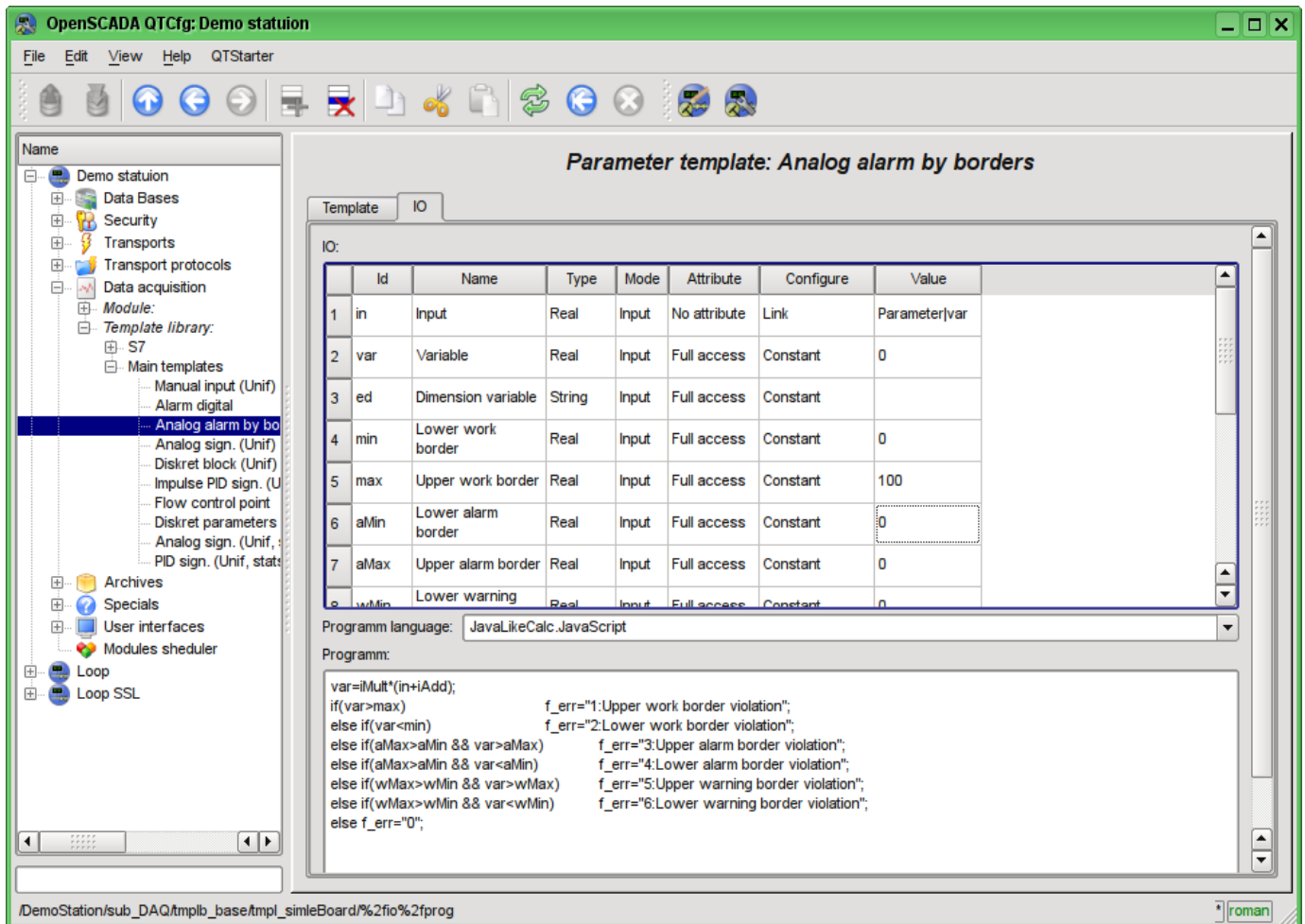


Fig. 9. The configuration tab of parameters' template of subsystem "Data acquisition".

The input/output field of the parameter's template provides the following properties of special purpose: "Attribute", "Configure" and "Value".

The "Attribute" property is the reflecting sign of the the i/o of the template on the resulting attribute of the parameter. There the following options for this property are provided:

- *No attribute* - input/output of the template's function does not reflect on the attribute;
- *Read only* - input/output of the template's function reflects on the attribute with read-only access;
- *Full access* - input/output of the template's function reflects on the attribute with full access.

The "Configure" property is the sign indicating the using of input/output of the template's function in the resulting configuration of the template on the logic level. The following options for this property are provided:

- *Constant* - available for setting only on the level of the configuration of parameter's template as a constant;
- *Public constant* - available for setting at the parameter of logic level in the configuration section of the template as a constant;
- *Link* - available for setting at the parameter of the logical level in the configuration section of the template in the form of link.

The field "Value" describes the preset value for the constants and configuration template of the external links. Template of the configuration of external links is used to describe the mechanism of grouping and automatic allocation of external links. The structure of the template of configuration of external links is the specific for each module of subsystem "Data acquisition", which uses the template's mechanism. In the case of the logic level module the allocation is made over the external attributes of the parameters with the template of configuration of the external link of the form: <Parameter>|<attribute>. Where <Parameter> is used to combine the parameters and place on the configuration form, and an attribute - for the associated

linkage of the attributes at the appointment of the parameter.

As an example of the template's using in Figure 10 lets show an images of the parameter of the logic level module "F3". In Fig.10 the tab "Template config" is presented', it serves for the configuration, including the linkage, of the parameter's template. In Fig.11 the tab "Attributes" is shown with the list of attributes and their values, created through the template.

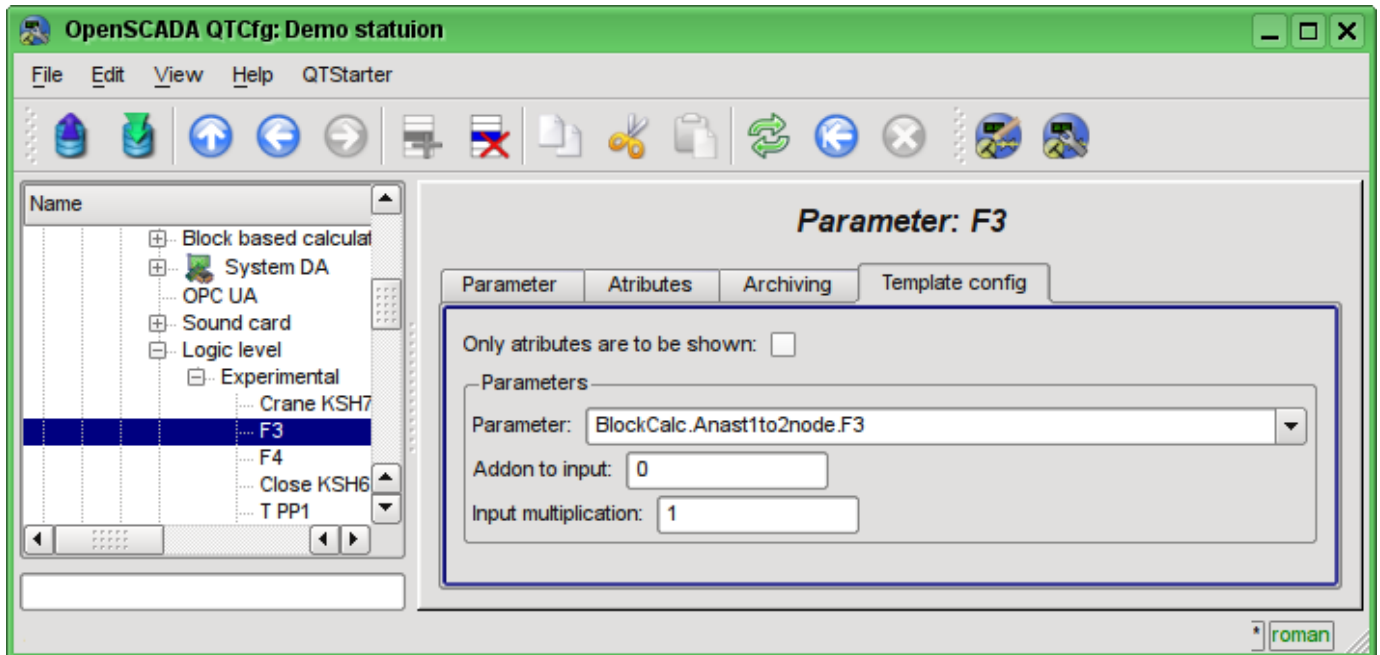


Fig. 10. The "Template config" tab of the "F3" parameter of the logic level module.

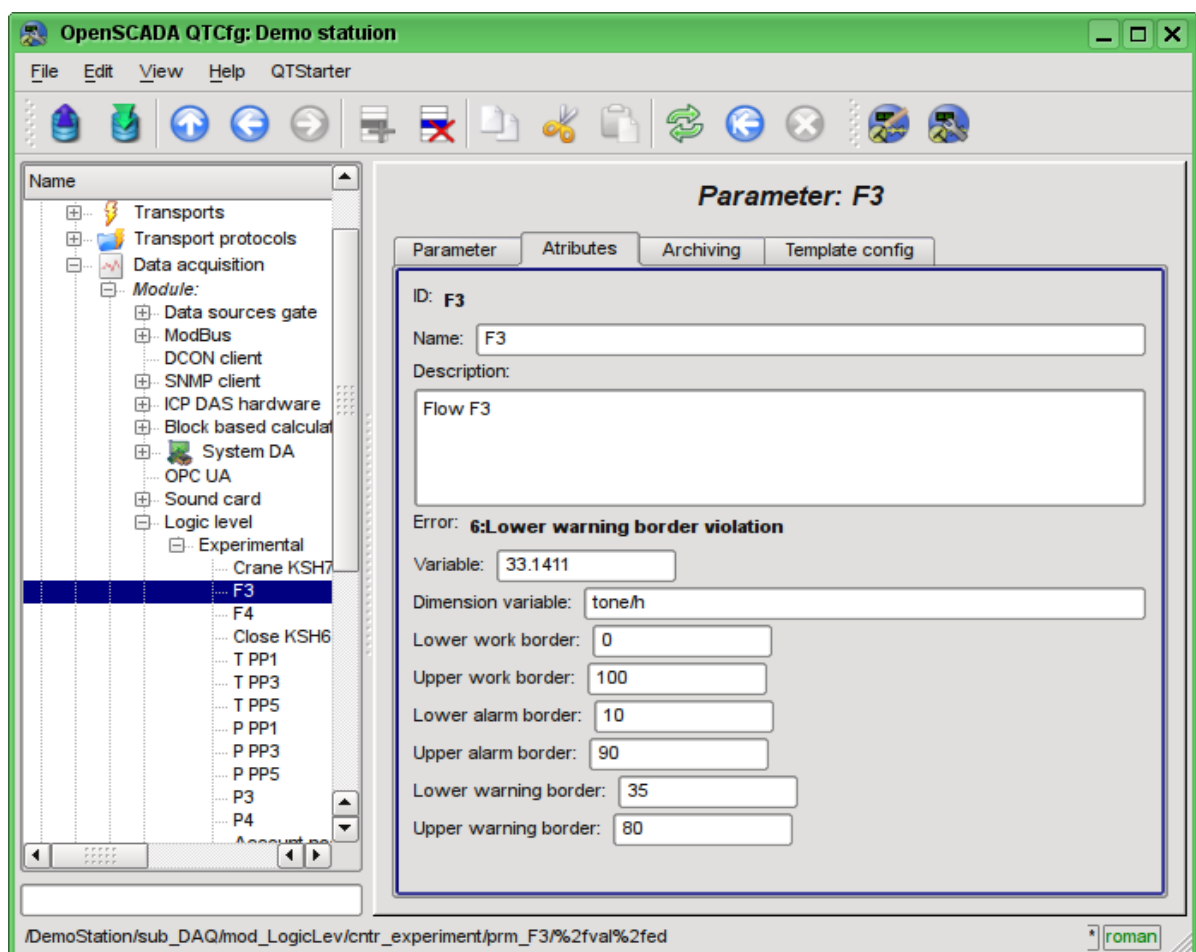


Fig. 11. The "Attributes" tab of the "F3" parameter of the logical level module.

4. Redundancy of the data sources

Redundancy in general and of the data sources in particular serves to increase the overall level of fault-tolerance of the solution by integrating the redundant nodes in collaboration with the main node. In case of failure of the main node the grab of the main node functions by the redundant one takes place. The redundant scheme can work in the mode of capacity allocation between the co-operating nodes.

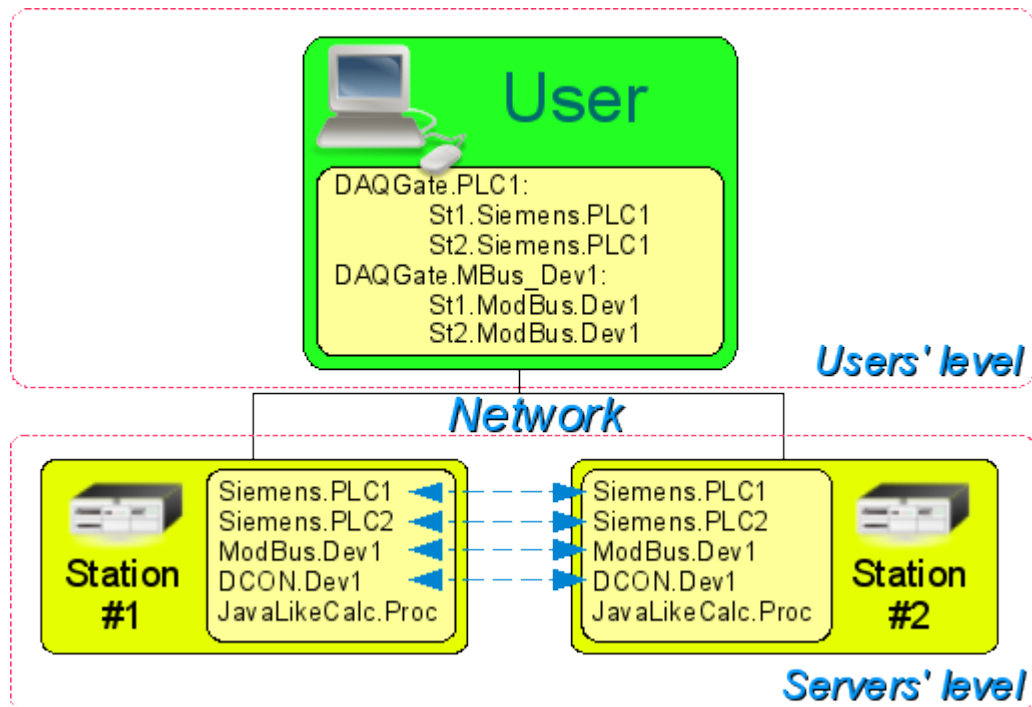


Fig. 12. Horizontal and vertical redundancy.

In the case of a subsystem "Data acquisition" of the OpenSCADA system the data redundancy (Figure 12) performs the following functions:

- Redundancy of the data acquisition mechanism. Typically, this function is realized without special arrangements by simply running of the parallel redundancy stations with the same configuration and working independently. However, in the case at the station, which works as PLC, such approach is unacceptable because of the simultaneous making of control actions and the absence of synchronization of calculators' data.
- Compensation of the data loss on the time of the node stop with the redundant node archive. There are two mechanisms of compensation. The first and the main mechanism implements the loading of the sections of the archive from the redundant station at the time of the station startup in general or of individual controllers of "DAQ" subsystem. the section of the archive is requested from the moment of the last record in the local archive and till the current time. The depth of the request is limited by the indicating of the limit time in the configuration of the redundancy. The second, complementary mechanism, performs the filling of the "holes" in the archive at the time of the actual user's request to the data. Such an approach on the one hand allows to make the predictable in time synchronization at startup and on the other hand - actually eliminates the data loss in the case of working at least one station during the entire time.
- Capacity allocation of data acquisition between the nodes. When creating complex allocated systems there can be an important question of predicting and optimizing of the overall system performance. Taking into account these problems the redundancy mechanism provides the execution of tasks of data acquisition of individual sources (OpenSCADA controllers) only at one station. The other stations' tasks would go to data synchronization mode with the executive station. In the case of loss of the connection with the executive station the task of the local data acquisition is started. It is also provided the possibility of optimal capacity allocation of the execution of data acquisition task's of the controllers' group between the stations.
- Optimization of the load on the external data sources through the data request from an external source by the only one node. In practice, we often meet highly loaded data sources or interfaces of access to the data sources, for which even the data acquisition by one station can be a problem and

would require reducing the acquisition periodicity, ie data quality. The mechanism of redundancy, except of capacity allocation between the stations as described above allows you to remove an additional load form the data source and its interfaces, thereby improving the quality of data.

- Prevention of some differences of data on different nodes associated with the mismatch of moments of time at the independent acquisition of data by individual nodes by means of receiving the data from the station with an active controller. In systems with redundant and high accountability it should be excluded or minimized the differences in the data at different stations, that means the real acquisition of data by one station and synchronization with these data of other stations.

Configuration of the redundancy starts with the addition of redundant stations in the list of OpenSCADA system stations in the tab "Subsystem" of the "Transports" subsystem (Fig.13). Then the whole configuration of the redundancy is made in the "Redundance" tab of subsystem "Data acquisition" (Fig. 14).

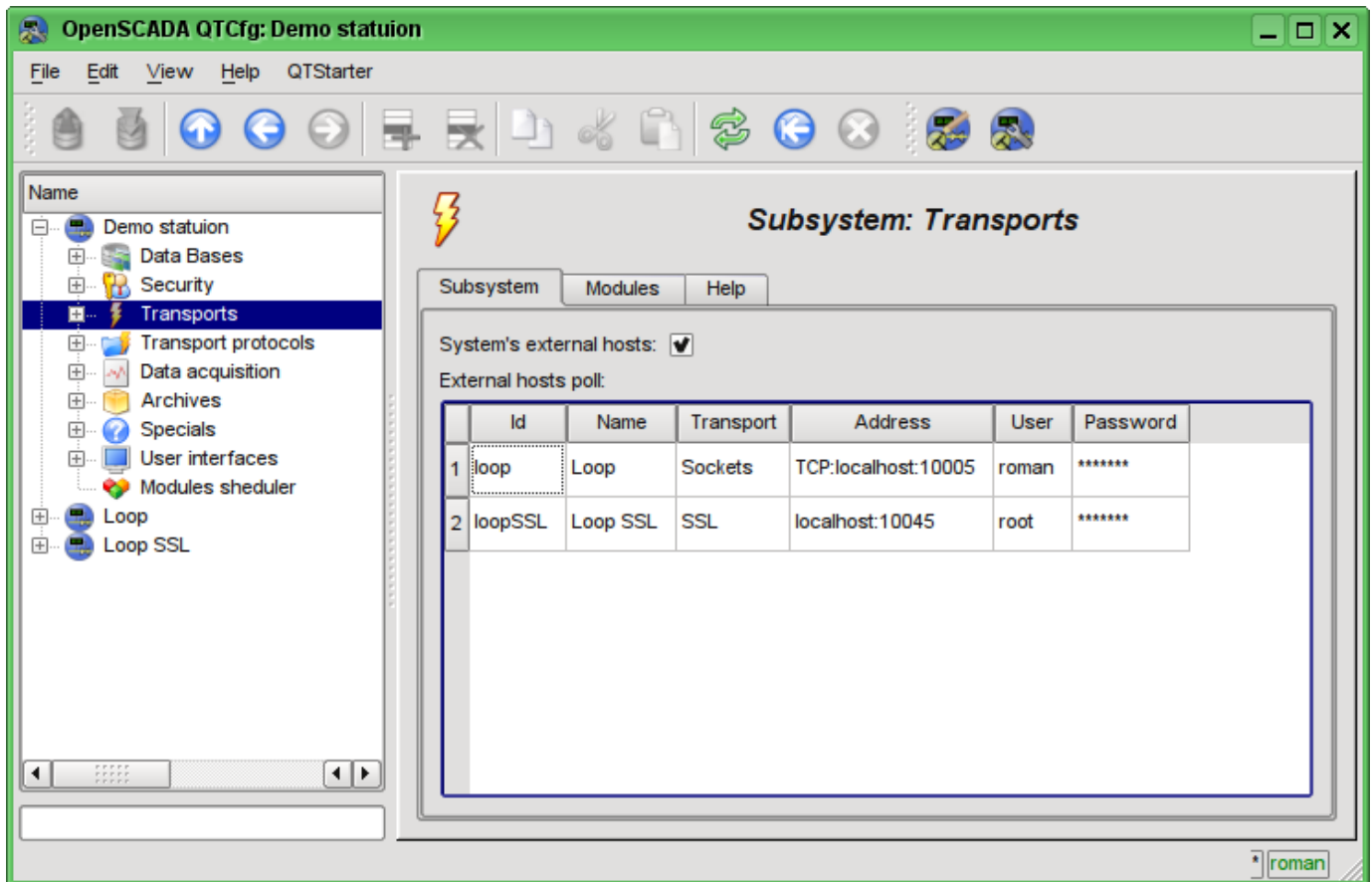


Fig. 13. The "Subsystem" tab of the "Transports" subsystem.

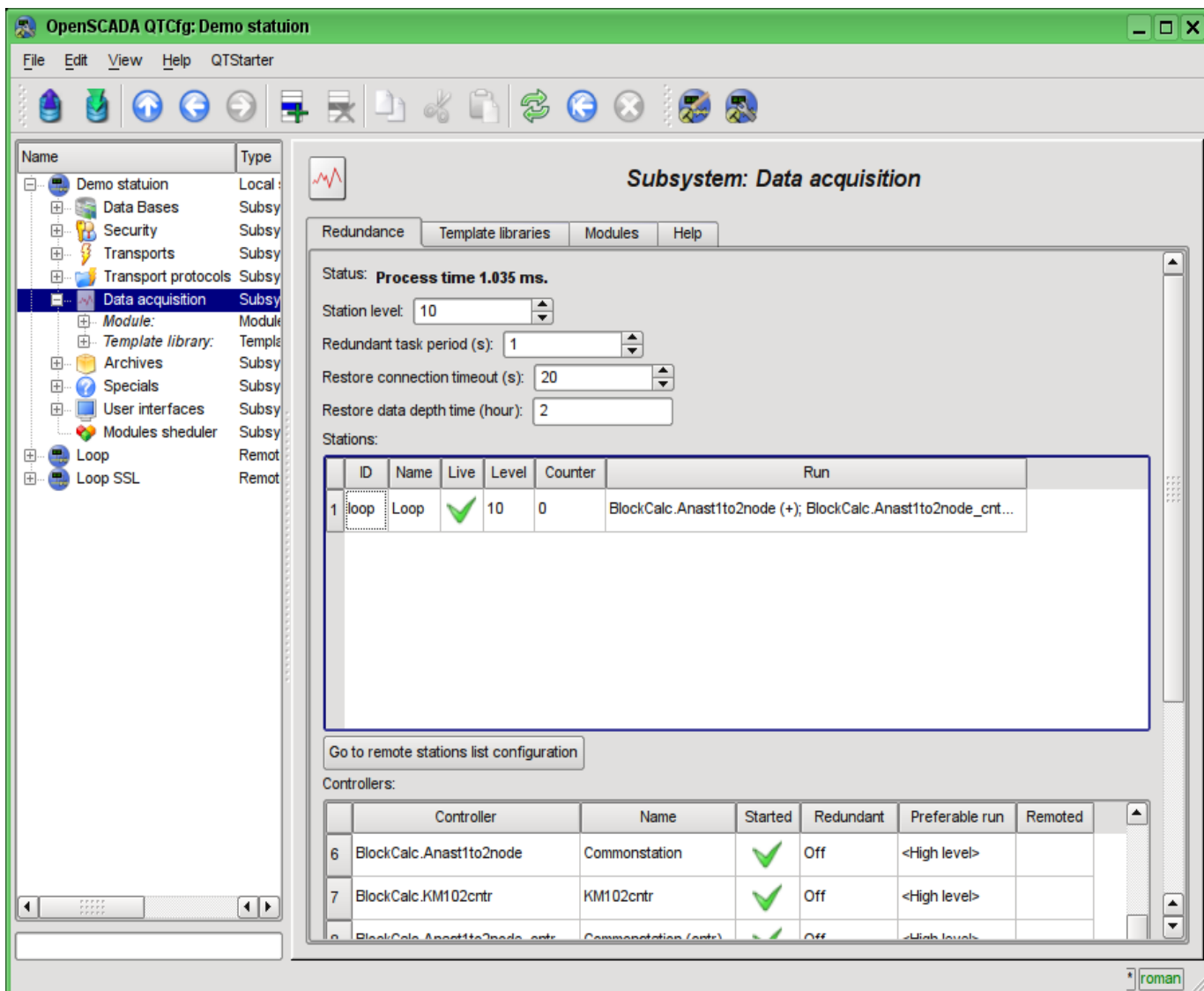


Fig. 14. The "Redundance" tab of the "Data acquisition" subsystem.

The service task of the redundancy mechanism is always running and executed at intervals which are prescribed in the appropriate configuration field. The real work on implementing the redundancy is carried out in the presence of at least one redundant station in the list of stations, and implies:

- Monitoring of the connection with external stations. In the monitoring process the requests to remote stations are made to get the information about them updated and to check connection. In the case of loss of connection with the station the repeat of connection to it is made through interval specified in the configuration field "Restore connection timeout". In the "Live" field of the station the current state of communication is displayed. In the "Counter" field the number of requests carried to the remote station, or the time remaining for the next connection attempt to the lost station is displayed. In the "Run" field there is a list of active controllers at the remote station with a sign of the local execution.
- Local planning of the controllers' execution in reserve. Planning is carried out in accordance with the station's level and preferences of controllers' execution.
- calling the data synchronization function for the local controllers working in the mode of synchronization of data from external stations. During the call, it is being prepared to request of the data from the remote station for the parameters in the controller starting from the time of the last request. On the request the only the values of modified attributes and sequence of values from an archive in case of loss of several cycles of values are returned.

To monitor the time spent in the cycle of redundancy tasks the field status is provided. When approaching the real time of execution to the cycle of the redundancy tasks it is recommended to increase the frequency of execution of this task!

For the controller of subsystem "Data acquisition" there is provided the modes of asymmetric and

symmetric redundancy. Asymmetric redundancy is working with the configuration of the controller of the remote station, as it is, and does not try to generalize it. Symmetrical mode supposes the synchronization of configuration of the controllers of stations with the configuration of the highest level station, and suggests the changes in the configuration of all controllers of the stations when changing it on the one of the stations. Currently this mode is not implemented!

Quick start OpenSCADA

An open system OpenSCADA is extremely modular, flexible and multi-functional SCADA-system. As a consequence of this the first contact with OpenSCADA can be quite complex because of the small chance of matching the previous experience of the user or complete lack of it with the methods of work in OpenSCADA. However, this is largely just a first impression, because the whole power of OpenSCADA is in the palm of the user, because of the abundance of which the user can get confused, and he may require considerable efforts to select the functions needed to solve his tasks.

For this reason, and to visualize the general concept of work in OpenSCADA this document is created. The document in the concise and understandable form shows the path from start of OpenSCADA to creation of the user interface elements on real examples. In addition, the document contains the chapter with recipes for the configuration, implementation, and typical problems of the user.

The document does not contain the detailed description of the concept and a deep dive into the details of OpenSCADA, and provides links to the appropriate OpenSCADA documents, containing such information.

Document description is synchronized with the implementation of the examples on the demonstration database (DB) of OpenSCADA. Consequently, the user must obtain the distribution kit of OpenSCADA with this database for illustrative study and testing the examples.

1. Terms, definitions and abbreviations

The automated workplace - Usually consists of a system unit of the computer system, display, mouse, sometimes with the keyboard, and other peripheral equipment that is used for visual representation of technological process data and making the control actions on the TP.

Lock (term) – notional boundary of technological parameter, in the case of its getting over the preset algorithm steps to prevent the accident are made. In some modes of TP (start) in accordance with the regulation it may be necessary to disable the lock (unlocking).

Unlocking (term) – process of the lock disabling for the duration of the TP working in the modes for which the regulation provides this operation. Attention, unlocking the technological parameters is strict accountable operation and the must be made by operational staff in the proper order.

Quittance (term) – the process of confirming the fact that operational staff drew attention to the failures of TP working. This process usually entails the adoption of measures by the operator to correct violations and pressing the appropriate button to stop the alarm.

PLC (abbreviation) – Industrial PLC. Microprocessor-based electronic device to which via computer-process interface (CPI) the signal of processing parameters are going. PLC acts the role of the direct data acquisition, processing and making the control actions by means of algorithms of automatic control. In addition the PLC provides data for the visualization of TP, and receives data of the manual intervention from the "top level" system.

Alarm (term) – process of notifying the operational staff of the violation of process or work of the automation equipment. Way of signaling may be of different types of impacts on human senses in order to attract attention. Often it is involved the following types of alarms:

- *Light alarm* – usually is done by changing the color of the graphic object (blinking) to emerging events and by the setting of static accidents colors (red and yellow) for acknowledged events.
- *Sound* – is made by an audible signal at the time of occurrence of the event. Type of alarm can be monotonous and the synthesized voice message with information about the violation.

TP (abbreviation) - Technological process. The whole complex of technological equipment of the production process.

CPI (abbreviation) – Computer-Process Interface. A number of devices or modules of PLC, to which are directly connected the signals from the sensors of TP for subsequent conversion from analog to digital form and vice versa. The transformation is carried out with aim of further processing of values of technological parameters in the PLC.

Alarm setpoint (term) - conventional boundary of the value of technological parameter, the overcoming of which is considered as the emergency situation. Usually the following boundaries are provided:

- *The upper and lower emergency boundaries* – boundaries of the emergency values of technological parameter.
- *The upper and lower warning boundaries* – boundaries of the prevention, regulation boundaries, of the violation of the technological parameter of the working range.
- *Failure* - sign of parameter getting over the hardware boundaries of technological equipment. Usually it characterizes the sensor failure, breakage of the communication channel with the sensor or PLC.

SCADA (abbreviation) - Supervisory Control And Data Acquisition. The software that performs complex tasks of data acquisition of TP, their archiving and presentation, as well as the making the control actions by the operator in manual mode.

2. Installation and start

The installation of distribution kit of OpenSCADA can be done in two ways. The first and the easiest way is to get packages for your distribution of the Linux operating system. The second - to build the OpenSCADA system from sources. In general, the installation procedure depends strongly on the used Linux distribution and to exhaustively describe it in this guide it does not seem possible! Therefore, you may need a deep familiarity with the mechanisms of software installation for the selected Linux distribution in its documentation.

If user does not have deep enough knowledge and skills in the chosen distribution of Linux, it is strongly recommended to choose the Linux distribution by the criterion of existence for it the packages of OpenSCADA in the repositories of the distribution, which will ensure an easy and problem-free installation!

If the user can not only install the OpenSCADA, but also the Linux distribution, for the first time he can use the "live" distribution of Linux, with the installed and ready for work or study demonstration of OpenSCADA. Currently are available "live" builds on the basis of distribution ALTLinux in the form of CD and Flash-images on the link: <ftp://ftp.oscada.org/OpenSCADA/Live>.

2.1. Installing OpenSCADA from packages

Installing OpenSCADA from packages, in its turn, can be made by two methods. The first - the simplest one, when packages of OpenSCADA are already present in the official or additional repositories of the distribution of used Linux, and installation of them - the question of running the typical program of packages' management followed by selection of the OpenSCADA packages. The second is when the packages of OpenSCADA are got and installed manually.

At the moment the system OpenSCADA packages can be found in the repositories of such distributions OS Linux: [ALTLinux](#) and distributions based on package base of [Fedora](#).

To check for OpenSCADA packages presence in the repositories of the used Linux distribution, as well as to download packages of OpenSCADA for manual installation you can download at the official OpenSCADA site (<http://oscada.org/en/zagruzka>).

Description of the installation from the repository of the selected Linux distribution we'll omit and refer the reader to the documentation of the appropriate distribution.

For the manually installation of OpenSCADA packages lets download them from the official website or from the other source. You can download packages of two types.

The first type is represented by a set of four packages:

- **openscada** - package with all files necessary to start OpenSCADA, including all modules;
- **openscada-demo** - files of the demo database with the configuration to start the demonstration;
- **openscada-doc** - all documentation on the OpenSCADA system;
- **openscada-devel** - development packages for the creation of the separate modules to the OpenSCADA system.

The second type is represented by the set of about forty packages with separation of OpenSCADA modules in separate packages:

- **openscada** - contains the OpenSCADA core, basic configuration and launching(starting) files;
- **openscada-DB.*** - "DB" subsystem's modules;
- **openscada-DAQ.*** - "Data acquisition" subsystem's modules;
- **openscada-Archive.*** - "Archives" subsystem's modules;
- **openscada-Transport.*** - "Transports" subsystem's modules;
- **openscada-Protocol.*** - "Transport protocols" subsystem's modules;
- **openscada-UI.*** - "User interfaces" subsystem's modules;
- **openscada-Special.*** - "Specials" subsystem's modules;
- **openscada-demo** - files of the demo database with the configuration to start the demonstration;
- **openscada-doc** - all documentation on the OpenSCADA system;
- **openscada-devel** - development packages for the creation of the separate modules to the OpenSCADA system.
- **openscada-plc** - virtual package containing dependencies for installing the typical configuration of the OpenSCADA as PLC;
- **openscada-server** - virtual package containing dependencies for installing the typical configuration of OpenSCADA as SCADA-server;
- **openscada-visStation** - virtual package containing dependencies for installing the typical configuration of OpenSCADA as visual SCADA-station.

The first type of the packages' set is provided for easy, manual installation, because it contains only four packages. The second type is designed to be placed in a repository of Linux distribution and for the following installation of them using the package manager, which made auto-dependency resolution. The second type of the packages' set allows you to install only the required components of OpenSCADA, thereby optimizing the working environment, which is not allowed by the packages of the first type.

If you are installing from the repository you should only select the package "openscada-demo". Everything else, according to the dependencies, will be selected and installed automatically.

Manual installation of RPM-packages of the first type can be made by the following command, after changing the working directory to the directory with the package:

```
# rpm -i openscada-demo-0.6.4.1-alt2.i586.rpm openscada-0.6.4.1-alt2.i586.rpm
```

Manual installation of DEB-packages of the first type is made by the following command, previously having changed the working directory to the directory with the package:

```
# dpkg -i openscada-demo_0.6.4.1-2_all.deb openscada_0.6.4.1-2_i386.deb
```

In the process of implementation it may cause bugs related to missing dependencies. The manual installation of the packages means that you'll solve them manually, like installing packages of OpenSCADA, or via the package manager of Linux distribution. To familiarize with the process of installing software in RPM-package you can by the click on: <http://skif.bas-net.by/bsuir/admin/node51.html>.

2.2. Installation from sources

If you can not get packages of OpenSCADA for the selected distribution, it remains the only one option of OpenSCADA building from the sources. The building process of OpenSCADA is described in details in the guide on the following link <http://wiki.oscada.org/HomePageEn/Doc/BuildFromSource>. However, it must be borne in mind that if you managed to build OpenSCADA from sources, then this document is not for you, and you probably can easily master the basic documents of OpenSCADA (<http://wiki.oscada.org/HomePageEn/Doc/ProgrammManual>).

This chapter is given here for completeness and integrity of the consideration of the question, because the required qualification level of the user for this chapter is much higher than the level of the document as a whole!

3. Initial configuration and start

After proper installation of the OpenSCADA with demo database no pre-configuration is required. If you want to perform a particular configuration, which differs from the base, then use the document of description the OpenSCADA program on the link: <http://wiki.oscada.org/HomePageEn/Doc/ProgrammManual/part4?v=7kf>.

Attention! The demonstration of OpenSCADA based on the demo database is not the same as that is usually provided by the commercial software vendors to demonstrate the possibilities, but to exclude or to complicate the normal operations by limiting the functions. Demonstration of OpenSCADA is fully-functional system that provides examples of implementation and configuration of various components. Based on the demo database of OpenSCADA one can easily create own projects, using the given resources.

To execute the OpenSCADA with demo database you can from the menu of the desktop environment in the "Graphics" section, "Demo of open SCADA system" with the characteristic icon (Fig. 3.1).

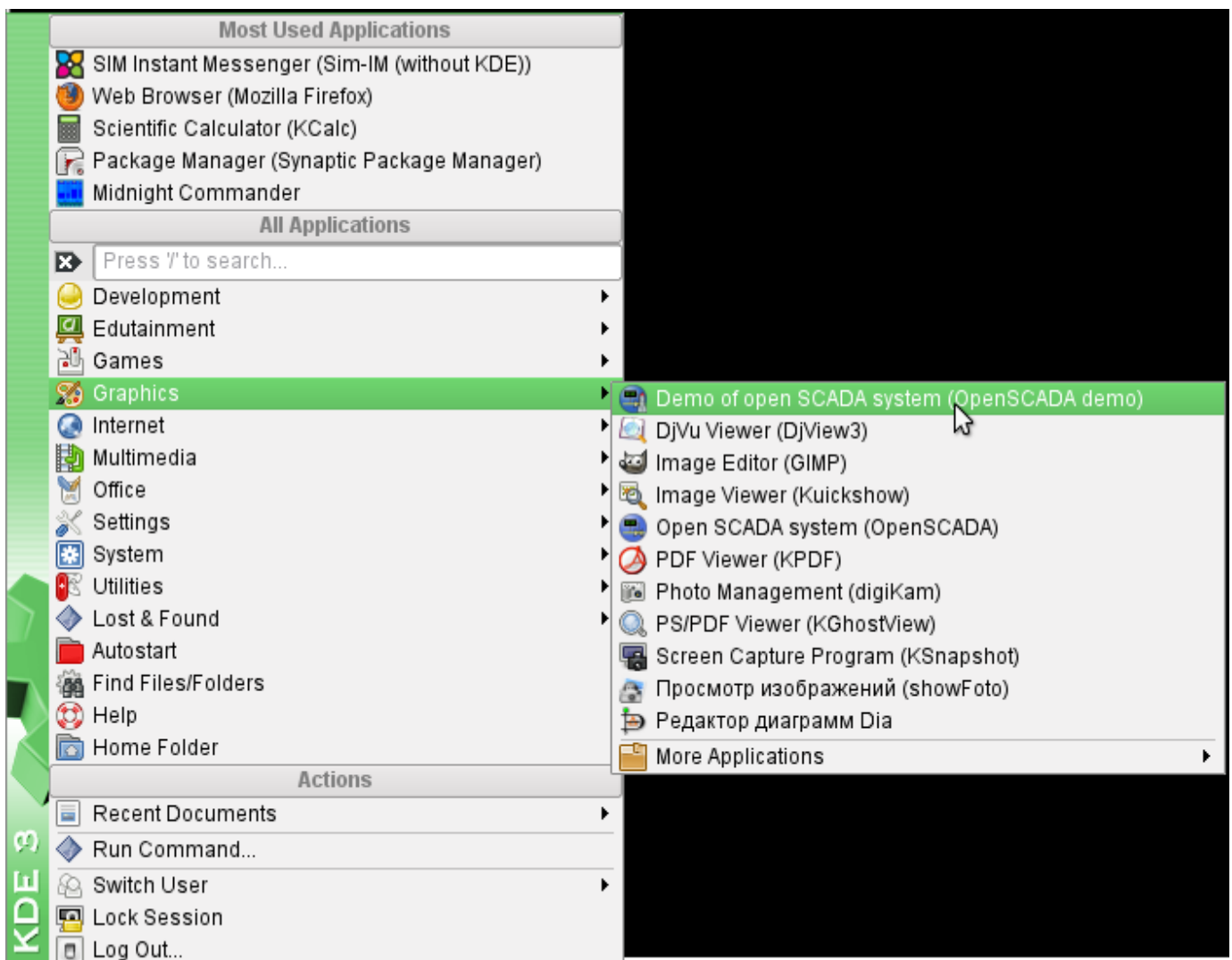


Fig. 3.1. Menu item of the desktop environment to start the demonstration of OpenSCADA.

Start also can be done from the console by the command:

```
# openscada_demo
```

After start we'll get the window of graphical configurator of OpenSCADA system - QTCfg (Fig.3.2) with the opened root page. Demo database specifically set up so that the first to appear when you start it would be the configurator's window. You can then open the window for creating graphical user interfaces, as well as run the project of user interface for execution.

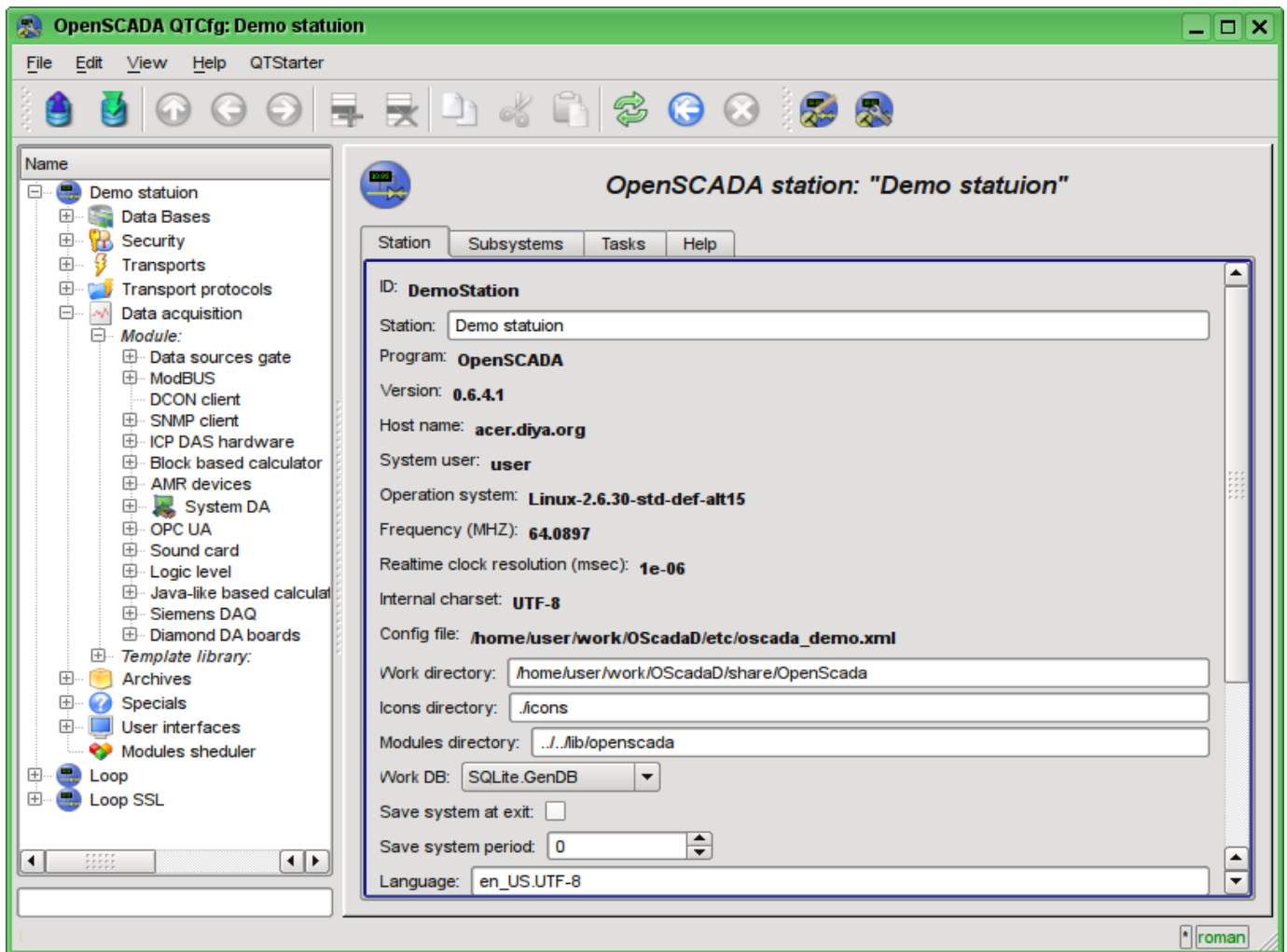


Fig. 3.2. OpenSCADA configurator - QTCfg, the root page.

Configurator of OpenSCADA is the main and sufficient mean for the configuration of any component of the system. Like many others components of OpenSCADA, configurator is implemented as a module. Besides the configurator QTCfg there may be available other configurators that performs the same function, but implemented on the basis of other technologies. For example, these are the Web-configurators: [WebCfg](#) and [WebCfgD](#).

All actions in the future, we will examine only in the configuration tool QTCfg, although all of them can be done in other configurators.

The structure of the interface of the configurator's window can be considered in detail by reference <http://wiki.oscada.org/HomePageEn/Doc/QTcf>. For us it is more important now to examine all the available interfaces of OpenSCADA, so click next to last icon in the top on the toolbar. After clicking on this icon the window of user interface development will be opened (Fig.3.3).

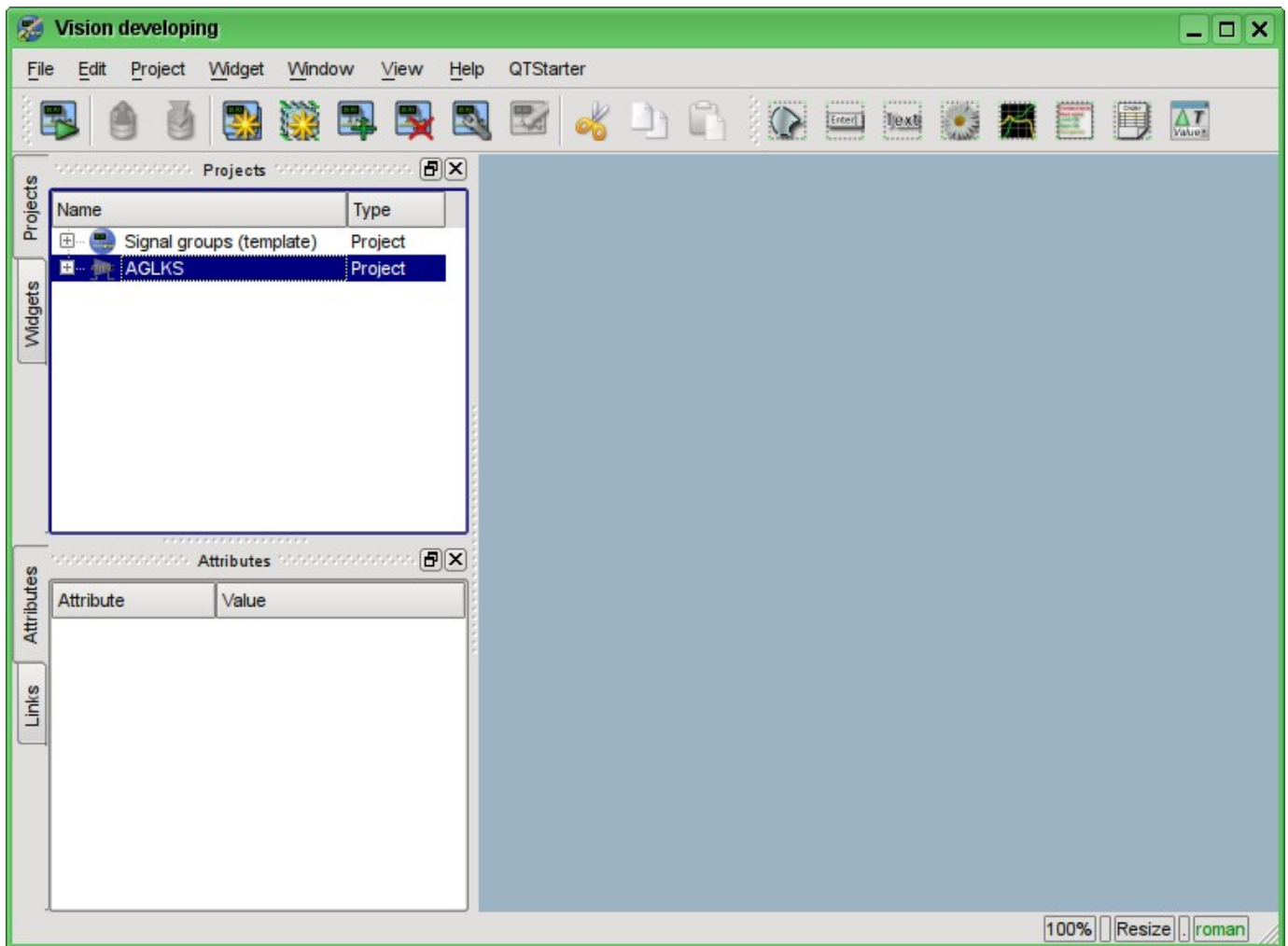


Fig. 3.3. Window of the UI development.

Then we can start the project "AGLKS" for execution. To do this, select it in the list of projects and run by clicking on the first left icon on the toolbar or in the the popup menu. The result will be the window of user interface (Fig.3.4).

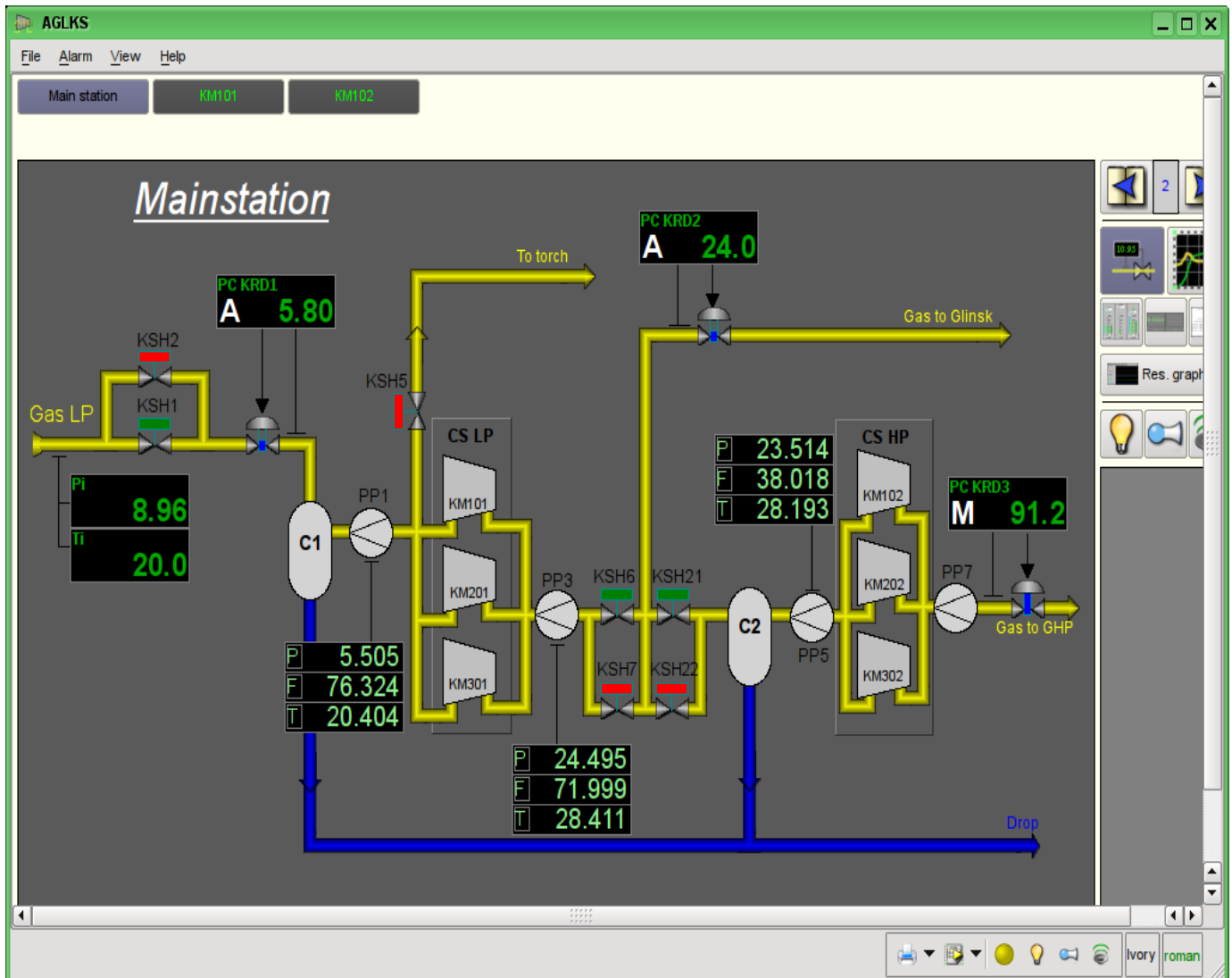


Fig. 3.4. Window of the user interface of the "AGLKS" project.

Building and executing of the user interfaces is implemented by the [Vision](#) module of the subsystem "User interfaces". In addition to this module it can be accessed the other modules of visualization. For example, OpenSCADA provides module [WebVision](#), which allows to execute projects, previously developed in the interface module "Vision", through the Web-based technologies and standard Web-browser. All actions in the future we will examine only in the interface of the "Vision" module.

So we ran the demonstration of OpenSCADA and familiarized with the main set of tools. In the future we will use them for configuration of OpenSCADA, creating the tasks of data acquisition, binding the collected data with the purpose of their processing and making the impacts, as well as to create the user interface of visualization of the received data and to make the control actions.

Lets close the window of the project "AGLKS" execution and the window of development of the user interface for the preparation to the study of the following chapters.

The whole process of configuration of SCADA-system to perform the functions of the "top level" can be divided into two stages:

- The configuration of data sources and creation the database (DB) of the parameters of these sources.
- Formation of a visual presentation of TP data by creating the operator interface in the form of mnemonic schemes, groups of graphs, groups of contours, documents, etc.

4. Working with Data Sources

The main function of any SCADA-system is to work with data sources, namely the inquiry of programmable logic controllers (PLC) and simple modules of CPI. For more details see the document "Data acquisition in OpenSCADA" on the following link: <http://wiki.oscada.org/HomePageEn/Doc/DAQ>.

Support of the one or another data source depends on the protocol or API, through which the source provides its data, and the availability for the protocol/API the module in the subsystem "Data acquisition" in OpenSCADA. The total list of modules of the subsystem "Data acquisition" and documentation on them can be found here <http://wiki.oscada.org/HomePageEn/Doc?v=9b7#h735-4> in the appropriate chapter.

Obtained from sources data subsequently are archived, processed and used for visual representation for the operator of TP.

4.1. Data inquiry of the TP device

As an example lets examine and create the inquiry for the air cooler device. Demo database contains the model of real-time of TP of compressor station of the six compressors. Data for two devices of air coolers "AT101_1" and "AT101_2" of the compressor station "KM101" are available on the protocol ModBUS/TCP on port 10502.

We will create the inquiry controller on the protocol ModBUS/TCP and get these data, thereby practically made the task of inquiry of real data, because from the external device our configuration will be different only in address of the device and addresses of the ModBUS registers.

For the data acquisition through ModBUS/TCP protocol in the OpenSCADA there is "ModBUS" module in the subsystem "Data acquisition". To add a new controller we will open the page of the modules "ModBUS" in the configurator ("Demo Station"->"Data acquisition"->"Module"->"ModBUS") and in the pop-up menu of the "ModBUS" item click "Add" (Fig. 4.1.1).

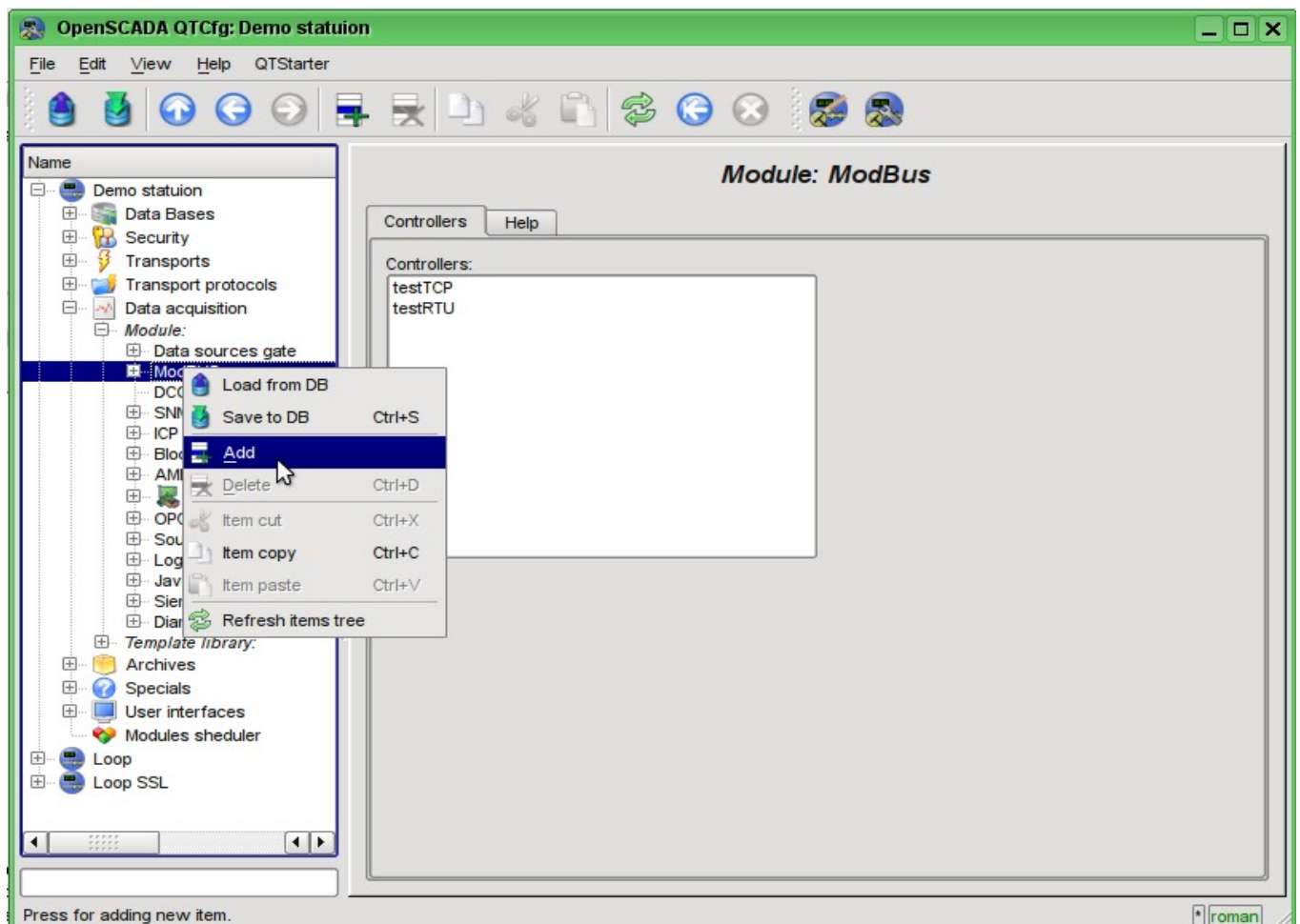


Fig. 4.1.1. Adding the controller in the "ModBUS" module of the subsystem "Data acquisition".

At the result of our actions the dialog window will appears (Fig.4.1.2) to enter the ID and name of the new controller. IDs of any objects in OpenSCADA are limited to 20 characters and they should be entered using English alphabet characters and numerals. In addition, it is desirable to start the ID with the letter. This is due to the fact that the identifier can later be used in scripts. The names of objects of OpenSCADA are limited to 50 characters and can be entered by any symbols. The names commonly used for display. If the name field is blank, instead it the identifier will be used to display. Enter the ID "KM101" and the name "KM 101".

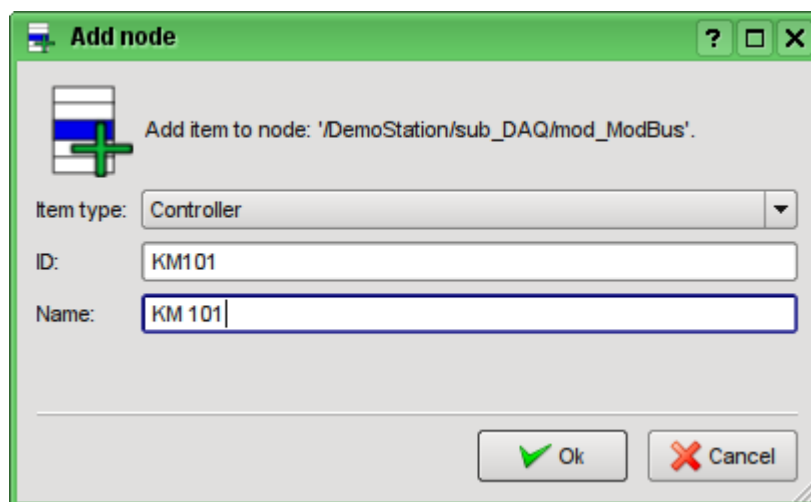


Fig. 4.1.2. Dialog to specify the ID and name of the new object.

After confirming we have a new controller's object. Lets choose it in the configurator and get acquainted with the settings (Fig.4.1.3).

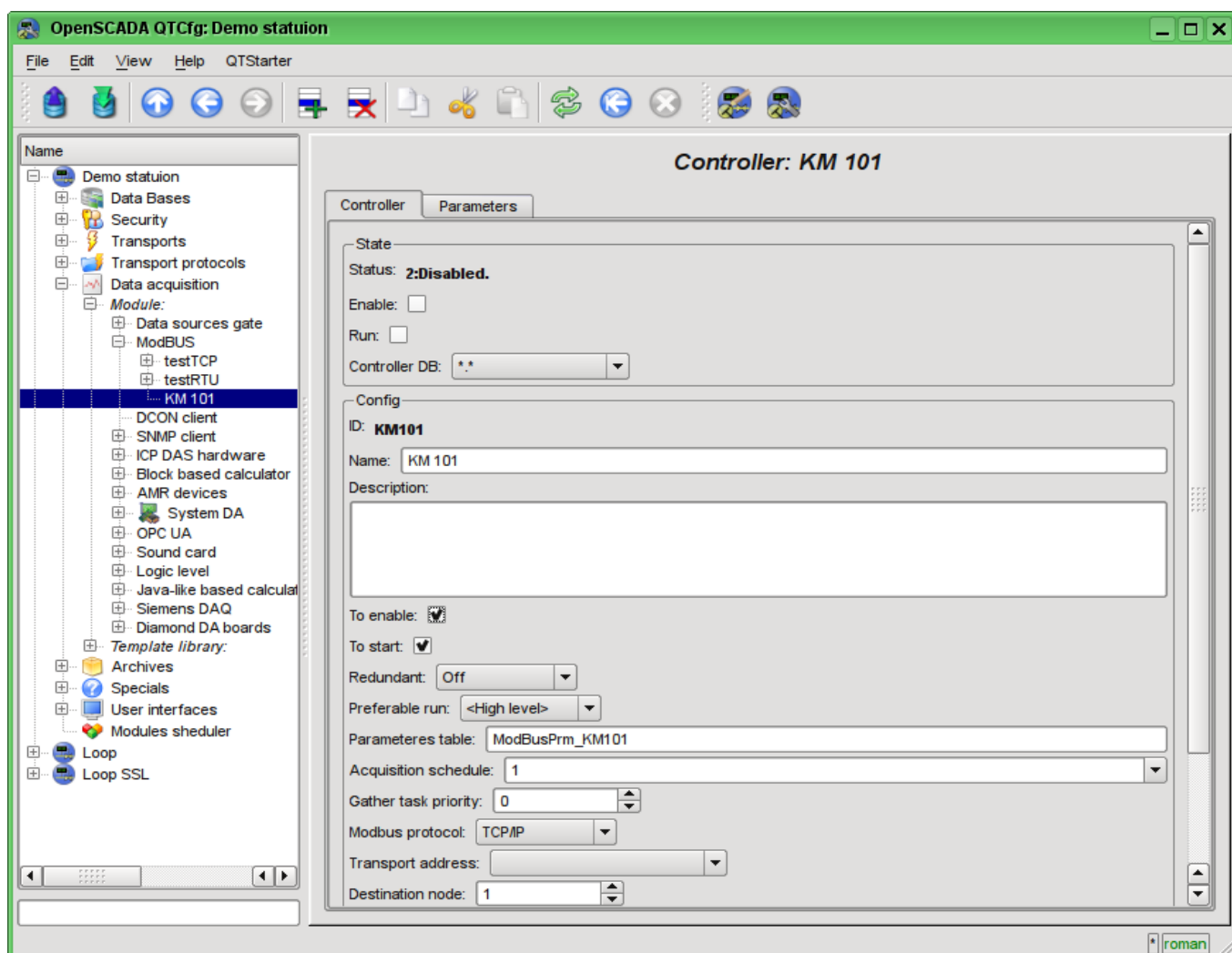


Fig. 4.1.3. The main tab of the controller's object settings of the ModBUS module.

Settings of the controller's object, as a rule, are specific for the different types of data sources and protocols. To familiarize in details with the settings of the controller's object of the "ModBUS" module you can using the link <http://wiki.oscada.org/HomePageEn/Doc/ModBus?v=btc#h871-13>. We'll examine the general configuration of the controller's object and the key settings for the "ModBUS" module.

With the help of the page of the controller's object in the section "Status" may be primarily assessed the current state of the controller's object and the real state of connection with the physical controller, as well as quickly to change it. For example, field "Status" contains the code of error and the textual description of the current state of connection with the controller, in this case the controller's object is disabled. We are able to enable it and start by setting the flags beside the appropriate fields. Enabled controller's object initializes the parameters objects, the running one runs the task of inquiry and provides an opportunity to transmit data to the controller through the attributes of the parameters. The DB field indicates which database to store in the configuration of the object. We will store the data the main database, ie leave field the default.

In the "Config" section the configuration of the controller's object is directly contained:

- "ID" and "Name" are the fields, we've just entered at the object's creation. The Name can be changed right here, but the ID can not be changed so simply. If you want to change the ID you must Cut (Ctrl+X) and Paste (Ctrl+V) the object and enter the desired ID.
- "Description" may contain the detailed description and purpose of the controller's object. In our case, the value of this field is not fundamental.
- "Enable" and "Run" indicated the state, in which to transfer the controller's object at start of OpenSCADA. Lets set both fields.
- Two next fields "Redundant" and "Preferable run" serves for the configuration the horizontal and vertical redundancy of the data sources at the different stations (<http://wiki.oscada.org/HomePageEn/Doc/DAQ?v=adg#h942-10>). We'll not touch them.
- "Parameters table" - contains the name of the database's table in which to store the configuration of parameters of the controller. Leave it default.
- "Acquisition schedule" - contains the configuration of the scheduler to run the inquiry task. To get the description of the format of the configuration of the field you can from the tooltip. The single number indicates the frequency of run in seconds. Let it be one second.
- "Gather task priority" - indicate the priority of the task (from -1 to 99). Priorities above zero are meaningful only when you start OpenSCADA from the privileged user. Leave this field unchanged.
- "ModBUS protocol" - indicates the variant of the ModBUS protocol. We are interested in the option "TCP/IP", so leave it as is.
- "Transport address" - indicates the outgoing transport of the subsystem "Transports", which is used to connect to the controller. In the case of "TCP/IP" option we need the transport module "[Sockets](#)". We'll examine the creating of the outgoing transport in "Sockets" in details below.
- "Destination node" - indicates the ModBUS node. In our case, it should be "1".
- "Data fragments merge" - includes the merging not related fragments of registers in the single block of the request, up to 100 registers, instead of generating individual requests. Allows you to reduce the total time of the inquiry. Lets set this option.
- "Connection timeout" - indicates how long to wait for the response from the controller and after which to report an error of connection. Zero indicates the use of time of transport. Unchanged.
- "Restore timeout" - specifies the time in seconds after which if there is no connection to retry to reconnect.

Lets save our changes to the database by clicking the second left icon on the toolbar.

Now, in the same manner as the controller's object, let's create the outgoing transport in the module "Sockets" ("Demo Station" ->"Transports"->"Sockets") through the context menu (Fig.4.1.4). And let's call the transport as well as the controller: "KM101" and the name - "KM 101". Note that in the "Item type" of the dialog (fig.4.1.2) you should choose the "Output transport".

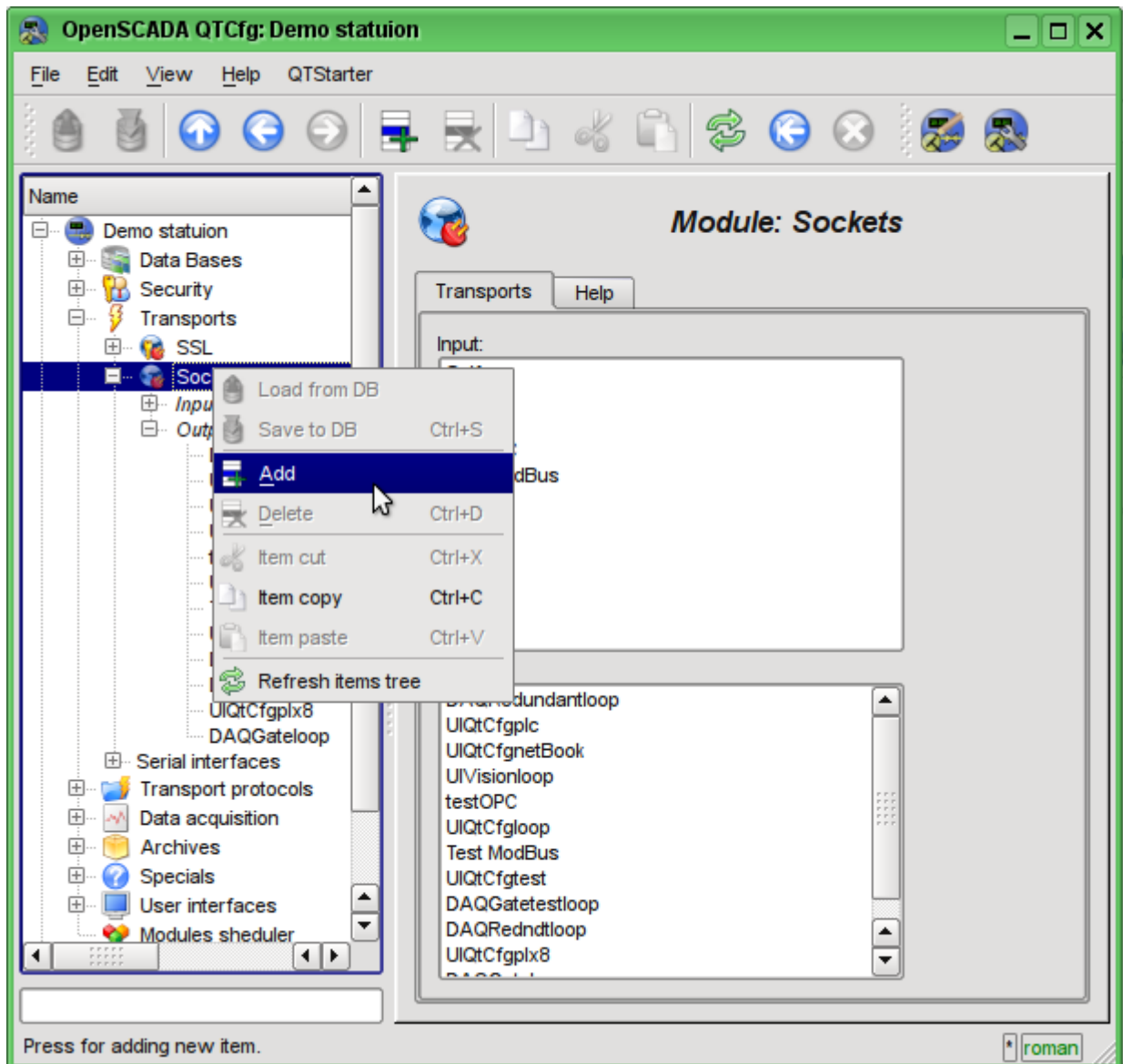


Fig. 4.1.4. Adding the outgoing transport in the module "Sockets" of subsystem "Transports".

The configuration page of outgoing transport is shown in Fig.4.1.5. This page also contains the section of the status and operational control. In the "Status" field the textual description of the current state of transport is contained. We can run it for execution by checking the box in front of the appropriate field. Running object of the transport initiates the connection to the external node. Field DB indicates the database to store the configuration of the object. We will store it in the main database.

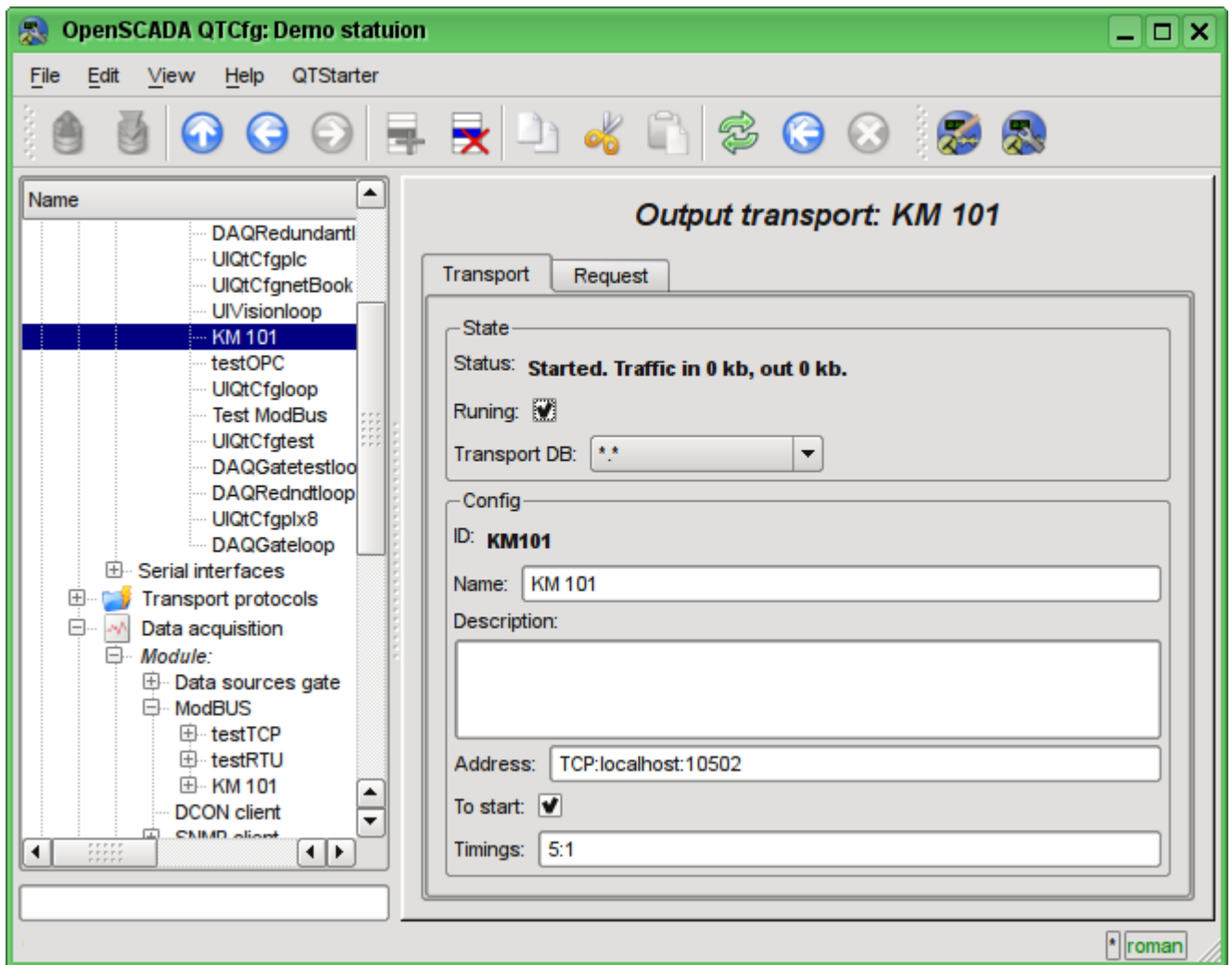


Fig. 4.1.5. The configuration page of the outgoing transport of the "Sockets" module of subsystem "Transports".

In the "Config" section the configuration of the transport object is contained:

- "ID" and "Name" contain the titles, which we entered when creating the object.
- "Description" may contain the detailed description and purpose of the object.
- "Address" specifies the type, address and mode of connection with the remote station. You can view the record format in the tooltip. Let's set this field to the value "TCP:localhost:10502".
- "To start" indicates in what state to transfer an object at start of OpenSCADA. Let's set the field.
- "Timings" indicate the duration of waiting for the response from the remote station. You can view the record format in the tooltip. Let us leave the value unchanged.

Let's save the transport and return to the configuration field "Transport address" of the controller's object and select the address "Sockets.KM101". Setting the controller's object is finished. The next step is configuration and choose the data you need to query from the controller. This setting is done by creating an object "Parameter" of the controller. The "Parameter" object allows you to describe the list of data obtained from the comptroller and to transmit them to the environment of OpenSCADA.

To add a new object of the parameter we will open in the configurator the page of our controller's object and on the popup menu of item "KM101" we'll click "Add". The parameter's object we'll call "AT101_1" and the name "AT 101_1".

The configuration page of the obtained parameter is shown in the Fig.4.1.6. This page contains the section of status and operational control. In the "Type" field it is contained the ID of the type of the

parameter, in this case it is only possible the "Standard" type (std). We can enable the parameter by checking the box of the appropriate field. The enabled parameter is involved in the process of exchange with the controller.

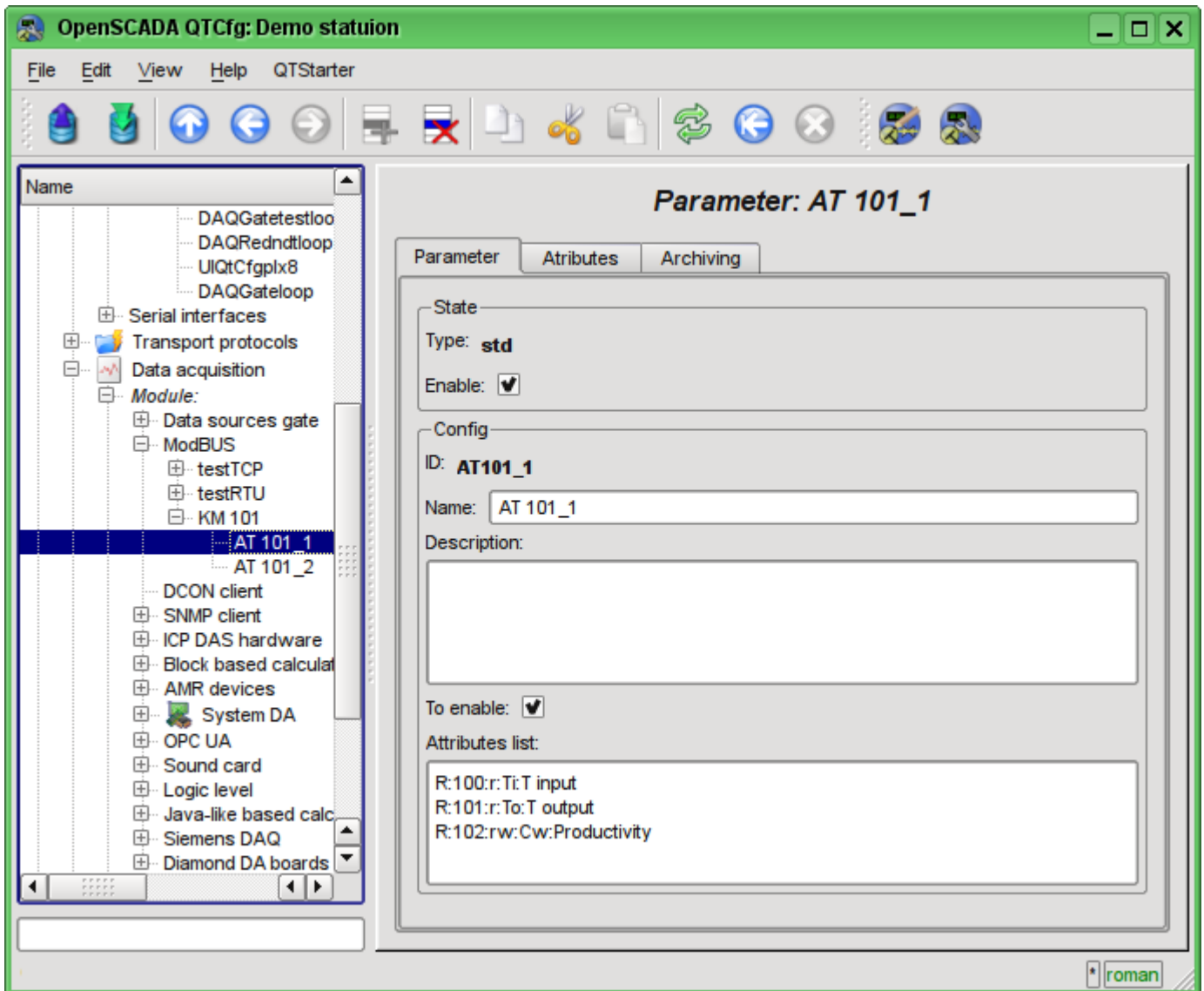


Fig. 4.1.6. Configuration page of the controller's parameter "ModBUS".

In the "Config" section the configuration of tge parameter's object is contained:

- "ID" and "Name" contain the titles, which we entered when creating the object.
- "Description" may contain the detailed description and purpose of the object.
- "To enable" indicates in what state to transfer an object at start of OpenSCADA. Let's set the field.
- "Atributes list" contains the configuration of attributes of parameters in relation of them to the registers and bits of ModBUS. You can view the record format in the tooltip. Let's set the contents of the text field as follows:
R:100:r:Ti:T input
R:101:r:To:T output
R:102:rw:Cw:Productivity.

Similarly, create the second option: "AT101_2" with the name "AT 101_2". The list of attributes for it let's set in:

```
R:103:r:Ti:T input
R:104:r:To:T output
R:105:rw:Cw:Productivity.
```

Let's save the both objects of the parameter. Now we can enable and run our controller to initiate the exchange. To do this, go back to the page of our controller's object and in the "Status" section let's set the flag "Run". If we do not miss something, the exchange is successfully started and in the "Status" field we'll get something like this, as it is shown in the Fig.4.1.7.

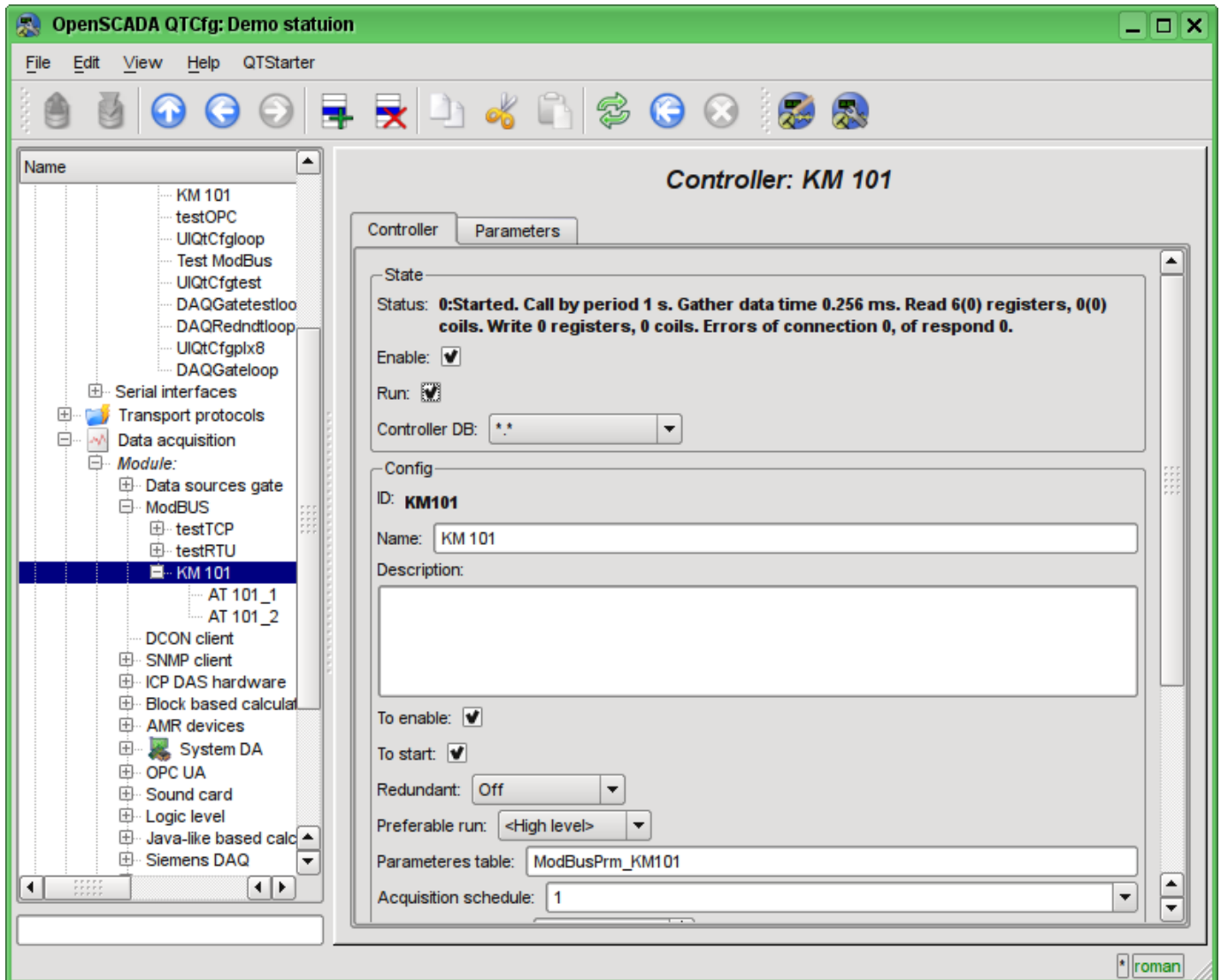


Fig. 4.1.7. The page of the controller's object if the exchange with the physical controller is successful.

In the case of a successful exchange with the physical controller, we'll obtain the described data of the controller in the infrastructure of OpenSCADA. You can see these data on the tab "Attributes" of our parameters AT101_1 (Fig.4.1.8) and AT101_2. Because the inquiry is regularly and at intervals of a second, then we can observe their changes by clicking the button "Refresh current page" on the toolbar.

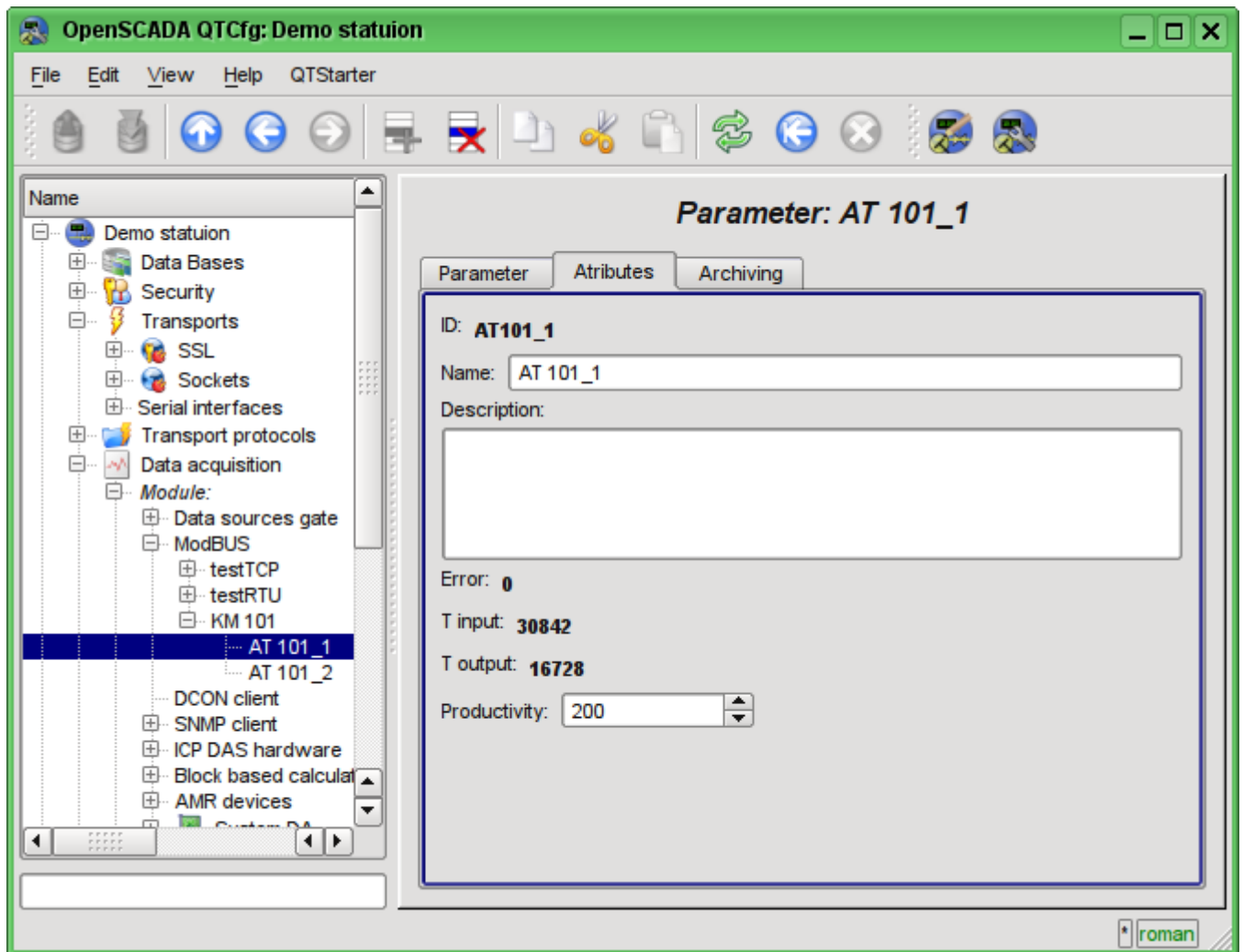


Fig. 4.1.8. The page of described attributes of the AT101_1 parameter.

The configuration of data acquisition is complete.

4.2. TP data processing

Frequently the initial data obtained from the data source are the "raw", ie unprepared or uncomfortable for the visual presentation, so you need to perform this preparation. In our example, we received the data that comes in the code from the scale inside the controller. Our task is to perform the calculation of real values from the received data. Data processing in OpenSCADA can be done, either during the visualization, and in the subsystem "Data acquisition". However, the mixing of the visualization process and processing of initial data makes the configuration confusing and makes the obtained images of the visualization unsuitable for reuse. For this reason, let's make the preparation of data in the subsystem "Data acquisition".

Calculations in the subsystem "Data acquisition" are done via the module of logic level "[LogicLev](#)" and the templates of parameters of the subsystem "Data acquisition". To familiarize with the concept of "logical level" you can here: <http://wiki.oscada.org/HomePageEn/Doc/DAQ?v=adg#h942-9>.

To make calculations in the module of the logic level you must first create the template of the parameter of subsystem "Data acquisition". To do this, let's open the page of templates' library "Main templates" ("Demo Station"->"Data acquisition"->"Template library"->"Main templates") and through the context menu we will create the template object "airCooler" with the name "Air cooler". The configuration page of the resulting object is shown in the Fig.4.2.1. This page contains the "State" section and the section of the operational control. We can make the template accessible by checking the box next to the corresponding field. Accessing templates can be connected to the data acquisition parameters, and the parameters will make calculations on this template. In the "Used" field the number of objects that use this template to calculate the image of the parameter is indicated. In the "Config" section only the familiar for us configuration fields are present.

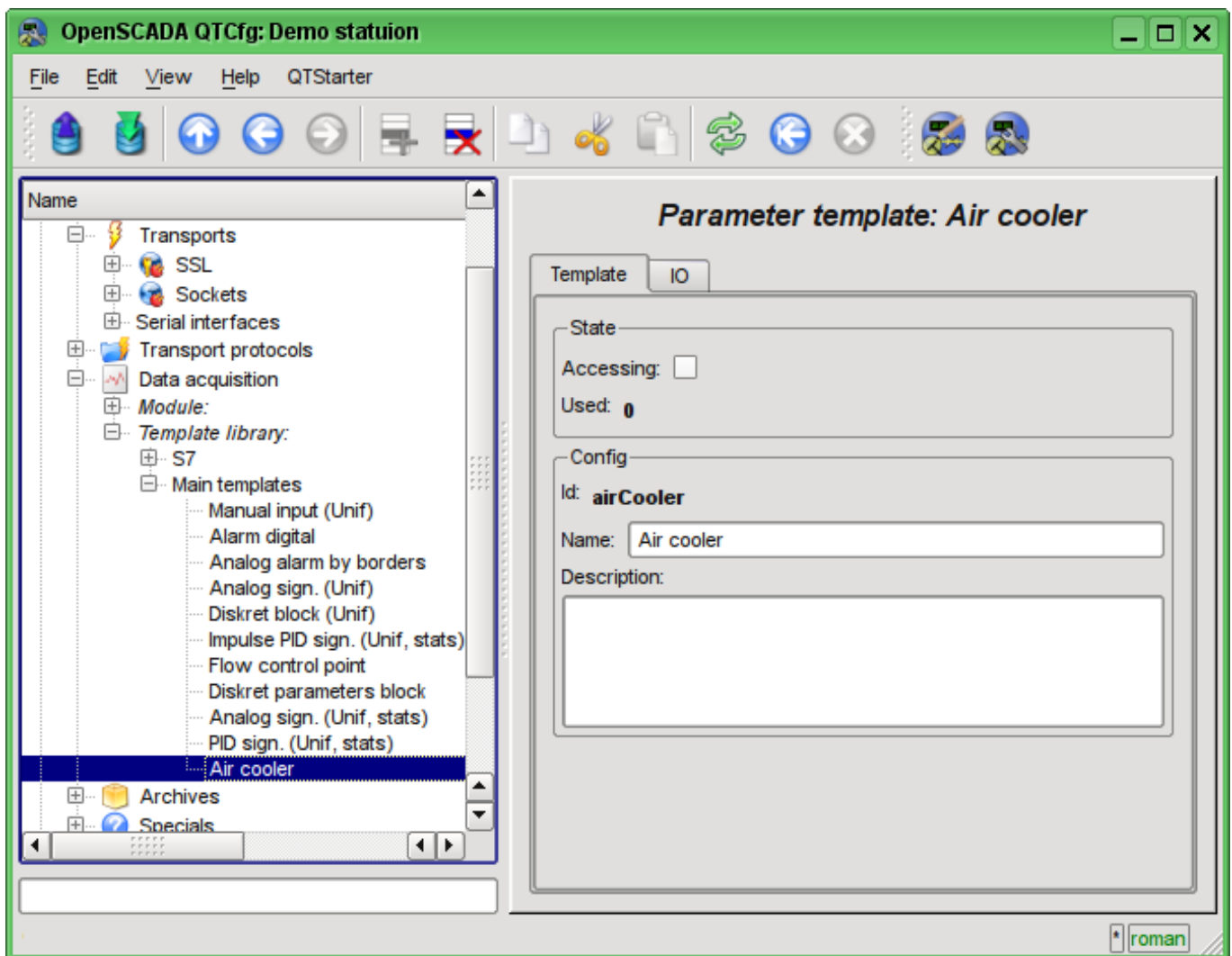


Fig. 4.2.1. The configuration page of the template's object.

The basic configuration and the formation of the template of parameter f data acquisition is made in the tab "IO" (Fig.4.2.2) of the template. The detailed description of the process of the template's formation can be found here: <http://wiki.oscada.org/HomePageEn/Doc/ProgrammManual?v=iuc#h932-6> .

Let's create in the template two properties fro the inputs ("TiCod", "ToCod"), two for outputs ("Ti","To") and one clear property ("Cw"). For the "TiCod", "ToCod" and "Cw" let's set the "Configure" flag to the "Link", this will let to link to them the "raw" source. For the "Ti" and "To" let's set the "Attribute" flag to the "Read only", and for the "Cw" - "Full access", we make it to form the three attributes of the resulting parameter of the data acquisition: two - read only and one with the full access..

The program language let's set to "JavaLikeCalc.JavaScript", and the program:

```
Ti=150*TiCod/65536;
To=100*ToCod/65536;
```

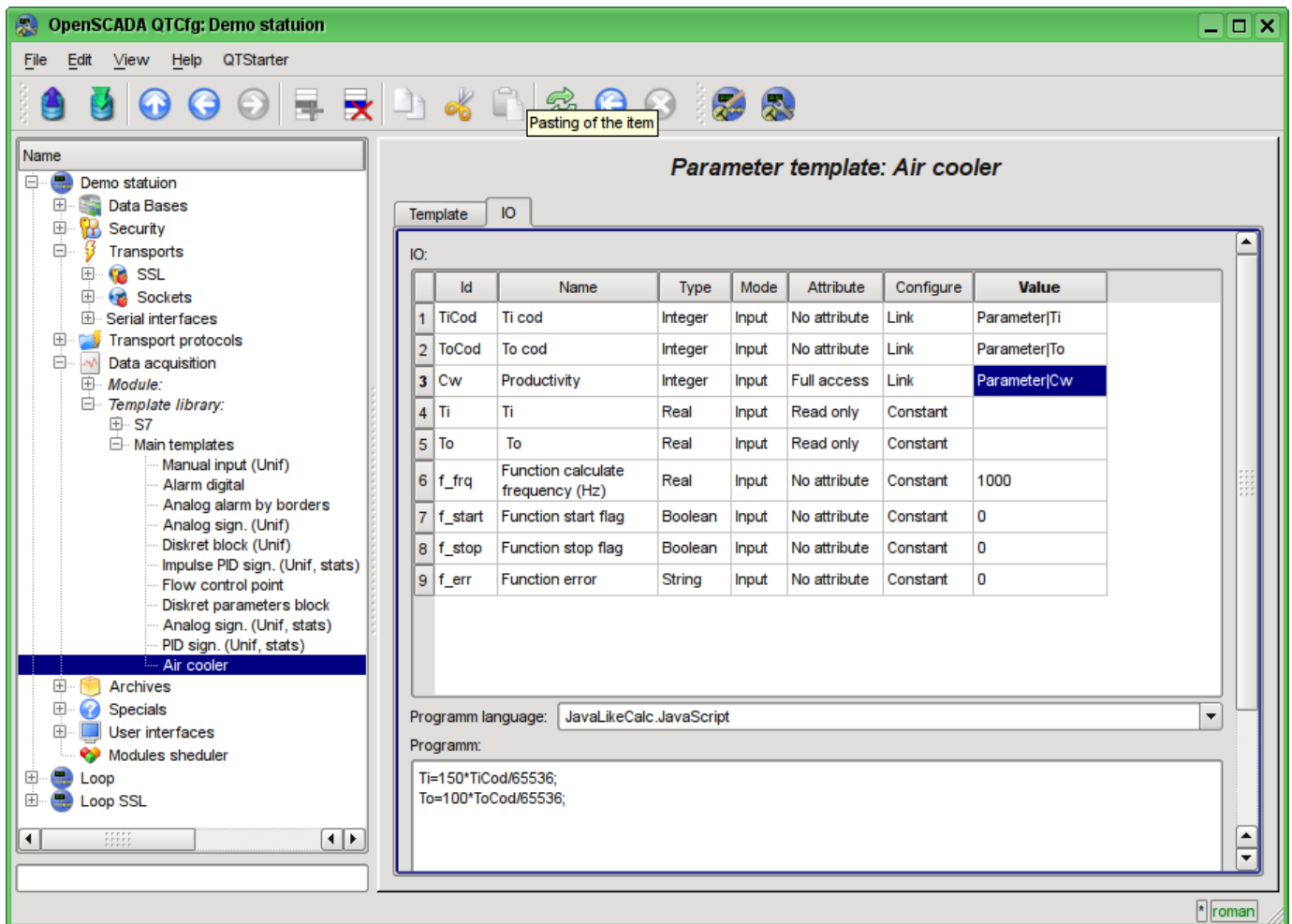


Fig. 4.2.2. Tab "IO" tab of the configuration page of the template's object.

Let's save the resulting template and set the accessibility flag.

Now we'll create the controller's parameters' objects in the "LogicLev" module of subsystem "Data acquisition". The controller and its parameters in the module "LogicLev" are identical to the previously created in the module "ModBUS" and they are created on the page: "Demo station"->"Data acquisition"->"Module"->"Logic level". The object of the controller and the parameters will be called identical to the objects in the module "ModBUS".

The object of the controller of the module "LogicLev" (Fig.4.2.3) has no specific settings and the default ones may not be touched.

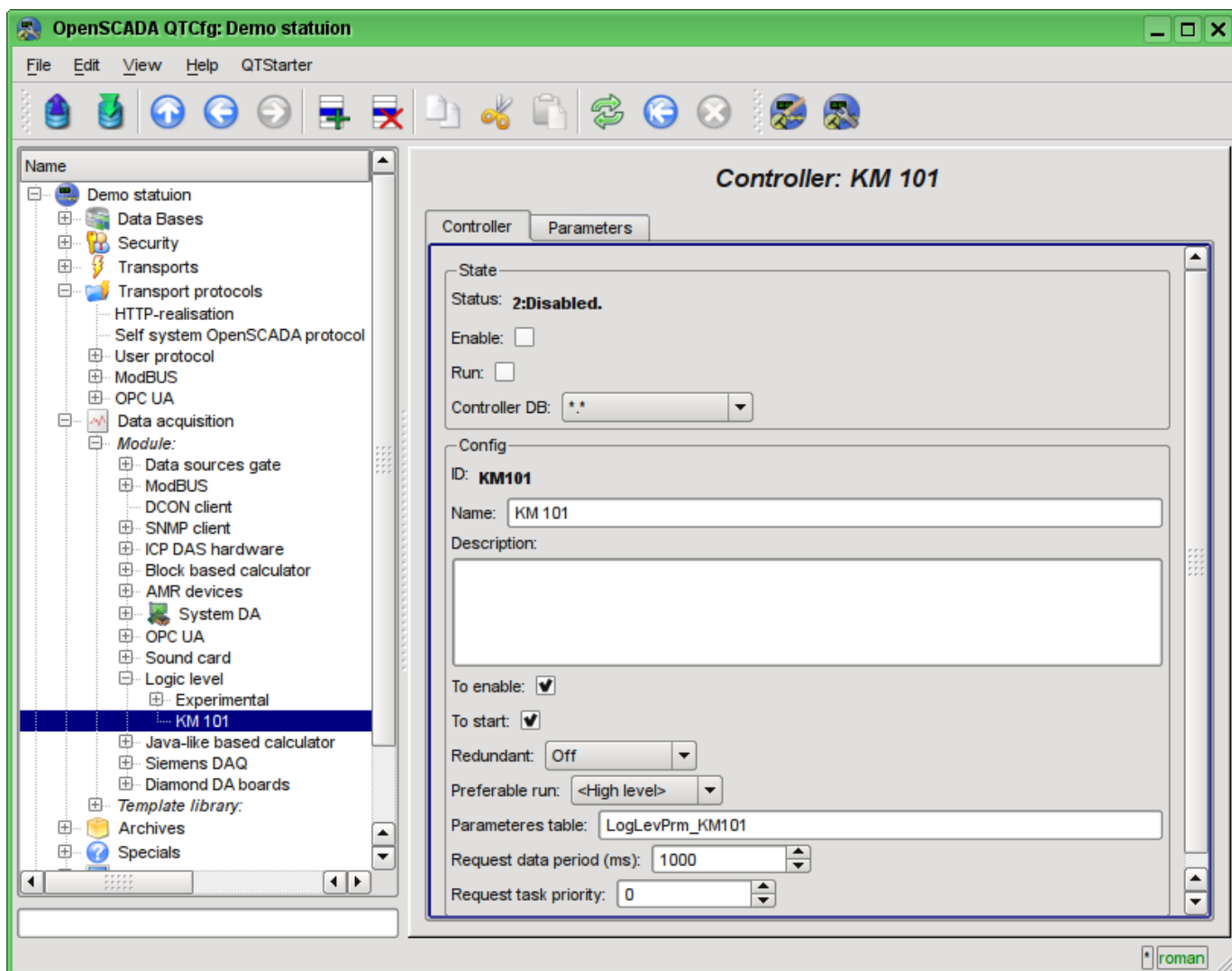


Fig. 4.2.3. The main tab of the configuration of the object of controller of the LogicLev module.

The object of the parameter of controller of the "LogicLev" module (Fig.4.2.4) has only the one specific setting - "Mode", where you need to set "Template" and select the address of the template, we have just created.

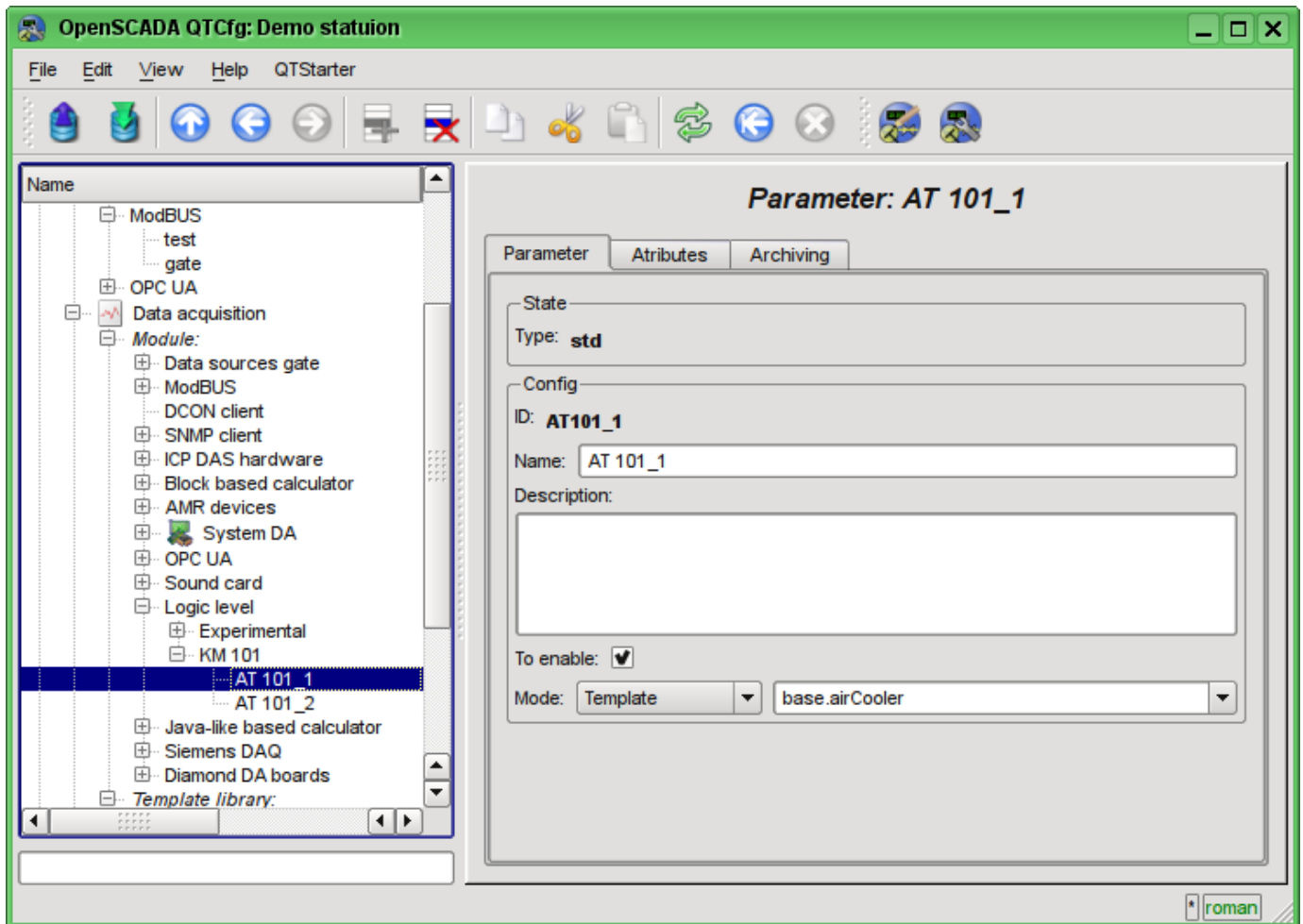


Fig. 4.2.4. Configuration page of the "LogicLev" controller's parameter.

In addition to the basic configuration of the parameter it is necessary to configure the attached template (Fig. 4.2.5). Configuration tab of the template appears in the parameter's mode "Enable". To enable the parameter it is possible by the previously enabling the controller. The flag "Only attributes are to be shown" allows you to set apart each link (Fig.4.2.6). Since we are made the following format of linkage in the template "Parameter|Ti", then all three links we can set simply by typing an address to the parameter in the "ModBus" controller. We shall specify the following addresses "ModBus.KM101.AT101_1" and "ModBus.KM101.AT101_2" in the appropriate parameters.

It should be noted that all the input fields addresses of objects in OpenSCADA provide a mechanism to set the address. This mechanism involves elemental choice, during which there is a movement in the interior. For example, typing the address "ModBus.KM101.AT101_1" first we will be able to choose the type of data source, including the "ModBus". By selecting "ModBus" in the list of available items for selection will be added to the module controllers "ModBus", among which will be "ModBus.KM101". Select the item "ModBus.KM101" add to the list of parameters of the controller, etc. to the final element in accordance with the hierarchy of objects (<http://wiki.oscada.org/Doc/OpisanieProgrammy?v=gax> # h827-6). To be able to return to levels above the selection list of all the elements are inserted into the higher levels of the current value of the address.

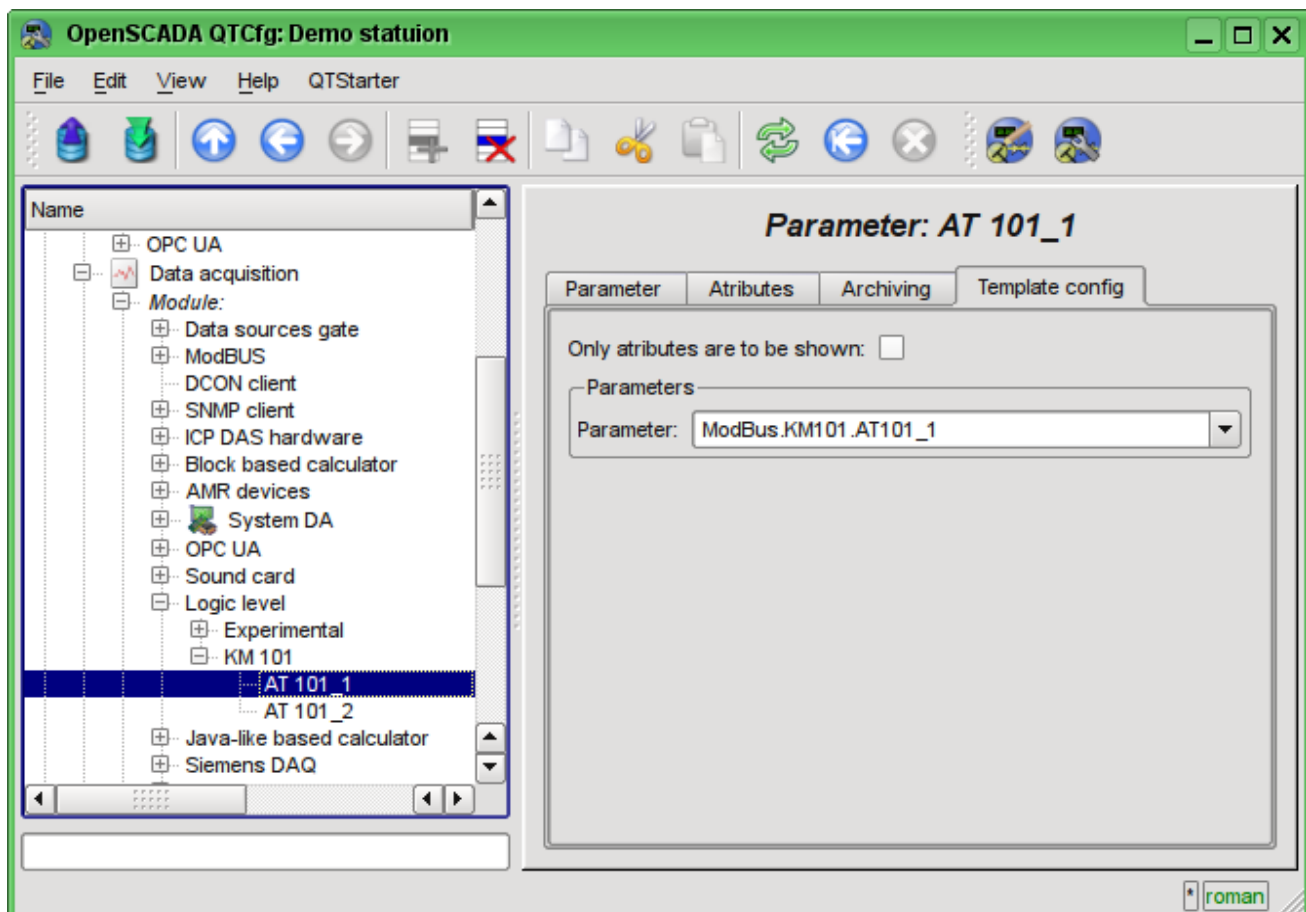


Fig. 4.2.5. The "Template config" tab of the "LogicLev" controller's parameter page.

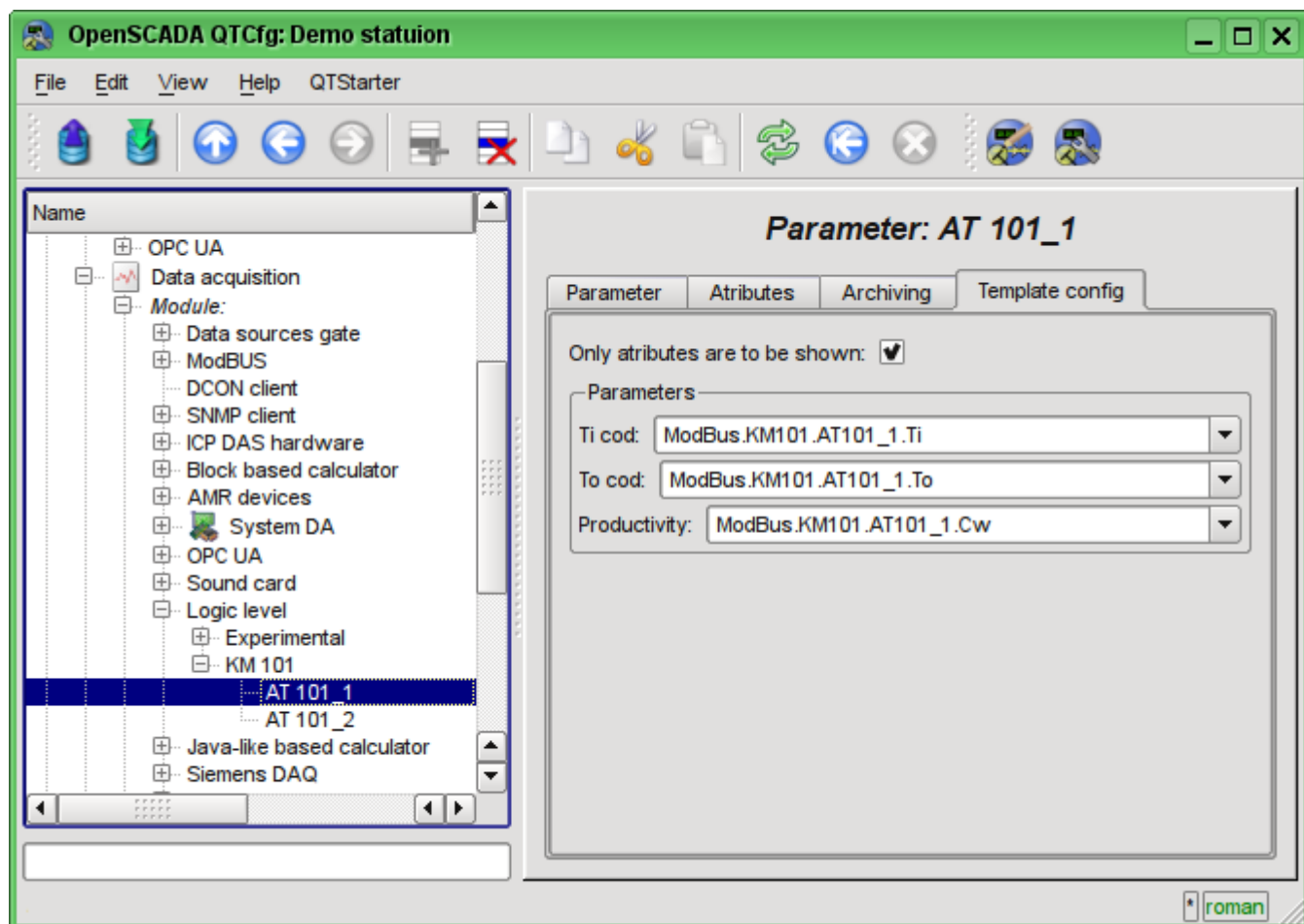


Fig. 4.2.6. The "Template config" tab of the "LogicLev" controller's parameter page with the links details.

Let's save the created objects of the controller and parameters. After this, run the controller for execution by setting the controller's flag "Run" in the "State". If we do not miss something, the calculation is successfully started and in the "State" we'll get something like the one on Fig.4.2.7.

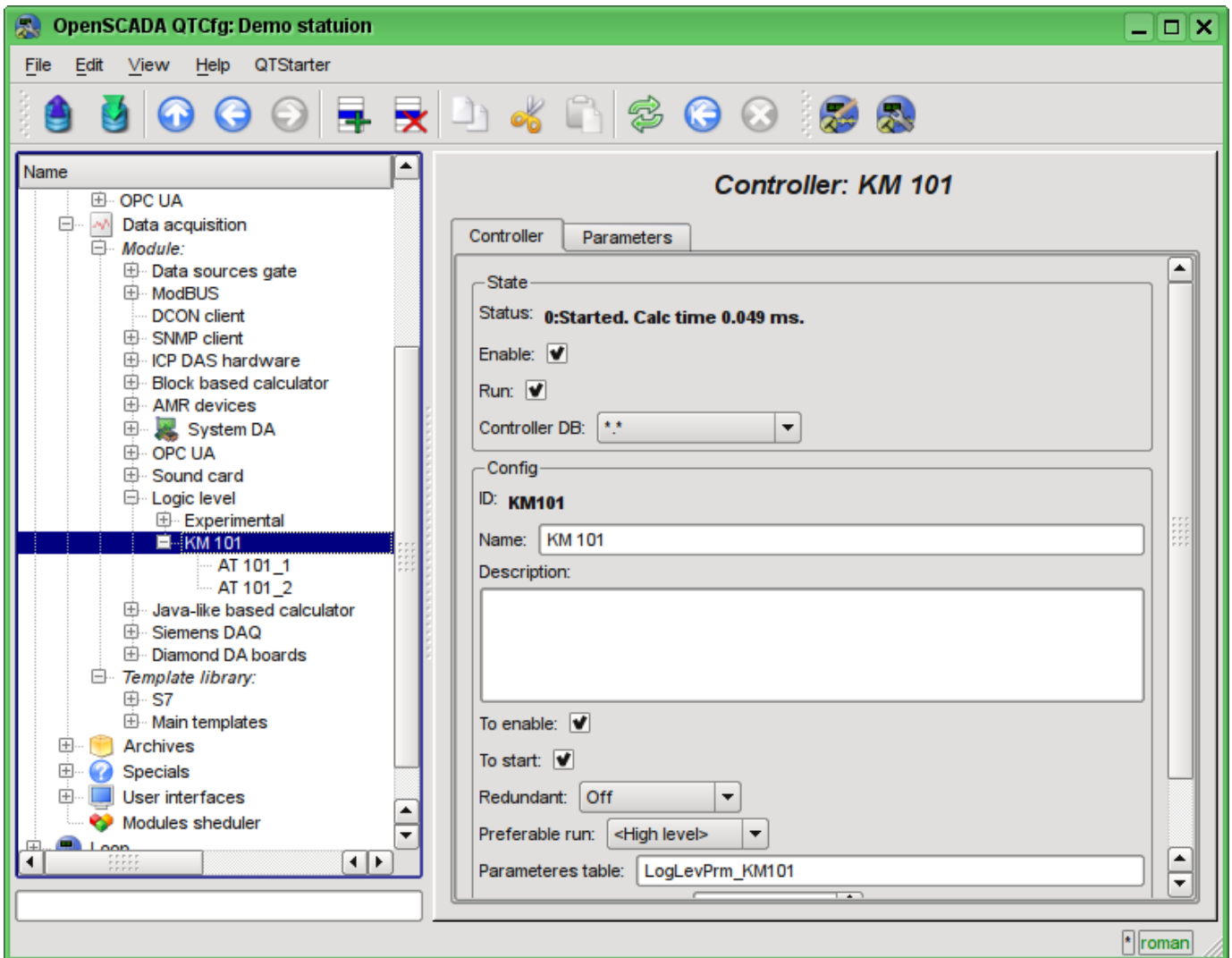


Fig. 4.2.7. The page of the controller's object if the calculation of the controller in the "LogicLev" module is successful.

In case of successful processing of the template's code in the parameters we'll obtain the processed data in the infrastructure of OpenSCADA. You can see these data on the tab "Attributes" of our parameters AT101_1 (Fig.4.2.8) and AT101_2.

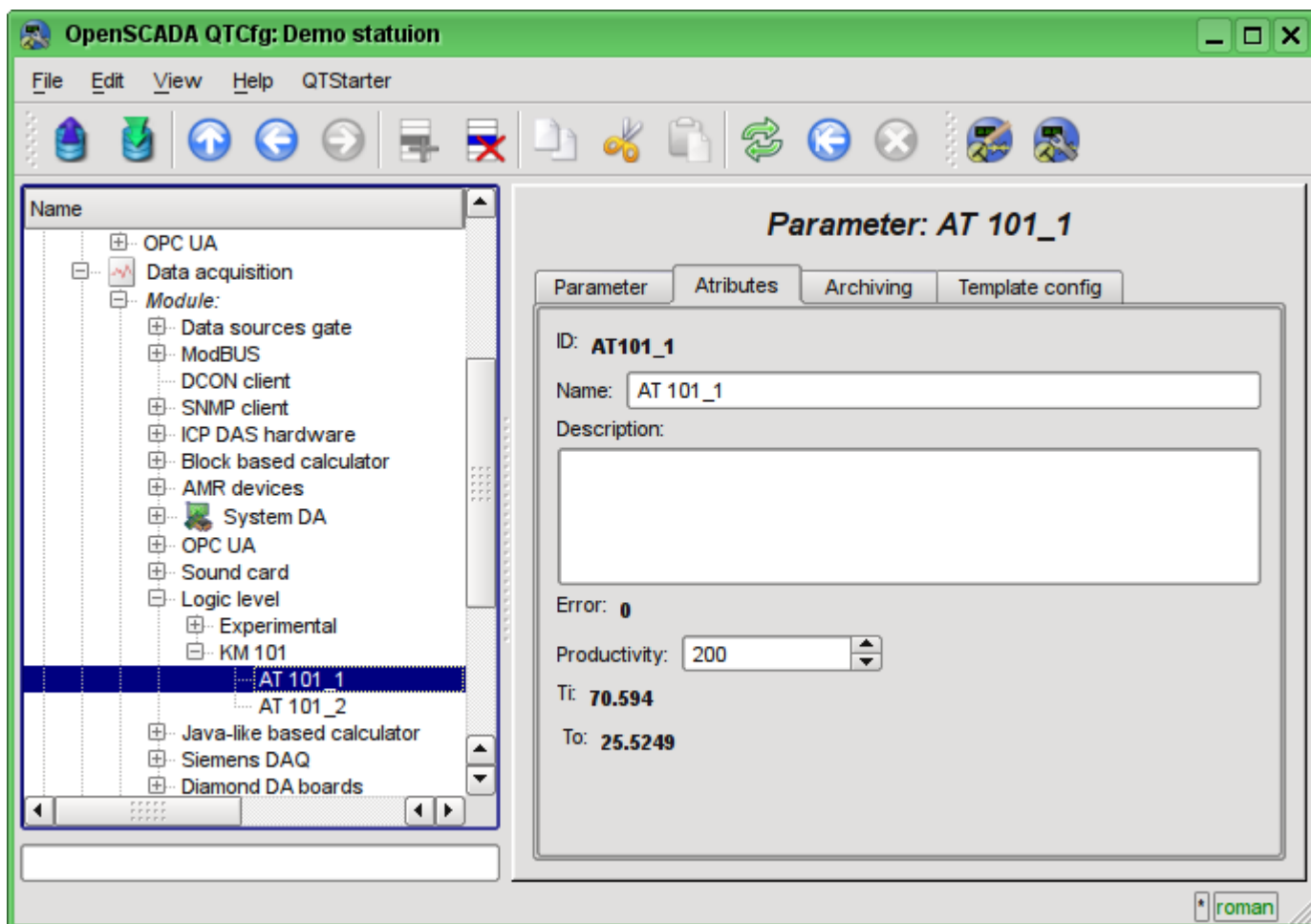


Fig. 4.2.8. The page of the attributes of the parameter AT101_1 of "LogicLev" module. The configuration of data processing is complete.

4.3. Enabling the TP data archiving

Many tasks require to keep the history of parameters of the TP. To activate the archiving of the attributes "Ti" and "To" of the AT101_1 and AT101_2 parameters in the previously created controller of the "LogicLev" module it is enough on the "Archiving" tab of the configuration page to choose which attributes are to be archived and by what archiers (Fig.4.3.1). We'll choose the archiving of "Ti" and "To" attributes in the "FSArch.1s" archiver.

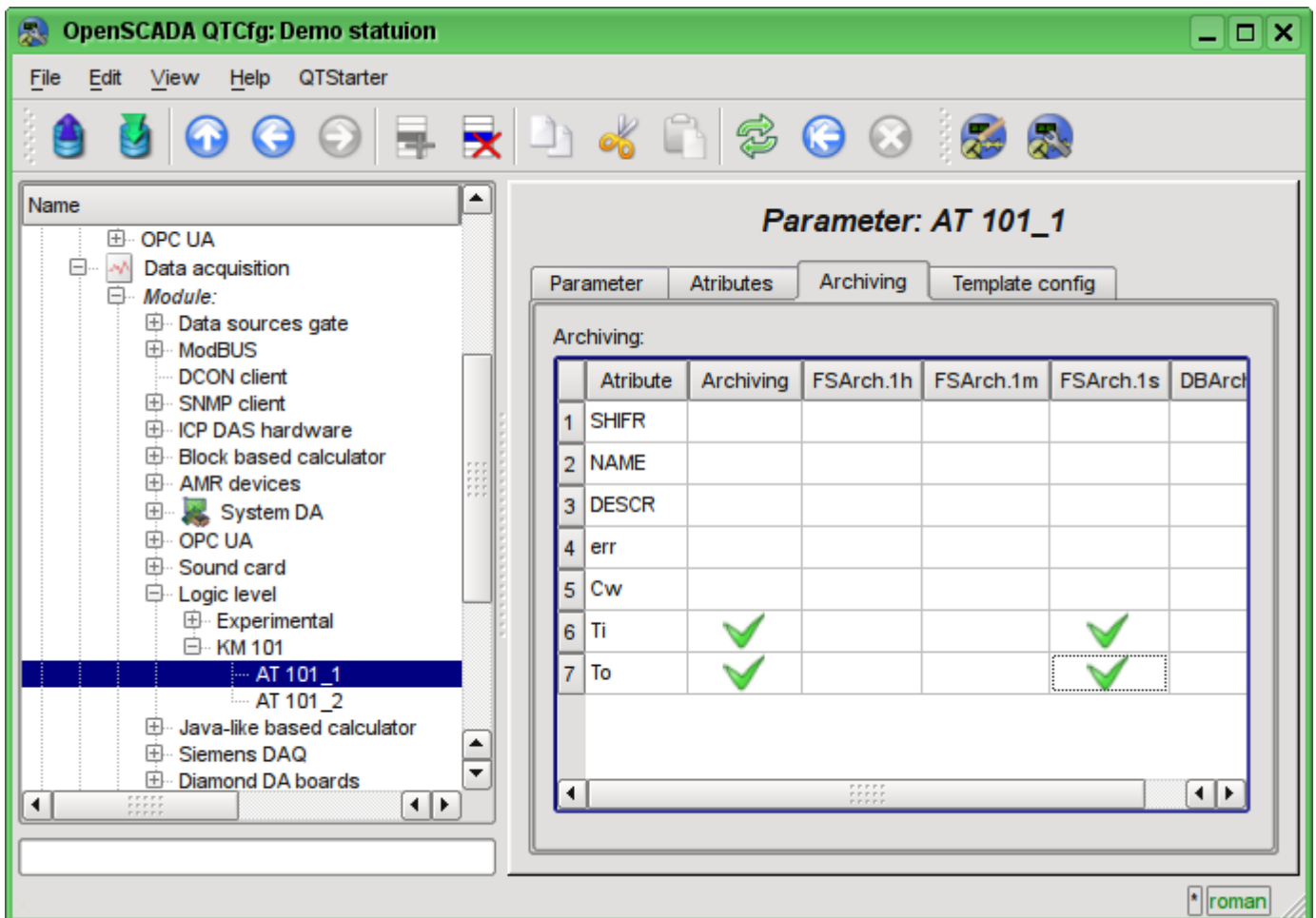


Fig. 4.3.1. The "Archiving" tab of the AT101_1 parameter of the "LogicLev" module.

As the result of this operation it will be automatically created the objects of archives for the selected attributes. For example, the archive's object for the attribute "Ti" of the AT101_1 parameter is presented at Fig.4.3.2.

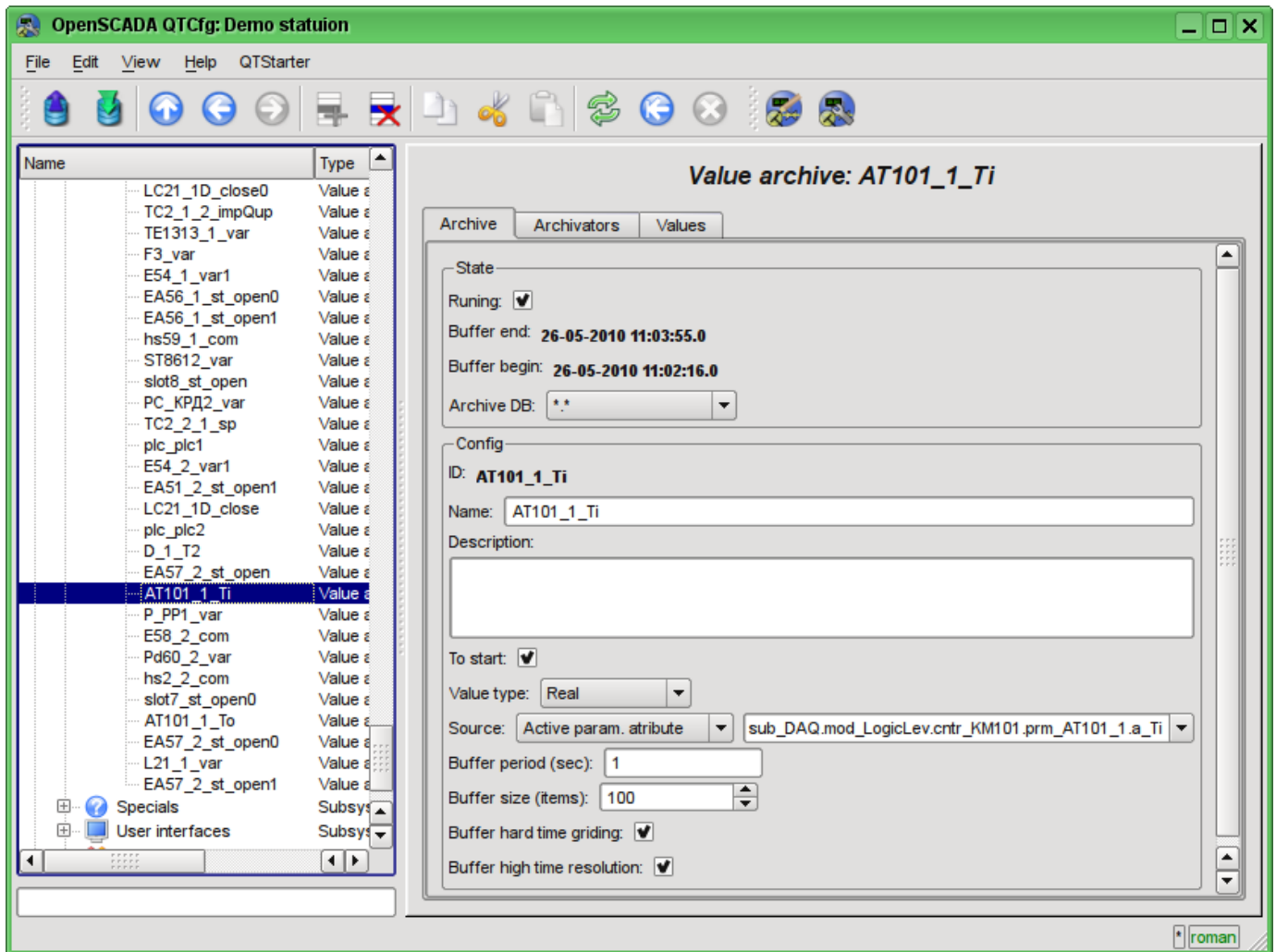


Fig. 4.3.2. The page of the archive's object of the "Ti" attribute of the AT101_1 parameter.

Usually the settings of the archive do not need to be change, but if you need the special configuration, it can be done on the aforesaid page. Often you may need to obtain the information about the archive. For example, find the archive's size, both in time and in the bytes, as well as to look at the graph(diagram) of the parameter (Fig.4.3.3).

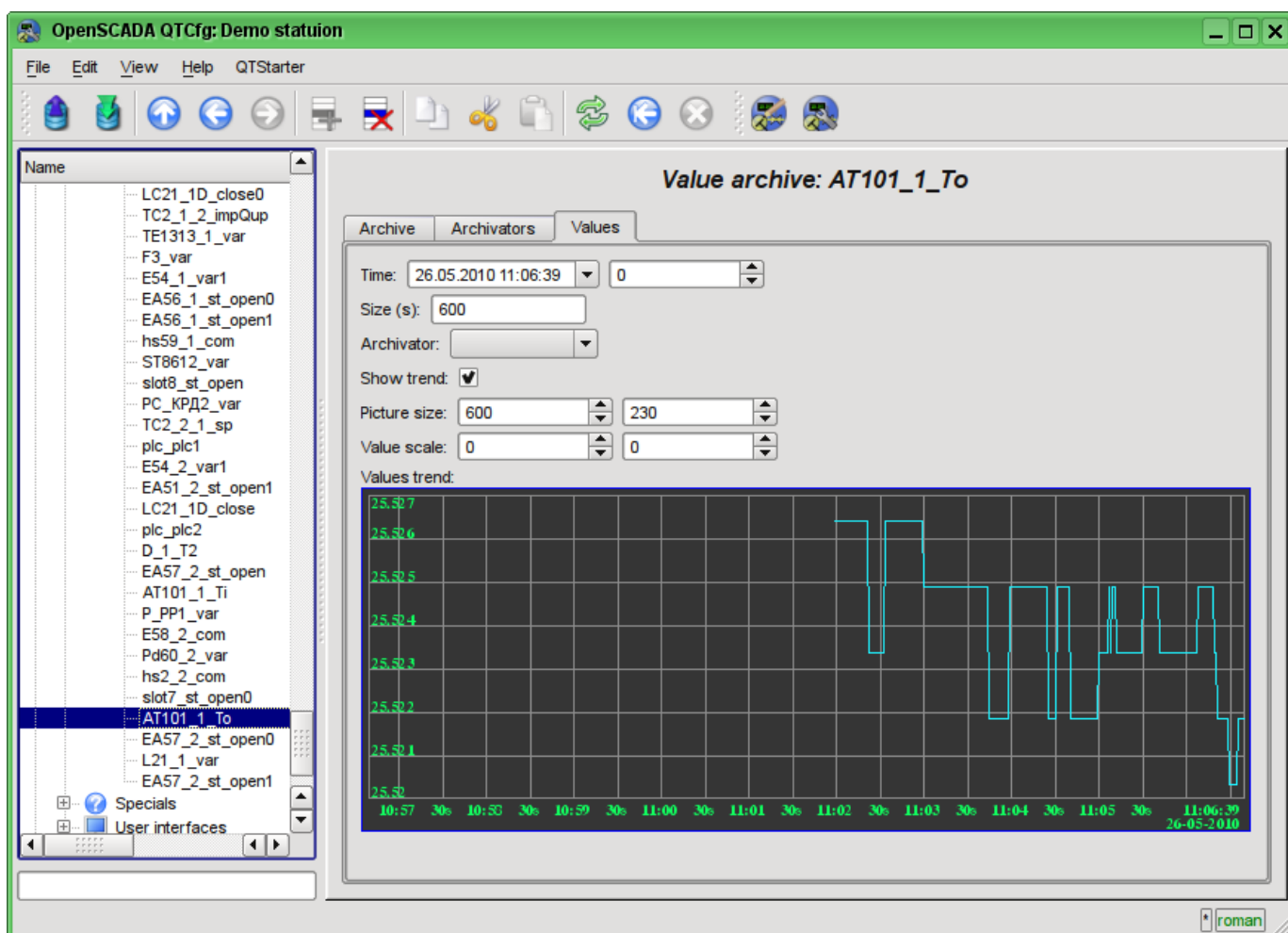


Fig. 4.3.3. The "Values" tab of the page of the archive's object of the "Ti" attribute of AT101_1 parameter.

5. The formation of visual presentation

The formation of visual presentation may be performed at three levels of complexity and the user can select any of them, depending on the level of his knowledge and availability of libraries with ready-made images and templates.

The first level requires a minimum qualification of the user, but implies the presence of template frames' libraries, which are needed to solve his task. Within the limits of the first level the user only has to know how to connect the dynamics to the template frames' pages and how to add new pages of the template frames.

The second level provides the additional ability to create new frames based on the finished complex elements, simply by their placement in the frame. To achieve this qualification level users will need libraries of complex elements needed to solve his tasks.

The third level requires that user is able to use of all the tools of the development environment of visual interfaces of OpenSCADA, including the creation of new complex elements and developing of the new user interfaces in the project.

All works on the visualization interface we will make in an environment of the "Vision" module of subsystem "User interfaces". To open the "Vision" interface window you should click the second icon on the right on the configurator toolbar. The result is the window previously shown in Fig.3.3.

5.1. Adding the template page in the project and linkage of the dynamics

Let's examine the first level of complexity task, when in the already designed interface it is necessary to link the dynamics to the template page. The concept of "Page's template" means the page on the basis of which with the help of inheritance it can be created a lot of final visualization pages with an individual list of the dynamics. The examples of these pages are: "Graphics group", "Contours group", "Overview frames panel" and "Result graphics". In the Fig.5.1.1 the template page "Graphics group" in the project tree "Signal groups (template)" is presented.

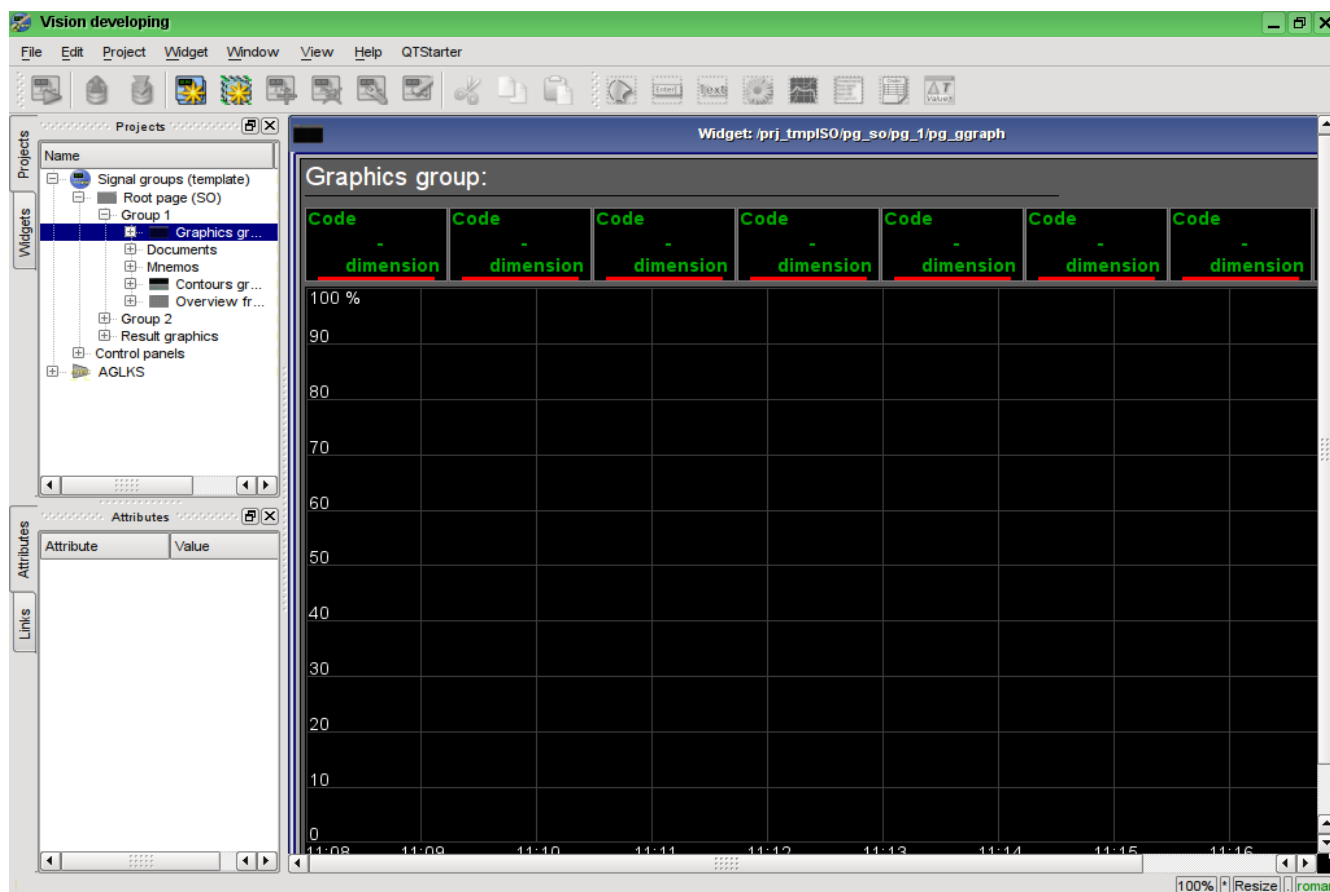


Fig. 5.1.1. The template page "Graphics group".

The "Graphics group" template page provides an opportunity to link up to eight signals for simultaneous display them on the diagram. Elements at the top will automatically hide for unspecified links.

Let's create the new group of graphs "Graphics 2" in the template container "Graphics group" of the first group of the root page of "Signal groups (template)". To do this, let's in the context menu of the "Graphics group" item select "Add visual item" (Fig.5.1.2). To enter the ID and name of the new visual item the dialog will appear (Fig.5.1.3). Enter the ID "2" and the name "Graphics 2".

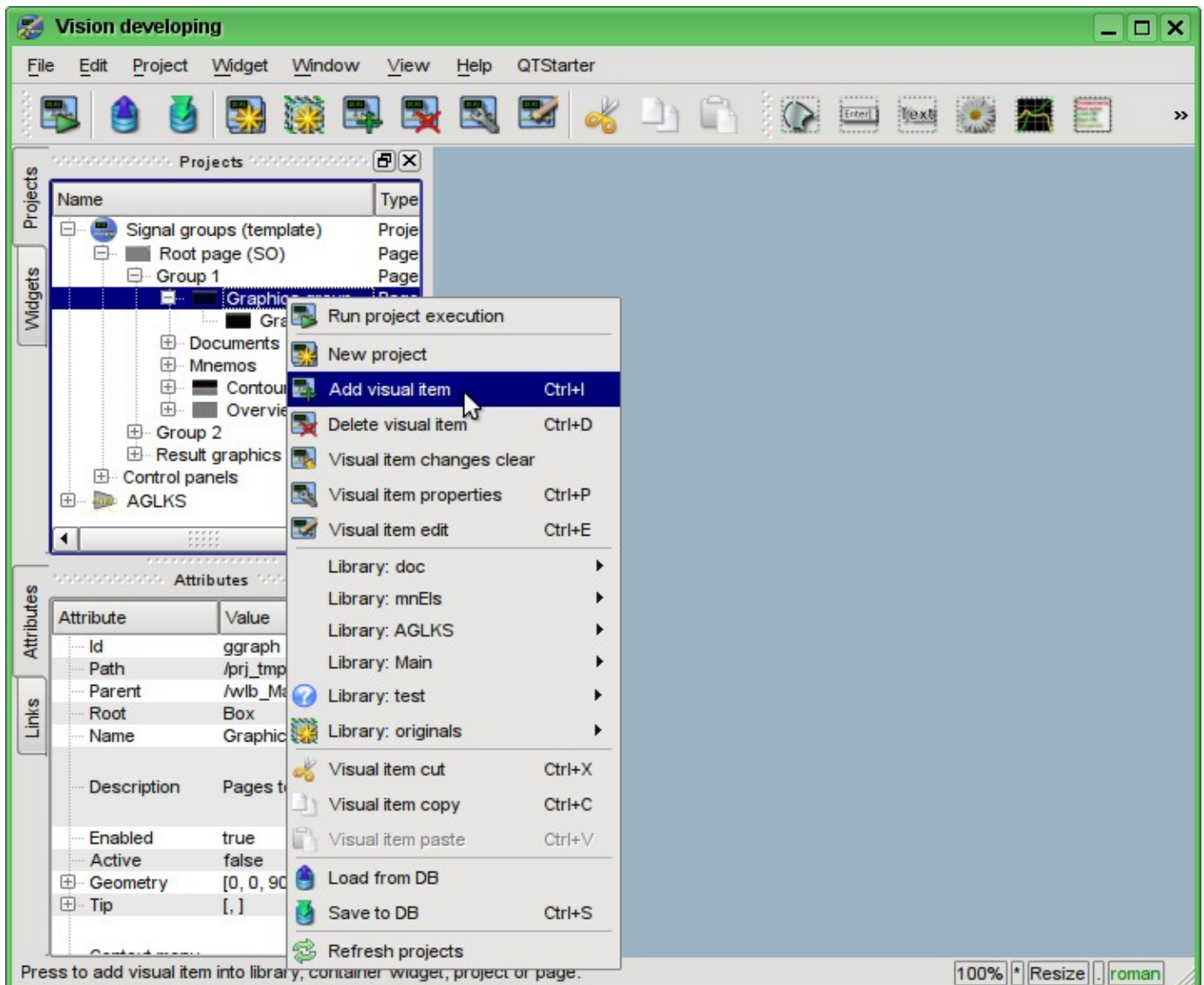


Fig. 5.1.2. Adding the "Graphics 2" group of graphs.

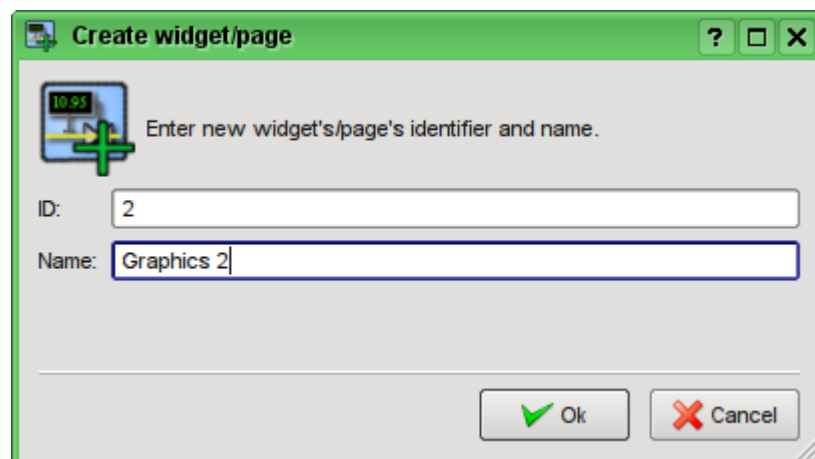


Fig. 5.1.3. Input dialog of the ID and name.

After confirming the name input it will be created the new page. However, for its activation, we need to enable it. You can enable this page in the dialog of the properties editing page (Fig.5.1.4). To open this page it is possible by selecting the menu item "Visual item properties "in the context menu of the newly created page.

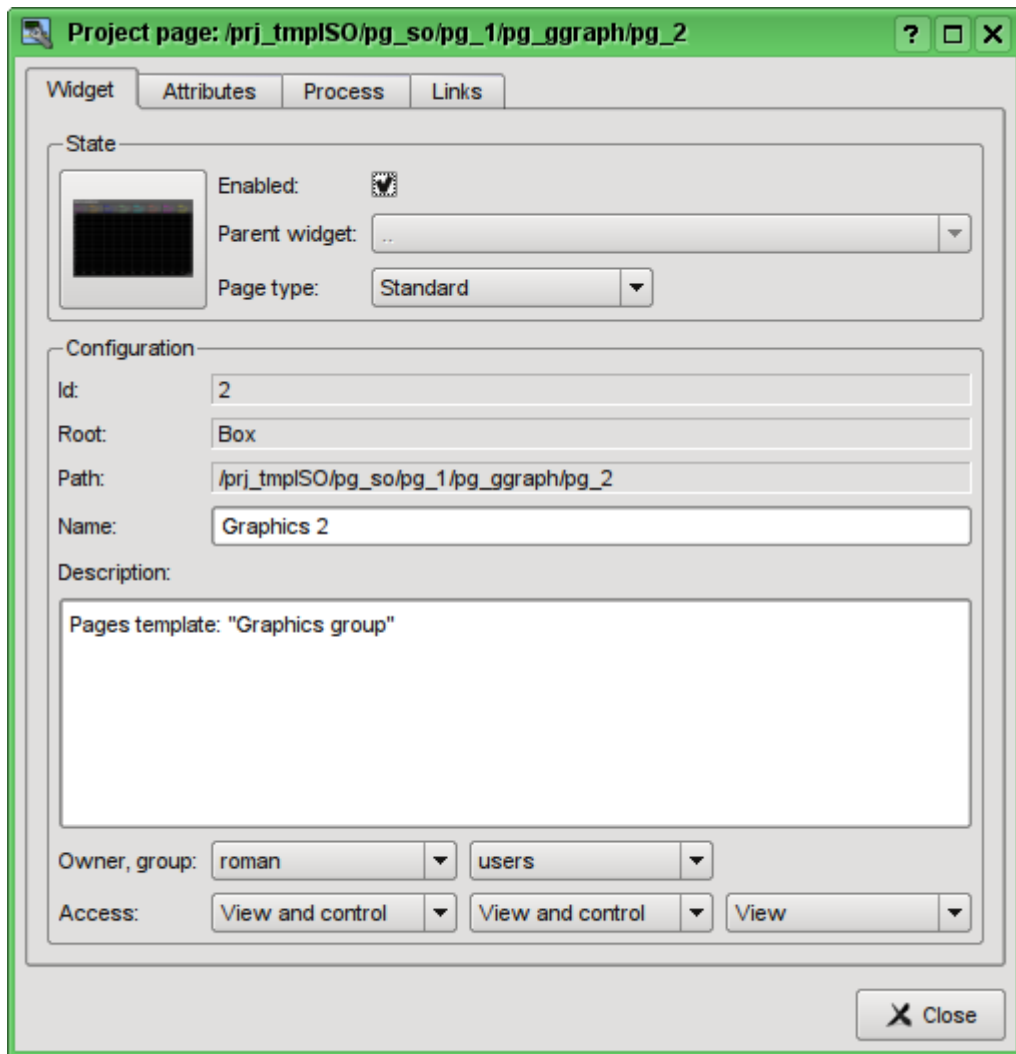


Fig. 5.1.4. Dialogue of the properties editing of the visual element.

After enabling the page you are ready to set links to the created in the previous chapter parameters of controllers. To do this, without leaving the dialog to edit the properties of the newly created page (Fig.5.1.4), click on the "Links" tab (Fig.5.1.5). On this tab, we can see the tree with the elements "el1" ... "el8". Unwinding any of the elements we'll see the "Parameter" branch, in this branch we need to specify or select the address of our attributes "Ti" and "To". Total we will fill the four elements. When filling out the elements the part of properties must be specified as constants. For example, it is necessarily needed to be specified:

- *name* - "val:AT101_1 Ti"
- *ed* - "val:deg.C"
- *max* - "val:150" (for Ti) and "val:100" (for To)
- *min* - "val:0"

If you foresee the existence of the attributes specified in the controller parameter's template as constant, it will be possible to specify only parameter, and the attributes will be set automatically.

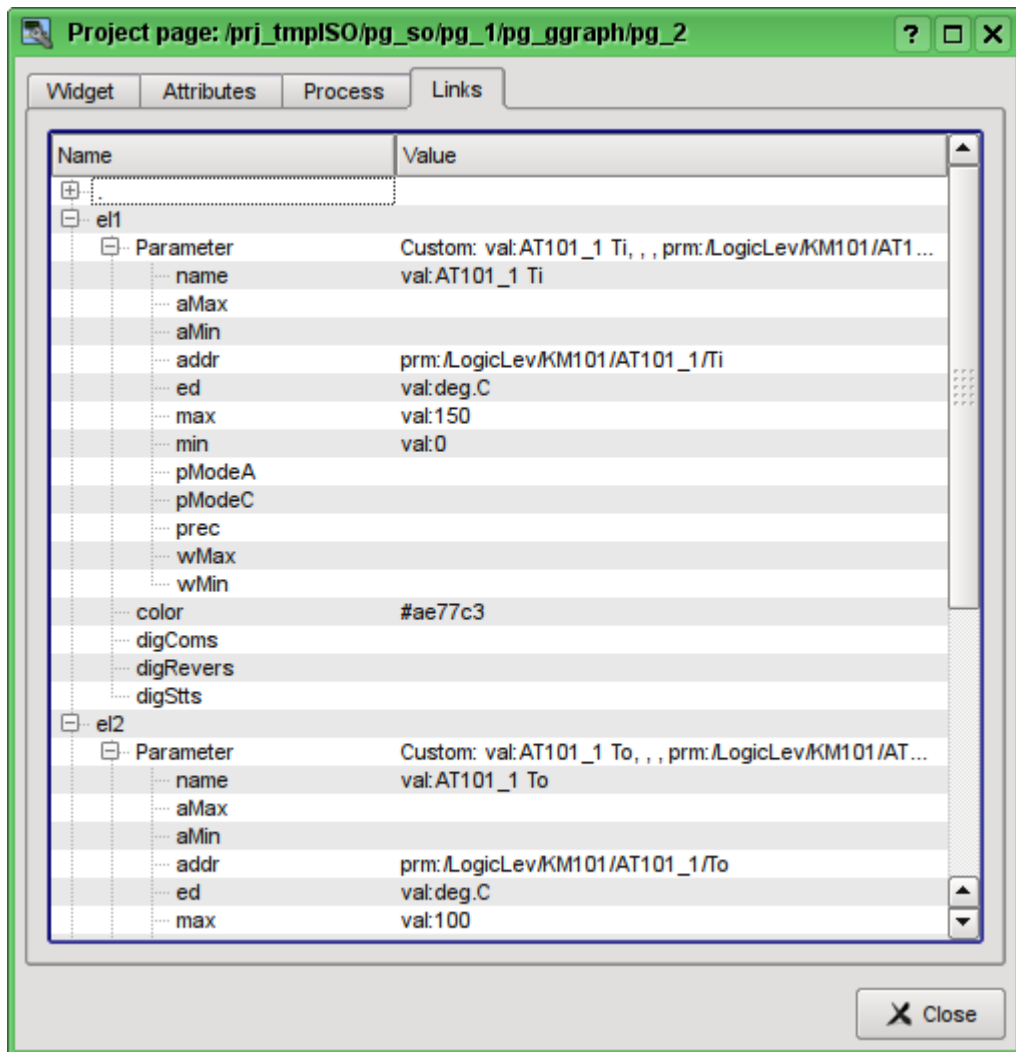


Fig. 5.1.5. The "Links" tab of the dialog of edit the properties of visual item.

Having finished the links entering, we can see the result of our efforts. To do this we'll close the editing properties dialog and run the "Signal groups (template)" for execution, about the run button we remember from the previous chapters. Then let's choose the graphics and switch to the second page. With error-free configuration, we should see something similar to that shown in Fig.5.1.6.

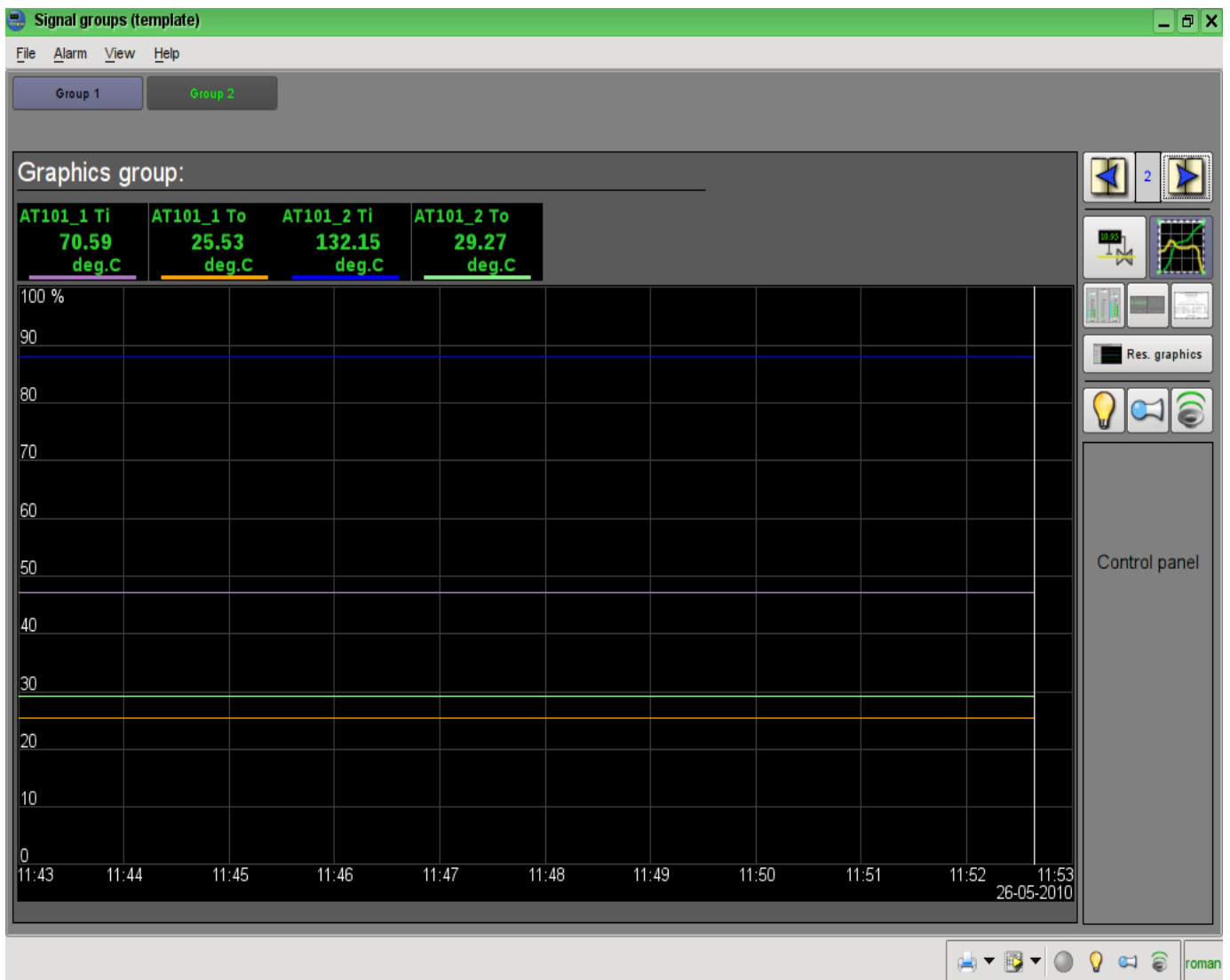


Fig. 5.1.6. The created group of graphs with the four signals linked.

5.2. The creation of the new frame, the mnemonic scheme

Let's raise the bar and create the new frame, on which we'll put the basic elements of our controllers' values displaying. Such frames are usually called the mnemonic schemes and in addition to the dynamics displaying, and even in the first place, contain the static image of the technological process in the mnemonic representation. We are not going to focus on the creation of statics and we'll add the dynamic elements and link them to the parameters of our controllers. We'll put the created frame to the tree of already known to us project.

New frames, destined later to be placed in the project, are to be created in the library of widgets. Let's create the new library of widgets "KM101" by the selecting of the vertical tab "Widgets" and in the context menu of the window of widgets' libraries click "New Library" (Fig.5.2.1). In the dialog of entering the name we'll indicate the identifier "KM101" and the name "KM 101" and then confirm.

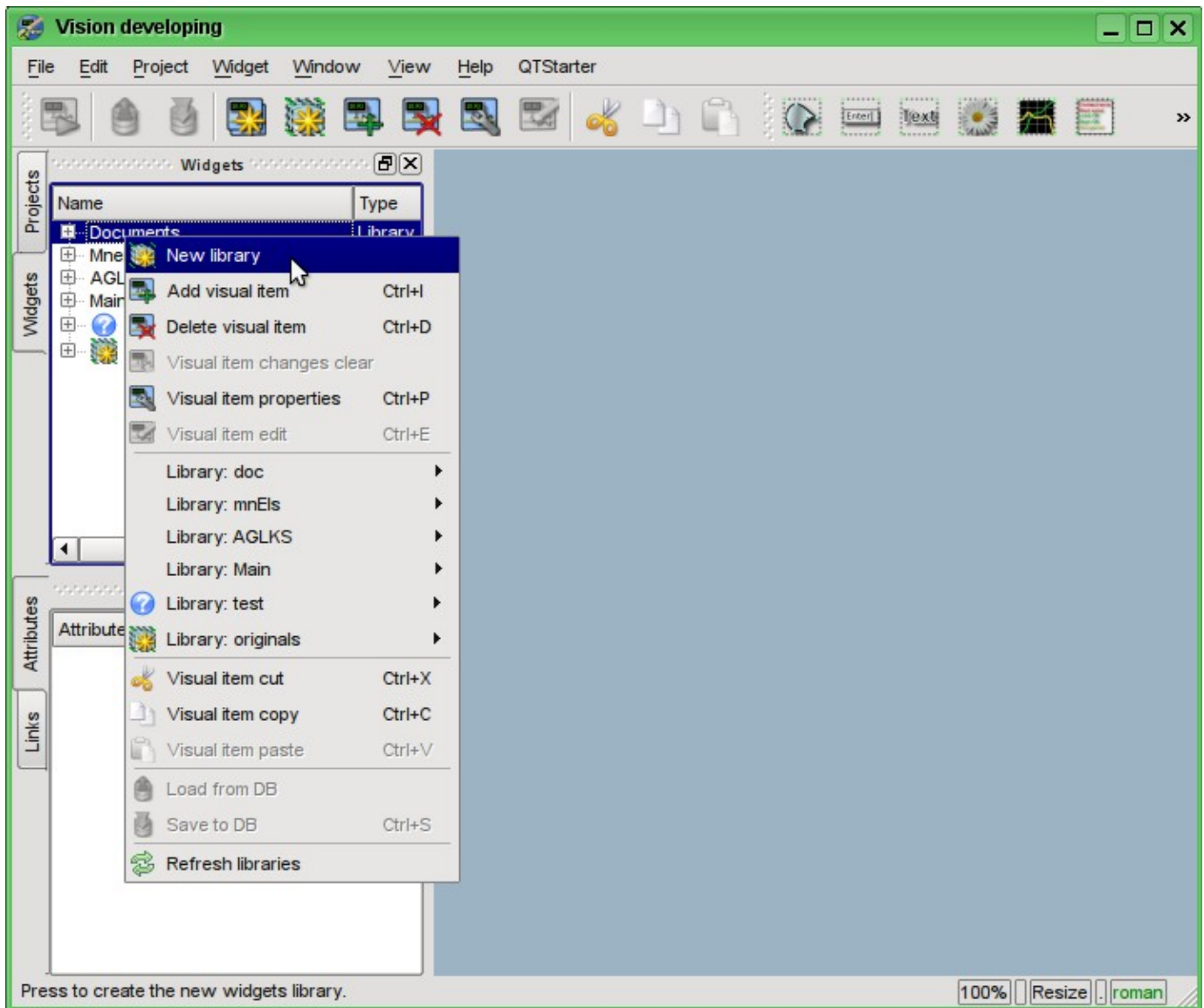


Fig. 5.2.1. Adding the new library of widgets.

Next we'll add the new frame "AT101" by selecting "Library: originals" -> "Elements box" in the context menu of the created library "KM101" (Fig.5.2.2). In the dialog of entering the name we'll indicate the identifier "AT101" and the name "AT 101" and then confirm. At the heart of any frame and the page must be based on an element of "Elements box" ("Box"), and therefore we have chosen it.

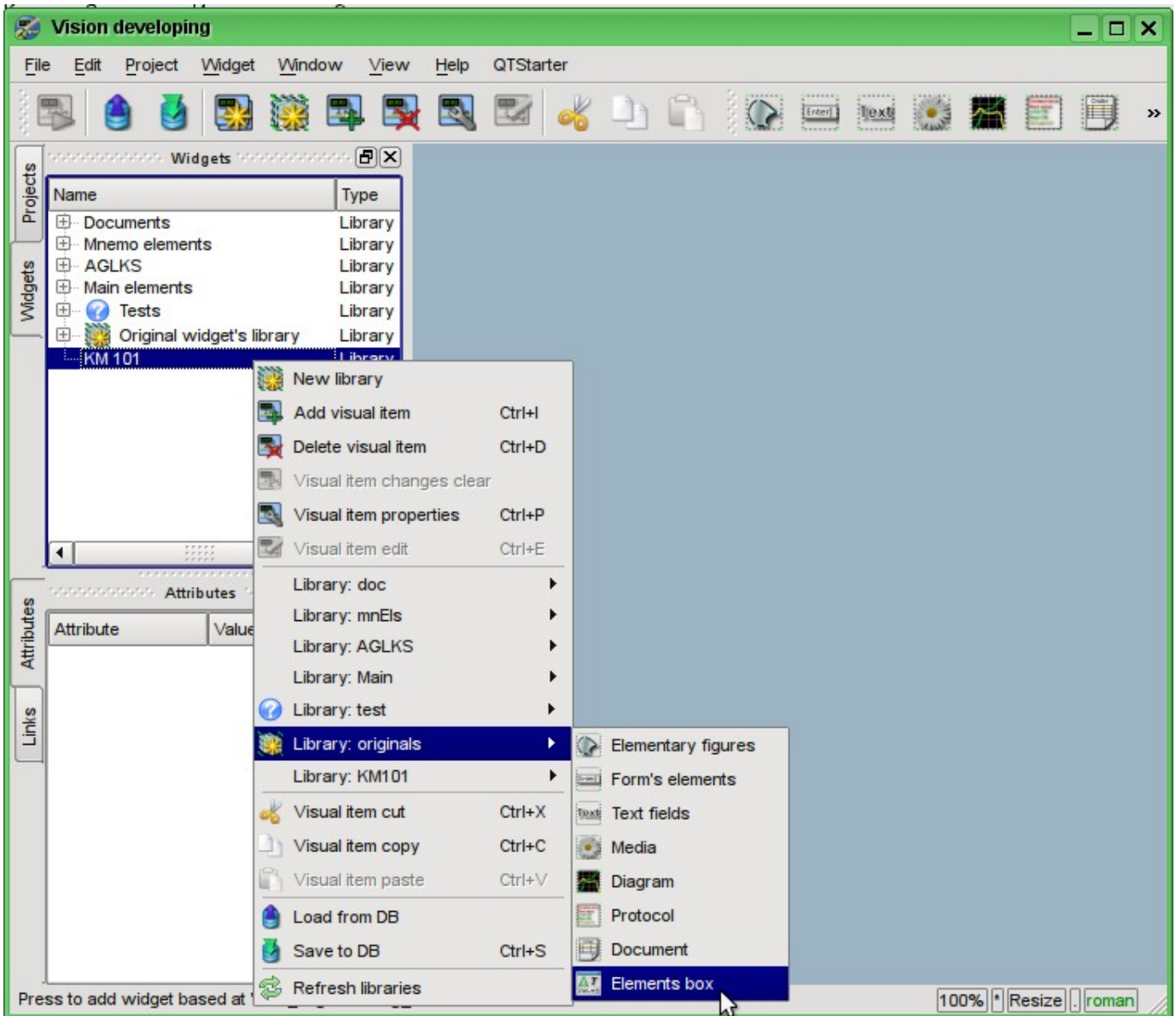


Fig. 5.2.2. Adding the new frame.

Immediately after the creation of the new frame element it is necessary to set its basic properties, characteristic to the mnemonic scheme frame. Properties or attributes of any visual element can be specified in the toolbar "Attributes", having pre-selected the visual element. Let's select the created frame "AT 101" and set the following properties:

- *Geometry:width* - 900;
- *Geometry:height* - 600;
- *Background:color* - "#5A5A5A";
- *Border:width* - 1;
- *Border:color* - "black".

The result will be an empty frame (Fig.5.2.3), ready to add items to it. To edit or view the the frame you should in the frame's context menu select the "Visual item edit".

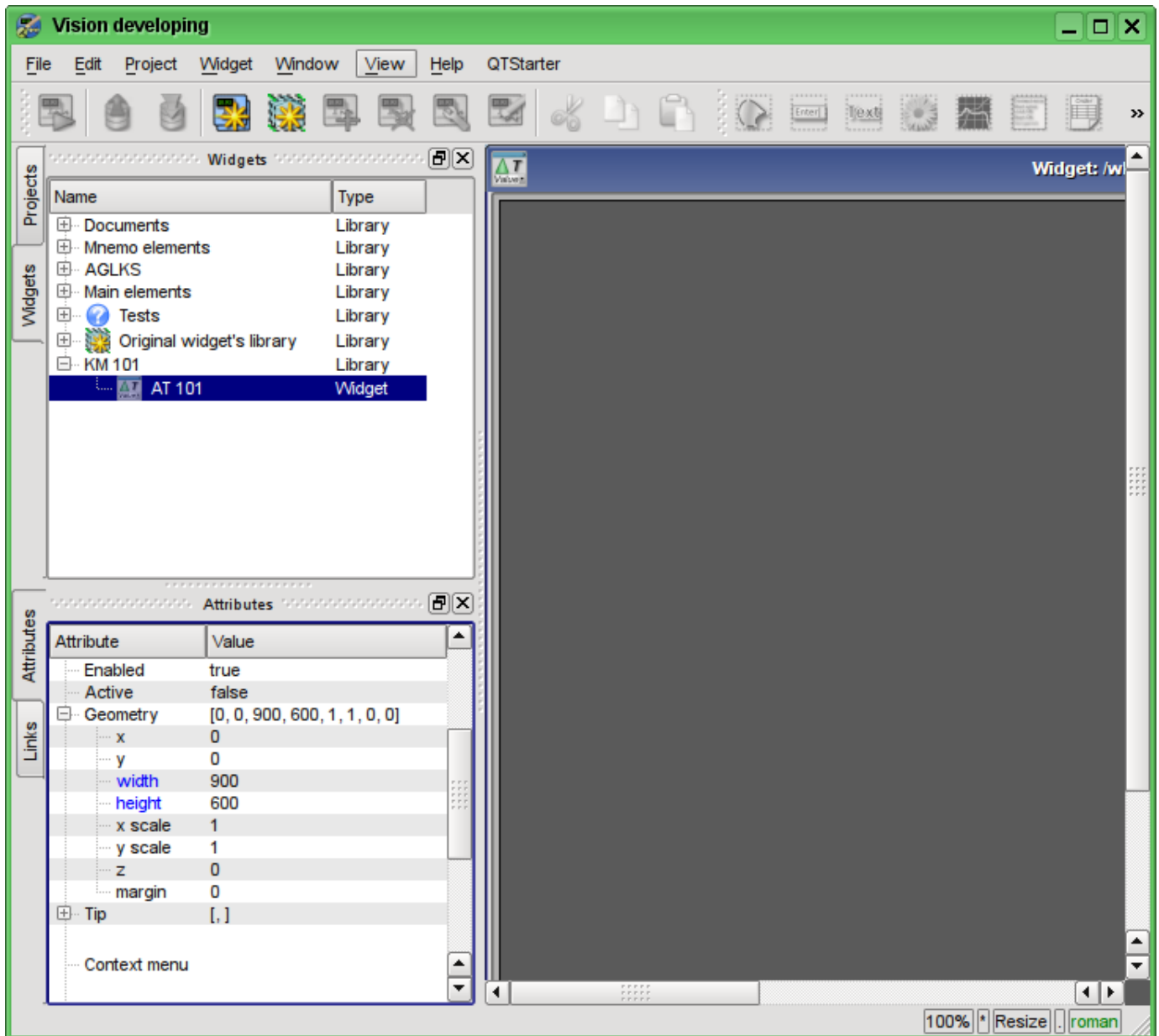


Fig. 5.2.3. The view of the new frame and set attributes for the mnemonic scheme.

Now let's add on frame the elements for the value of the analog parameters displaying for our four signals. To place an element for displaying an analog signal to the mnemonic scheme it is necessary to select our mnemonic scheme, and then in the window's menu to select the "Widget" -> "Library: Main&" -> "Analog show" after which the cursor with an image of this element will appear, which should be moving to the desired location on the mnemonic scheme and then the left mouse button should be pressed. At the time of adding the dialog asking the name of the new element will appear. We'll add this way the four elements which we'll call: "A1_Ti", "A1_To", "A2_Ti" and "A2_To". The added elements can be subsequently positioned as needed by simply selecting and dragging them by the mouse. After such manipulations, we should get the mnemonic scheme with the view, similar to Fig.5.2.4.

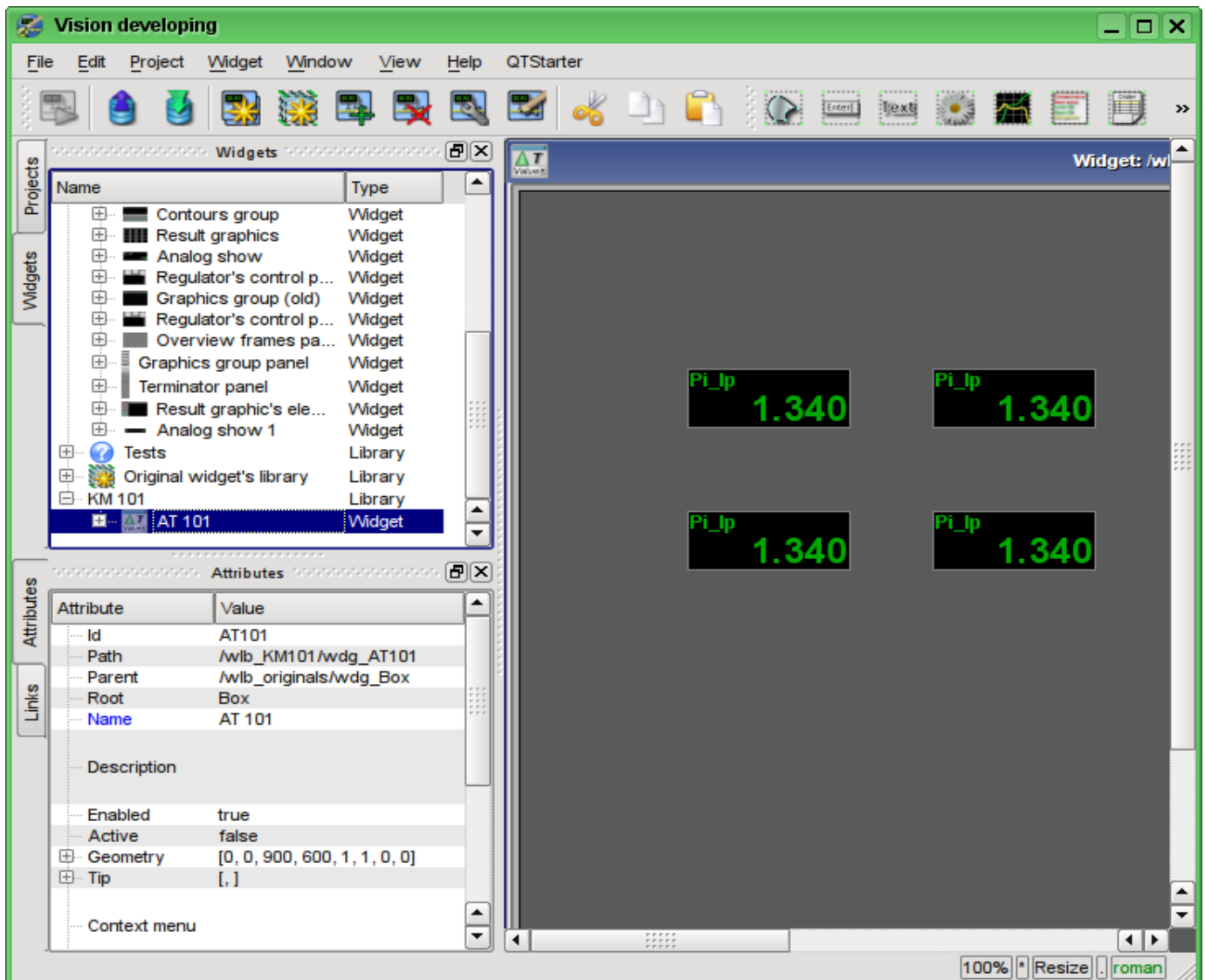


Fig. 5.2.4. The view of the new frame and set attributes for the mnemonic scheme.

This procedure of the creating the mnemonic scheme we'll consider to be finished. Save the new library of widgets "KM101" and proceed to the stage of the placing our mnemonic scheme in the project's tree of "Signal groups (template)".

Let's put our mnemonic scheme to the branch of the "Signal groups (template)" -> "Root page (SO)" -> "Group 1" -> "Mnemos" by selecting in the context menu for the "Mnemos" item the item "Library: KM101" -> "AT 101". The identifier for the new mnemonic scheme let's set to "2" and the name field let's leave blank. Immediately after adding it is necessary to set the basic property of the mnemonic scheme "Page:group" to the value "so".

Next you need to make an already familiar to us the operation from the previous chapter, namely the setting of links to the created in the previous chapter the parameters of controllers. To do this let's open the dialogue of the properties editing of the mnemonic scheme on the "Links" tab (Fig.5.2.5). On this tab, we'll see the tree with the elements of "A1_Ti", "A1_To", "A2_Ti" and "A2_To". Unwinding any of the

elements, we'll see the "Parameter" branch, in this branch we are to specify or select the address of our attributes "Ti" and "To", respectively. When filling out the elements the part of the properties must be specified as constants. For example, necessarily must be specified:

- *pName* - "val:AT101_1 Ti"

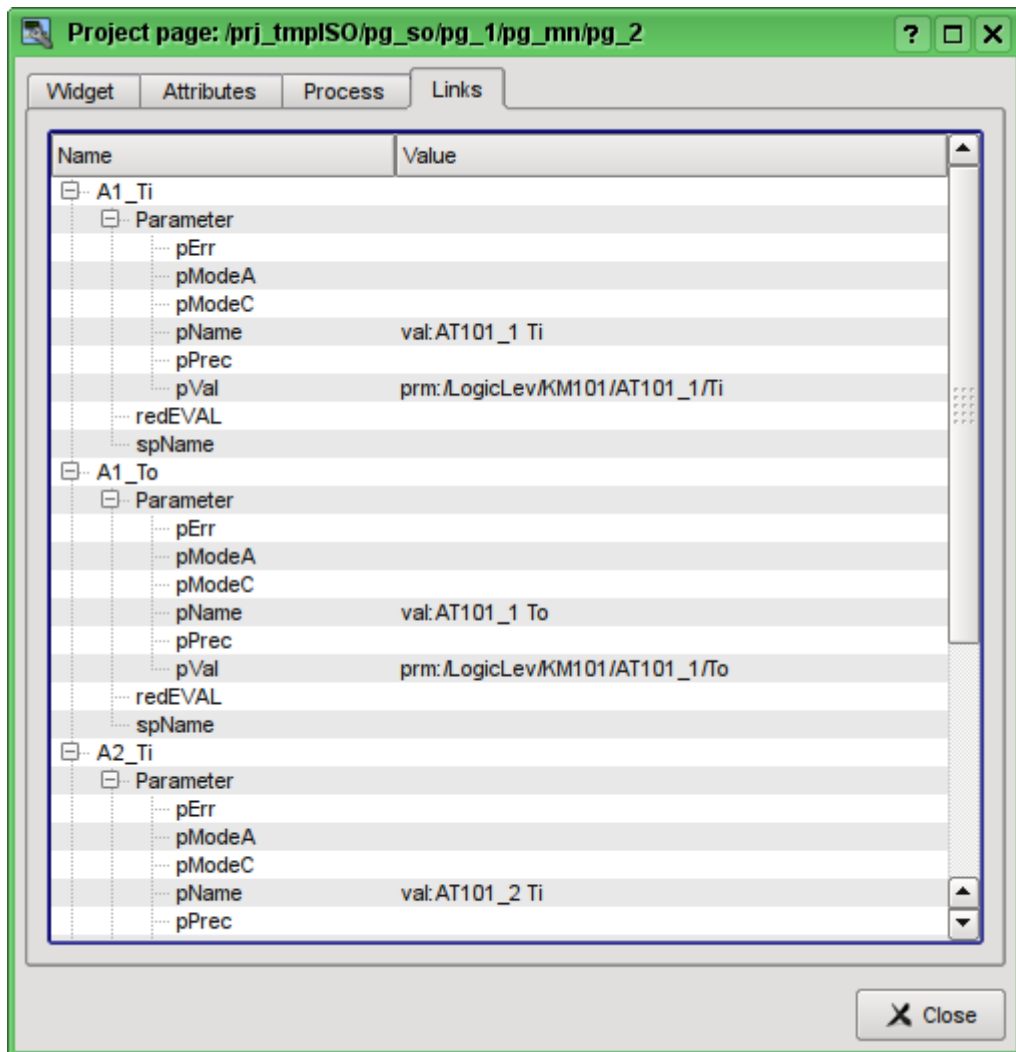


Fig. 5.2.5. The "Links" tab of dialog of editing the properties of the mnemonic scheme.

Now we can save our mnemonic scheme and verify what we have. To do this, we'll close the properties dialog and run the "Signal groups (template)" for execution. Then switch to the second mnemonic scheme by the paging buttons. With error-free configuration, we should see something similar to that shown in Fig.5.2.6.

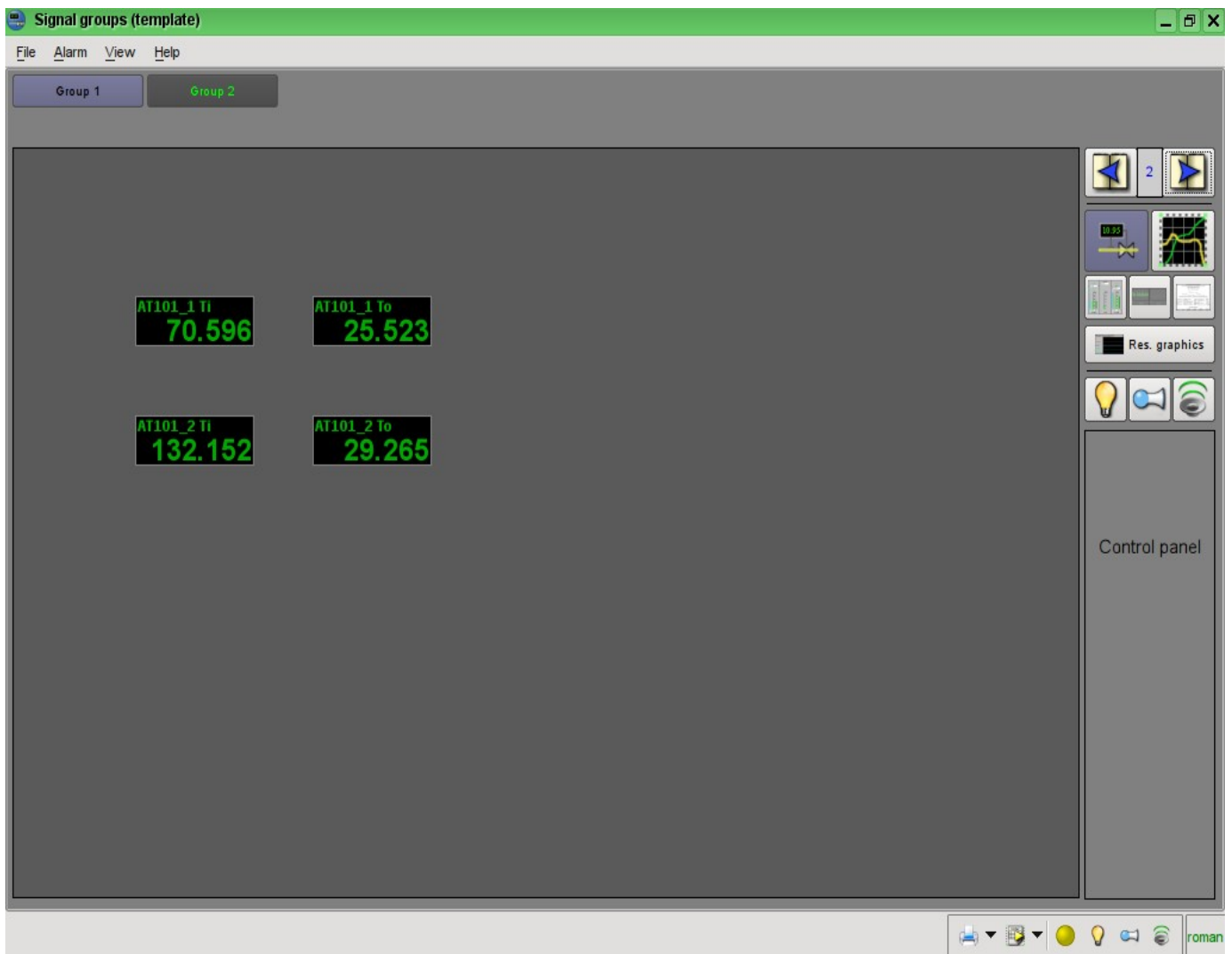


Fig. 5.2.6. The created mnemonic scheme with four linked signals.

5.3. Creation of the new complex element

Let's proceed to the objectives of the third level of complexity, namely the creation of an complex element. Creating of the new complex element, which includes a combination of basic primitives, can be made in several stages. As an example, let's examine the task, consisting of two stages:

- Creation the widget "Air cooler" on the basis of the primitive "Elementary figures".
- Creation the final grouped widget "Cooler" based on the primitive "Elements box".

5.3.1. Creation the widget "Air cooler" on the basis of the primitive "Elementary figures".

The widget will be created in our previously made library "KM101". To do this we'll make right mouse button click on this library and select the item "Library: originals"->"Elementary figures", as it is shown in Figure 5.3.1.1. For a new element let's write the "air_cooler" identifier and the name "Air cooler".

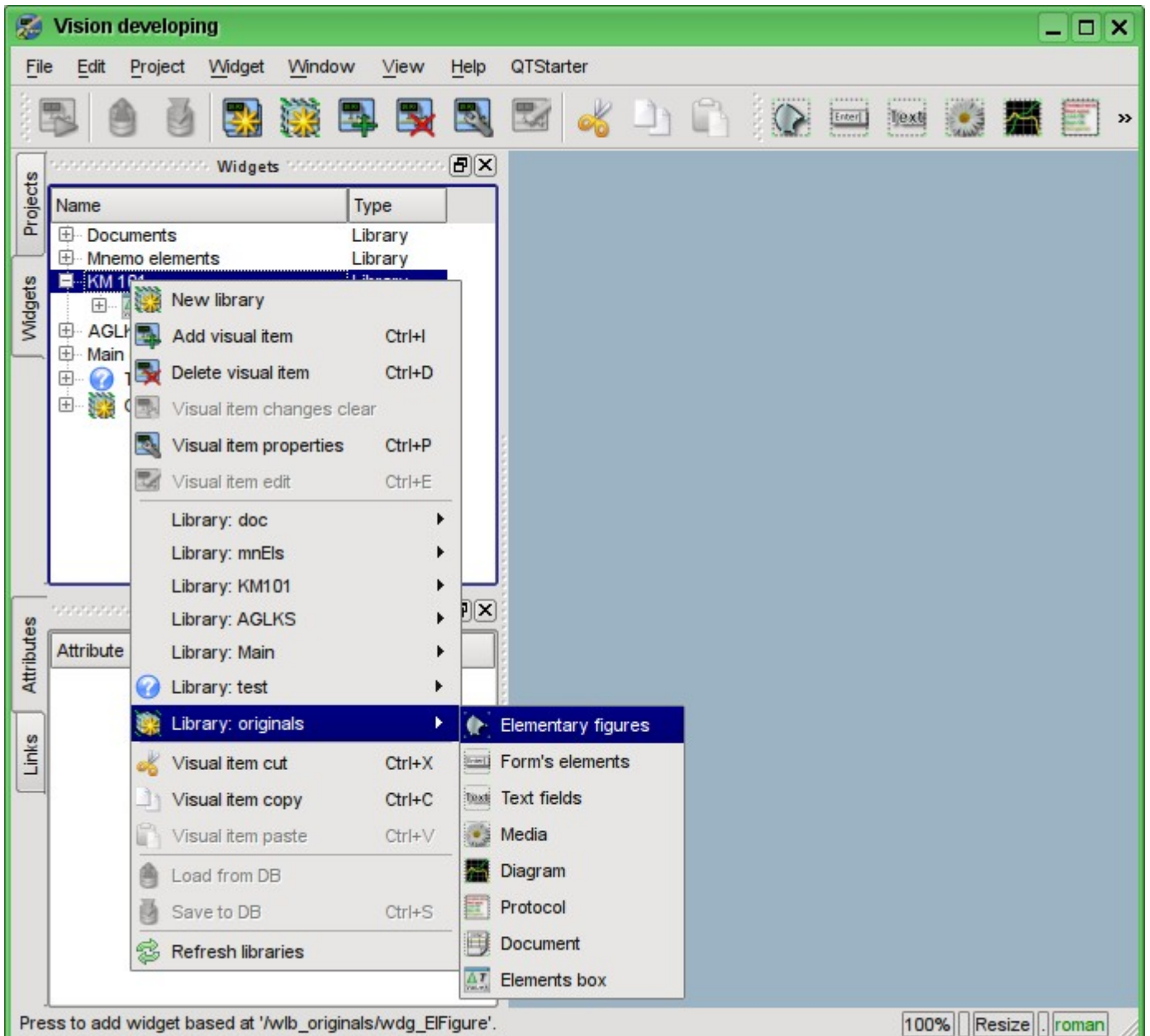


Fig. 5.3.1.1. Adding the widget based on the primitive "Elementary figures" to the "KM101" library.

After confirmation, we will have a new widget's object with the name "Air cooler". Select it in the widget library "KM101" and open for editing via the context menu of the new element. Let us now set in the "Attributes" tab in the "Geometry" section width and height of the widget to the 200 pixels (Fig. 5.3.1.2).

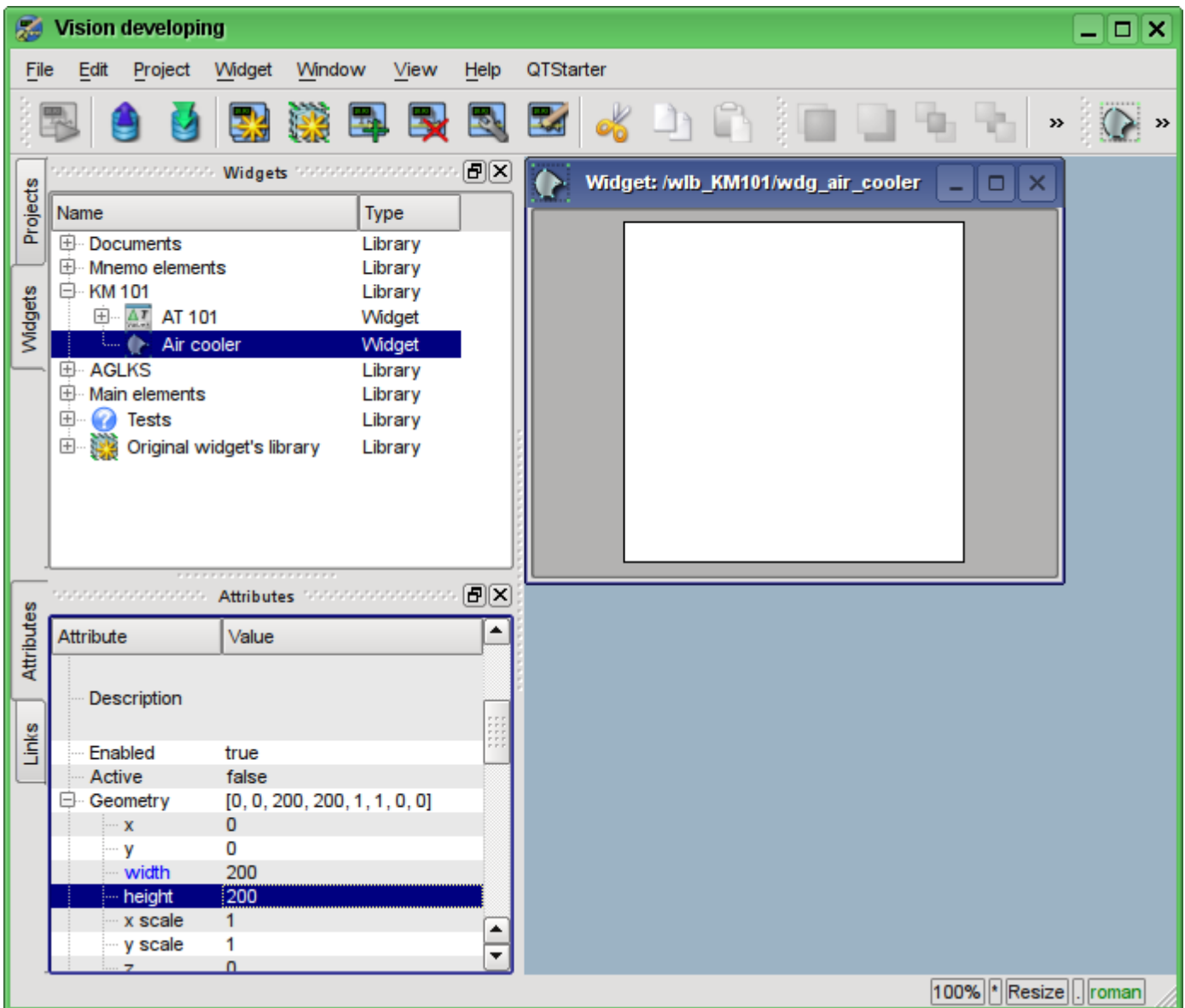


Fig. 5.3.1.2. Specifying the geometric sizes of the widget.

Now let's draw the visual presentation of the widget. This procedure can be done in two ways described below:

- To draw the desired image by the mouse, using the "Line", "Arc", "Bezier curve" and "Fill." The corresponding panel ("Elementary figure tools") appears after entering the edit mode (drawing). To enter this mode it is possible as shown it is shown in Fig. 5.3.1.3, or by double clicking the left mouse button on the body of the widget.
- Manually fill in the "Element's list", by entering the list of required elements and coordinates of points.

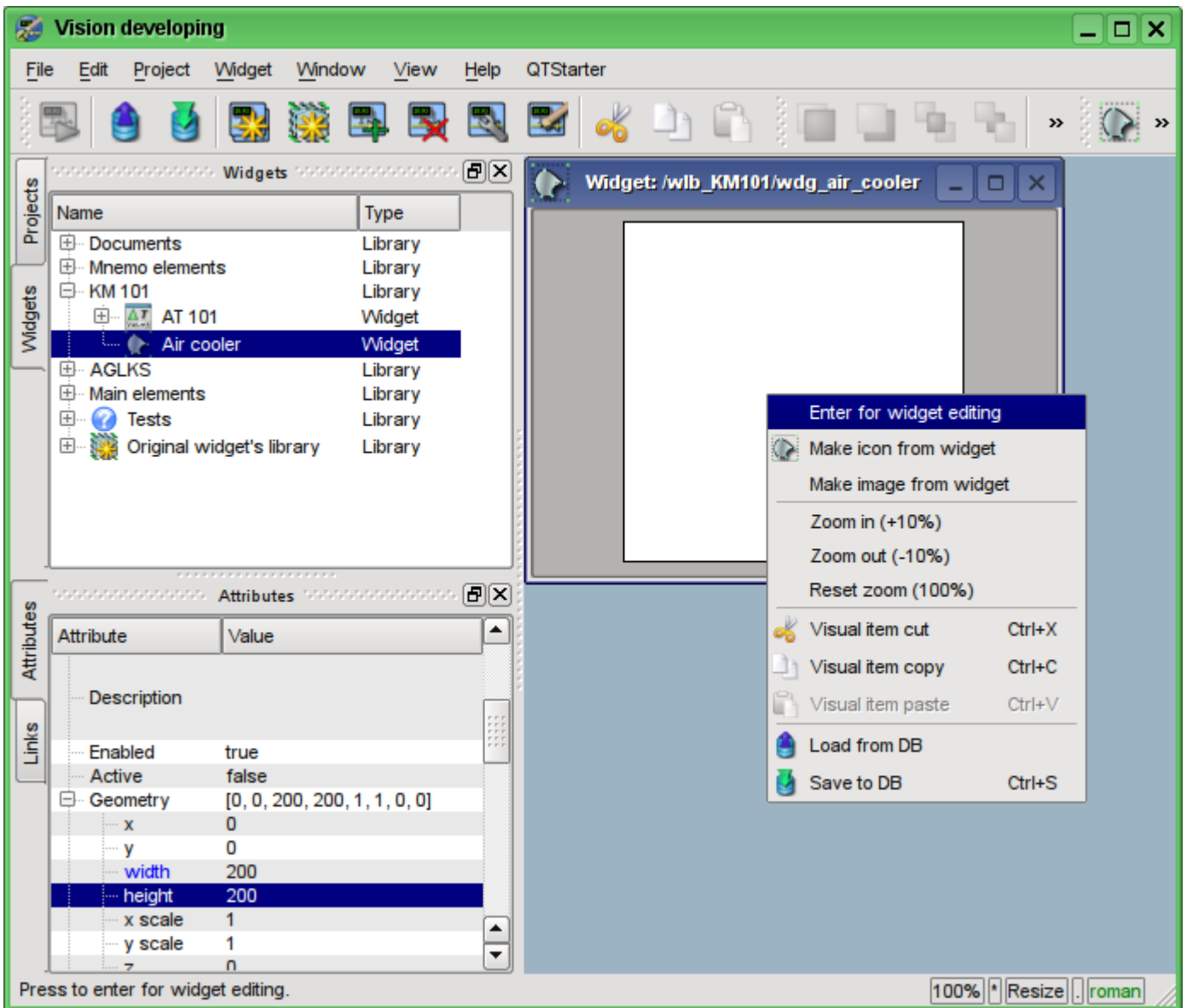


Fig. 5.3.1.3. Entrance to the mode of drawing the widget, based on the primitive "Elementary figures."

In our example, we'll use the second method. To do this in the "Element's list" of the attributes inspector let's enter the list below and press "Ctrl" + "Enter".

```

line:(20|80):(100|20)
line:(100|20):(180|80)
line:(180|80):(100|140)
line:(100|140):(20|80)
line:(100|20):(100|140)
line:(20|80):(180|80)
line:(50|165):(100|140)
line:(100|140):(150|165)
line:(150|165):(50|165)
fill:(20|80):(100|20):(180|80):(100|140)
fill:(50|165):(100|140):(150|165)

```

All the points in our case are specified in the static form, since it is not provided the dynamics and change of coordinates in the mode of execution, and all the other parameters are left by default.

As a consequence, our widget will take the form shown in Fig. 5.3.1.4.

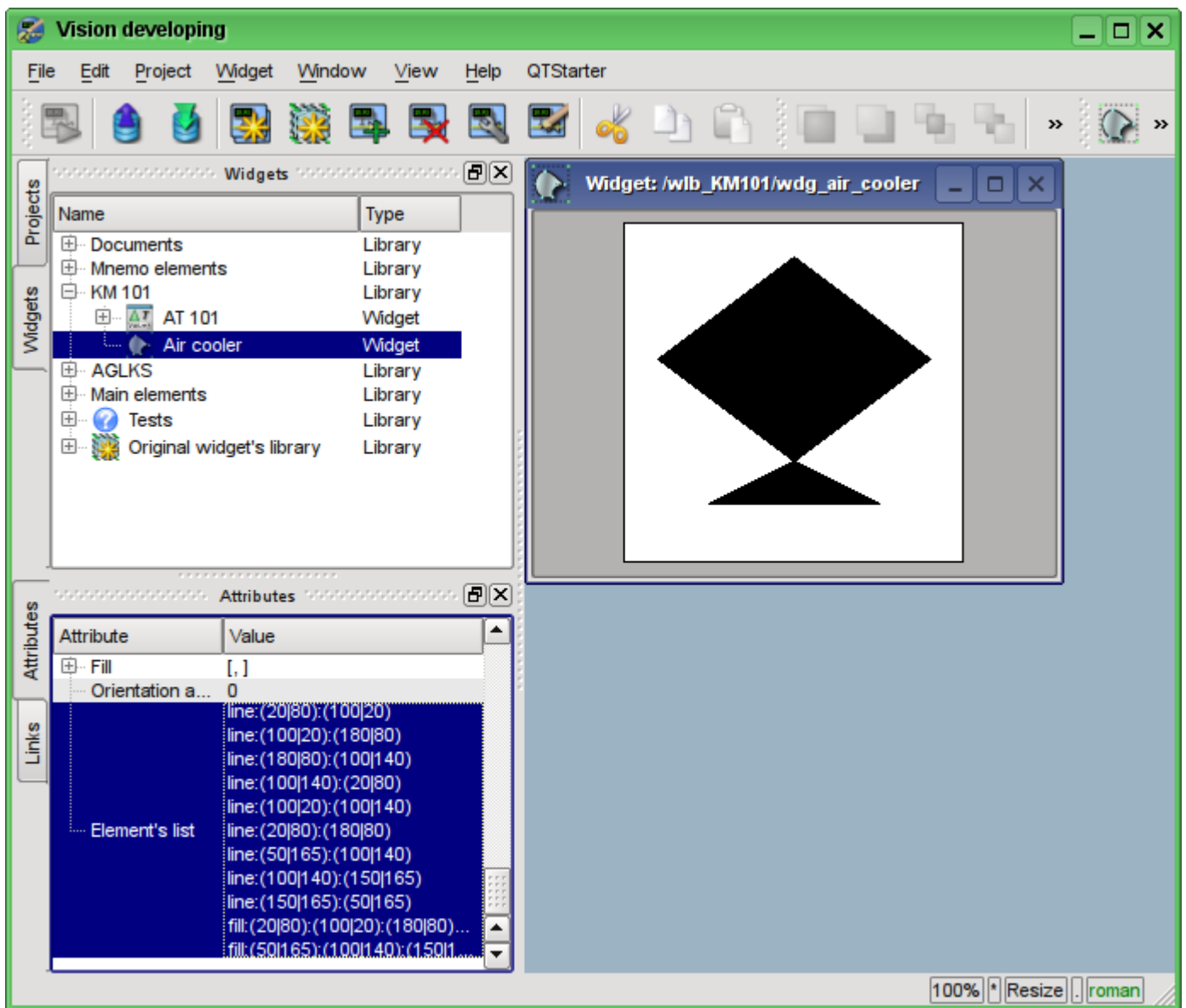


Fig. 5.3.1.4. The image corresponding to the "Element's list" of the widget.

Now let's change the fill color (black, if you do not specify any other (default)) in the tab "Attributes" in the "Fill" section to "lightgrey" (Figure 5.3.1.5). Color can be set as with [# ColorKeywords color names](#) and in the format #RRGGBB (#RRGGBB-AAA).

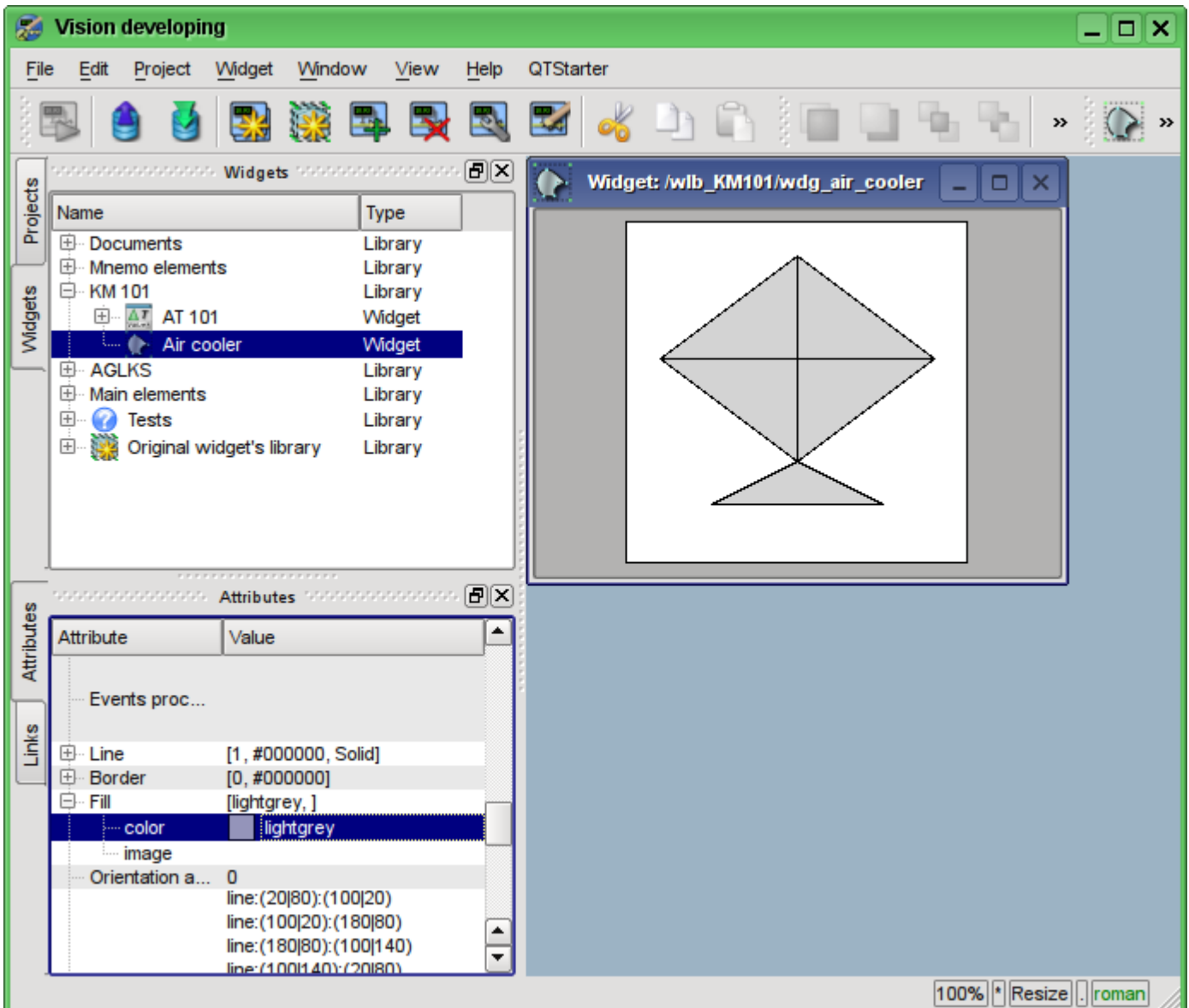


Fig. 5.3.1.5. Change the color of the fill.

Let's create an icon for our widget, which will be visible in the widgets' tree of the library "KM101" (Figure 5.3.1.6).

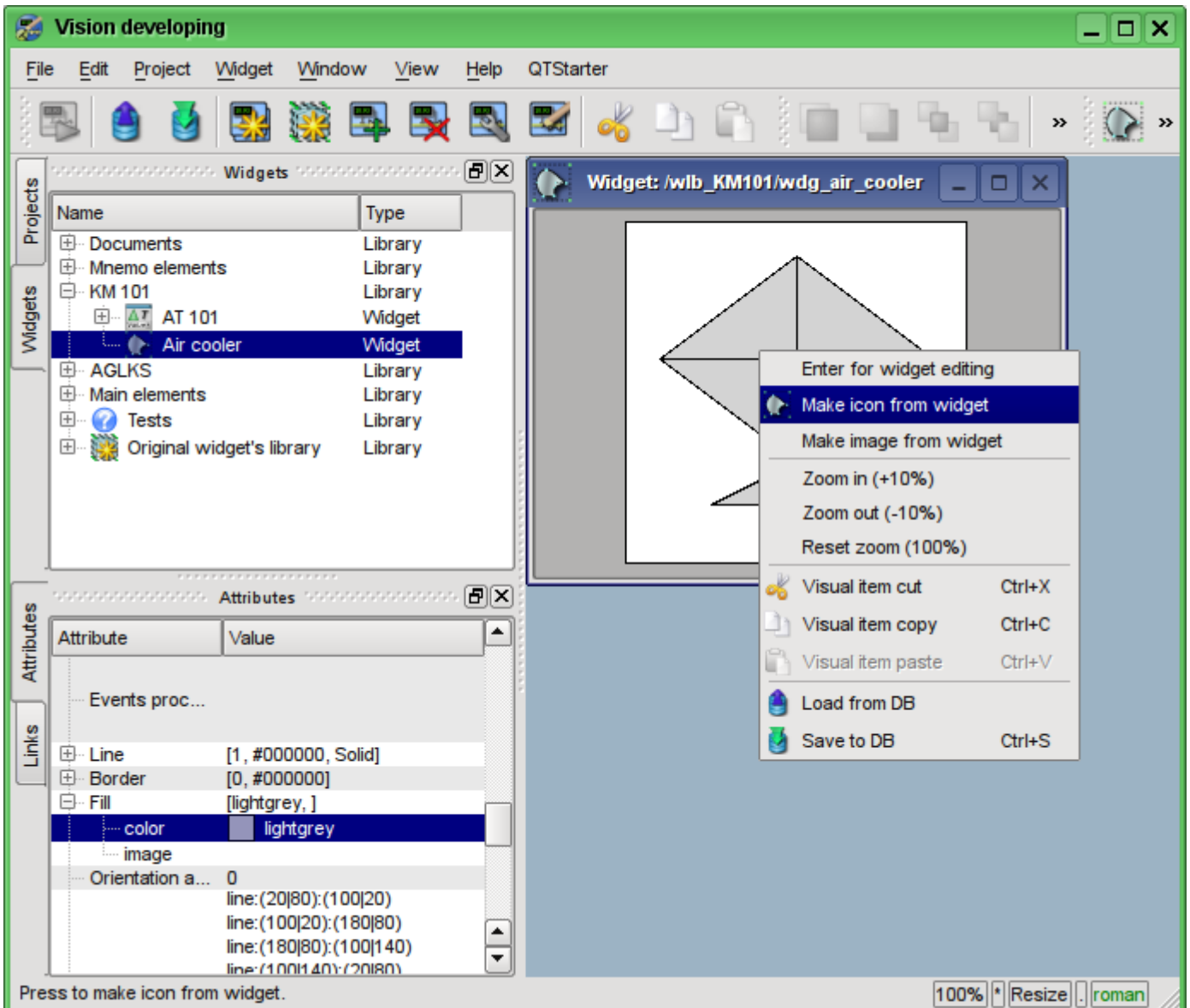


Fig. 5.3.1.6. Creating an icon for the widget.

The process of creating the first widget is completed. We'll now turn to the stage of layout and the creation of the resulting widget.

5.3.2. Creation the final complex widget "Cooler" on the basis of the primitive "Elements box"

The resulting widget we'll create in the "KM 101" library. To do this we must click the right mouse button on the library and select the primitive "Elements box", as it is shown in Figure 5.3.2.1. For a new element let's specify the identifier "elCooler" and the name of "Cooler".

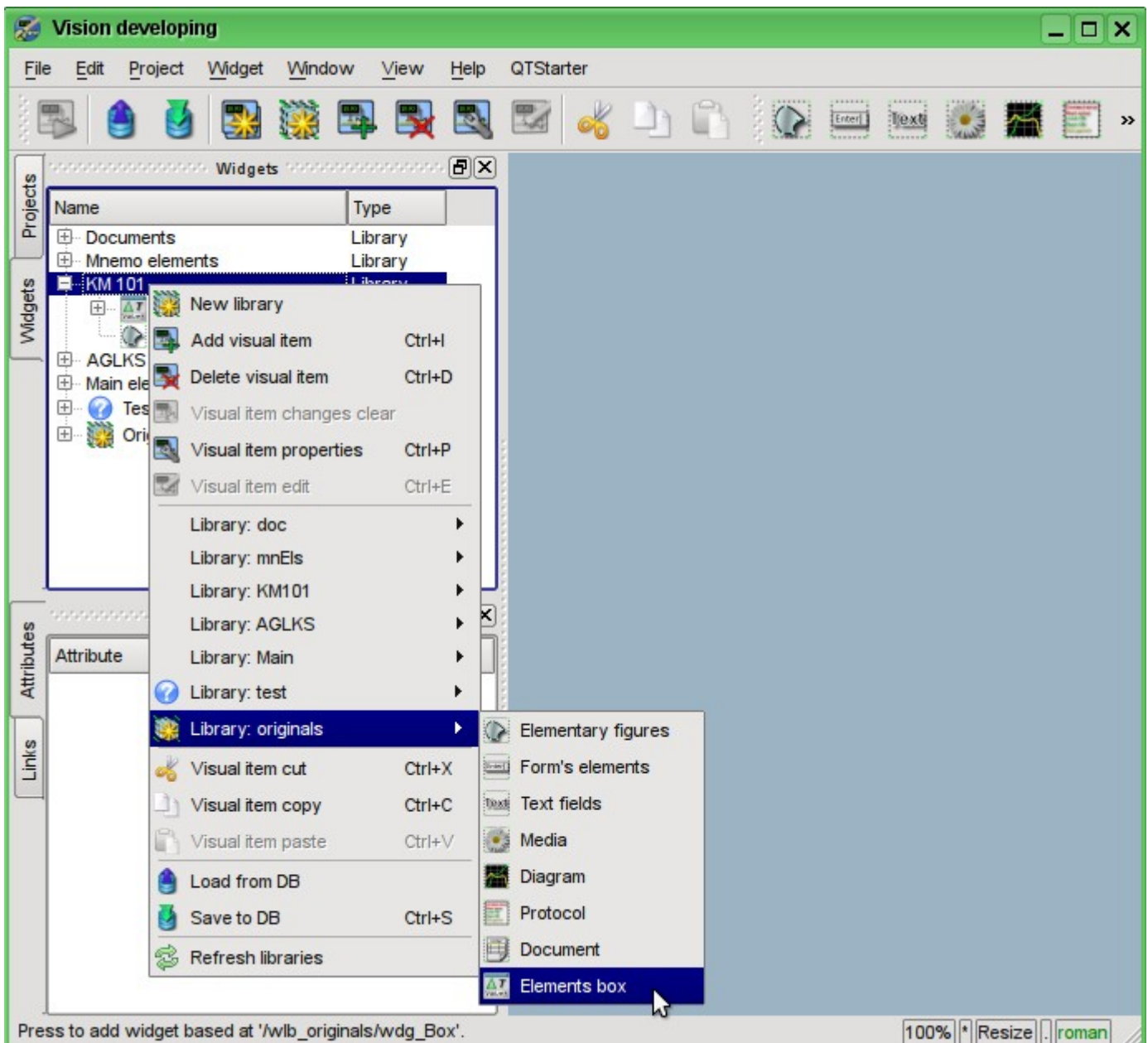


Fig. 5.3.2.1. Adding the widget based on the primitive "Elements box" to the "KM 101" library.

After confirmation, we'll have the new widget object with the name "Cooler". Select it in the widget library "KM 101" and open for editing. Let us now set the width and height of the widget in the 250 and 200 pixels respectively in the "Attributes" tab in the "Geometry" section.

Let's take the previously created element "Air cooler"(air_cooler) and drag him (clicking on it by the left mouse button and moving the cursor of the mouse to the body of the widget, then let the button) to the newly created widget (see Figure 5.3.2.2).

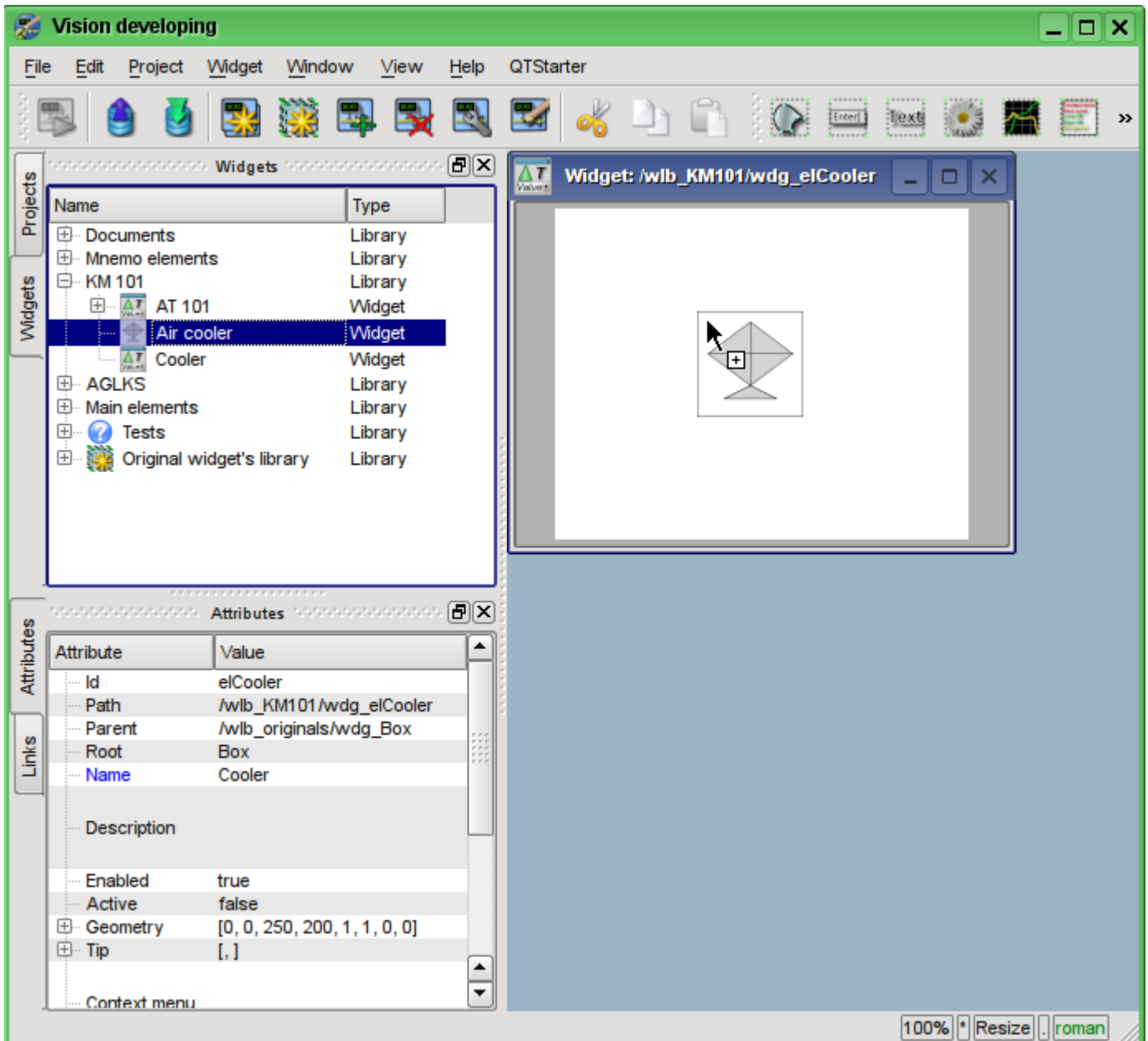


Fig. 5.3.2.2. Drag and Drop of the widget "air_cooler" to the widget-container "elCooler".

The dialogue window will appear to enter the ID and name of the new widget. ID and the name can be set arbitrarily. We will input the "air_cooler" ID and the name we'll leave blank (it will be inherited from parent - the element "air_cooler"). Thus, the newly-created widget inside the container "elCooler" inherits the element - "Air cooler" ("air_cooler"). After confirming the entry of ID and name the widget "Air cooler" ("air_cooler") will be added to our widget container "elCooler" (Figure 5.3.2.3)

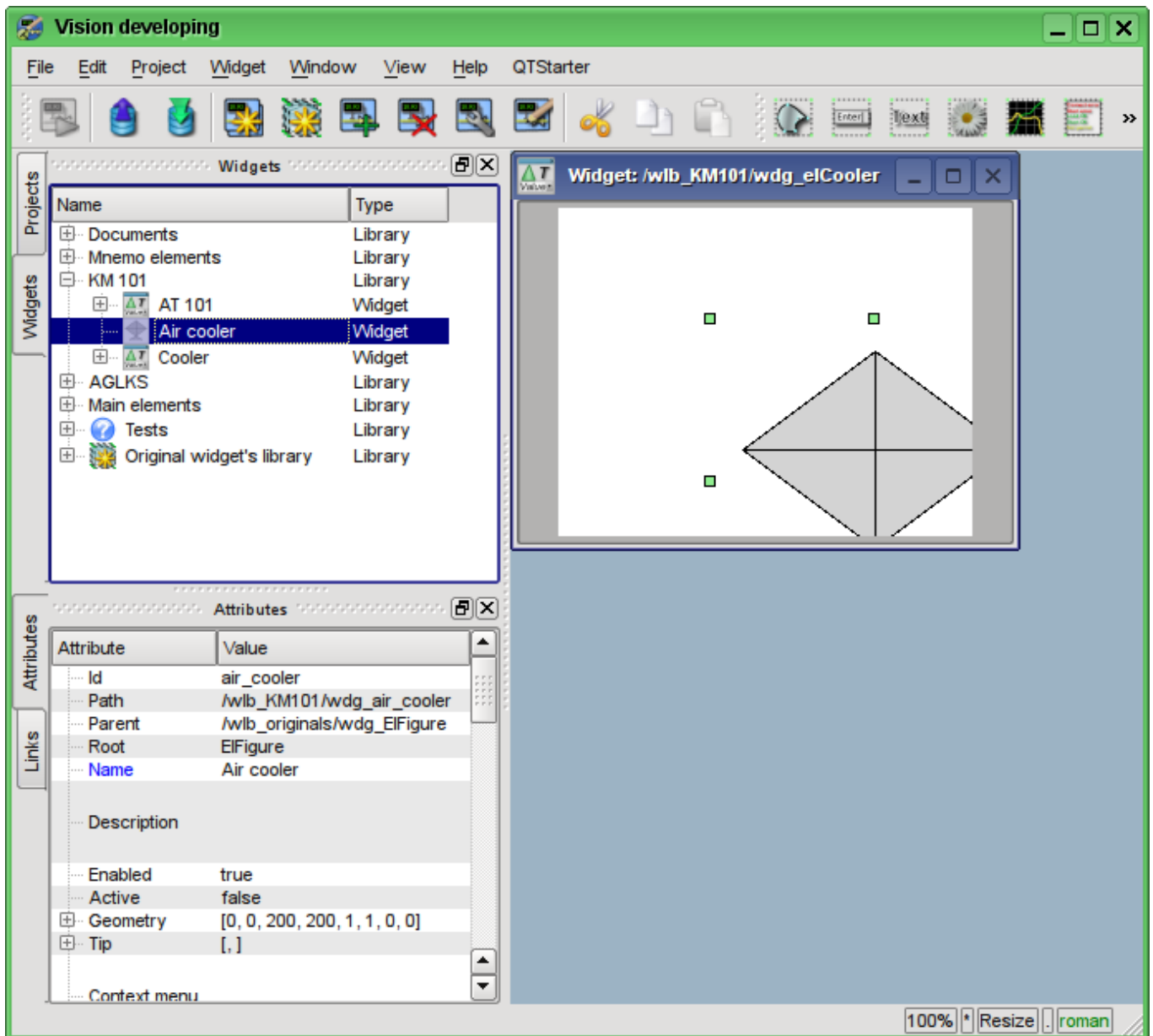


Fig. 5.3.2.3. Adding the inherited widget "air_cooler".

Let's set on the attributes panel of the widget in the "Geometry" section the coordinates "x" and the "y" of upper left corner of the widget to 25 pixels and 0 respectively.

Next, unwind the library "Mnemo elements", find there the "Cooler" element (cooler2) and drag it to the widget-container. This element will dynamically display the productivity of the air cooler. As the result it will appear the dialog window for entering the ID and name of the new widget. Enter the ID "cooler2" and the name again let's leave blank. Thus, the newly-created widget inside the container "elCooler" will inherit the element of the library "Mnemo elements" - "Cooler" ("cooler2"). After confirming the entry of the ID and name the widget "Cooler" ("cooler2") will be added to our widget-container "elCooler". If necessary, raise the widget "cooler2" over the widget "air_cooler" within the widget-container "elCooler" from the toolbar below. Let's specify in the "Attributes" tab in the "Geometry" section the coordinates "x" and "y" of the upper left corner of the widget "Cooler" to the 75 and 30 pixels respectively. Change in the inherited widget "Cooler" alpha channel (transparency) of the fill color. To do this in the "Attributes" tab in the fields "Color1" and "Color2" we'll change the colors by the adding "-200" to them, where the 200 - the value of transparency ("0" - fully transparent, while "255" - the fully opaque), as it is shown in Fig. 5.3.2.4.

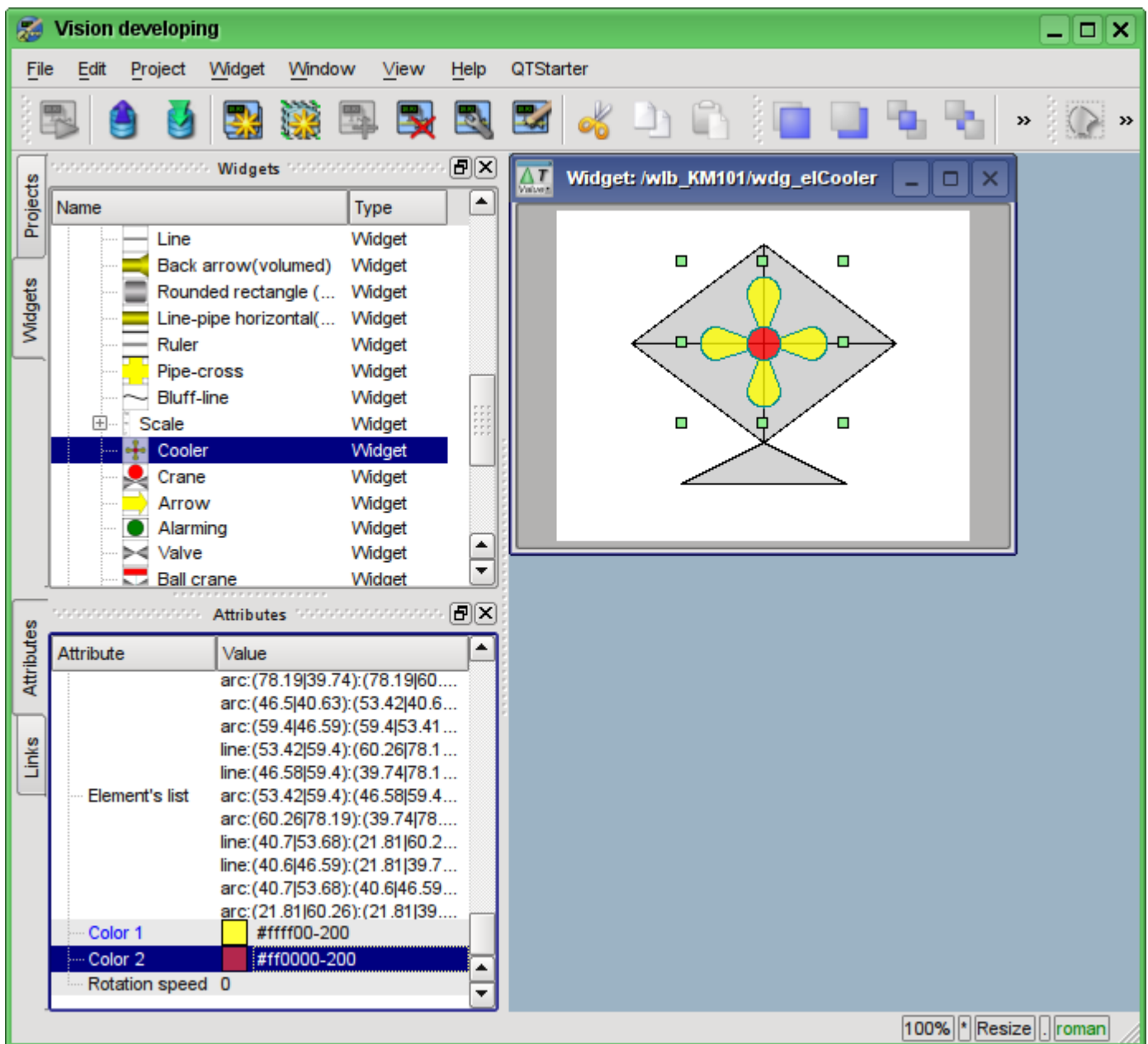


Fig. 5.3.2.4. Change the fill colors transparency in the inherited widget "cooler2".

Now let's add to the widget-container "elCooler" two text fields based on the primitive "Text", in order to display the input and output temperatures of the flow. To do this in the library "KM 101" we'll select the widget "Cooler" and then click on the visual items toolbar on the icon of the primitive "Text", as it is shown in Figure 5.3.2.5.

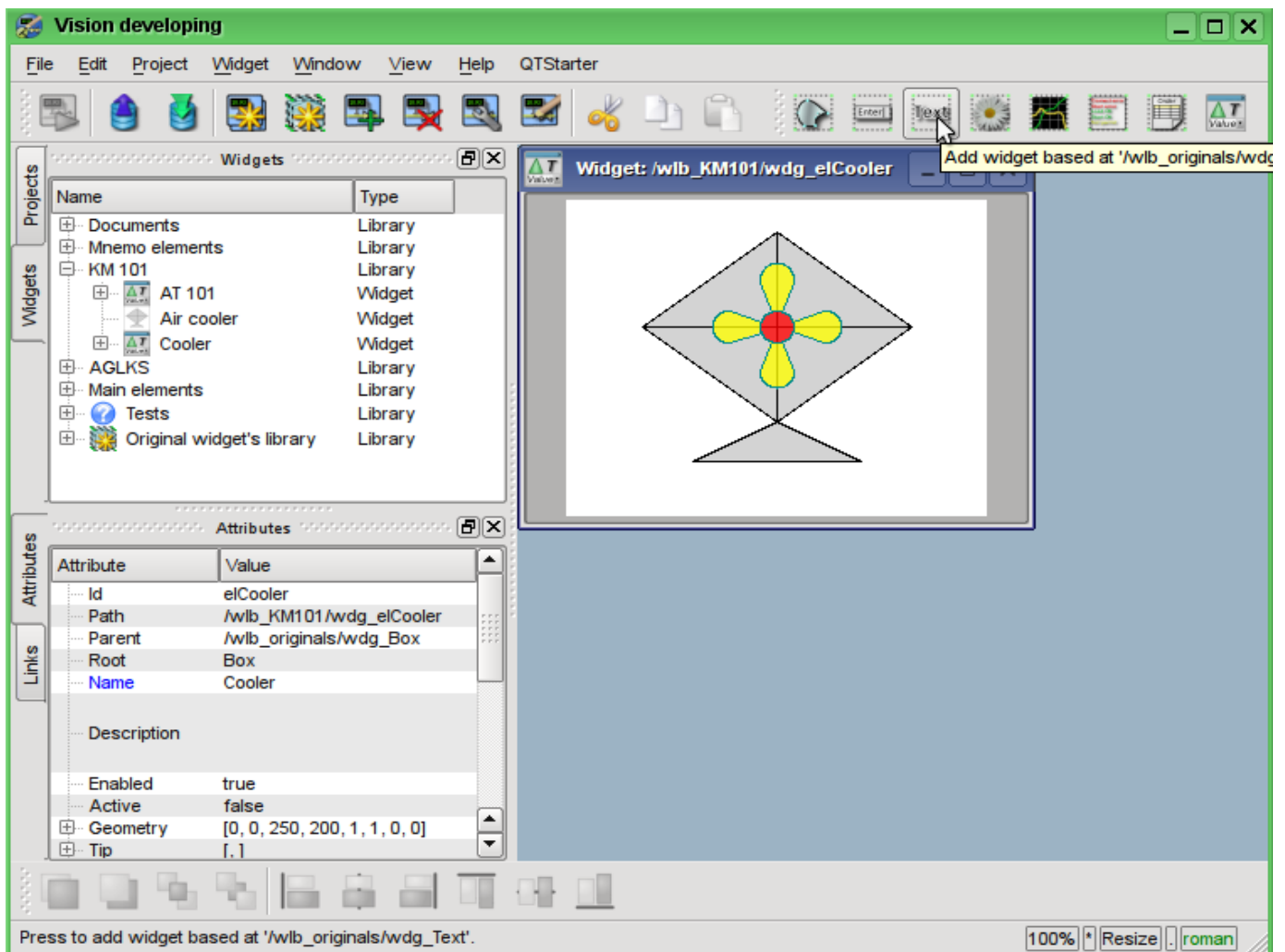


Fig. 5.3.2.5. Adding the new element to the container, based on the primitive "Text."

The dialog of the ID and name of the newly created element entering will appear. Enter the ID "Ti" for the first text field, and the name field we'll leave blank. Let's define the geometric sizes and the coordinates of the upper-left corner of the widget, as it is shown in Fig. 5.3.2.6

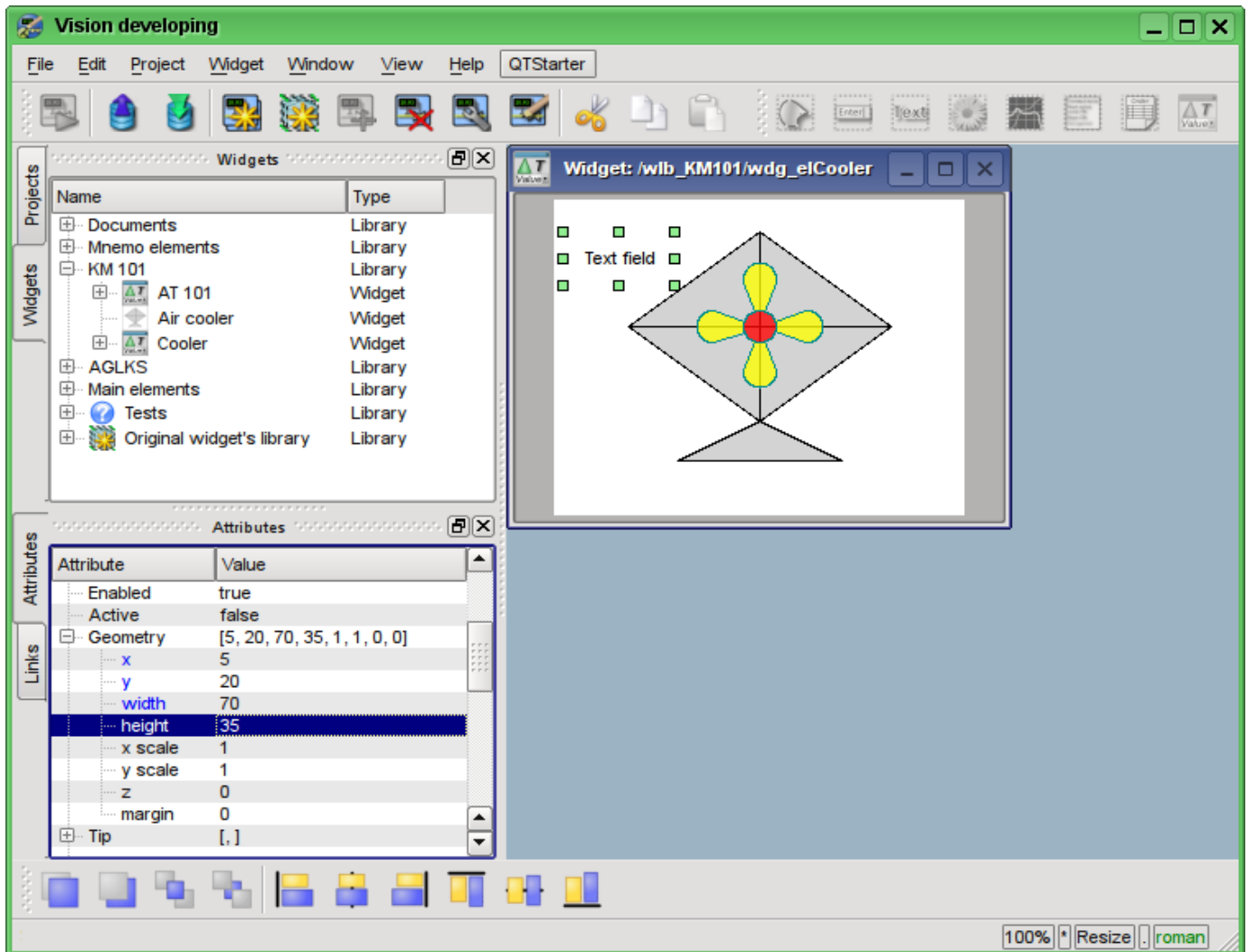


Fig. 5.3.2.6. Specifying the geometry of the widget "Ti".

Let's change the size of the font for this element and make it bold (Fig. 5.3.2.7). Note that the modified field in the inherited widgets are highlighted in blue for easy tracking of changes and their subsequent "cleaning" (rollback) with right mouse button click on the changed attribute.

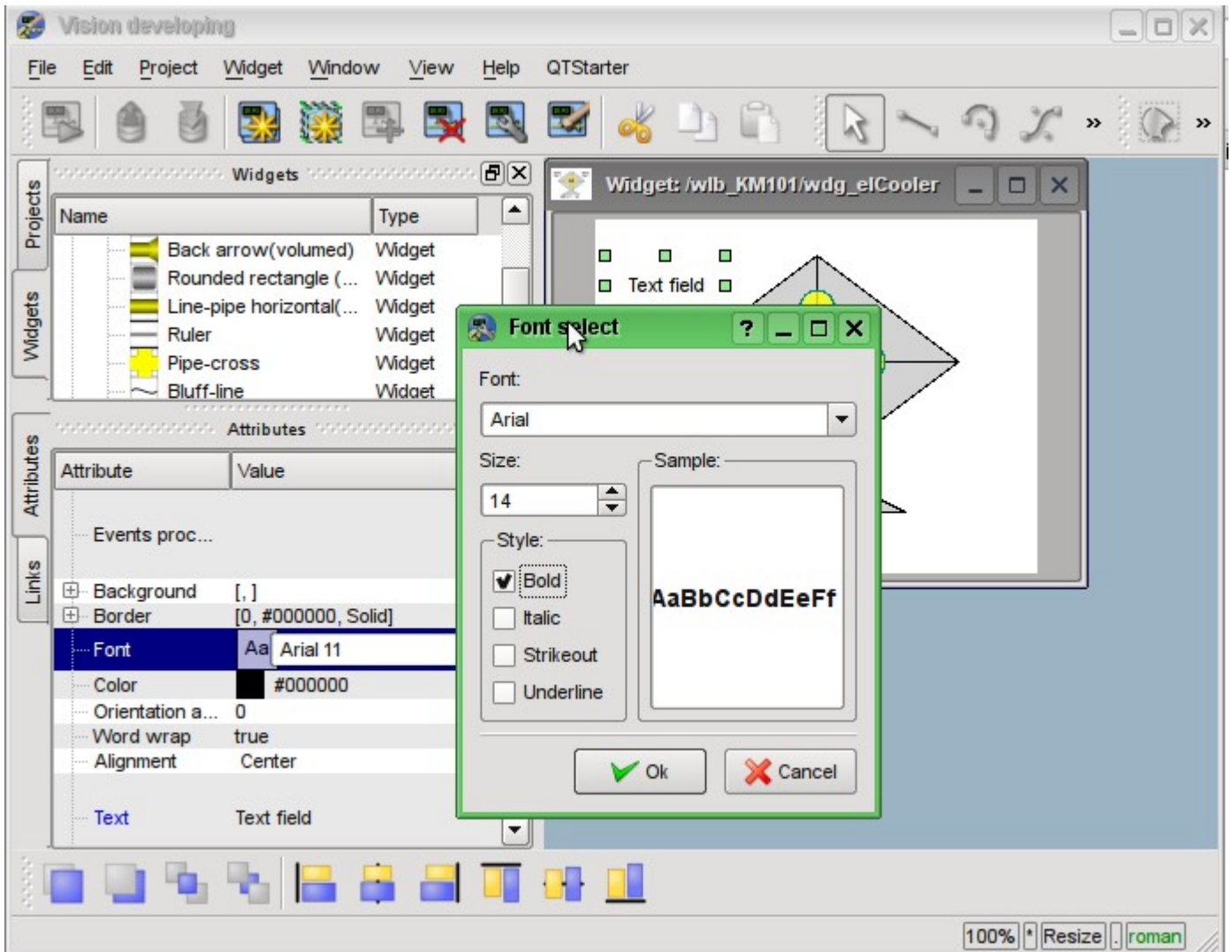


Fig. 5.3.2.7. Changing the font size for the widget "Ti".

Now we'll change the field "Text" of the "Ti" widget, indicating the presence of the argument "%1" in it, in which it will be subsequently transferred the real value of the input temperature (Fig. 5.3.2.8).

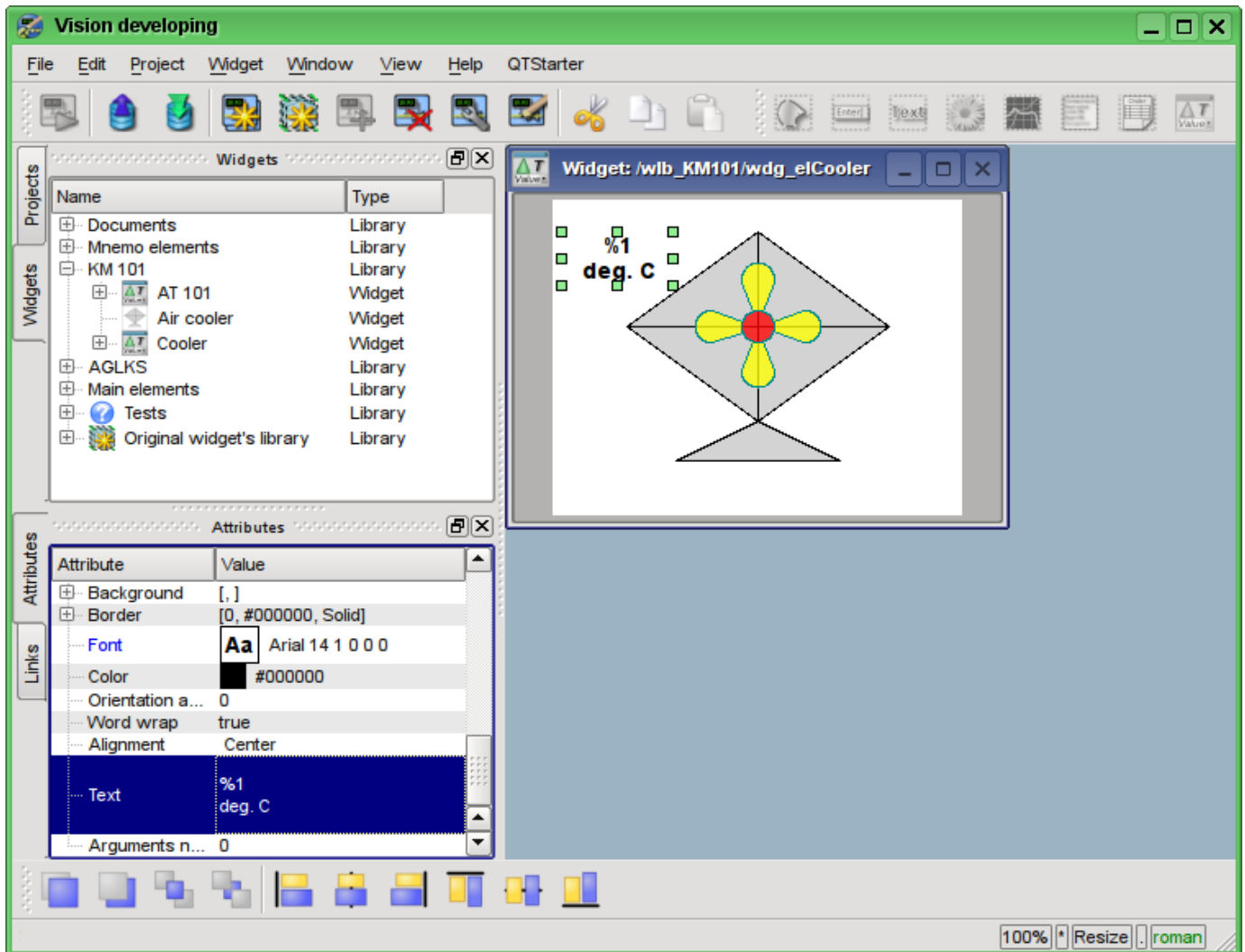


Fig. 5.3.2.8. Changing the field "Text" and an indication of the argument's presence in it for the widget "Ti".

Next in the list of attributes of the "Ti" widget in the section "Number of arguments" let's enter "1" and configure the argument (Figure 5.3.2.9). The number "300.25" is entered only for the purpose of clarity, in the execution mode it will be changed by the real value of the input temperature.

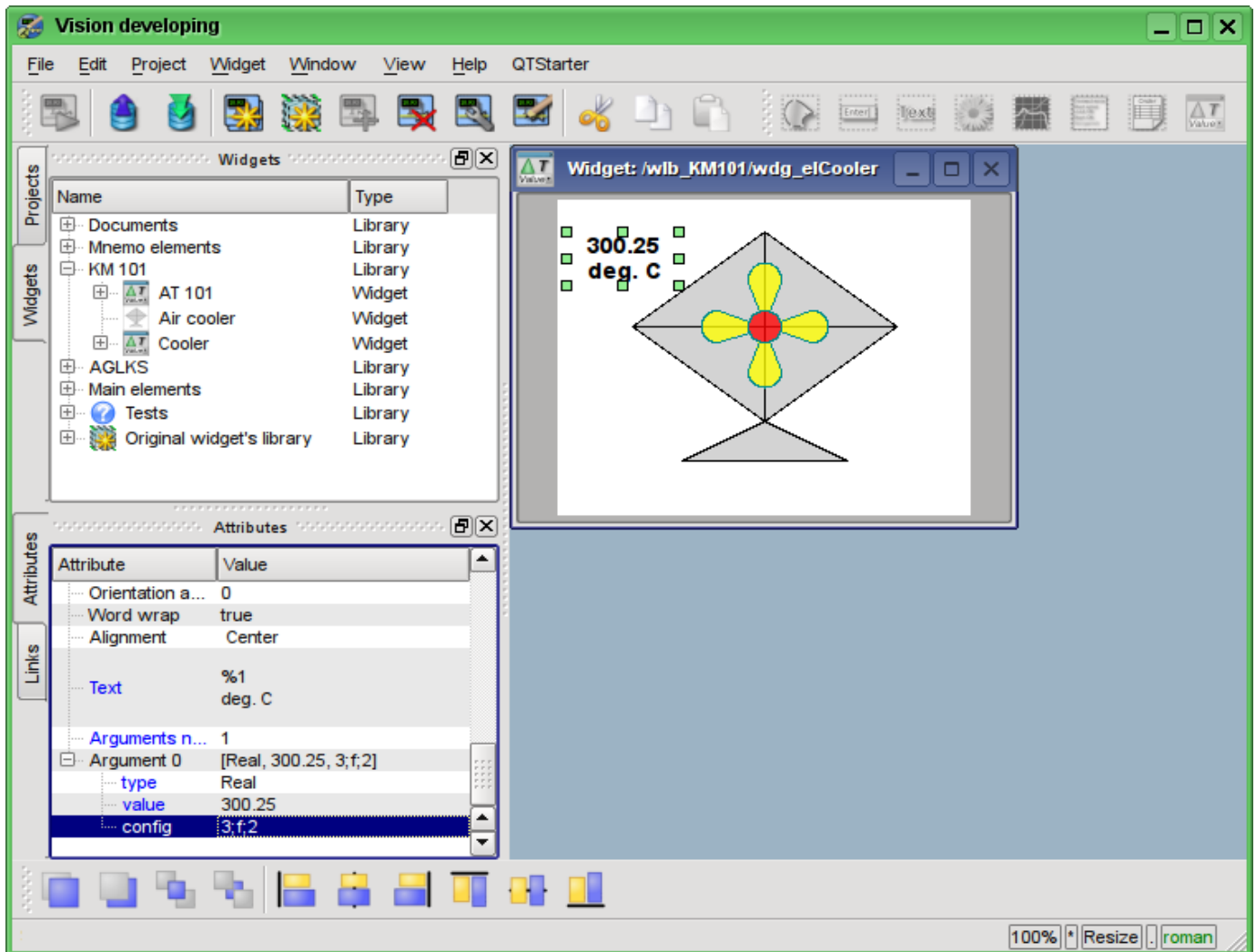


Fig. 5.3.2.9. The configuration of the argument for the "Ti" widget.

Now we'll copy the "Ti" widget in order to create an equivalent widget "To" (output temperature), as it is shown in Figure 5.3.2.10.

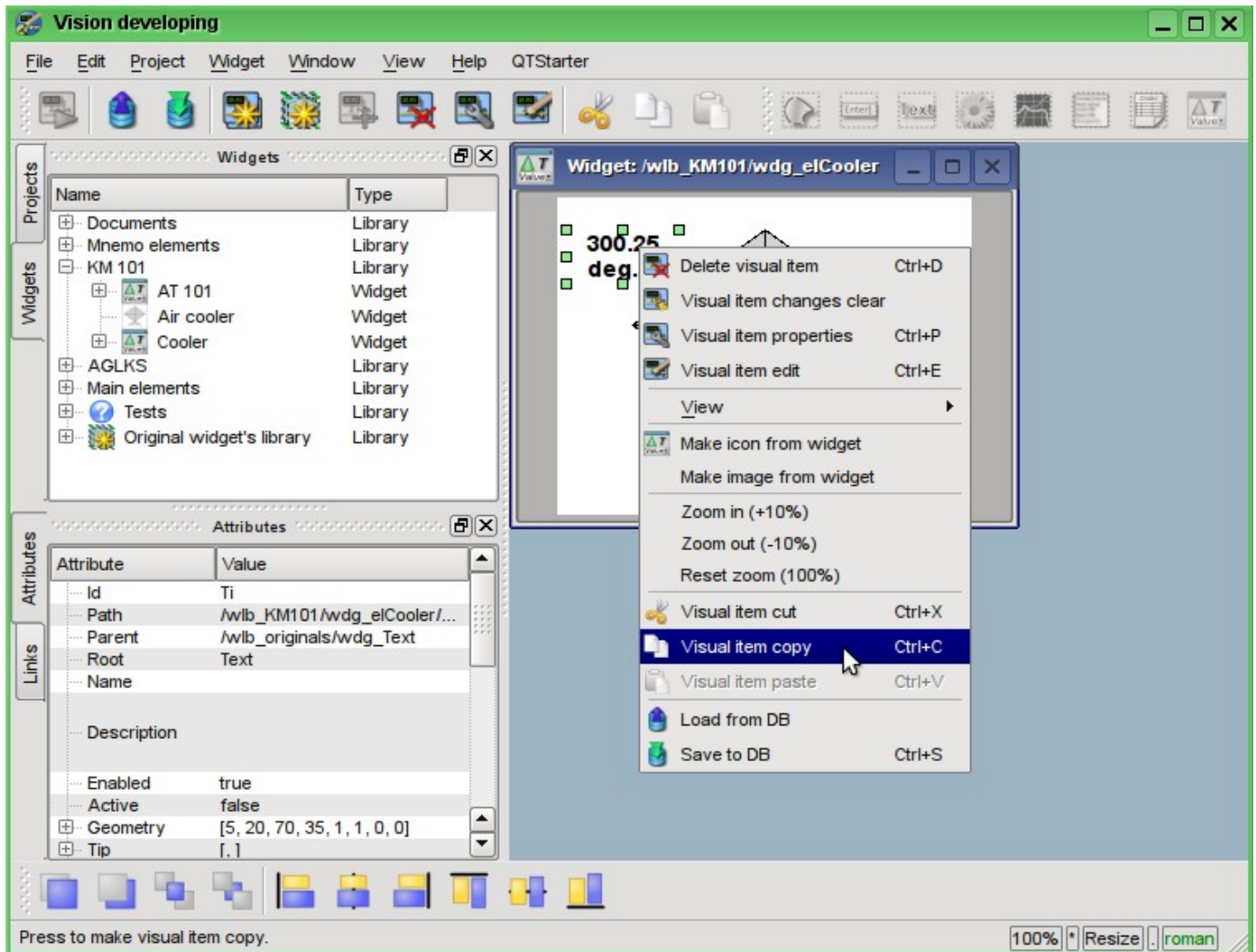


Fig. 5.3.2.10. Copying of the "Ti" widget.

Let's paste the widget in the widget-container "Cooler" in the library "KM 101" (Fig. 5.3.2.11). In the dialog of the ID and the name entering for the newly created widget in the field "ID" we'll write "To", and the name field we'll leave blank.

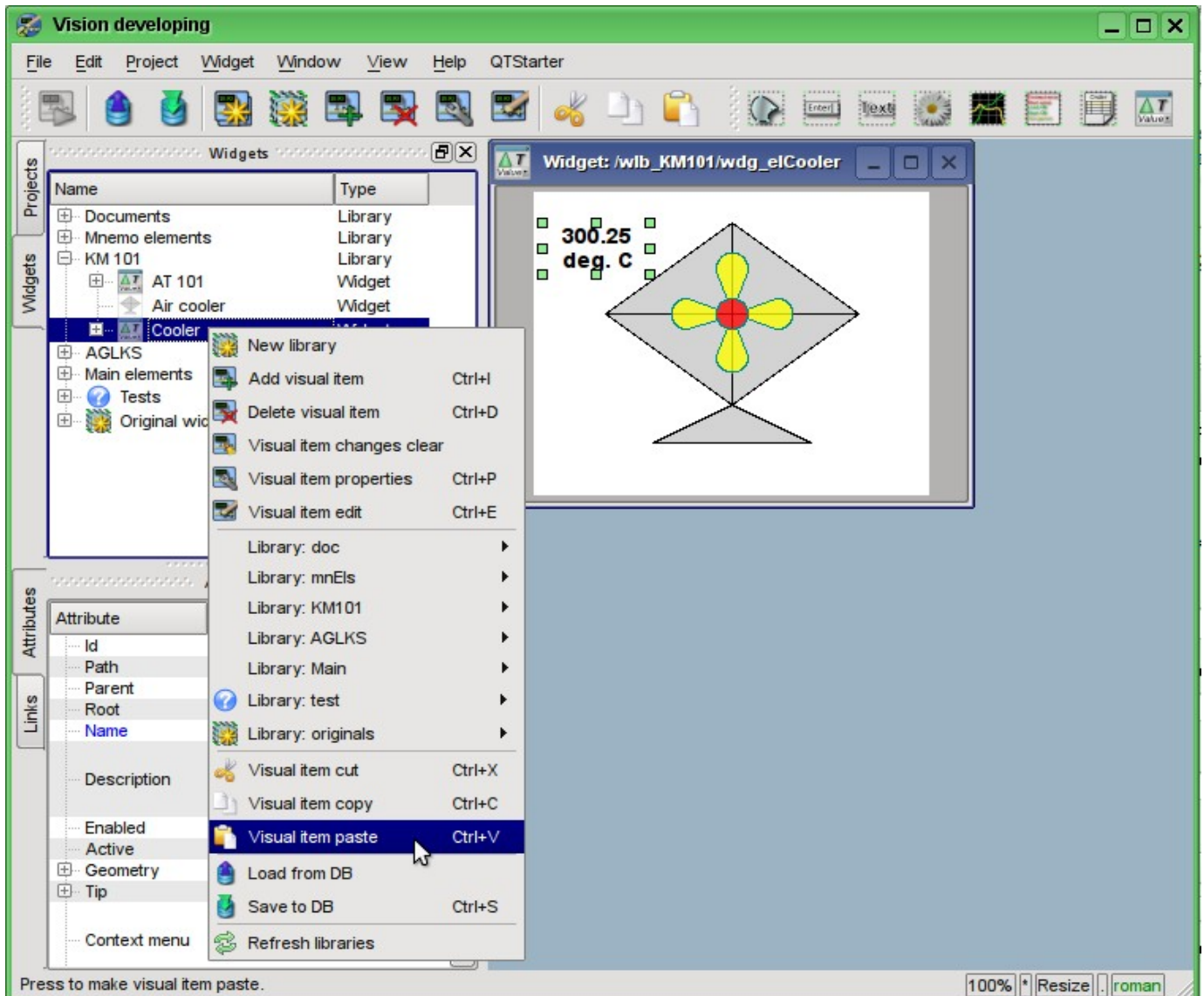


Fig. 5.3.2.11. Paste the copied widget to the widget-container "Cooler" of the library "KM 101".

Let's change the geometry of the "To" widget, as it is shown in Fig. 5.3.2.12.

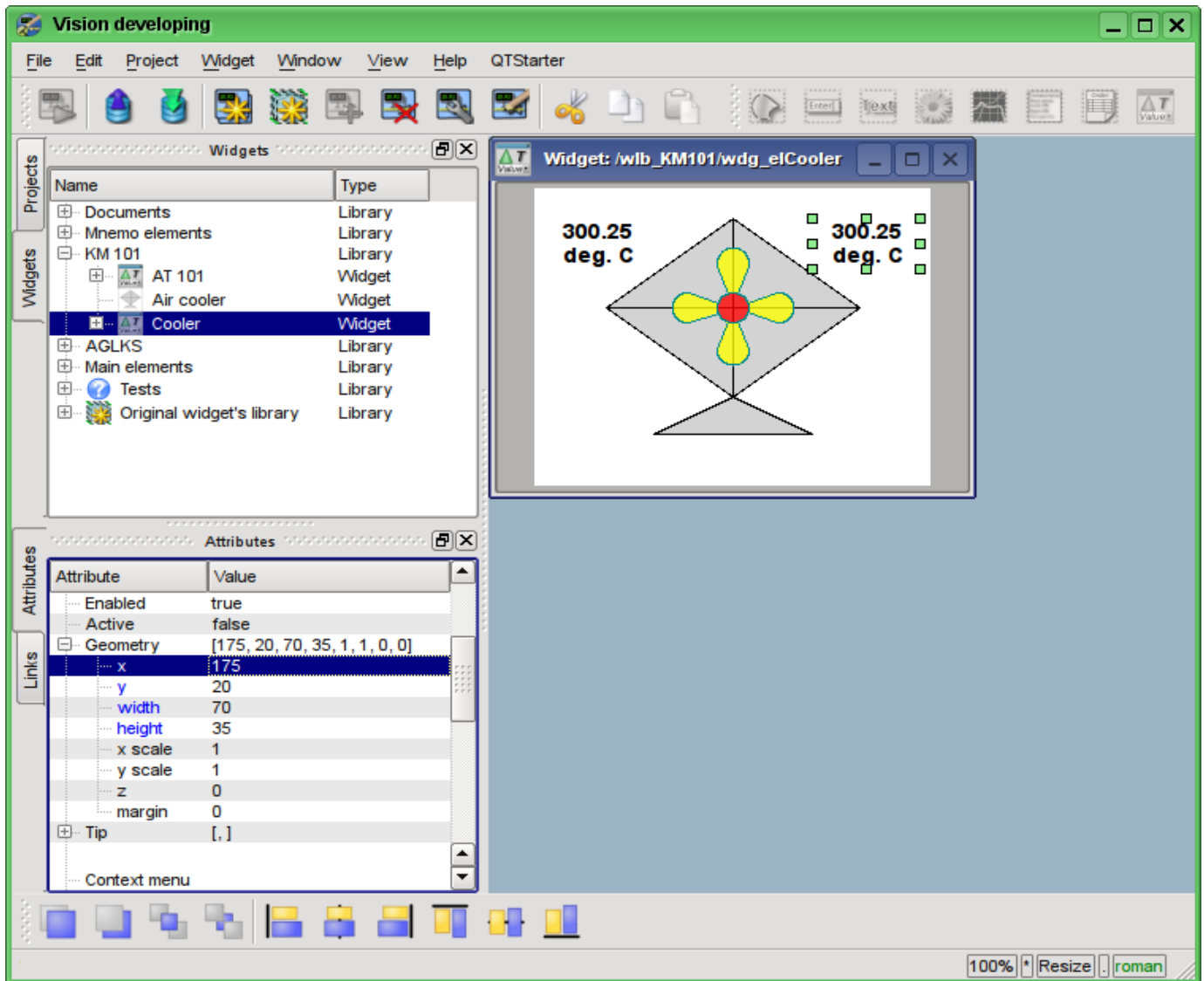


Fig. 5.3.2.12. Changing the geometry of the "To" widget.

Now let's add the widget based on the primitive "Form's elements", which will be used as the ComboBox to select the productivity values of the cooler. The identifier will be "cw", and the "Name" field we'll leave blank. (Figure 5.3.2.13)

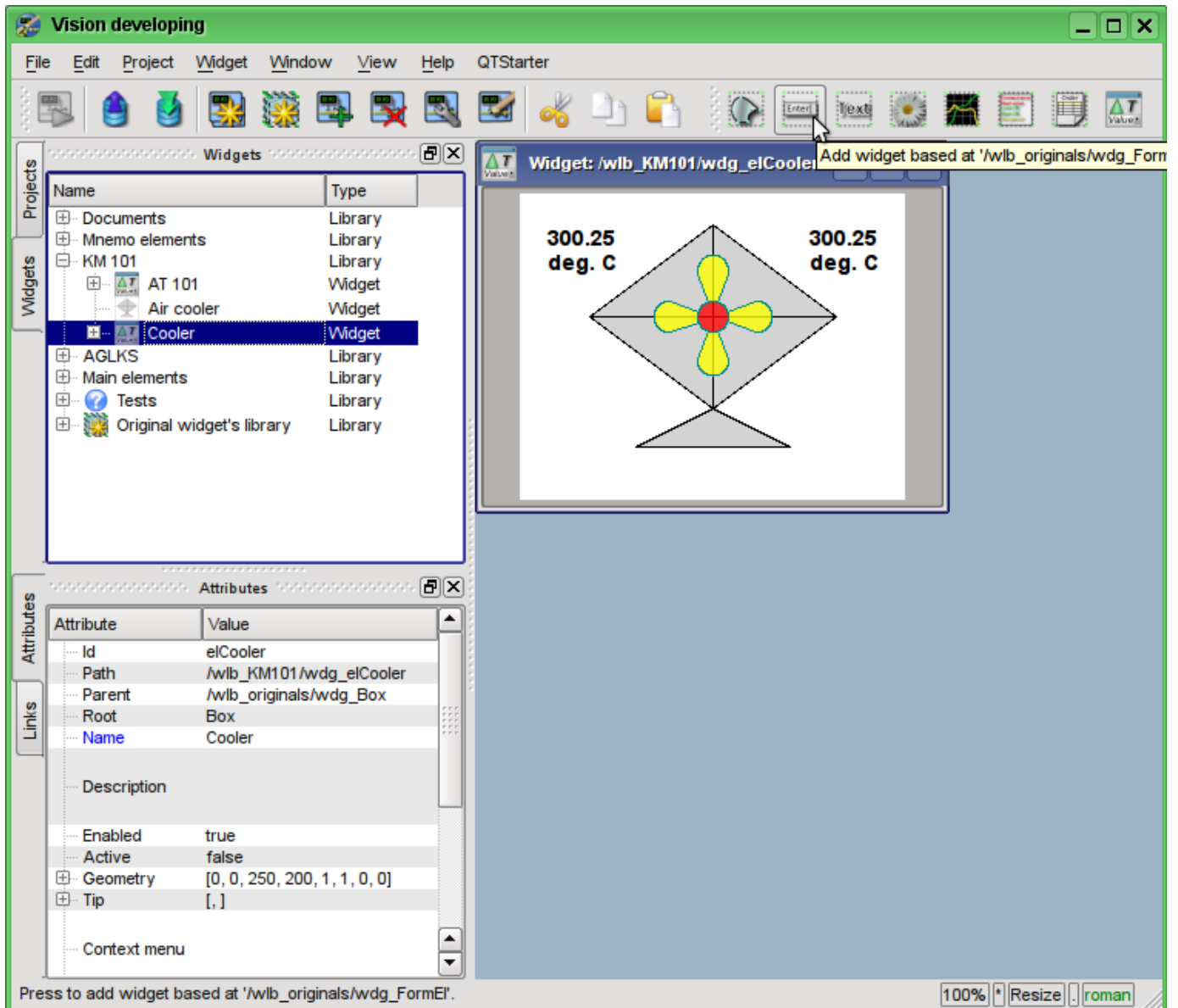


Fig. 5.3.2.13. Adding the widget based on the primitive "Form's elements".

Let's set the parameters of the "Geometry" section of the "Attributes" tab for the newly added widget: coordinates "x", "y" of the upper left corner, width and height of 60, 158, 60 and 40, respectively. Let's change the "Element type" to the Combo Box, as it is shown in Fig. 5.3.2.14

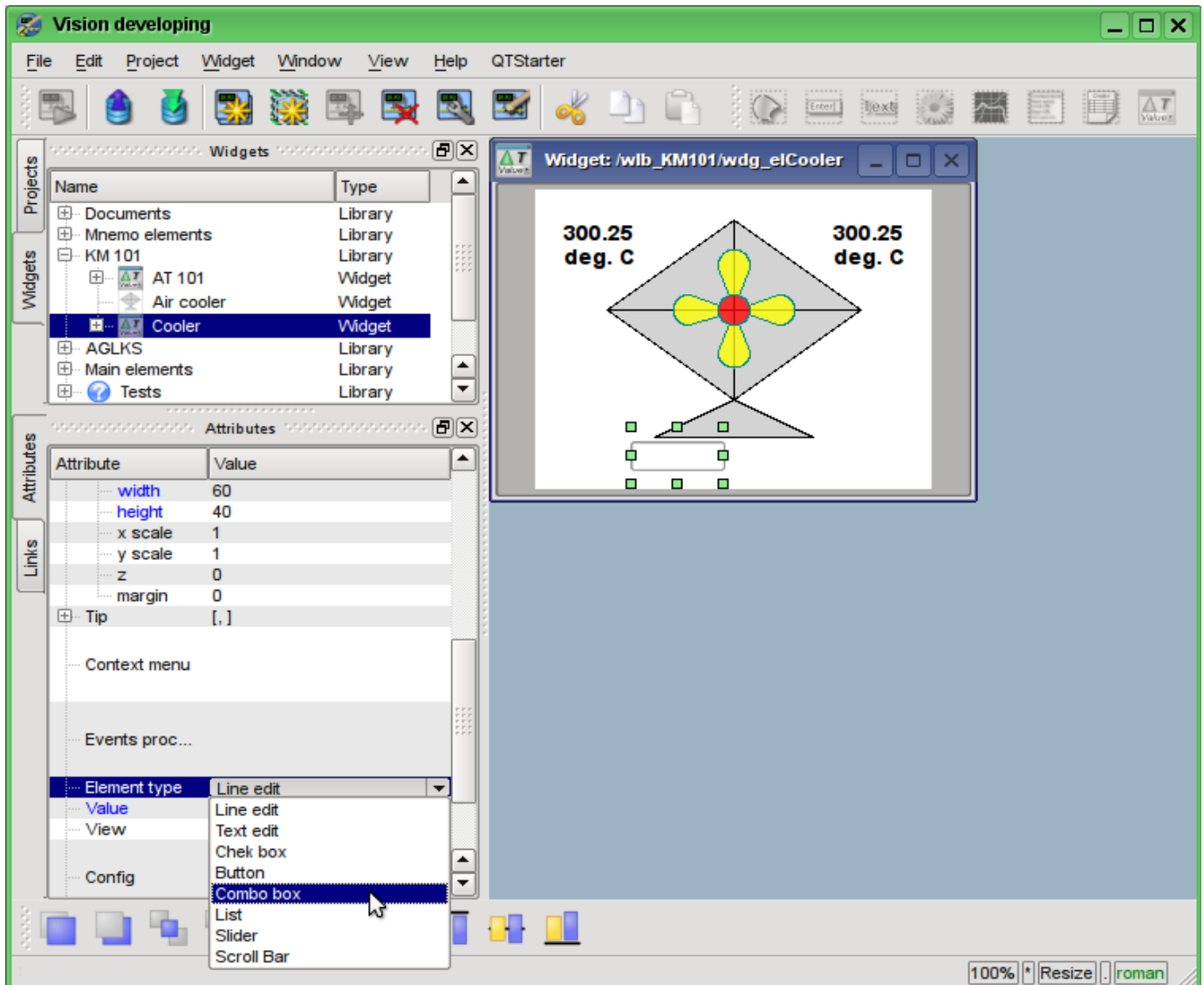


Fig. 5.3.2.14. Change the "Geometry" and "Element type" for the newly created widget.

Let's fill the fields: "Value", "Items" and "Font", as it is shown in Fig. 5.3.2.15. In addition, it is important to raise the combobox above all elements and make it active. To activate the combo box widget, you need to set the appropriate property for it.

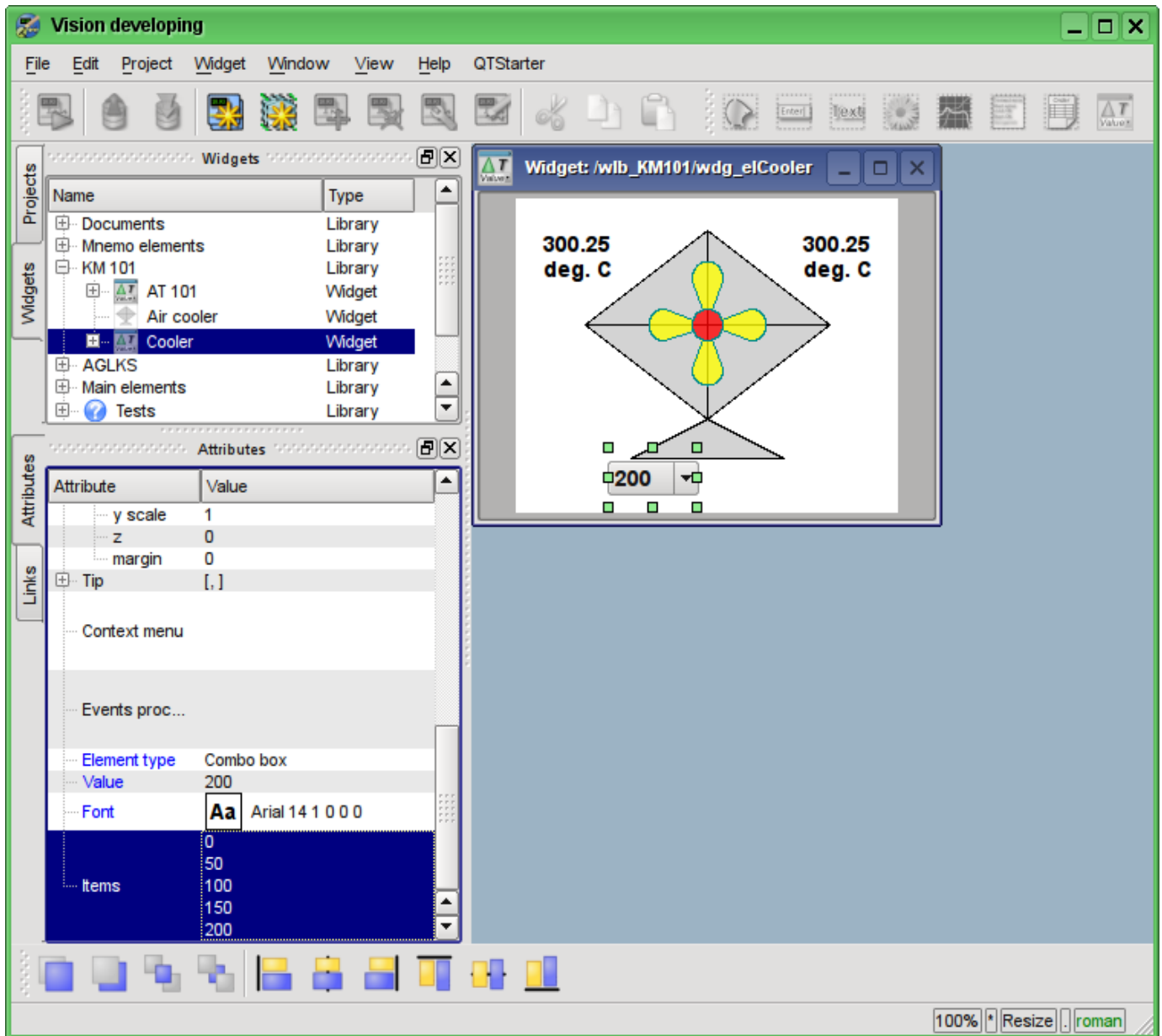


Fig. 5.3.2.15. Filling the parameters of the "cw" ComboBox.

To display the cooler productivity dimensions we'll add the widget on the basis of the "Text" primitive. Let's make the same procedure as for the "Ti" widget. The identifier of the newly created widget will be "dimension", the geometry: coordinates of the upper-left corner "x", "y" will be set to the 125 and 168, respectively, while width and height - at 60 and 20, respectively. Let's change the font size to "14 bold", and in the field "Text" let's type "rpm", that will be our dimension (Fig. 5.3.2.16).

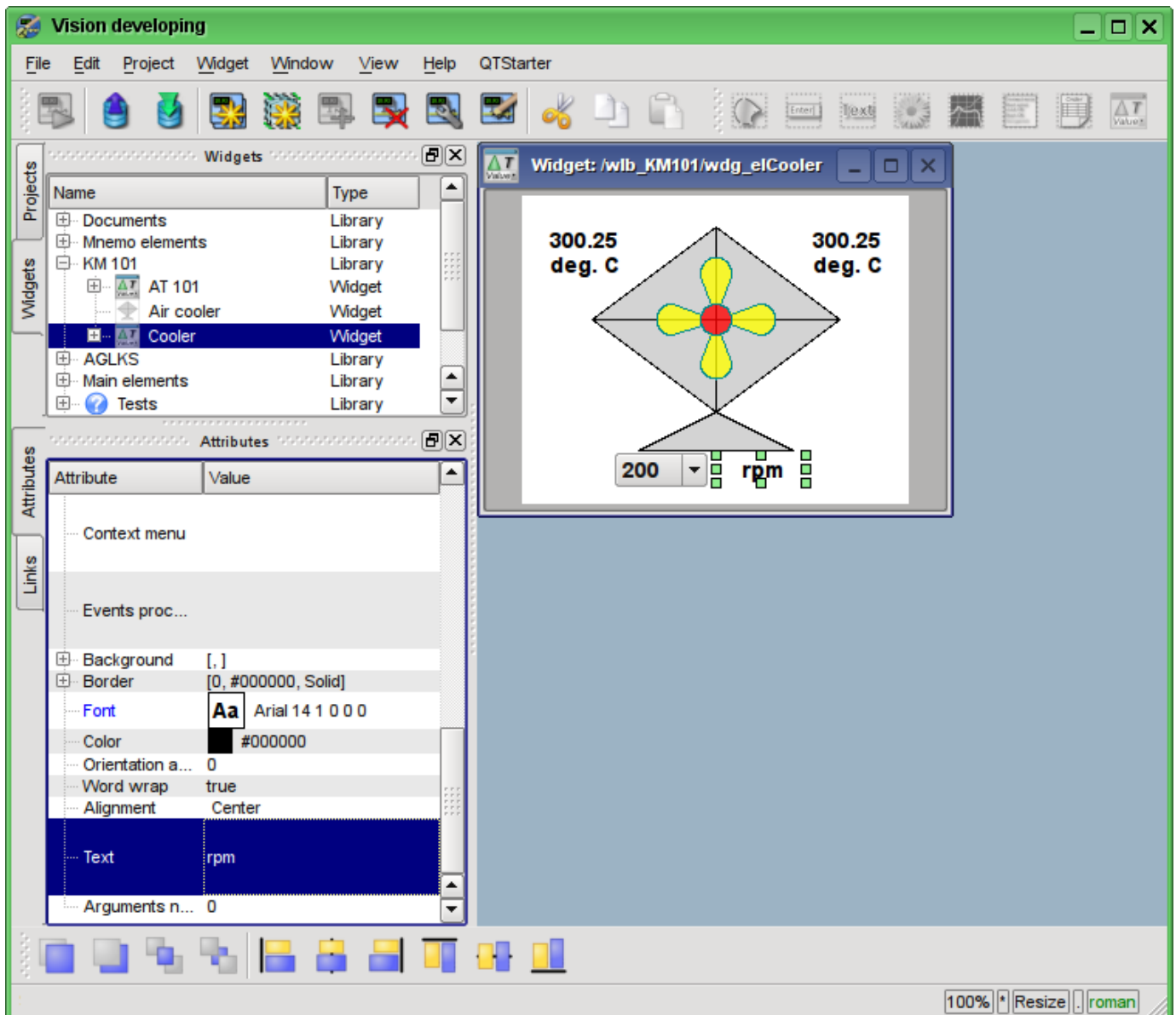


Fig. 5.3.2.16. Adding the "dimension" widget, based on the primitive "Text" and changing of its settings.

To add the processing logics for the widget "Cooler" (elCooler) we'll open the dialog of the properties editing of the visual element and select the "Process" tab. On this tab we can see the tree of widget's attributes and the field for the program code for the attributes' processing. To solve our task, we must add three attributes: Ti, To, Cw (Fig. 5.3.2.17). To add an attribute you should unwind the root element ".", select any element inside the root one and click "Add attribute" button below.

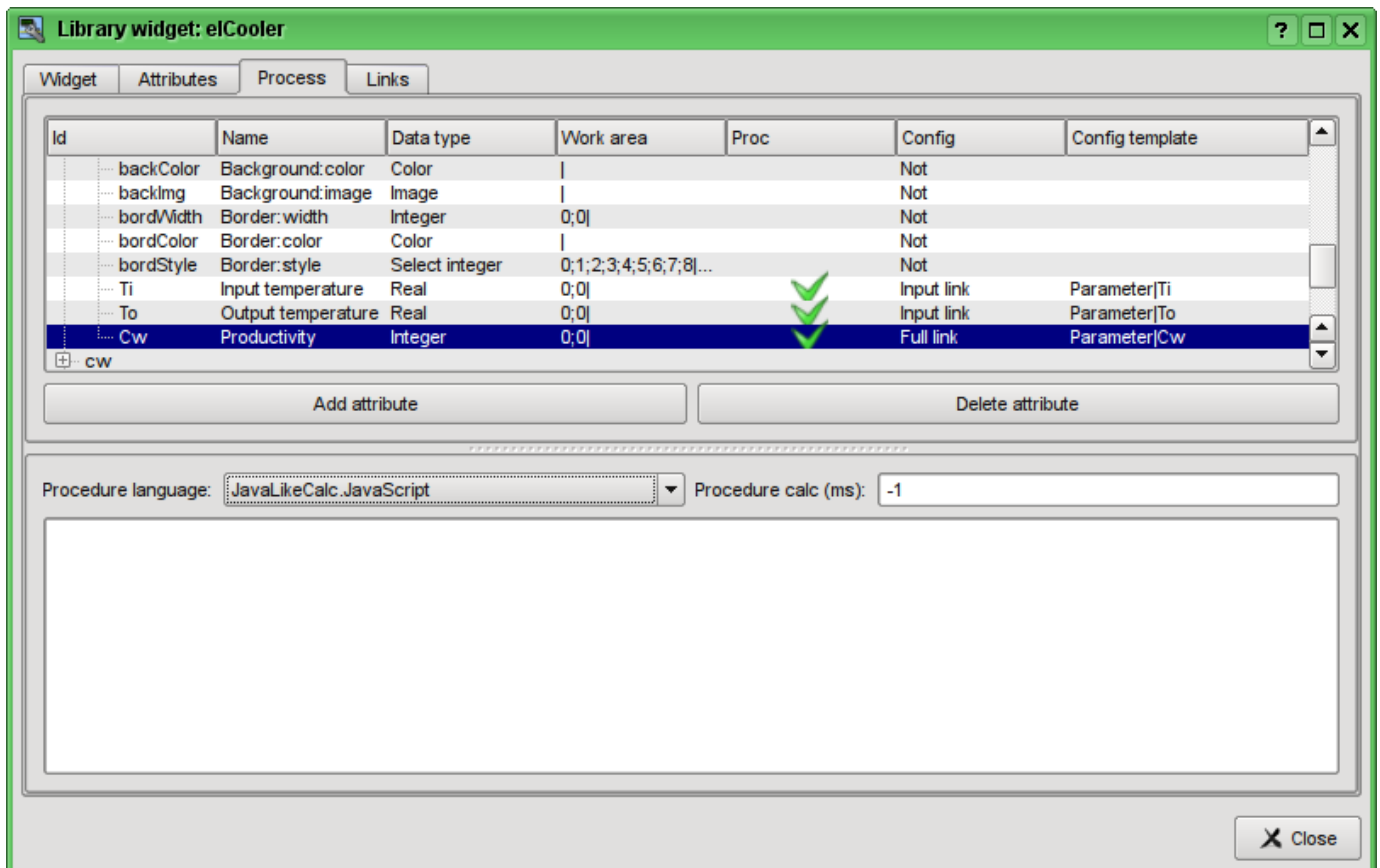


Fig. 5.3.2.17. Adding the three attributes for the element "elCooler" of the library "KM 101".

Further we'll enable the processing of "value" attribute of combo box "cw", as it is shown in Fig. 5.3.2.18. Similarly, enable the processing of the "arg0val" attribute for Ti and To, as well as the "speed" attribute of the "cooler2" element.

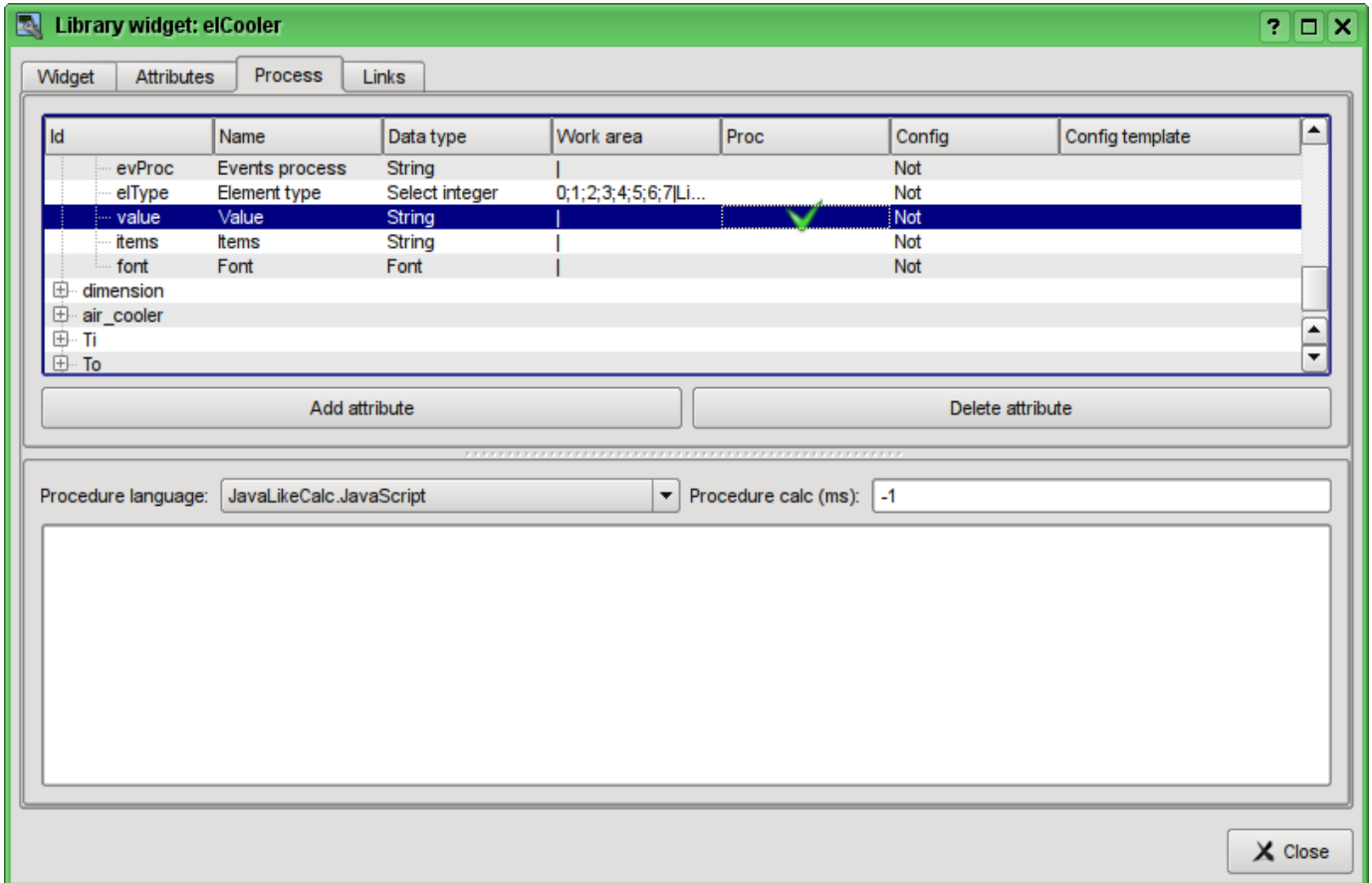


Fig. 5.3.2.18. The enabling of the processing of the "value" attribute of the combo box "cw".

At the end let's set the user programming language for the program to the "JavaLikeCalc.JavaScript" and write the program to process this widget:

```
Ti_arg0val = Ti;
To_arg0val = To;

ev_wrk=ev_rez="";
off=0;
while(true)
{
    ev_wrk=Special.FLibSYS.strParse(event,0,"\n",off);
    if( ev_wrk == "" ) break;
    if( ev_wrk == "ws_CombChange:/cw" ) Cw = cw_value;
    else ev_rez += ev_wrk+"\n";
}
cw_value = Cw;
cooler2_speed = Cw/5;
```

The resulting view of the Process tab of the "elCooler" widget of the "KM 101" library will have the form shown in Fig. 5.3.2.19.

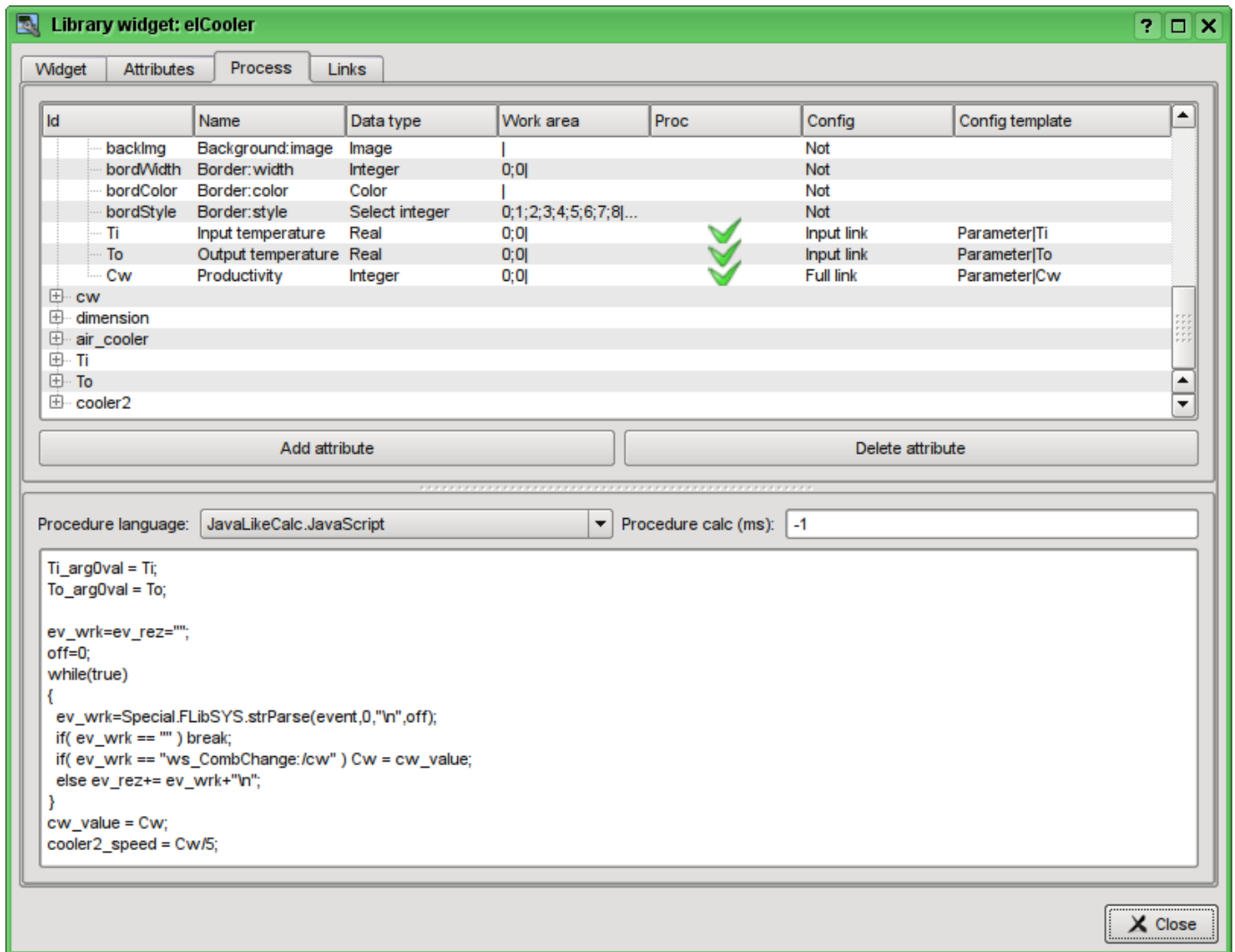


Fig. 5.3.2.19. The resulting view of the Process tab of the "elCooler" widget of the "KM 101" library.

Let's close the dialogue of the properties of visual element editing, create an icon on the basis of our element, close the inner editing window and save it all.

The development of the complex element is finished.

5.3.3. Adding the complex element to the mnemonic scheme

To test the operability and evaluate the results of our efforts let's add the created widget to the mnemonic scheme, developed in chapter 5.2. We'll repeat this operation for two coolers "AT101_1" and "AT101_2".

To do this we'll open the frame of mnemonic scheme "AT 101" for editing. Then grab by the "mouse" our complex element and drag to mnemonic scheme, where we drop it in the desired position. In the dialog we'll enter the identifiers "AT101_1" and "AT101_2" respectively. The field "Name" is blank. Added element we'll place the way we desire. After such manipulations, we should get the mnemonic scheme with the view, similar to Fig.5.3.3.1.

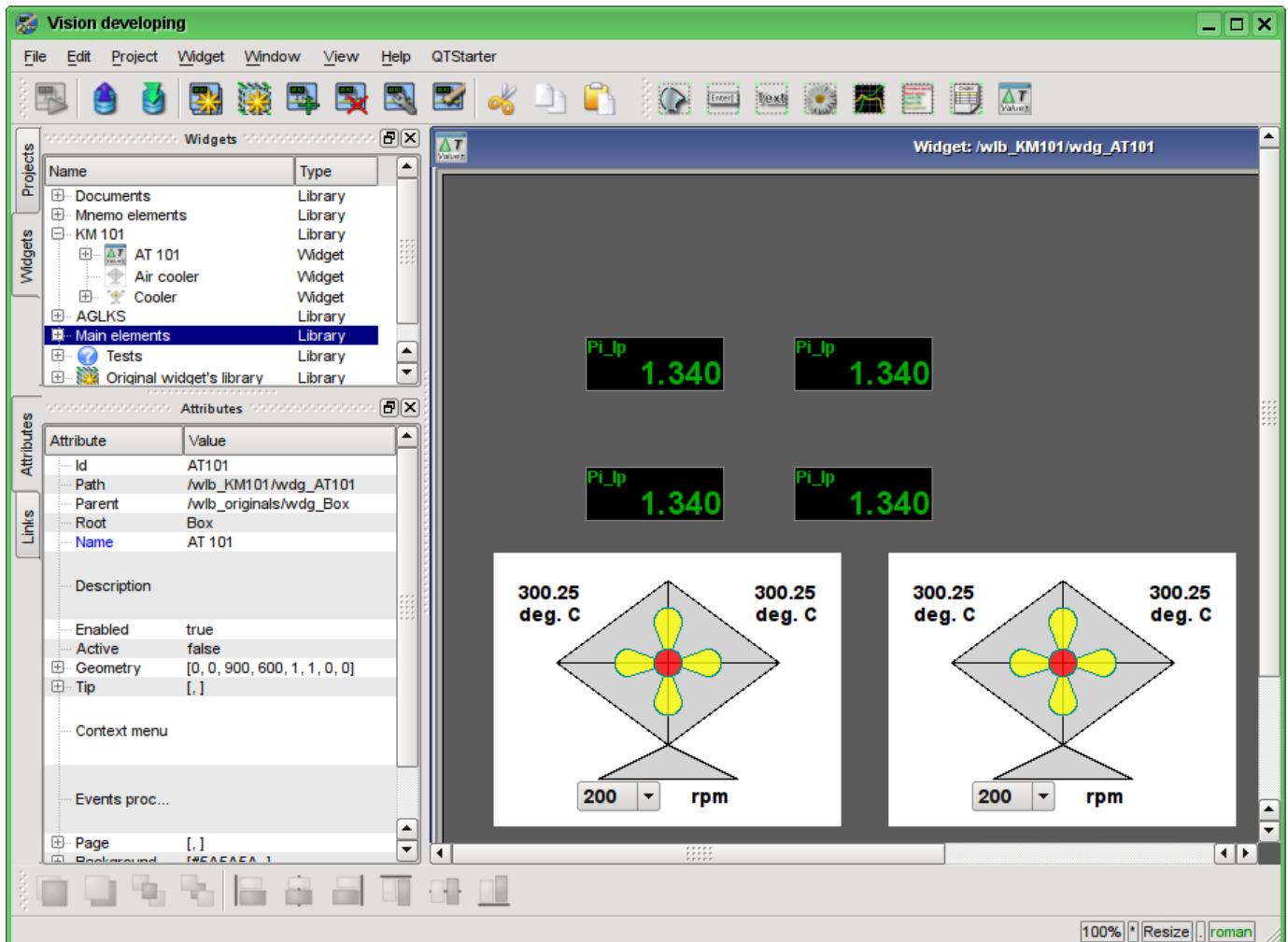


Fig. 5.3.3.1. The view of the mnemonic scheme with complex elements.

Let's save the new mnemonic scheme and close its window. Then move on to the project and open this mnemonic scheme in the project's tree "Signal groups (template)"->"Root page (SO)"->"Group 1"->"Mnemos"->"AT 101". As you can see, our new elements are appeared here automatically. And we only need to connect the links to the new elements. To do this we'll open the dialog of editing the properties of the mnemonic scheme on the "Links" tab (Fig.5.3.3.2). On this tab, we can see the tree with the elements of "AT101_1" and "AT101_2". Unwinding any of the elements, we'll see the "Parameter" branch just with the "Ti", "To" and "Cw" attributes, thus we can simply specify the address of the parameter "prm:/LogicLev/KM101/AT101_1" in the "Parameter" field and attributes will be placed automatically.

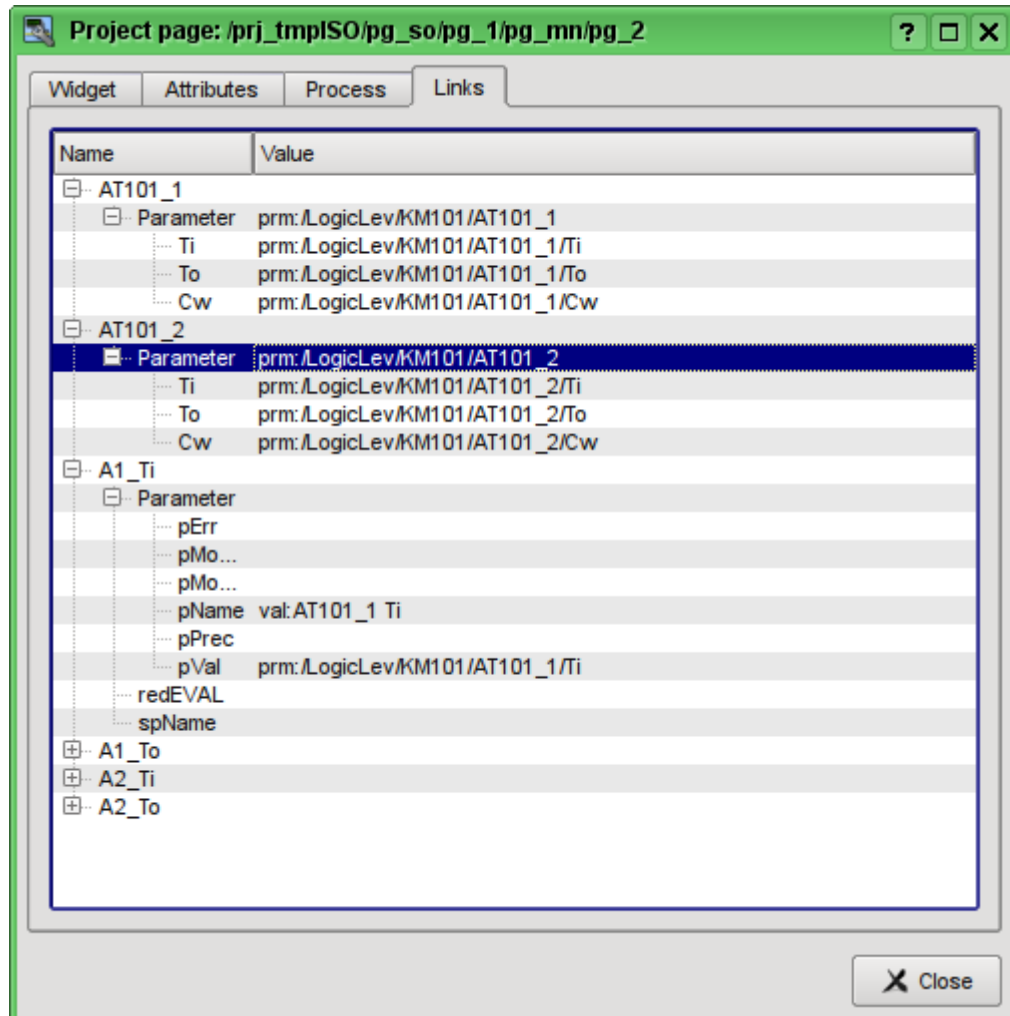


Fig. 5.3.3.2. The "Links" tab of the dialog of editing the properties of the mnemonic scheme.

Let's save our mnemonic scheme and verify what we have. To do this, close the dialog of the properties and run the "Signal groups (template)" for execution. Then switch to the second mnemonic scheme with the help of paging buttons. With error-free configuration, we should see something similar to that shown in Fig.5.3.3.3.

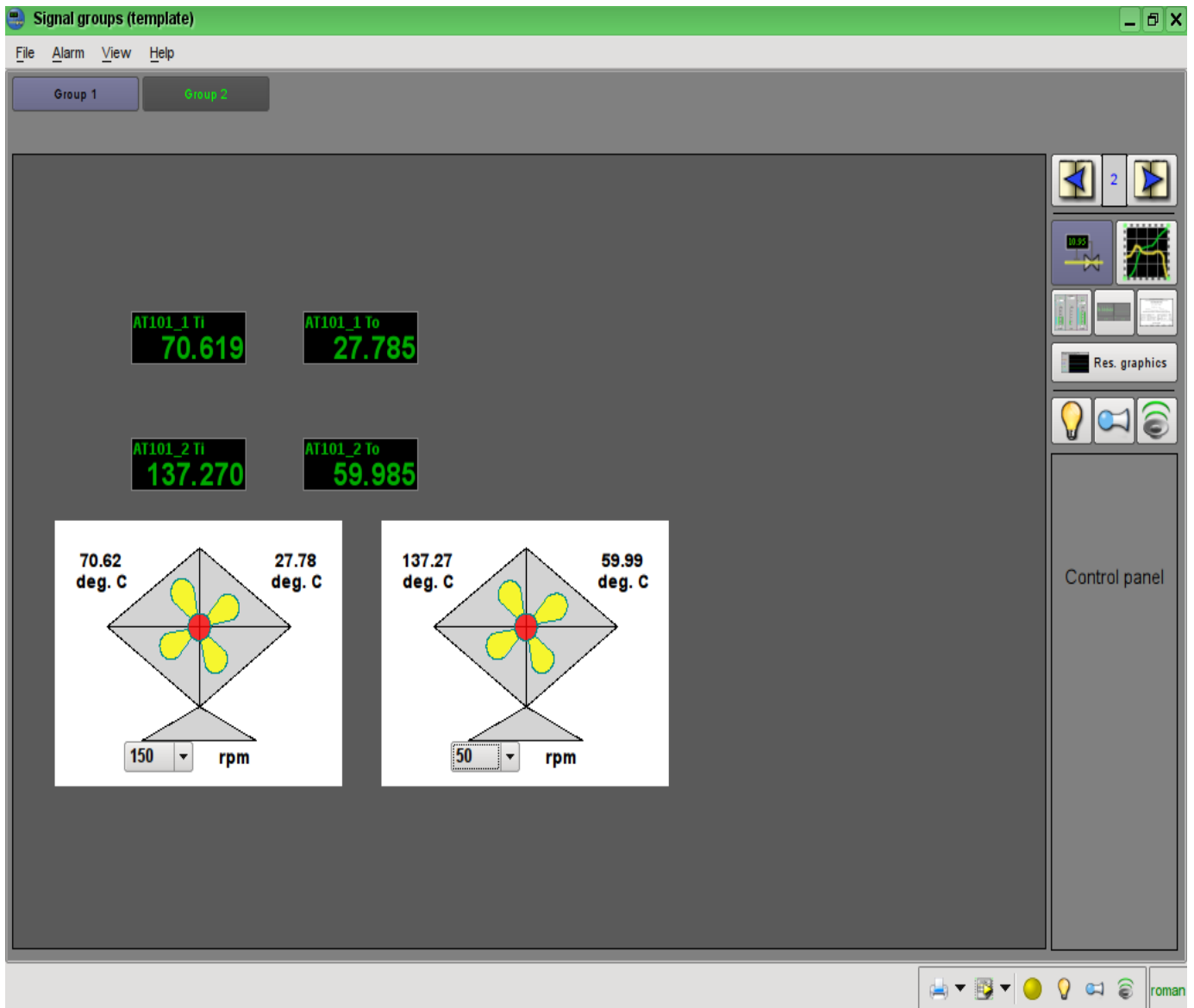


Fig. 5.3.3.3. The resulting mnemonic scheme.

On this mnemonic scheme through our complex elements we can not only observe but also to control the productivity of coolers, simply by changing the value in the combo box. Changing the productivity, we can see the changes in temperature. History of changes we can see on the created in the chapter 5.1 the group of graphs.

6. Recipes

This section is intended to provide the descriptions of recipes for solving the common problems and tasks of the user. Recipes to be placed in this section may be offered by the users.

Conclusion

This document describes in detail the basic process of creating the user interface elements, with preparation and configuration of the data source. In general, you can quickly get an idea of the work with the OpenSCADA system, and purposefully look for solutions of associated problems.

Module of subsystem “Archives”<FSArch>

<i>Module:</i>	FSArch
<i>Name:</i>	Arhivator on the file system
<i>Type:</i>	Archive
<i>Source:</i>	arh_FSArch.so
<i>Version:</i>	1.4.1
<i>Author:</i>	Roman Savochenko
<i>Description:</i>	Archive module. Provides archiving functions for messages and values on the file system.
<i>License:</i>	GPL

The module is designed for archiving messages and values of OpenSCADA on the file system.

Any SCADA system provides the ability to archive the collected data, i.e. formation of history of the changes (dynamics) of processes. Archives conditionally can be divided into two types: archives of messages and archives of values.

A feature of the archives of messages is that so-called events are archived. The characteristic feature of the events is its time of occurrence. The archives of messages are usually used for archiving, messages in the system, i.e. conducting of logs and reports. Depending on the source the messages can be classified according to different criteria. For example, this may be the reports of emergency situations, the reports of actions of the operators, reports of the glitches of connection and others.

A feature of the archives of values is their frequency, measured in the time lag between two adjacent values. Archives of values are used for archiving the history of continuous processes. As the process is continuous, it can only be archived by introducing the notion of quantization of time interviewing, because otherwise we get the archives of infinite dimensions in view of continuity of the nature of the process. In addition, practically, we can get value from the time limited by the data sources. For example, a fairly high-quality data sources in the industry, are rarely allowed to receive data at a frequency of more than 1kHz. And this is without taking into account of the sensors themselves, which have even less qualitative characteristics.

For conducting of archives in the system OpenSCADA the subsystem «Archives» is provided. This subsystem, according to the types of archives, consists of two parts: an archives of messages and archives of values. The subsystem, in general, is a module that allows you to create archives based on the different nature and methods of storing of data. This module provides a mechanism for the archiving on the file system for both: for the flow of messages, and for the flow of values.

1. Message Archiver

Archives of messages are formed by archiver. There can be the set of archivers, with individual settings, allowing to share archiving of different classes of messages.

The archiver of messages of this module allows you to store data in XML files or in the flat-text format. Markup language XML is a standard format that is easily understood by a lot of exterior applications. However, opening and reviewing of the files in this format requires considerable resources. On the other hand, the flat-text format requires far fewer resources, although not uniform, but also requires knowledge of its structure to deal with.

In any case, both formats are supported and the user can select any of them in accordance with his requirements.

Files of the archive are named by archivers based on the date of the first messages in the archive. For example so: <2006-06-21 17:11:04. Msg>.

Files of the archive can be limited in size and time. After exceeding the limit a new file is created. Maximum number of files in a directory of the archiver can also be restricted. After exceeding the limit on the number of files old files will be deleted!

In order to optimize the use of disk space archivers support package of old archives by gzip packer. Packaging is made after a long non-use of the archive.

When you are using the archives in the form of XML, appropriate files are loaded entirely! For a long time unused archives unloading timeout of access to the archive is used, after the exceeding of which the archive is unloaded from memory and then is packaged.

Module provides additional settings for the archiving process (Fig. 1).

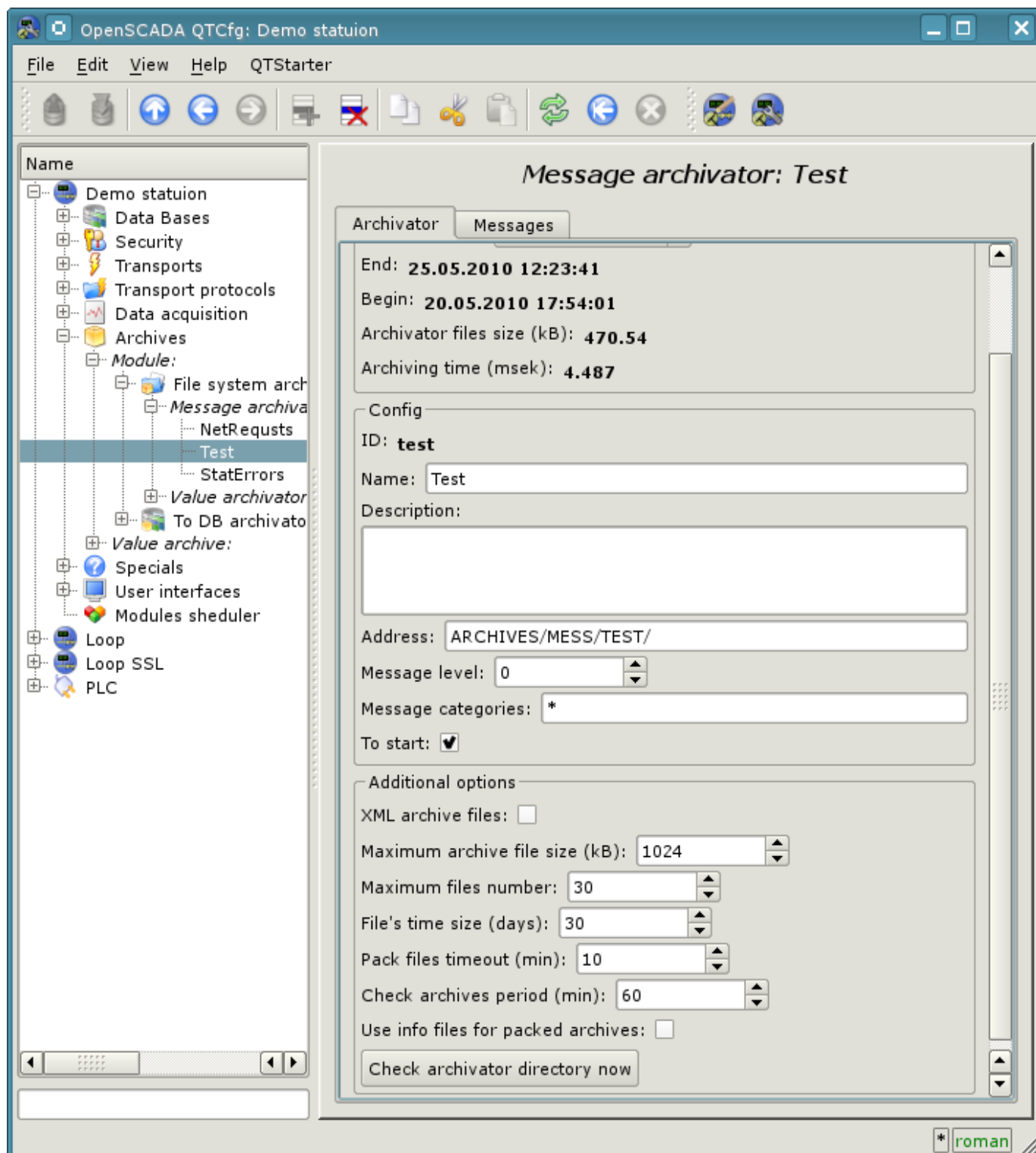


Fig.1. Additional settings of an archiving process of messages by module FSArch.

Those parameters include:

- Selecting of XML-format archive files.
- The maximum size of a single archive file.
- The maximum number of archived files.
- Limiting the size of the archive over time.
- Timeout of package of archive files.
- The frequency of inspection of files by archiver to search for new archives and deleting the old ones.
- The info-files using for packed archive files.
- The command of immediately verification of directory of archiver. It can be used with the placement in the directory of archiver of files of archives from other stations.

1.1. File format of archive messages

The table below shows the syntax of the archive file based on the XML-language:

Tag	Description	Attributes	Contains
FSArch	The root element. Identifies the file as belonging to the module.	<i>Version</i> — version of the archive file; <i>Begin</i> — the start time for the archive (hex – UTC in seconds from 01/01/1970); <i>End</i> — the end time for the archive (hex – UTC in seconds from 01/01/1970).	(m)
m	Tag of the single message.	<i>tm</i> — time of creation of the message (hex – UTC in seconds from 01/01/1970); <i>tmu</i> — microseconds of message's time; <i>lv</i> — message level <i>cat</i> — category of message.	Text of message

Archive file on the basis of the flat text consists of:

- header in the format: [FSArch <vers> <charset> <beg_tm> <end_tm>]
Where:
 - <vers> — version of the archiving module;
 - <charset> — code page of the file (usually UTF8);
 - <beg_tm> — UTC start time for the archive from 01.01.1970, in hexadecimal form;
 - <end_tm> — UTC end time for the archive 01.01.1970, in hexadecimal form.
- records of the messages in the format: [<tm> <lev> <cat> <mess>]
Where:
 - <tm> — message time in format <utc_sec:usec>, where:
 - *utc_sec* — UTC time from 01.01.1970, in hexadecimal form;
 - *usec* — microseconds of time, in decimal form.
 - <lev> — the level of importance of the message;
 - <cat> — category of the message;
 - <mess> — text of the message.

Text of the message and its category are coded to exclude separator symbols (space character).

1.2. Example of the archive of messages file

Example of the contents of an archive file in format of the XML language:

```
<?xml version='1.0' encoding='UTF-8' ?>
<FSArch Version="1.3.0" Begin="4a27dfbc" End="4a28c990">
<m tm="4a28cd01" tmu="942937" lv="4"
  cat="/DemoStation/sub_DAQ/mod_DiamondBoards/">dscInit error.</m>
<m tm="4a28cd12" tmu="466631" lv="4"
  cat="/DemoStation/sub_Transport/mod_Sockets/out_HDDTemp/">Connect to Internet
  socket error: Operation now in progress!</m>
</FSArch>
```

Example of the contents of the archive file in the format of flat text:

```
FSArch 1.3.0 UTF-8 4a27dfbb 4a28cd12
4a28cd11:295857 1 /DemoStation/ Start!
4a28cd11:296091 1 /DemoStation/sub_Transport/ Start%20subsystem.
4a28cd11:304391 1 /DemoStation/sub_DAQ/mod_DAQGate/cntr_test/ Enable%20controller!
4a28cd11:306362 1 /DemoStation/sub_DAQ/mod_ModBus/cntr_testTCP/ Enable%20controller!
4a28cd11:310956 1 /DemoStation/sub_DAQ/mod_ModBus/cntr_testRTU/ Enable%20controller!
4a28cd11:313845 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_Anast1to2node/ Enable
%20controller!
4a28cd11:531765 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM102cntr/ Enable
%20controller!
4a28cd11:557546 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_Anast1to2node_cntr/ Enable
%20controller!
4a28cd11:616320 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM101/ Enable%20controller!
4a28cd11:770404 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM102/ Enable%20controller!
4a28cd11:935745 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM201/ Enable%20controller!
4a28cd12:64148 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM202/ Enable%20controller!
4a28cd12:212514 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM301/ Enable
%20controller!
4a28cd12:331423 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM302/ Enable%20controller!
4a28cd12:462627 1 /DemoStation/sub_DAQ/mod_System/cntr_AutoDA/ Enable%20controller!
4a28cd12:466631 4 /DemoStation/sub_Transport/mod_Sockets/out_HDDTemp/ Connect%20to
%20Internet%20socket%20error:%20Operation%20now%20in%20progress!
4a28cd12:499705 1 /DemoStation/sub_DAQ/mod_SoundCard/cntr_test/ Enable%20controller!
4a28cd12:502482 1 /DemoStation/sub_DAQ/mod_LogicLev/cntr_experiment/ Enable
%20controller!
4a28cd12:620560 1 /DemoStation/sub_DAQ/mod_JavaLikeCalc/cntr_testCalc/ Enable
%20controller!
4a28cd12:624907 1 /DemoStation/sub_DAQ/mod_Siemens/cntr_test/ Enable%20controller!
4a28cd12:644620 1 /DemoStation/sub_DAQ/mod_DAQGate/cntr_test/ Enable%20controller!
4a28cd12:665980 1 /DemoStation/sub_Archive/ Start%20subsystem.
4a28cd12:843813 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_Anast1to2node/ Start
%20controller!
4a28cd12:845059 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM102cntr/ Start
%20controller!
4a28cd12:845555 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_Anast1to2node_cntr/ Start
%20controller!
4a28cd12:845983 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM101/ Start%20controller!
4a28cd12:846778 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM102/ Start%20controller!
4a28cd12:847440 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM201/ Start%20controller!
4a28cd12:849979 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM202/ Start%20controller!
4a28cd12:850851 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM301/ Start%20controller!
4a28cd12:851417 1 /DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM302/ Start%20controller!
4a28cd12:852073 1 /DemoStation/sub_DAQ/mod_System/cntr_AutoDA/ Start%20controller!
4a28cd12:854718 1 /DemoStation/sub_DAQ/mod_LogicLev/cntr_experiment/ Start
%20controller!
4a28cd12:889380 1 /DemoStation/sub_Archive/ Start%20subsystem.
4a28cd12:909319 1 /DemoStation/sub_UI/mod_VCAEngine/ Start%20module.
```

2. Values Archiver

Archives of values are formed particularly by archivers of the values for each registered archive. There can be a lot of archivers with individual settings that allow to divide the archives by various parameters, such as the accuracy and depth.

Archive of values is an independent component, which includes buffer processed by archivers. The main parameter of archive of value is a source of data. As a source of data may make the attributes of the parameters of subsystem “Data acquisition”, as well as other external data sources (passive mode). Other sources of data could be: network archivers of remote OpenSCADA systems, environment of programming of systems OpenSCADA etc. No less important parameters are the parameters of the archive buffer. From the parameters of the buffer the opportunity of working of archivers depends on. Thus, the frequency of values in the buffer should be no more than the frequency of the fastest archiver, a buffer size not less than double the amount for the slowest archiver. Otherwise, the possible loss of data!

The overall scheme of archival of values vividly depicted in Fig. 2.

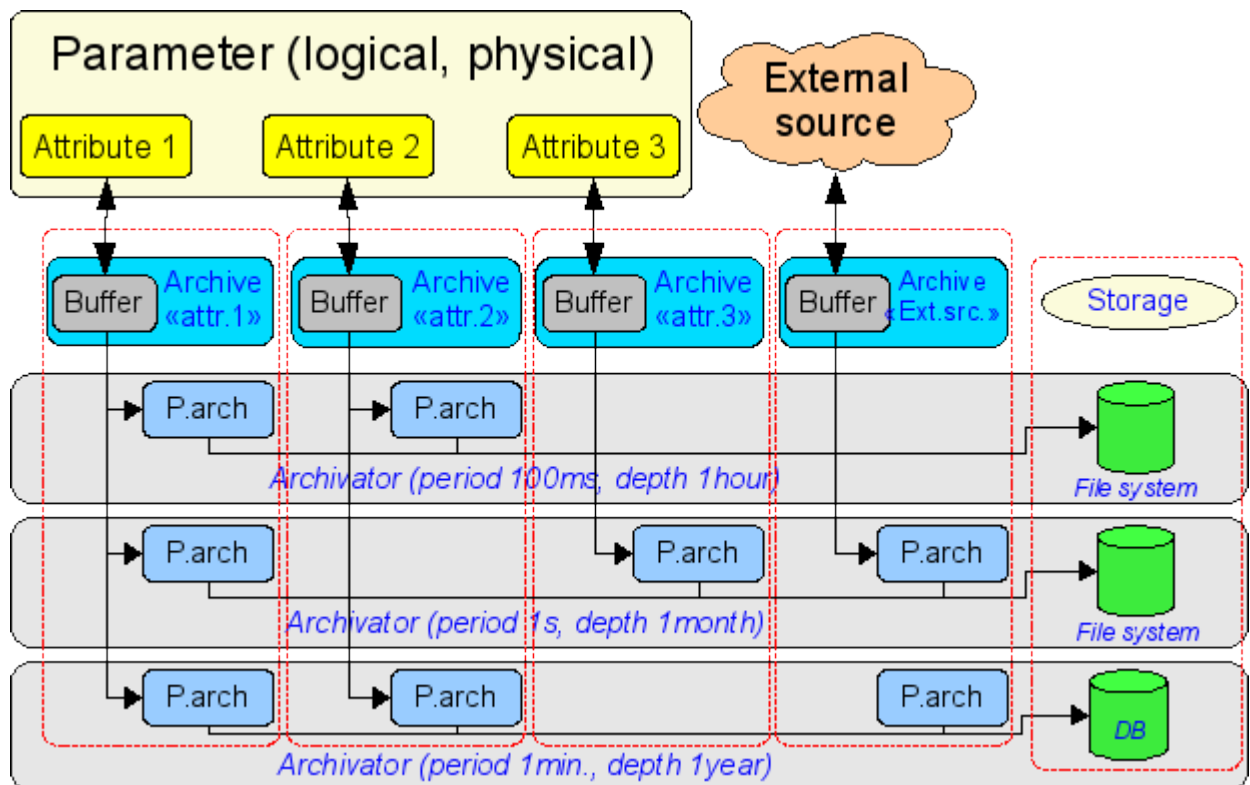


Fig.2. The overall scheme of process of archival values of module FSArch.

Files of archives are named by archivers based on the date of the first value in the archive and archive identifier. For example in this way: <MemInfo_use 2006-06-17 17:32:56.val>.

Files of archives can be limited in time. After exceeding the limit the new file is created. Maximum number of files in a directory of archiver also may be limited. After exceeding the limit on the number of files old files will be deleted!

In order to optimize the use of disk space archivers support package of old archives by gzip packer. Packaging is made after a long non-use of the archive. For fast archives connection allow to other systems you can enable info-files using for packed files, that prevent all files forward unpackaging at other system.

Module provides additional settings for the archiving process (Fig. 3).

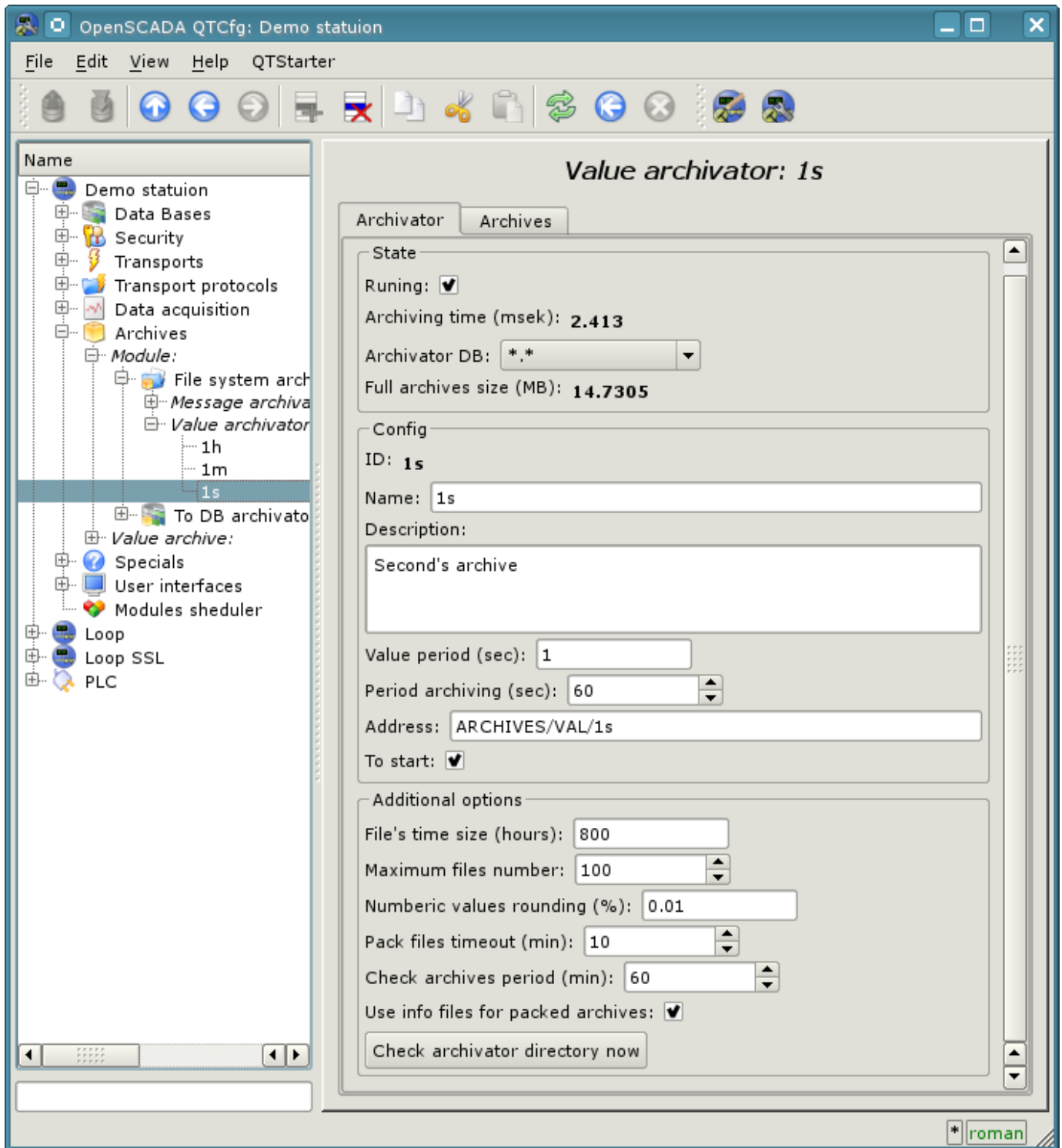


Fig.3. Additional settings of an archiving process of values by module FSArch.

2.1. File format of archive values

To implement the archiving to the file system the following requirements are to be done:

- quick (easy) access to add to the archive and reading from the archive;
- the possibility of changing the values of the existing archive (to fill holes in duplicate systems);
- cycle (size restrictions);
- the possibility of the compression by the method of packaging the same values sequence that preserves the possibility of quick access (consistent packaging);
- the possibility of packaging obsolete data by standard archivers (gzip, bzip2 ...), with the possibility of extracting on access.

In accordance with the above requirements archiving is organized by method of plurality of files (for each source). Cyclical of archive sold at the file level, ie a new file is created, and the oldest one is removed. For fast compression the method of tightening to the last equal value is used. For this purpose, the bit archiving table is provided with the size of one to one with the number of stored data. Ie each bit corresponds to the single value in the archive. The presence of bit indicates the presence of value. For the thread of the same values bits reduced to zero. In the case of the string archive the table is not a bit but the byte one and contains the length of the appropriate value. In the case of reception of the thread of equal values, the length will be zero and the first same value will be read. As the table is bite one, the archive will be able to keep strings with the length more than 255 characters. Thus, the methods of storage can be divided into a method of fixed and not fixed data size. The overall structure of the archive is shown in Fig. 4.

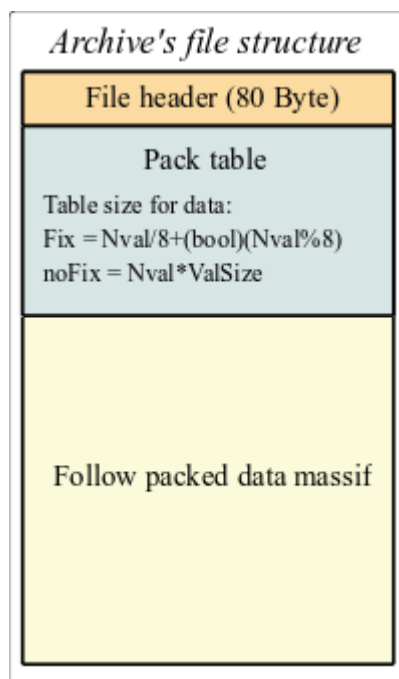


Fig. 4. The overall structure of the value archive.

When you create a new archive file there is formed: the title (the structure of the title is in the table 1), zero bit table of package of the archive and the first false value. Thus, the archive will be initialized with false values. In the future, the new values will be inserted in the area of values with adjustment of index table of packaging. It follows that the passive archives will dwindle in the files with the size of the title and the bit table.

Table 1. The structure of the header of archive file

Field	Description	Size in bite(bit)
f_tp	System name of the archive («OpenSCADA Val Arch.»)	20
archive	Name of the archive to which the file belongs.	20
beg	Start time of the archive data (mkc)	8
end	End time of the archive data (mkc)	8

Field	Description	Size in bite(bit)
period	Periodicity of the archive (MKC)	8
vtp	Type of value in the archive (Boolean, Integer, Real, String)	(3)
hgrid	Criterion of using of hard grid in the buffer of the archive	(1)
hres	Criterion of using of time of high resolution (mcs) in the buffer of the archive	(1)
reserve	Reserve	14
term	The symbol of the end of the header of file (0x55)	1

Explaining of the mechanism of consistent packaging is given in Fig. 5. As can be seen from the figure a sign of the package contains a length (not fixed types) or a sign of the package (fixed types) of the separately taken value. This means that to obtain the desired value of displacement it is necessary to sum up the length of previous valid values. The implementation of this operation each time and for each value is highly invoice operation. Therefore, the mechanism of caching of displacement of the values is provided. The mechanism caches displacement of values through predefined their quantity, as well as cashes the last value for which the access is made (separately for reading and writing).

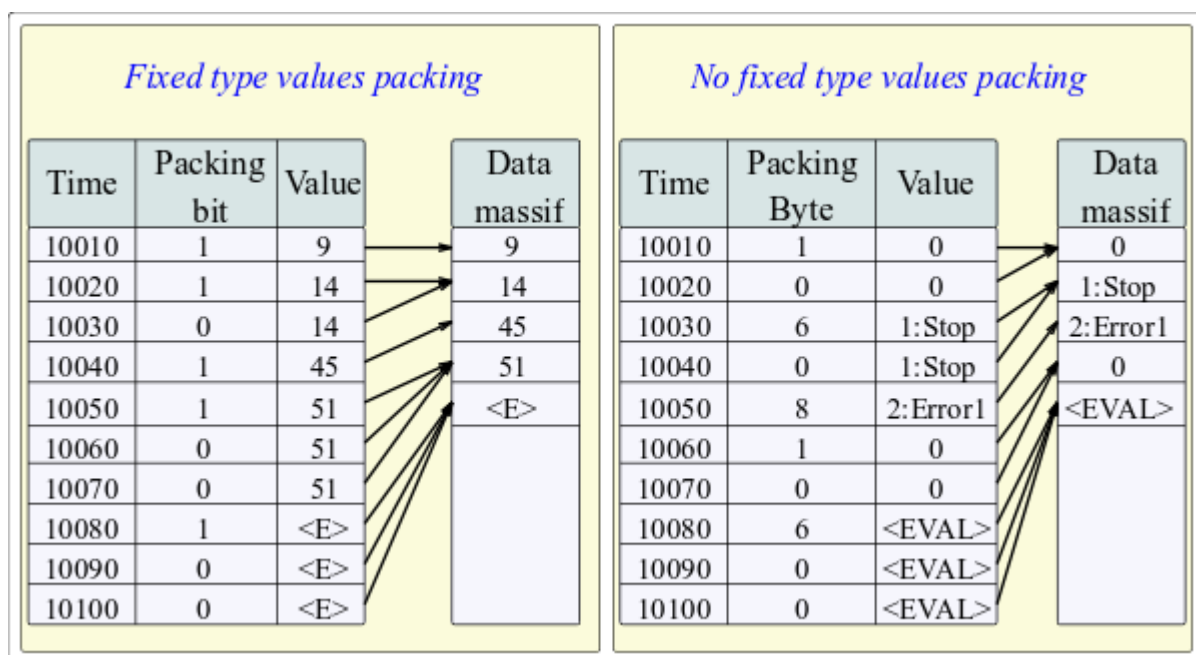


Fig. 5. The mechanism of follow packaging of values.

Changes of the values in the existing archive is also provided. However, given the necessity to implement the shifting of the tail of the archive, it is recommended to perform this operation as sparingly as possible and with as far as possible large blocks.

3. Efficiency

In the design and implementation of the module it was built mechanisms improving the process of archiving.

The first mechanism is a mechanism of block (frame-accurate or transactive) location of data in the files of the archives of values. Such an arrangement allows to achieve a maximum speed of archiving, and thus allows to archive more data streams at the same time. The experience of the practical using showed that the system of K8–3000 with a regular IDE hard drive is able to archive to 300000 data streams at a frequency of 1 second, or K5–400 system with the IDE drive (2.5”) can archive to 100 parameters with 1 millisecond intervals.

The second mechanism is the package of current values, and outdated files of archives to optimize the use of disk space. There are two packaging mechanisms: the consistent package (archives of values), and a mechanism of finish packaging of archives by means of standard packer (gzip). This approach allowed to achieve high productivity in the process of archiving of current data with the effective mechanism of consistent compression. And finish packaging by means of standard packer of obsolete archives completes the overall picture of the compact storage of large volumes of data. Statistics of practical using, in real noise signal (the worst situation), showed that the extent of consistent packaging is 10%, and the extent of the full packaging was 71%.

Module of subsystem “Archives” <DBArch>

<i>Module:</i>	DBArch
<i>Name:</i>	Архиватор on the DB
<i>Type:</i>	Archive
<i>Source:</i>	arh_DBArch.so
<i>Version:</i>	0.9.2
<i>Author:</i>	Roman Savochenko
<i>Description:</i>	Archive module. Provides archiving functions for messages and values on the DB.
<i>License:</i>	GPL

The module is designed for archiving messages and values of OpenSCADA to a database maintained by OpenSCADA.

Any SCADA system provides the ability to archive the collected data, i.e. formation of history of the changes (dynamics) of processes. Archives conditionally can be divided into two types: archives of messages and archives of values.

A feature of the archives of messages is that so-called events are archived. The characteristic feature of the events is its time of occurrence. The archives of messages are usually used for archiving, messages in the system, i.e. conducting of logs and reports. Depending on the source the messages can be classified according to different criteria. For example, this may be the reports of emergency situations, the reports of actions of the operators, reports of the glitches of connection and others.

A feature of the archives of values is their frequency, measured in the time lag between two adjacent values. Archives of values are used for archiving the history of continuous processes. As the process is continuous, it can only be archived by introducing the notion of quantization of time interviewing, because otherwise we get the archives of infinite dimensions in view of continuity of the nature of the process. In addition, practically, we can get value from the time limited by the data sources. For example, a fairly high-quality data sources in the industry, are rarely allowed to receive data at a frequency of more than 1kHz. And this is without taking into account of the sensors themselves, which have even less qualitative characteristics.

For conducting of archives in the system OpenSCADA the subsystem «Archives» is provided. This subsystem, according to the types of archives, consists of two parts: an archives of messages and archives of values. The subsystem, in general, is a module that allows you to create archives based on the different nature and methods of storing of data. This module provides a mechanism for the archiving on the file system for both: for the flow of messages, and for the flow of values.

1. Message Archiver

Archives of messages are formed by archiver. There can be the set of archivers, with individual settings, allowing to share archiving of different classes of messages.

The archiver of messages of this module stores data in a database table, which is named by the following way: DBAMsg_{ArchID}. Where:

- *ArchID* — archiver identifier.

The size of the table of archive may be limited in time. After exceeding the limit the old records will be deleted!

Module provides additional settings for the archiving process. This module has only one such parameter and it determines the size of the archive over time.

Table of the database archiver has the following structure: {**TM**, **TMU**, **CATEG**, **MESS**, **LEV**}. Where:

- *TM* — UTC time of the message, seconds from (01.01.1970). In the DB, containing a specialized type of storage date and time, can be used this specialized type.
- *TMU* — microseconds of time
- *CATEG* — message category.
- *MESS* — text of the message.
- *LEV* — level of the message.

2. Values Archiver

Archives of values are formed particularly by archivers of the values for each registered archive. There can be a lot of archivers with individual settings that allow to divide the archives by various parameters, such as the accuracy and depth.

Archive of values is an independent component, which includes buffer processed by archivers. The main parameter of archive of value is a source of data. As a source of data may make the attributes of the parameters of subsystem “Data acquisition”, as well as other external data sources (passive mode). Other sources of data could be: network archivers of remote OpenSCADA systems, environment of programming of systems OpenSCADA etc. No less important parameters are the parameters of the archive buffer. From the parameters of the buffer the opportunity of working of archivers depends on. Thus, the frequency of values in the buffer should be no more than the frequency of the fastest archiver, a buffer size not less than double the amount for the slowest archiver. Otherwise, the possible loss of data!

The overall scheme of archival of values vividly depicted in Fig. 1.

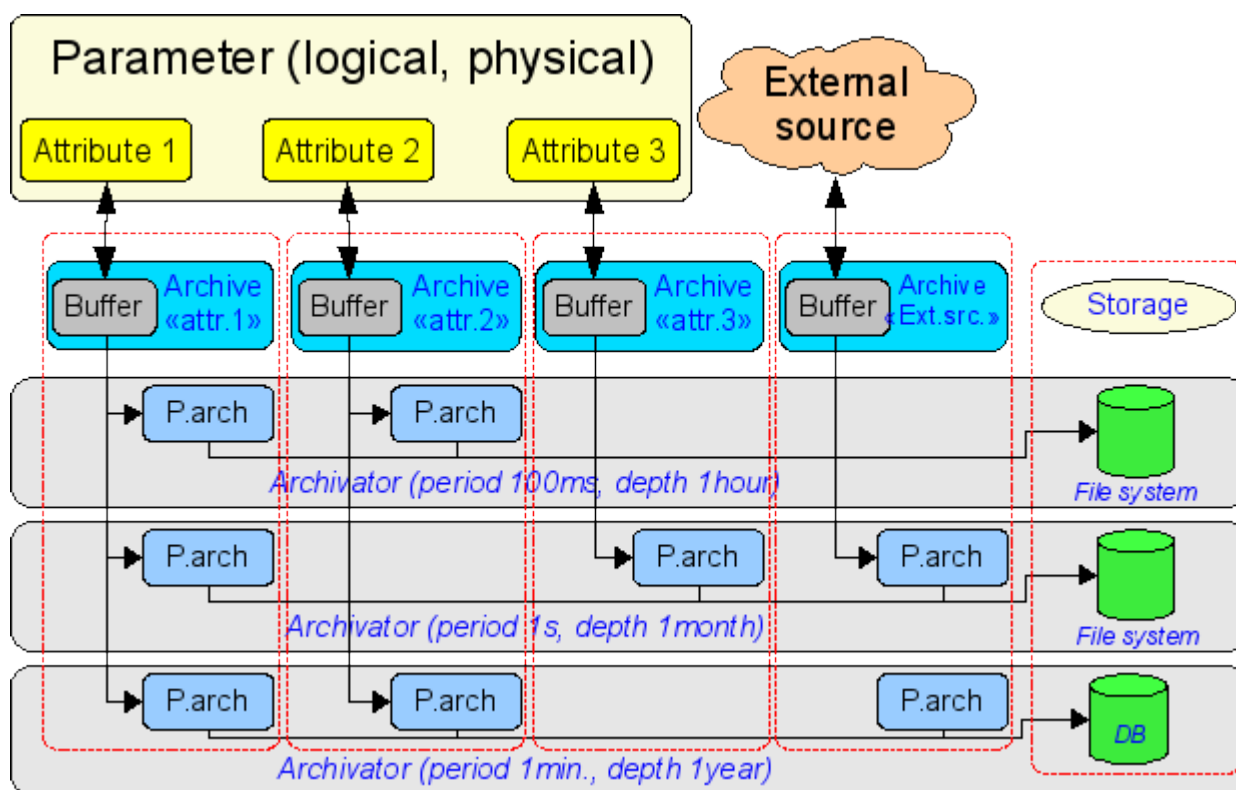


Fig.1. The overall scheme of the process of archiving by module DBArch.

Archive of this module stores data in a database table, which is called by the following way: DBAVI_{ArchID}_{ArchiveID}. Where:

- *ArchID* — identifier of the archiver of values.
- *ArchiveID* — identifier of the archive.

The size of the table of archive may be limited in time. After exceeding the limit the old records will be deleted!

Module provides additional settings for the archiving process. This module has only one such parameter and it determines the size of the archive over time.

Table of database archiver of values is as follows: **{TM, TMU, VAL}**. Where:

- *TM* — UTC time of the value, the second from (01.01.1970). In the databases, containing a specialized type of storage date and time, it can be used this type of specialization.
- *TMU* — Time value in microseconds.
- *VAL* — The value, type of value is determined by the type of the column.

3. Informational table of the archival tables

To store the beginning, end and other information of archives in archival tables the informational table with the name of the module is created: «DBArch». This table has the structure: **{TBL, BEGIN, END, PRM1, PRM2, PRM3}**. Where:

- *TBL* — Name of the table of the archive.
- *BEGIN* — Beginning of data in the archive.
- *END* — End of data in the archive.
- *PRM1* — Optional parameter 1.
- *PRM2* — Optional parameter 2.
- *PRM3* — Optional parameter 3.

Module of the subsystem “DB” <DBF>

<i>Module:</i>	DBF
<i>Nmae:</i>	DB DBF
<i>Type:</i>	DB
<i>Source:</i>	bd_DBF.so
<i>Version:</i>	2.0.2
<i>Author:</i>	Savochenko Roman
<i>Description:</i>	DB module. It provides the support of *.dbf files, version 3.0.
<i>License:</i>	GPL

The module is designed to provide in the system OpenSCADA support of the type of database files *.dbf. The module is based on the library for work with dbf files for “Complex2” firm “DIYA” Ltd. The module allows you to perform operations on databases, tables and contents of tables.

1. Operations over the database

The operations of opening and closing of the database is supported, with the possibility of creating a new database when you open and delete existing at the close. In terms of the subsystem “DB” of system OpenSCADA opening of DB is its registration for further using of it in the system.

Under the DB, in the case of the dbf-files it is meant the directory containing the dbf-files. Therefore, operation of the creating and deleting of the database – creates and deletes the directory where the table (dbf-files) are stored. The role of the address of database plays the full name of the directory with dbf-files. Access to the database is defined by the system rights of access to the directory.

The module supports coding of data in the correct code page. To this purpose, for the database as a whole, you can specify a working code page. During the work it will be carried out data coding, database coding, from the DB code page to the system code page of OpenSCADA and backwards.

2. Operations over the table

The operations of opening and closing of the table with the possibility of creating a new table when you open and deleting the existing one at the closing are supported.

Actually dbf-file is the table. Creation and deletion of tables implies creation and deletion of dbf-file. Table name is the name of dbf-file in the directory of DB. Access to the table are define by the rights of access to dbf-file.

3. Operations over the contents of the table

- Scanning of the records of the table;
- Request the values of these records;
- Setting the values of these records;
- Removing the records.

API of subsystem “DB” suppose the access to the contents of the table on the value of key(s) fields. Thus, the operation of request of the record implies the preset of key columns of the object TConfig, which will fulfill the request. Creating a new record(string) is the installation of the values

of record, which does not exist.

The module allows you to dynamically change the structure of the database tables DBF. Thus, in the event of a discrepancy of the table and the structure determined by record, the structure of the table will be reduced to the the required structure of record. In the case of the request of the value of the record, and mismatching of the structures of record and the table there will be available only to the values of common elements of the record and table. The module does not track the order of the elements in the record and in the structure of the table!

While access to the values of the tables the synchronization is used by through the capture of the resource to have access to the table. This avoids the destruction of data in the case of multi-access!

The types of the elements of dbf-file that correspond to types of elements of system OpenSCADA in the following way:

The types of fields of system OpenSCADA	Type of field of dbf-file
TFld::String	“C”
TFld::Integer, TFld::Real	“N”
TFld::Boolean	“L”

4. Productivity of DB

Measurement of productivity of DB were carried out by the test “DB” of the module of system tests "SystemTests", by performing operations over the records of the structure: <name char (20), descr char (50), val double (10.2), id int (7), stat bool>.

Operation	K8–3000+,256M,120G
<i>Creation of the 1000 records (sek):</i>	1.07
<i>Updating of the 1000 records (sek):</i>	1.6
<i>Getting of the 1000 records (sek):</i>	1.0
<i>Deleting of the 1000 record (sek):</i>	0.95

Module of the subsystem “DB” <MySQL>

<i>Module:</i>	MySQL
<i>Name:</i>	DB MySQL
<i>Type:</i>	DB
<i>Source:</i>	bd_MySQL.so
<i>Version:</i>	1.6.2
<i>Author:</i>	Savochenko Roman
<i>Description:</i>	DB module. It provides the support for DB MySQL.
<i>License:</i>	GPL

Module <MySQL> gives to the system OpenSCADA support of DB MySQL. MySQL database is a powerful multi-platform database available for free license. Manufacturer of MySQL database is the company MySQL AB <http://www.mysql.com>. The module is based on the library with API of the manufacturer of DB MySQL. The module allows you to perform operations over databases, tables and contents of tables.

1. Operations over the database

The operations of opening and closing of the database is supported, with the possibility of creating a new database when you open and delete existing at the close. In terms of the subsystem “DB” of system OpenSCADA opening of DB is its registration for further using of it in the system. It also supported the operation of requesting the list of tables in the database.

DB MySQL address by string of following type: [*<host>*; *<user>*; *<pass>*; *<bd>*; *<port>*; *<u_sock>*; *<names>*]. Where:

- *host* — the name of the host on which the database server MySQL works;
- *user* — the name of the user of database;
- *pass* — user password to access the database;
- *bd* — the name of the database;
- *port* — port to listen to by the database server (default is 3306);
- *u_sock* — the name of UNIX-socket in the case of local access to the database (/var/lib/mysql/mysql.sock).
- *names* — MySQL SET NAMES charset.

In the case of local access to the database in the same host, you must use the UNIX socket. For example: [*;roman; 123456; OpenSCADA;;/var/lib/mysql/ mysql.sock*]

In the case of remote access to the database you must use the host name and port of the server of the database. For example: [*server.nm.org;roman;123456;OpenSCADA;3306*]

2. Operations over the table

The operations of opening and closing of the table with the possibility of creating a new table when you open and deleting the existing one at the closing, and also the operation of the requesting of the table's structure are supported.

3. Operations over the contents of the table

- scanning of the records of the table;
- request the values of these records;
- setting the values of these records;
- removing the records.

API of subsystem “DB” suppose the access to the contents of the table on the value of key(s) fields. Thus, the operation of request of the record implies the preset of key columns of the object TConfig, which will fulfill the request. Creating a new record(string) is the installation of the values of record, which does not exist.

The module allows you to dynamically change the structure of the database tables MySQL. Thus, in the event of a discrepancy of the table and the structure determined by record, the structure of the table will be set to the required structure of record. In the case of the request of the value of the record, and mismatching of the structures of record and the table there will be available only to the values of common elements of the record and table. The module does not track the order of the elements in the record and in the structure of the table!

The module is realised support multilanguage text variables. For fields with multilanguage text variable create the column of separated language in format **<lang>#<FldID>** (en#NAME). In this time the base column contain value for base language. The columns of separated languages created by needs, in time saving to DB and execution OpenSCADA in correspond language. If for work language value no present then will used value for base language.

The types of the elements of DB MySQL correspond to types of elements of system OpenSCADA in the following way:

The types of fields of the system OpenSCADA	Types of fields of DB MySQL
TFld::String	char (n), text, mediumtext
TFld::Integer	int (n), DATETIME [for fields with a flag TFld::DateTimeDec]
TFld::Real	double(n, m)
TFld::Boolean	tinyint(1)

4. Access rights

MySQL database provides a powerful mechanism for the separation of access, which is to selectively identify the access for user of the database to specific SQL-commands. The following table lists the operation over the database and the required access to the commands of these operations.

Operation	SQL-commands
Creation of the database and tables	CREATE
Deleting of the database and tables	DROP
Adding of records	INSERT
Deleting the values of records	DELETE
Getting the values of records	SELECT
Setting the values of records	UPDATE
Manipulation with the structure of the table	ALTER

5. Productivity of DB

Measurement of productivity of DB were carried out by the test “DB” of the module of system tests "SystemTests", by performing operations over the records of the structure: <name char (20), descr char (50), val double (10.2), id int (7), stat bool>.

Operation	K8-3000+, 384M, 120G, MySQL 5.0.51 (local)	MySQL 4.0.24 (remote)
<i>Creation of the 1000 records (sec.):</i>	0.67	0.99
<i>Updating of the 1000 records (sec.):</i>	0.67	1.33
<i>Getting of the 1000 records (sec.):</i>	0.38	0.49
<i>Deleting of the 1000 record (sec.):</i>	0.23	0.34

Module of the subsystem “DB” <SQLite>

<i>Module:</i>	SQLite
<i>Name:</i>	DB SQLite
<i>Type:</i>	DB
<i>Source:</i>	bd_SQLite.so
<i>Version:</i>	1.6.2
<i>Aurhor:</i>	Savochenko Roman
<i>Description:</i>	DB module. It provides the support for DB SQLite.
<i>License:</i>	GPL

Module <SQLite> gives to the system OpenSCADA support of DB SQLite. DB SQLite is a small, embedded database which supports the SQL-queries. SQLite DB is distributed under a free license. To familiarize with the database it is possible on the website of the database – <http://sqlite.org>. The module is based on the library with API of the manufacturer of DB SQLite. The module allows you to perform operations over databases, tables and contents of tables.

1. Operations over the database

The operations of opening and closing of the database is supported, with the possibility of creating a new database when you open and delete existing at the close. In terms of the subsystem “DB” of system OpenSCADA opening of DB is its registration for further using of it in the system. It also supported the operation of requesting the list of tables in the database.

SQLite database is addressed by specifying the name of the database file in the following format: [*<FileDBPath>*]. Where:

- *FileDBPath* - full path to DB file (./oscada/Main.db).
Use empty path for a private, temporary on-disk database create.
Use ":memory:" for a private, temporary in-memory database create.

The module supports coding of data in the correct code page. To this purpose, for the database as a whole, you can specify a working code page. During the work it will be carried out data coding, database coding, from the DB code page to the system code page of OpenSCADA and backwards.

2. Operations over the table

The operations of opening and closing of the table with the possibility of creating a new table when you open and deleting the existing one at the closing, and also the operation of the requesting of the table's structure are supported.

3. Operations over the contents of the table

- scanning of the records of the table;
- request the values of these records;
- setting the values of these records;
- removing the records.

API of subsystem “DB” suppose the access to the contents of the table on the value of key(s) fields. Thus, the operation of request of the record implys the preset of key columns of the object TConfig, which will fulfill the request. Creating a new record(string) is the installation of the values of record, which does not exist.

The module allows you to dynamically change the structure of the database tables SQLite. Thus, in the event of a discrepancy of the table and the structure determined by record, the structure of the table will be

set to the required structure of record. In the case of the request of the value of the record, and mismatching of the structures of record and the table there will be available only to the values of common elements of the record and table. The module does not track the order of the elements in the record and in the structure of the table!

The module is realised support multilanguage text variables. For fields with multilanguage text variable create the column of separated language in format <lang>#<FldID> (en#NAME). In this time the base column contain value for base language. The columns of separated languages created by needs, in time saving to DB and execution OpenSCADA in correspond language. If for work language value no present then will used value for base language.

The types of the elements of DB SQLite correspond to types of elements of system OpenSCADA in the following way:

The types of fields of the system OpenSCADA	Types of fields of database SQLite
TFld::String	TEXT
TFld::Integer, TFld::Boolean	INTEGER
TFld::Real	DOUBLE

4. Access rights

Access rights to the database are defined by the rights of access to the separately taken file of the database. Module supports the work with SQLite database files in read-only mode, such as demonstrations.

5. Productivity of DB

Measurement of productivity of DB were carried out by the test “DB” of the module of system tests "SystemTests", by performing operations over the records of the structure: <name char (20), descr char (50), val double (10.2), id int (7), stat bool>.

Operation	K8–3000+, 256M, 120G, SQLite 3.4.2
<i>Creation of the 1000 records (sec.):</i>	0.45
<i>Updating of the 1000 records (sec.):</i>	0.50
<i>Getting of the 1000 records (sec.):</i>	0.2
<i>Deleting of the 1000 record (sec.):</i>	0.2

Module of the subsystem “DB” <FireBird>

<i>Module:</i>	FireBird
<i>Name:</i>	DB FireBird
<i>Type:</i>	DB
<i>Source:</i>	bd_FireBird.so
<i>Version:</i>	0.9.5
<i>Author:</i>	Savochenko Roman
<i>Description:</i>	DB module. It provides the support for DB FireBird.
<i>License:</i>	GPL

Module <FireBird> gives the system OpenSCADA support of DB FireBird and InterBase. DB FireBird is a small, embedded database, with the functions of a network database that supports SQL-queries. DB FireBird is built on a commercial DBMS Interbase and distributed under a free license. To familiarize with the database it is possible on the website of the database – <http://www.firebirdsql.org>. The module is based on the library with API of the manufacturer of DB. The module allows you to perform operations over databases, tables and contents of tables.

1. Operations over the database

The operations of opening and closing of the database is supported, with the possibility of creating a new database when you open and delete existing at the close. In terms of the subsystem “DB” of system OpenSCADA opening of DB is its registration for further using of it in the system. It also supported the operation of requesting the list of tables in the database.

DB FireBird is addressed by specifying the database file name, username and password. In general, the address database is written in this way: [*<file>;<user>;<pass>*]. Where:

- *file* – the full name of the database file;
- *user* – user of the database on behalf of which the access is made;
- *pass* – password for the user on behalf of which the access is made;

The module supports coding of data in the correct code page. To this purpose, for the database as a whole, you can specify a working code page. During the work it will be carried out data coding, database coding, from the DB code page to the system code page of OpenSCADA and backwards.

2. Operations over the table

The operations of opening and closing of the table with the possibility of creating a new table when you open and deleting the existing one at the closing, and also the operation of the requesting of the table's structure are supported.

3. Operations over the contents of the table

- scanning of the records of the table;
- request the values of these records;
- setting the values of these records;
- removing the records.

API of subsystem “DB” suppose the access to the contents of the table on the value of key(s) fields. Thus, the operation of request of the record implies the preset of key columns of the object TConfig, which will fulfill the request. Creating a new record(string) is the installation of the values of record, which does not exist.

The module allows you to dynamically change the structure of the database tables FireBird. Thus, in the

event of a discrepancy of the table and the structure determined by record, the structure of the table will be set to the required structure of record. In the case of the request of the value of the record, and mismatching of the structures of record and the table there will be available only to the values of common elements of the record and table. The module does not track the order of the elements in the record and in the structure of the table!

The module is realised support multilanguage text variables. For fields with multilanguage text variable create the column of separated language in format <lang>#<FldID> (en#NAME). In this time the base column contain value for base language. The columns of separated languages created by needs, in time saving to DB and execution OpenSCADA in correspond language. If for work language value no present then will used value for base language.

The types of the elements of DB FireBird correspond to types of elements of system OpenSCADA in the following way:

The types of fields of the system OpenSCADA	Types of fields of database FireBird
TFld::String	VARCHAR, BLOB SUBTYPE TEXT
TFld::Integer	INTEGER
TFld::Real	DOUBLE
TFld::Boolean	SMALLINT

4. Access rights

Access rights to the database are defined by the rights of DB.

5. Productivity of DB

Measurement of productivity of DB were carried out by the test “DB” of the module of system tests "SystemTests", by performing operations over the records of the structure: <name char (20), descr char (50), val double (10.2), id int (7), stat bool>.

Operation	K8–3000+, 256M, 120G, FireBird 2.0.3 (Local SuperServer)	FireBird 2.0.3 (Remote SuperServer)
<i>Creation of the 1000 records (sec.):</i>	1.23	2.76
<i>Updating of the 1000 records (sec.):</i>	4.43	6.92
<i>Getting of the 1000 records (sec.):</i>	2.31	4
<i>Deleting of the 1000 record (sec.):</i>	1.01	2.39

Module of the subsystem “DB” <PostgreSQL>

<i>Module:</i>	PostgreSQL
<i>Name:</i>	DB PostgreSQL
<i>Type:</i>	DB
<i>Source:</i>	bd_PostgreSQL.so
<i>Version:</i>	0.9.0
<i>Author:</i>	Maxim Lysenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	DB module. It provides the support for DB PostgreSQL.
<i>License:</i>	GPL

Module <PostgreSQL> gives to the system OpenSCADA support of DB PostgreSQL. PostgreSQL database is a powerful multi-platform database available for free license. Manufacturer of PostgreSQL database is the PostgreSQL Global Development Group www.postgresql.org. The module is based on the library with API of the manufacturer of DB PostgreSQL. The module allows you to perform operations over databases, tables and contents of tables.

1. Operations over the database

The operations of opening and closing of the database are supported, with the possibility of creating a new database when you try to open one and delete the existing at the close. In terms of the subsystem "DB" of system OpenSCADA opening of DB is its registration for further using of it in the system. It also supported the operation of requesting the list of tables in the database.

DB PostgreSQL address by string of following type:

[<host>;<hostaddr>;<user>;<pass>;<bd>;<port>;<connect_timeout>]. Where:

- host - the name of host to connect to. If this begins with a slash, it specifies Unix-domain communication rather than TCP/IP communication; the value is the name of the directory in which the socket file is stored.
- hostaddr - Numeric IP address of host to connect to. This should be in the standard IPv4 address format, e.g., 172.28.40.9. If your machine supports IPv6, you can also use those addresses. TCP/IP communication is always used when a nonempty string is specified for this parameter.
- user - the name of the user of database;
- pass - user password to access the database;
- bd - the name of the database;
- port - port to listen to by the database server (default is 5432);
- connect_timeout - maximum wait for connection, in seconds. Zero or not specified means wait indefinitely. It is not recommended to use a timeout of less than 2 seconds.

In the case of local access to the database in the same host the address string should be as follows: [;roman;123456;OpenSCADA;;10]

In the case of remote access to the database you must use the address and port of the server of the database. For example:[server.nm.org;192.168.2.1;roman;123456;OpenSCADA;;10]

2. Operations over the table

The operations of opening and closing of the table with the possibility of creating a new table when you open and deleting the existing one at the closing, and also the operation of the requesting of the table's structure are supported.

3. Operations over the contents of the table

- scanning of the records of the table;
- request the values of these records;
- setting the values of these records;
- removing the records.

API of subsystem “DB” suppose the access to the contents of the table on the value of key(s) fields. Thus, the operation of request of the record implies the preset of key columns of the object TConfig, which will fulfill the request. Creating a new record(string) is the installation of the values of record, which does not exist.

The module allows you to dynamically change the structure of the PostgreSQL database tables. Thus, in the event of a discrepancy of the table and the structure determined by record, the structure of the table will be set to the required structure of record. In the case of the request of the value of the record, and mismatching of the structures of record and the table there will be available only to the values of common elements of the record and table. The module does not track the order of the elements in the record and in the structure of the table!

The module provides the support of multilanguage text variables. For fields with multilanguage text variables the columns of the appropriate language are created in format <lang>#<FldID> (en#NAME). In this time the base column contain value for base language. The columns of other languages are created by needs, at the time of saving to DB and execution OpenSCADA with appropriate language. In the case of the value's absence for the language it will be used the values for basic language.

The types of the elements of DB PostgreSQL correspond to types of elements of system OpenSCADA in the following way:

The types of fields of the system OpenSCADA	Types of fields of DB PostgreSQL
TFld::String	character(n), character varying(n), text
TFld::Integer	integer, bigint, timestamp with time zone [for the fields with the flag TFld::DateTimeDec]
TFld::Real	double precision
TFld::Boolean	smallint

4. Access rights

PostgreSQL database contains some mechanism of separation of access, which is to specify the user privileges for database. The table below lists the necessary privileges for the work in the OpenSCADA.

Operation	SQL-commands
Creation of the DB	CREATEDB
Creation of the connection	LOGIN

5. Productivity of DB

Measurement of productivity of DB were carried out by the test "DB" of the module of system tests "SystemTests", by performing operations over the records of the structure: <name char (20), descr char (50), val double (10.2), id int (7), stat bool>. OpenSCADA was launched with the demo configuration.

Operation	K8-3000+, 384M, 120G, PostgreSQL 8.3 (local)	PostgreSQL 8.3 (remote)
Creation of the 1000 records (sec.):	0.89	1.04
Updating of the 1000 records (sec.):	1.02	1.1
Getting of the 1000 records (sec.):	0.61	0.63
Deleting of the 1000 record (sec.):	0.36	0.4

The module of subsystem “Data acquisition” <DiamondBoards>

<i>Module:</i>	DiamondBoards
<i>Name:</i>	Diamond cards of data acquisition
<i>Type:</i>	DAQ
<i>Source:</i>	daq_DiamondBoards.so
<i>Version:</i>	1.2.1
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides an access to the cards of data acquisition from Diamond Systems. Includes support for Athena motherboard.
<i>License:</i>	GPL

The module provides for the system OpenSCADA support of dynamic data sources, based on the cards for data collection of Diamond Systems company (<http://diamondsystems.com>). The module is built on the basis of a universal driver of the manufacturer of board. Universal driver is available for almost all known software platforms in the form of a library. Universal driver has been received at <http://www.diamondsystems.com/support/software>. The driver was included in the distribution kit of OpenSCADA, therefore, for the building of the module external libraries are not required.

The boards of data acquisition of Diamond Systems represent the modules of expansion of the PC/104 format. Boards may include: analog IO (input/outputs), digital IO, and counters. Complete set of cards can vary greatly. There can be contained only one type of IO or many others. In addition, the function of data acquisition can be given to the system boards of this company. For example, the motherboard Athena contains: 16 AI, 4 AO, 24 DIO.

The module provides support for analog and digital IO. The of analog inputs (AI) is supported in two modes: direct acquisition and the acquisition on interruption. The method of the acquisition on interruption allows to achieve the maximum frequency of interrogation which is supported by the hardware. In the case of Athena, the frequency achieves 100 kHz. The process of acquisition on interruption data becomes the second frames and placed in the archives buffer.

In the case of interrogation of the analog channels on interruption is not possible to configure individually each channel. Such an opportunity is provided only through direct interrogation.

Discrete channels are usually bi-directional and grouped into 8 channels. Each group of channels can be separately designate direction. The module provides the ability to configure a group of discrete parameters.

Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level.

1. Data controller of Diamond boards

Board of Diamond Systems configured by creating the controller in the system OpenSCADA and configuration of it. Example of the tab of configuration of the controller of the board is shown in Figure 1.

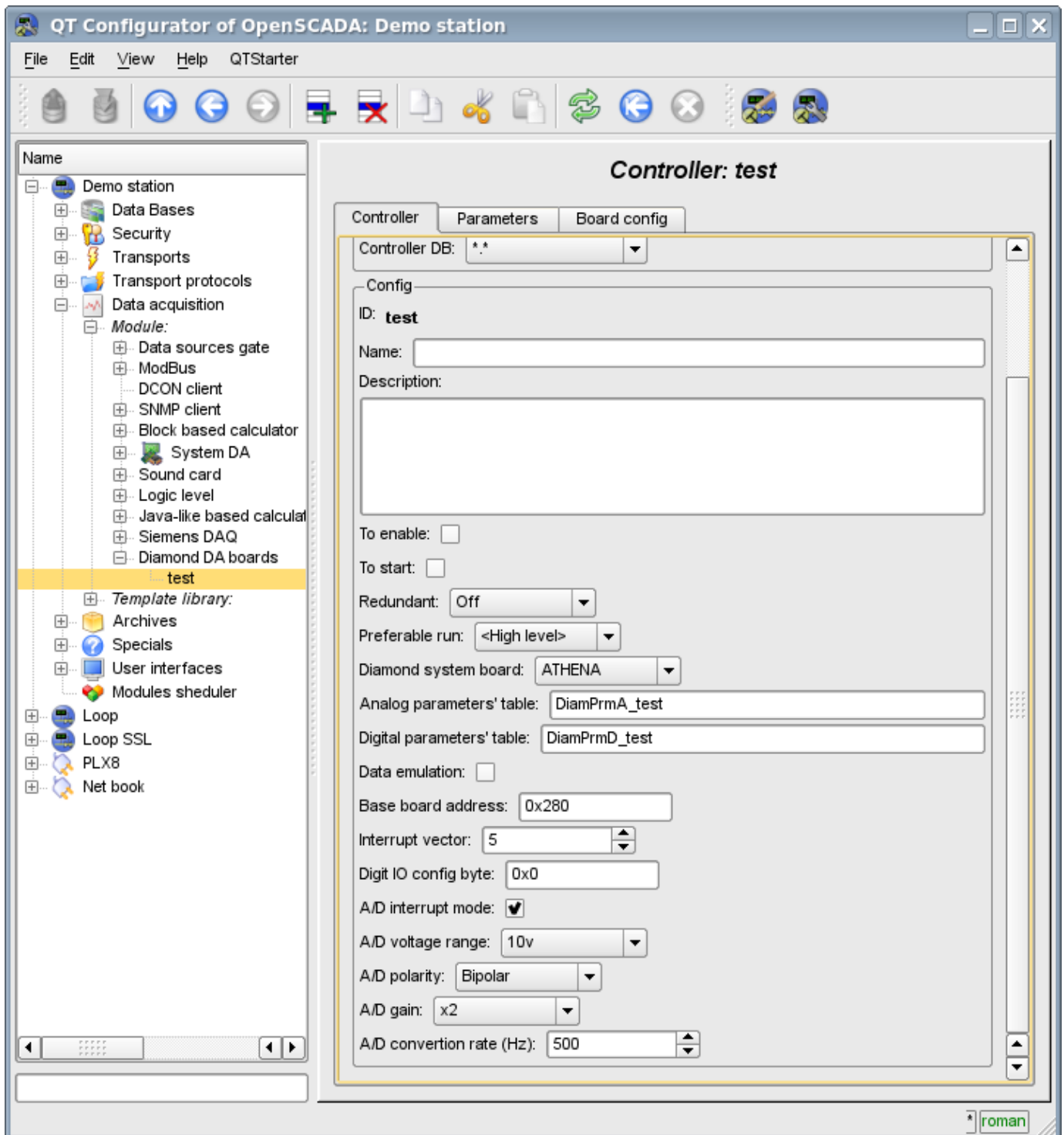


Fig.1. Tab of configuration of the controller/board of Diamond Systems.

Using this form, it can be set:

- The status of the controller(card), as follows: Status, "Enable", "Run" and the name of the database containing the configuration.
- Identifier, name and description of the controller(card).
- The status, in which the controller is to be transferred at the boot time:"To enable" and "To start".
- Horizontal mode of redundancy and performance preference of the controller.
- Type of the card of Diamond Systems company.

- The names of tables for storing of the configuration of analog and discrete parameters of the controller.
- The switching on of high-speed emulation mode of the data source.
- Base address and hardware interruption of the board(for the acquisition on interruption).
- Sign of the acquisition of analog inputs on the interruption and the frequency of data acquisition on the same channel.
- The overall configuration of the converter of analog inputs on the following structure: the range of input voltage, polarity and amplification of the channels.

In the mode of direct interrogation of analog inputs hardware interrupt of the card, frequency of analog inputs interrogation and the strengthening of the analog converter are not available.

To configure ports of digital inputs / outputs on the controller's page there is the tab of the configuration (fig. 2).

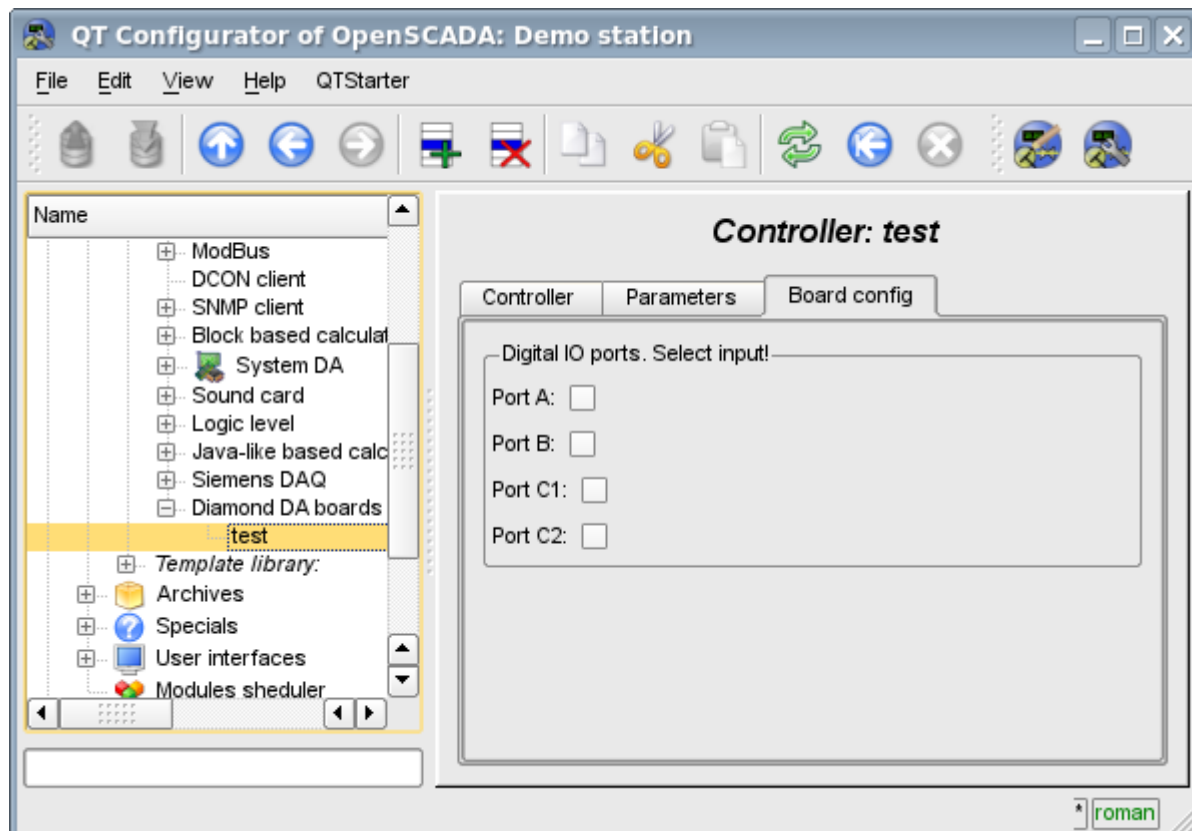


Fig.2. Tab of configuration of digital inputs / outputs ports.

2. Parameters of the Diamond controller

Module provides the information on two types of parameters: the digital and analog. Each type of the parameter is stored in the database and, consequently, has its own tab configuration. Tab of the configuration of analog parameters is presented in Fig.3. Configuration tab of digital parameters is presented in Fig.4.

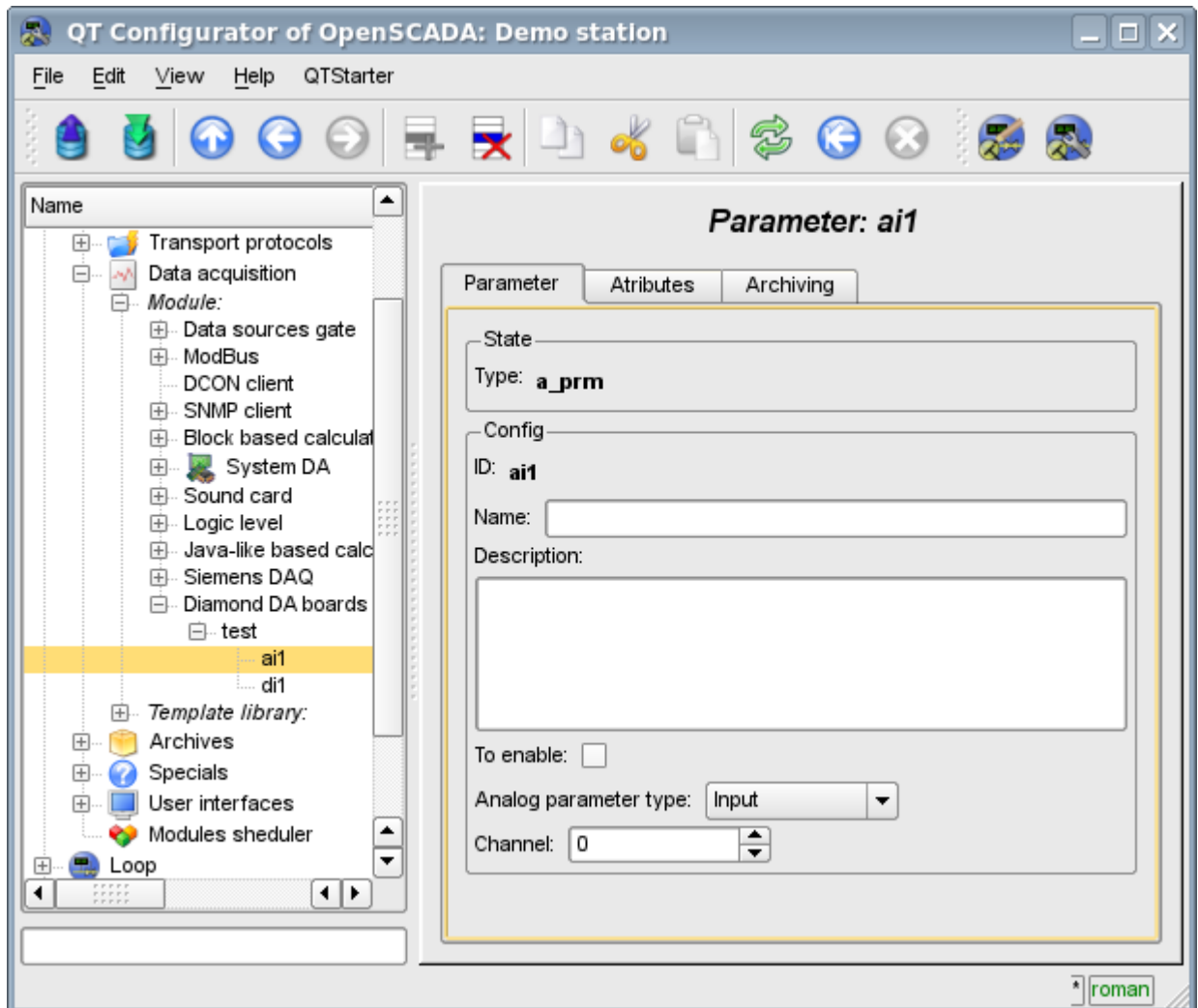


Fig.3. Tab of the configuration of analog parameters.

Using the form of configuration of analog parameters it can be set:

- Mode of the parameter, namely "Enabled" and type of the parameter.
- Id, name and description of the parameter.
- The state in which the parameter is to be transferred at boot time: "To enable".
- The orientation of the parameter - "Input" or "Output".
- Physical channel of the parameter.
- Strengthening of the channel in the case of input(for direct interrogation).

To access the values of analog parameters are attributes must be formed. For analog inputs:

- the percentage value (value);
- input voltage (voltage);
- ADC code (code).

For analog outputs are set:

- the percentage value (value);
- output voltage (voltage).

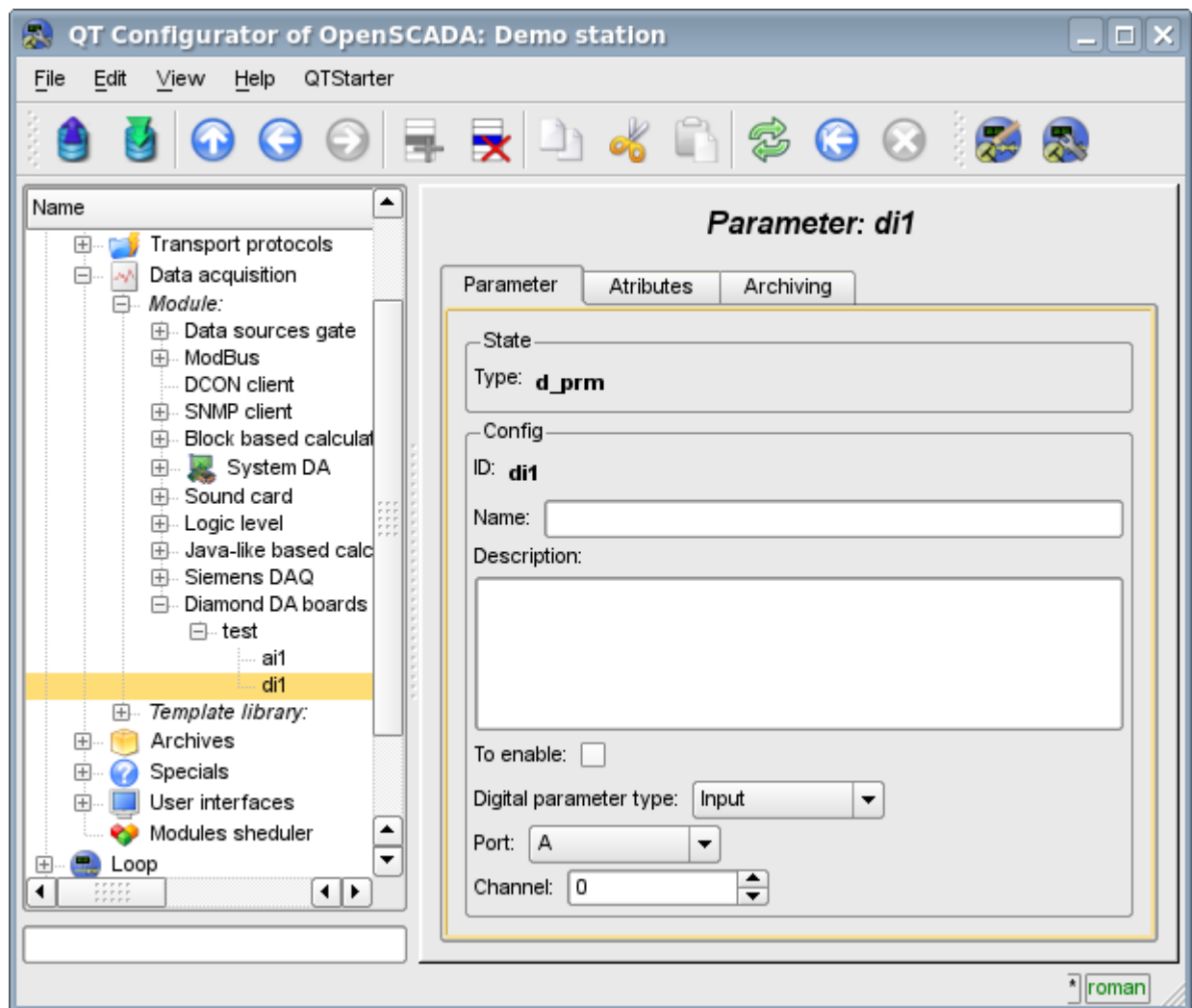


Fig.4. Configuration tab of digital parameters.

Using the Configuration tab of digital parameters there can be set:

- Mode of the parameter, namely, "Enable" and the type of parameter.
- Id, name and description of the parameter.
- The state in which the parameter is to be transferred at boot time: "To enabled".
- The orientation of the parameter - "Input" or "Output".
- Physical port and number of the channel.

To access the values of digital parameters the attribute, which provides the input value or inserts the new one, must be formed.

Links

Used version the Linux driver from Diamond systems: dscud5.91linux.tar.gz

The patch for build driver at kernel Linux 2.6.29, used for data gathering by interrupt: lastkernels.patch

The module of subsystem “Data acquisition” <System>

<i>Module:</i>	System
<i>Name:</i>	Data acquisition of OS
<i>Type:</i>	DAQ
<i>Source:</i>	daq_System.so
<i>Version:</i>	1.7.2
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides data acquisition from the OS. Supported data sources of OS Linux: HDDTemp, LMSensors, Uptime, Memory, CPU etc.
<i>License:</i>	GPL

The module is a sort of gateway between the system OpenSCADA and OS (operating system). The module receives data from various data sources of the OS and allows to manage components of the OS (in the future).

The module provides the ability to automatically search for the supported and active data sources with the establishment of parameters for access to them as well as the implementation of the function of the horizontal reservation, namely, working in conjunction with the remote station of the same level.

1. The controller of data

To add a data source of operating system there is created and configured the controller in the system OpenSCADA. Example of the configuration tab of the controller of the given type depicted in Fig. 1.

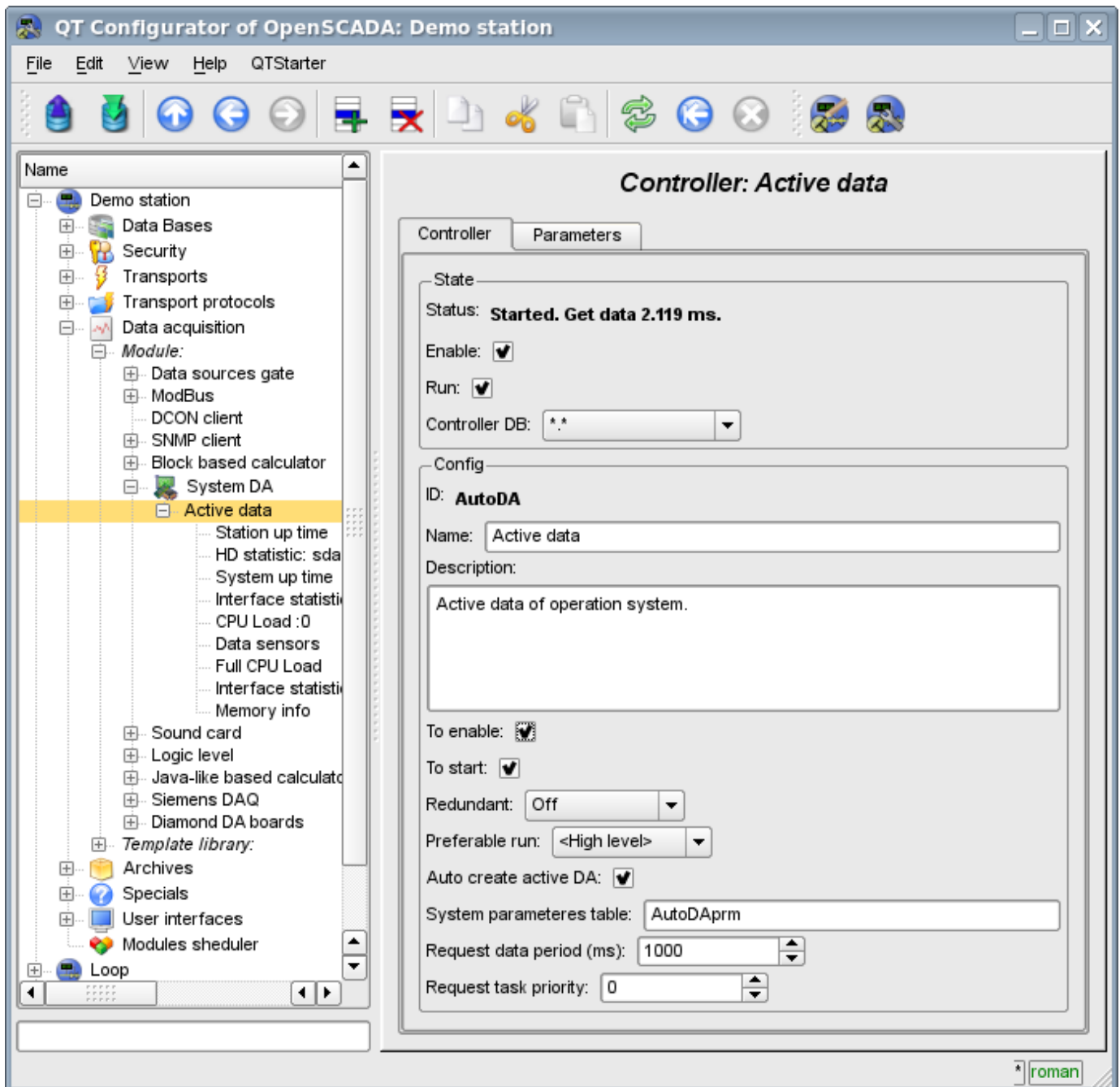


Fig.1. Tab of configuration of the controller.

From this tab you can set:

- The state of the controller, as follows: Status, "Enable", "Run" and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state in which the parameter is to be transferred at boot time: "To enable", "To start".
- Horizontal reservation mode and preference of the performance of the controller.
- Feature "Automatic search of active data sources and the creation of parameters for them".
- Name of table to store the configuration of the controller parameters.
- The period and the priority tasks of the interrogation of data sources.

2. Parameters

Module *System* provides only one type of parameters – “All parameters”. Additional configuration fields of the parameters of the module (Fig. 2) are:

- part of the system;
- optional (depending on the data source).

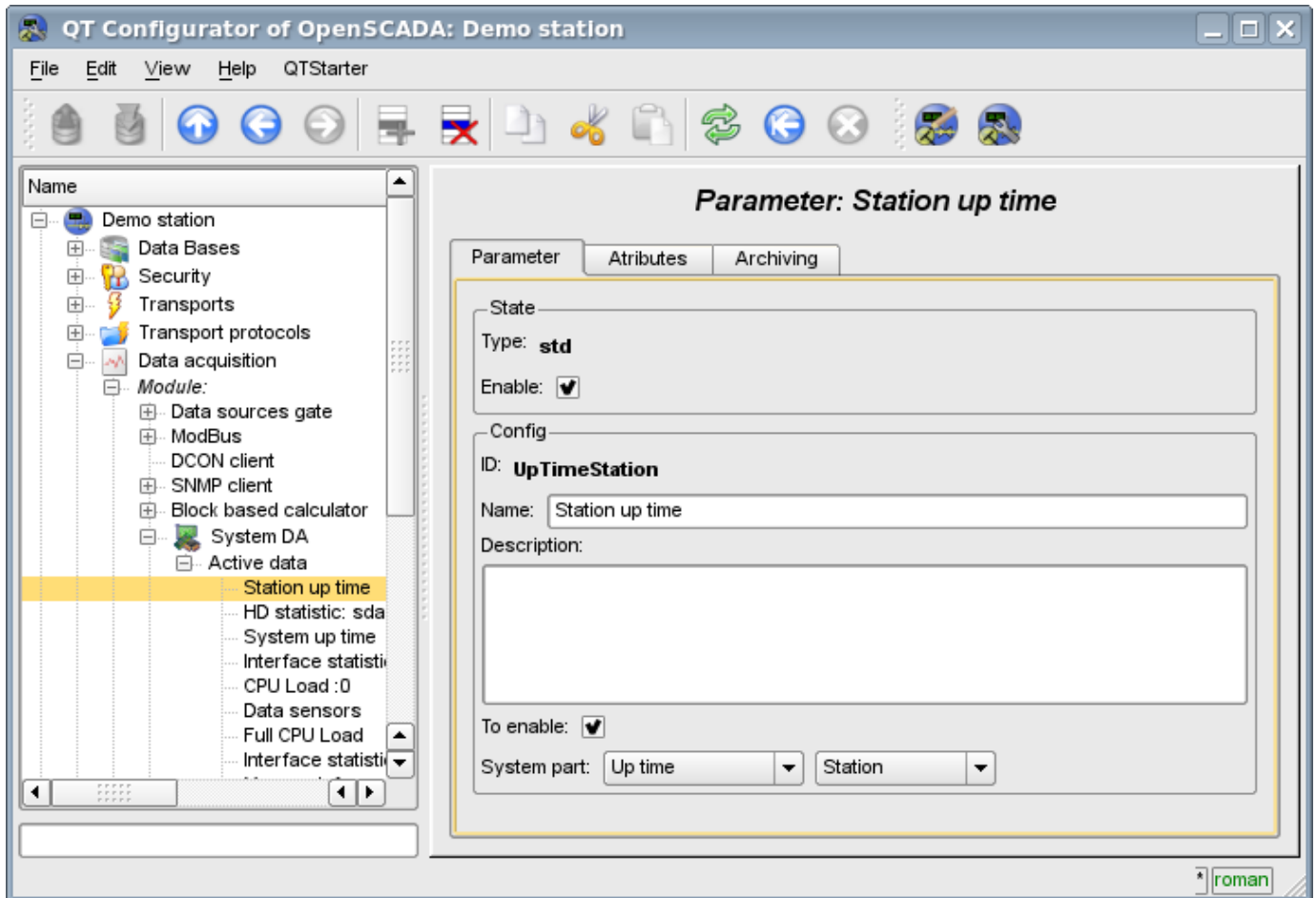


Fig.2. Tab of configuration of the parameter.

The table below there is a list of supported data sources of the operating system, the value of the additional configuration field and attributes of the parameters.

Data source	Value of the additional configuration field	Attributes of the parameter	Demands
Processor unit (CPU)	Name/number of the process. It can be a number of processor or to be «in general» for all processors <gen>.	<ul style="list-style-type: none"> • [real] load:Load (%) • [real] sys:System (%) • [real] user:User (%) • [real] idle:Idle (%) 	
Memory (MEM)	Not used	<ul style="list-style-type: none"> • [dec] free:Free (κБ); • [dec] total:Total (κБ); • [dec] use:Used (κБ); • [dec] buff:Buffers (κБ); • [dec] cache:Cache (κБ); • [dec] sw_free:Swap, free (κБ); • [dec] sw_total:Swap, total (κБ); • [dec] sw_use:Swap, used (κБ). 	

Data source	Value of the additional configuration field	Attributes of the parameter	Demands
Sensors (sensors)	Not used	Attributes are defined by sensors that are available on the motherboard. For each sensor the unique attribute is created.	The library libsensors or program mbmon is used. Higher priority in the use is given to the library libsensors, because mbmon has problems on multicore architectures.
HDD temperature (hddtemp)	HDD. Disks, available in the system.	<ul style="list-style-type: none"> • [string] disk:Name; • [string] ed:Unit of measurement; • [real] t:Temperature. 	It must be installed configured and running as a service program hddtemp
Uptime (uptime)	Uptime: <ul style="list-style-type: none"> • System; • Station. 	<ul style="list-style-type: none"> • [dec] full:Seconds full; • [dec] sec:Seconds; • [dec] min:Minutes; • [dec] hour:Hours; • [dec] day:Days. 	
HDD Smart (hddsmart)	Disk. Disks, available in the system.	Attributes are defined by SMART-fields available for this disc. For each field the unique attribute is created.	It must be installed and available smartctl utility.
HDD statistics (hddstat)	Disk or partition. Disks or partitions, available in the system.	Attributes: <ul style="list-style-type: none"> • [dec] rd:Read (Kb); • [dec] wr:Written (Kb). 	
Net statistics (netstat)	Network interface. Network interfaces, available in the system.	Attributes: <ul style="list-style-type: none"> • [dec] rcv:Received (Kb); • [dec] trns:Transferred (Kb). 	

The module of subsystem “Data acquisition” <BlockCalc>

<i>Module:</i>	BlockCalc
<i>Name:</i>	Block calculator.
<i>Type:</i>	DAQ
<i>Source:</i>	daq_BlockCalc.so
<i>Version:</i>	1.4.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides a block calculator.
<i>License:</i>	GPL

The module of subsystems «DAQ» BlockCalc provides the system OpenSCADA with the mechanism for creating custom calculations. The mechanism of calculations based on the formal language of block circuits(functional blocks).

Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level. In addition to the synchronization of the archives of values and archives of attributes of parameters the module implements synchronization of computational templates, in order to shockless catch of the algorithms.

Languages of block programming based on the notion of circuits(functional blocks). Moreover, depending on the substance of the block, block circuits may include: logic, relay logic circuits, a model of technological process, and more. The essence of the block circuit is that it contains a list of blocks and relations between them.

From a formal point of view a block is an element(function), which has inputs, outputs, and an algorithm for computing. Basing on the concept of programming area, block is a frame of values associated with the object of function.

Of course, the inputs and outputs of blocks may be needed to be connected for a solid block scheme. The following types of links are provided:

- Interblock, connecting the input of one block to the output of another one, the input of one block to another one's input and output of one block to the input of another one;
- Interblock remote, connection of blocks of controllers of different block circuits of the module;
- Coefficients, the transformation of input into the constant, all inputs / outputs by default are initiated as a constant;
- External attribute of the parameter.

Conditionally, connections of blocks can be represented as links between the blocks as a whole(Fig. 1) or detailing of the links(Fig. 2). In the process of binding parameters of blocks the connection of parameters of any type is acceptable. Thus, in the process of computation automatically casting of types will be done.

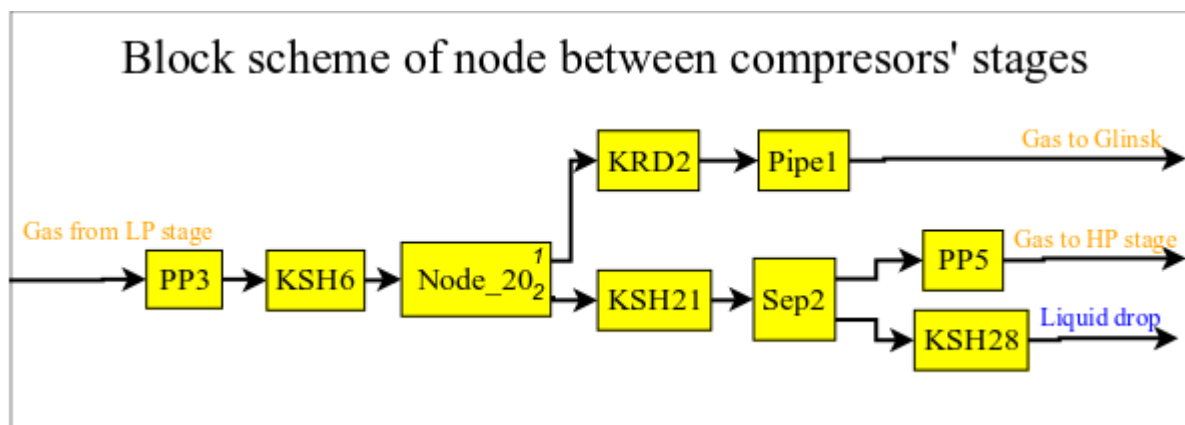


Fig. 1. The general connection between the blocks of block scheme

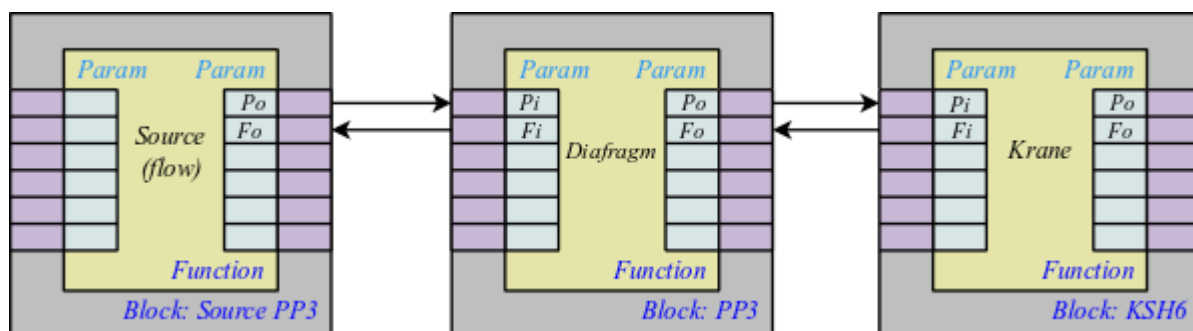


Fig. 2. Detailed links between blocks

1. The controller of the module

Each controller of this module contains the block circuit, which he computes with the specified period. In order to provide calculated data in the system OpenSCADA the parameters can be created in the controller. Example of the configuration tab of the controller of the given type depicted in Fig. 3.

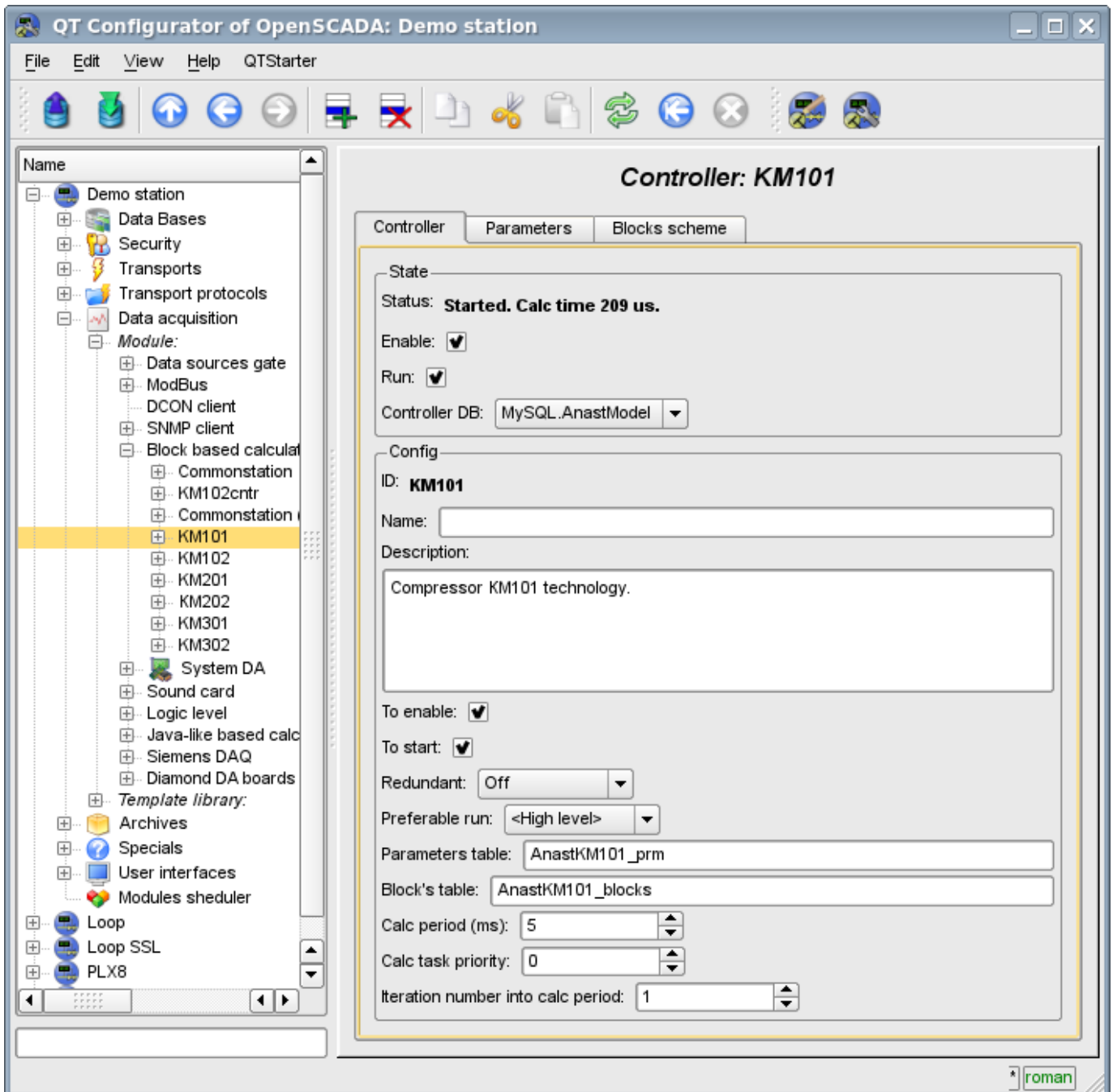


Fig. 3. Tab configuration of the controller.

From this tab you can set:

- The state controller, as follows: State, “Enabled”, “Running” and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller is to be translated at boot time: “Enabled” and “Running”.
- Horizontal mode of redundancy and performance preference of the controller.
- The names of tables to store the parameters and blocks of the controller.
- The period, priority and number of iterations in one cycle of calculating task of the block scheme of the controller.

2.The block scheme of the controller

The block scheme is formed by means of the tab controller's blocks, configuration of the block (Figure 4) and its connections (Fig. 5).

Blocks of block scheme can connect both among themselves and to the attributes of the parameters. Blocks themselves do not contain the structure of input/output(IO), but contain values, based on the IO-structure of related function. Function for linking with a block is used from the object model of the system OpenSCADA.

Any block may at any time be removed from the process and be reconfigured and then may be again included in the process. Communications between the blocks can be configured without exception blocks from the processing and stopping of the controller. All IO values without connections can be changed during processing.

Using tab of the blocks you can:

- Add/remove a block in the block scheme.
- To monitor the total number, number of enabled and the number of processing blocks.

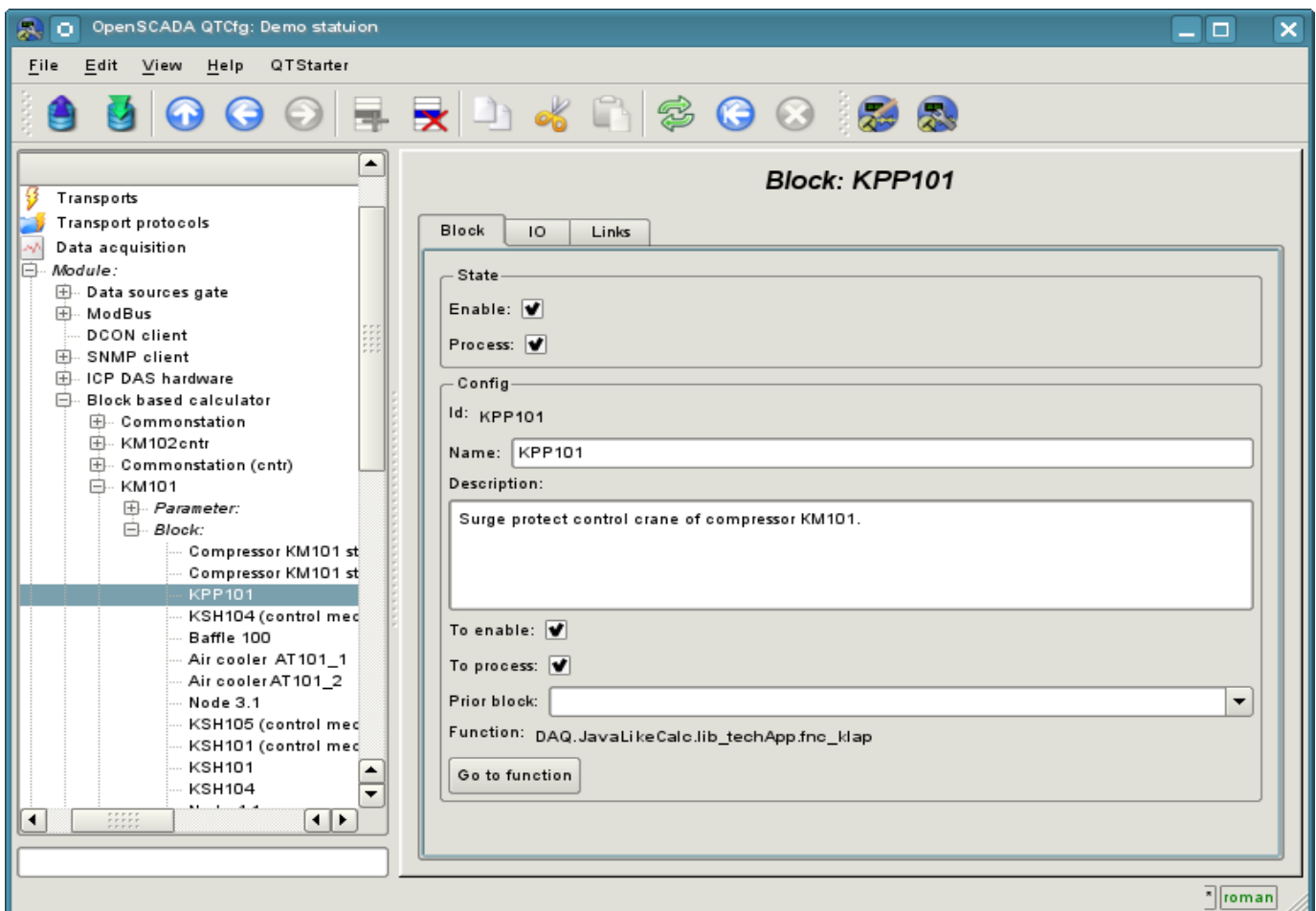


Fig. 4. Configuration tab of the block scheme.

Using the form of block configuration it can be set:

- The state of the block, as follows: “Enabled” and “Processed”.
- Id, name and description of the block.
- The state in which the block is to be translated at boot time: “Enabled” and “Running”.
- Set block which must calc before this block.
- Appoint a working function from the object model. Back to the function for familiarization.

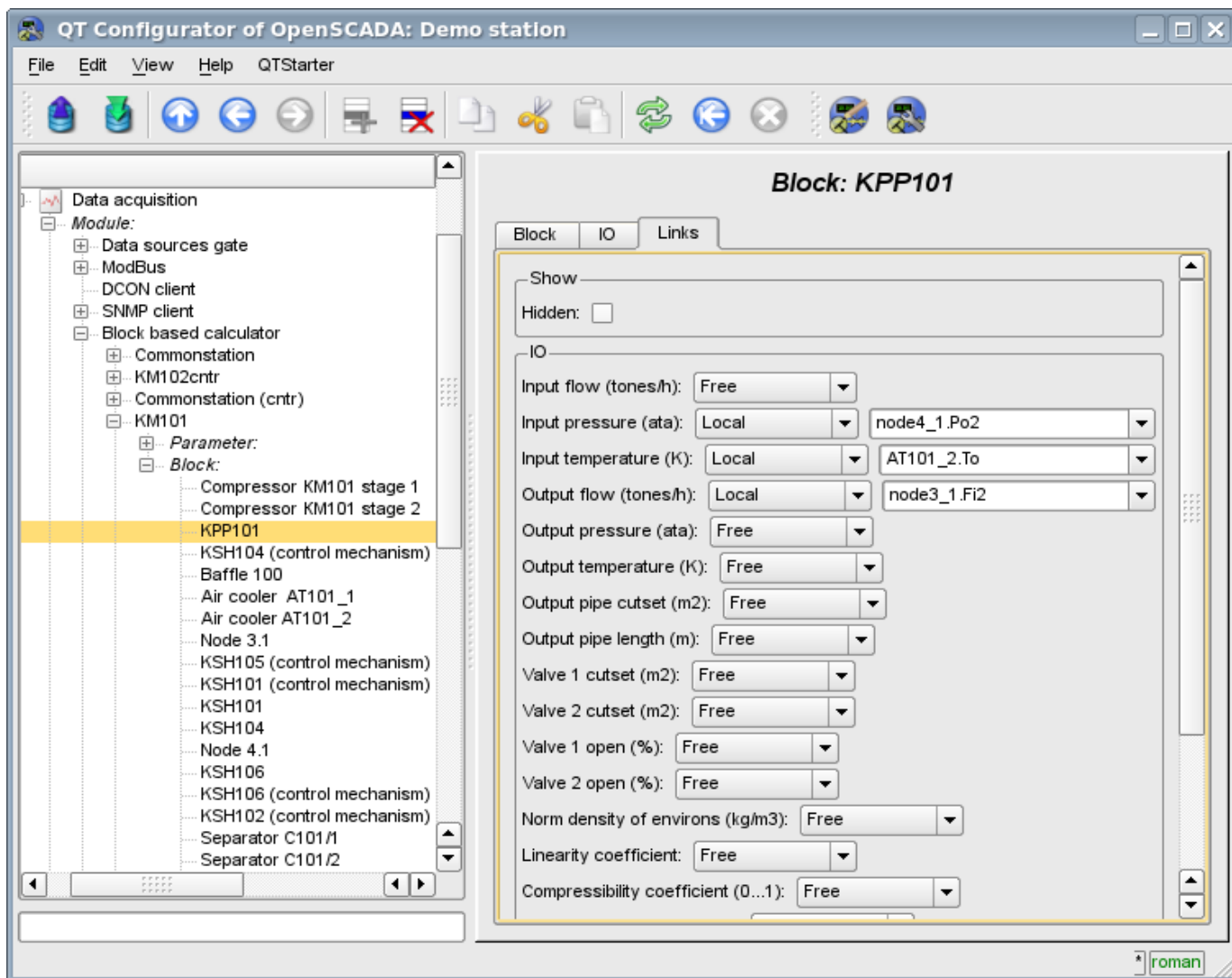


Fig. 5. Configuration tab of links of the block of the block scheme.

Using the configuration tab of links of the block of the block scheme the links can be set for the parameter of each block separately.

The following types of links are supported:

- Inter-block. Connecting the block input to the output of another block, the input of one block to another's input and output of one block to the input of another.
- Distant inter-block. The connection of blocks from various controllers of the module.
- Ratio. The transformation of the input to a constant. All inputs/outputs by default are initiated as constants.
- External attribute of the parameter.

To set values for the parameter of the block there is the corresponding tab (Fig.6).

In accordance with the custom functions in the system OpenSCADA the four main types of IO are supported: integer, float, boolean and string.

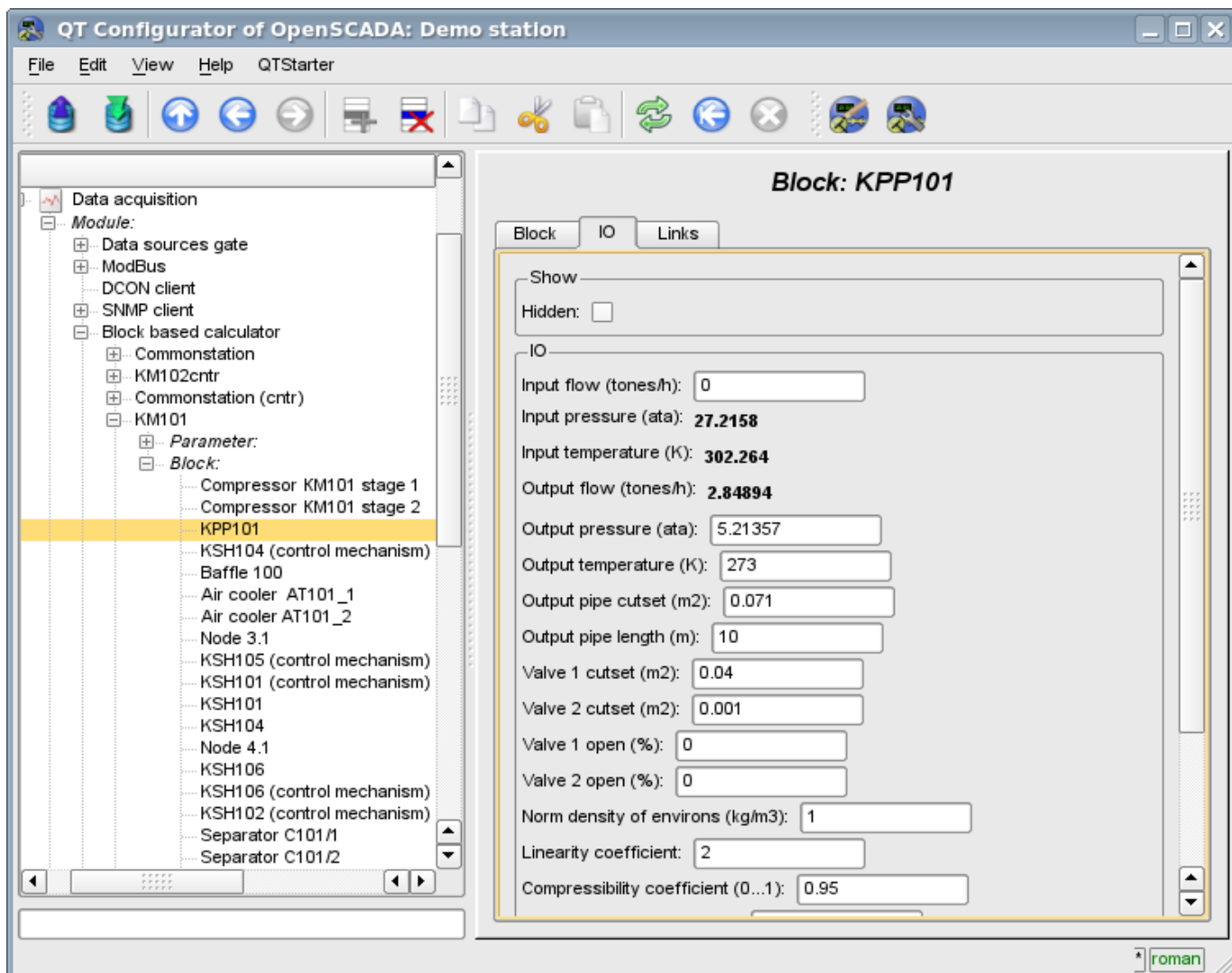


Fig. 6. Configuration tab of values of parameters of block of the block scheme.

3. Parameters of the controller

The module provides only one type of parameters – the “Standard”. The parameter used to reflect the data, calculated in the blocks, on the attributes of the controller's parameters. Example of the configuration tab of the parameter is shown in Fig.7.

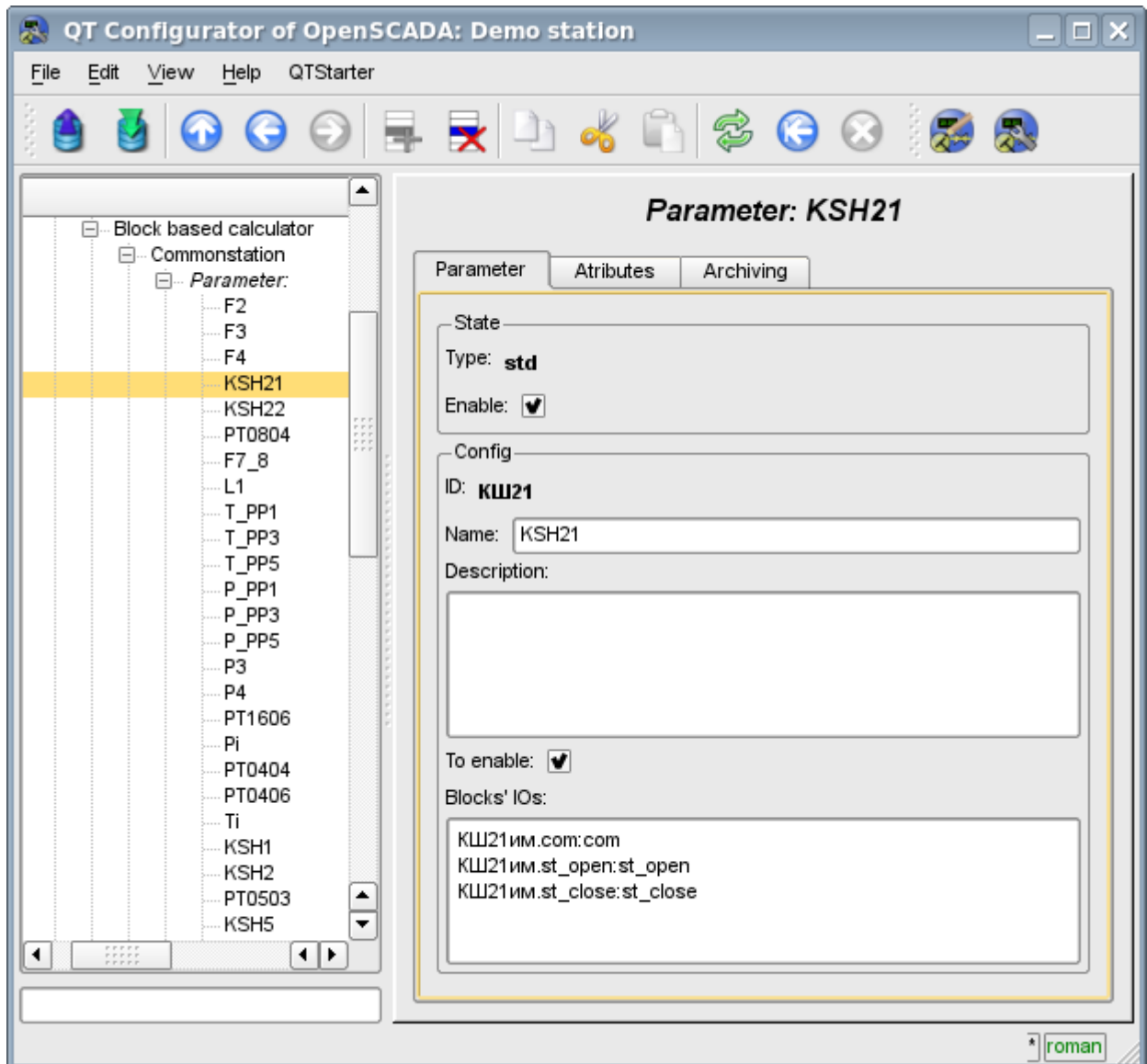


Fig. 7. Configuration tab of values of parameters of the controller.

From this tab you can set:

- The state of the parameter, as follows: “Enabled” and type of the parameter.
- Id, name and description of the parameter.
- The state in which the parameter must be translated at boot time: “Enabled”.
- The list of attributes that are reflected on the parameters of the blocks. It is created as the list of elements in the format: `<BLK>.<BLK_IO>:<AID>:<ANM>`. Where:
 - `<BLK>` - block ID, block schemes ID; for constant value set to:
 - '*s' - string type;
 - '*i' - integer type;
 - '*r' - real type;
 - '*b' - boolean type.
 - `<BLK_IO>` - parameter of the block or of the the block scheme; for constant value set to attribute value;
 - `<AID>` — attribute of the parameter ID;
 - `<ANM>` — name of the attribute of parameter.

4. Copying of the block schemes

To simplify and expedite the development of complex and repetitive block schemes the mechanism of copying of the elements of block scheme both individually and block schemes entirely is provided. The mechanism of copying is integrated into the kernel of OpenSCADA and operates transparently.

The module of subsystem “Data acquisition” <JavaLikeCalc>

<i>Module:</i>	JavaLikeCalc
<i>Name:</i>	Calculator based on Java-like language.
<i>Type:</i>	DAQ
<i>Source:</i>	daq_JavaLikeCalc.so
<i>Version:</i>	1.8.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides based on java like language calculator and engine of libraries. The user can create and modify functions and libraries.
<i>License:</i>	GPL

The module of controller *JavaLikeCalc* provides a mechanism for creating of functions and libraries on Java-like language. Description of functions on Java-like language is reduced to the binding parameters of the function with algorithm. In addition, the module has the functions of the direct computation by creation of the computing controllers.

Direct computations are provided by the creation of controller and linking it with the function of this module. For linked function it is created the frame of values, with which the periodically calculating is carried out.

The module implements the functions of the horizontal redundancy, namely, working in conjunction with the remote station of the same level. In addition to the synchronization of the archives of values and archives of attributes of parameters the module implements synchronization of computational function, in order to shockless catch of the algorithms.

Parameters of functions can be freely created, deleted or modified. The current version of the module supports up to 65535 parameters of the function in the sum with the internal variables. View of the editor of functions is shown in Figure 1.

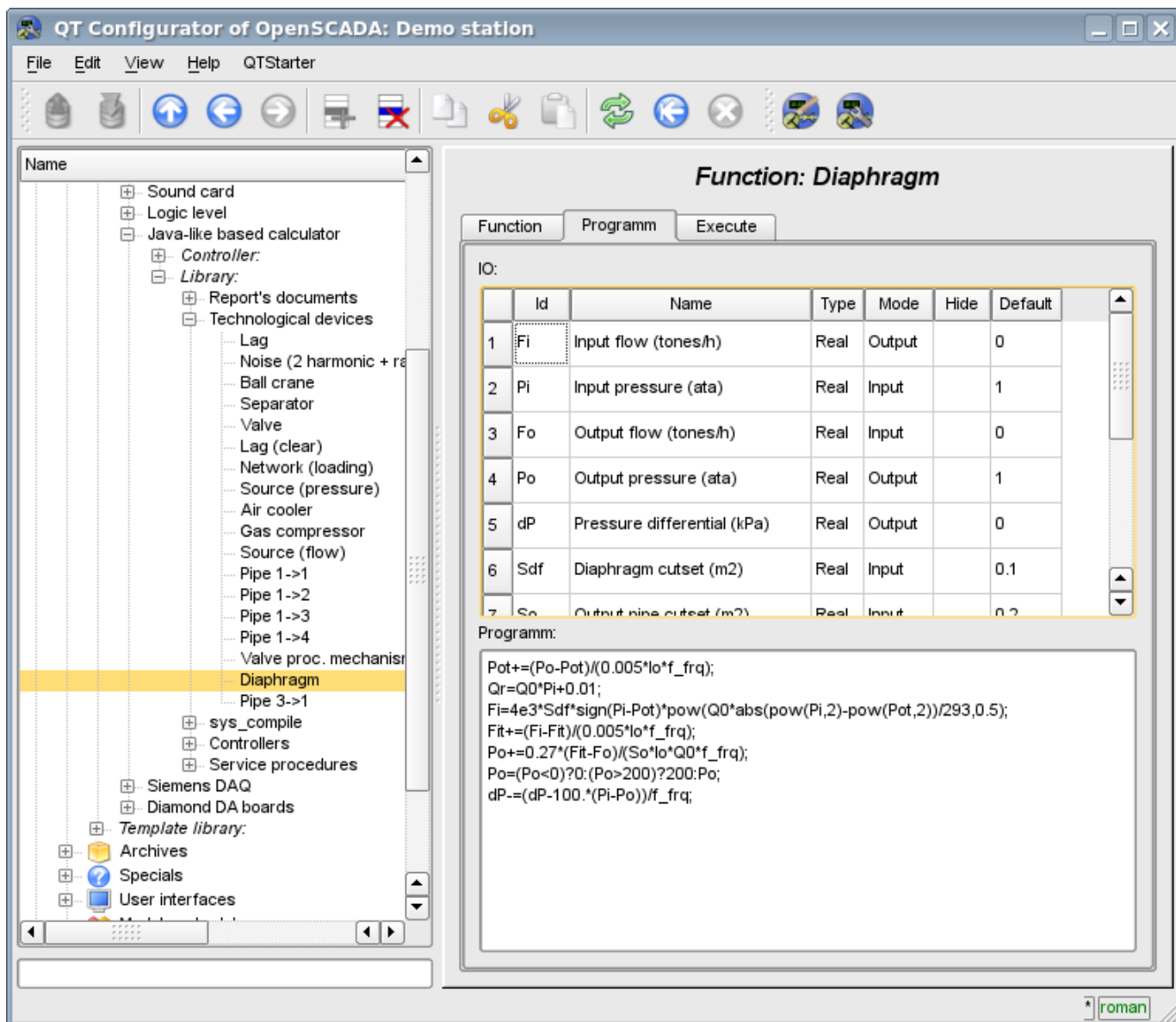


Fig.1. View of the editor of functions.

After any program changing or configuration of parameters recompiling of the programs with forestalling of linked with function objects of values of TVAlCfg is performed. Language compiler is built using well-known generator grammar «Bison», which is compatible with the not less well-known tool Yacc.

The language uses the implicit definition of local variables, which is to define a new variable in the case of assigning a value to it. This type of local variable is set according to the type of the assigning value. For example, the expression $\langle Qr=Q0*Pi+0.01; \rangle$ will define Qr variable with the type of variable Q0.

In working with various types of data language uses the mechanism of casting the types in the places where such casting is appropriate.

To comment the sections of code in the language it is provided $\langle \langle // \rangle \rangle$ and $\langle \langle / * \dots * / \rangle \rangle$ characters. Everything that comes after $\langle // \rangle$ up to the end of the line and between $\langle \langle / * \dots * / \rangle \rangle$, is ignored by the compiler.

During the code generation language compiler produces an optimization of constants and casting the types of the constants to the required type. Optimizing of the constants means the implementation of computing of the constants in the process of building of the code under the two constants and paste the result in the code. For example, the expression $\langle y=pi*10; \rangle$ reduces to a simple assignment $\langle y=31.4159; \rangle$. Casting the types of constants to the required type means formation of the constant in the code which excludes the cast in the execution process. For example, the expression $\langle y=x*10 \rangle$, in the case of a real type of the variable x, is transformed into $\langle y=x*10.0 \rangle$.

The language supports calls of the external and internal functions. Name of any function in general is perceived as a character, test for ownership of which by a particular category is done in the following order:

- keywords;
- constants;
- built-in functions;
- external functions, object's functions and OpenSCADA nodes functions (DOM) ;
- already registered characters of variables, object's attributes and hierarchy of objects DOM;
- new attributes of the system parameters;
- new function parameters;
- new automatic variable.

Call of the external function, attribute of system parameters is written as an address to the object of dynamic tree of the object model of the system OpenSCADA in the form of: `<DAQ.JavaLikeCalc.lib_techApp.klapNotLin>`.

To provide the possibility of writing custom procedures for the administration of the various components of OpenSCADA module provides the implementation of API pre-compilation of custom procedures of individual components of OpenSCADA on the implementation of Java-like language. These components are already: Templates of the parameters of subsystem “Data acquisition” and Visual control area (VCA).

1. Java-like language

1.1. Elements of language

Keywords: if, else, while, for, break, continue, return, using, true, false.

Constants:

- decimal: numerals 0–9 (12, 111, 678);
- octal: numerals 0–7 (012, 011, 076);
- hexadecimal: numerals 0–9, letters a-f or A-F (0x12, 0XAB);
- real: 345.23, 2.1e5, 3.4E-5, 3e6;
- boolean: true, false;
- string: «hello».

Types of variables:

- integer: $-2^{31} \dots 2^{31}$;
- real: $3.4 * 10^{308}$;
- boolean: false, true;
- string: length up to 255 symbols and without next string went.

Built-in constants: pi = 3.14159265, e = 2.71828182, EVAL_BOOL(2), EVAL_INT(-2147483647), EVAL_REAL(-3.3E308), EVAL_STR("<EVAL>")

Attributes of the parameters of system OpenSCADA (starting from subsystem DAQ, as follows <Type of DAQ module>.< Controller>.<Parameter>.<Attribute>).

The functions of the object model of the system OpenSCADA.

1.2. Operations of language

Operations supported by the language presented in the table below. The priority of operations is reduced from top to bottom. Operations with the same priority is composed of one color group.

Symbol	Описание
()	Call of the function.
{ }	Program blocks.
++	Increment (post and pre).
--	Decrement (post and pre).

Symbol	Описание
-	Unary minus.
!	Logical negation.
~	Bitwise negation.
*	Multiplication.
/	Division.
%	The remainder of integer division.
+	Addition
-	Subtraction
<<	Bitwise shift left
>>	Bitwise shift right
>	Greater
>=	Greater than or equal to
<	Less
<=	Less than or equal to
==	Equals
!=	Unequal
	Bitwise «OR»
&	Bitwise «AND»
^	Bitwise «Exclusive OR»
&&	Boolean «AND»
	Boolean «OR»
?:	Conditional operation (i=(i<0)?0:i;)
=	Assignment.
+=	Assignment with addition.
-=	Assignment with subtraction.
*=	Assignment with multiplication.
/=	Assignment with division.

1.3. Embedded functions of language

To ensure a high speed in mathematical calculations module provides embedded mathematical functions that are called at the level of commands of virtual machine. Predefined mathematical functions:

- $\sin(x)$ - sine x ;
- $\cos(x)$ - cosine x ;
- $\tan(x)$ - tangent x ;
- $\sinh(x)$ - hyperbolic sine of x ;
- $\cosh(x)$ - hyperbolic cosine of x ;
- $\tanh(x)$ - hyperbolic tangent of x ;
- $\text{asin}(x)$ - arcsine of x ;
- $\text{acos}(x)$ - arc cosine of x ;
- $\text{atan}(x)$ - arctangent of x ;
- $\text{rand}(x)$ - random number from 0 to x ;
- $\lg(x)$ - decimal logarithm of x ;
- $\ln(x)$ - natural logarithm of x ;
- $\exp(x)$ - exponent of x ;
- $\text{pow}(x, x1)$ - erection of x to the power $x1$;
- $\text{max}(x, x1)$ - maximum value of x and $x1$;

- `min(x,x1)` - minimum value of `x` and `x1`;
- `sqrt(x)` - the square root of `x`;
- `abs(x)` - absolute value of `x`;
- `sign(x)` - sign of `x`;
- `ceil(x)` - rounding the number `x` to a greater integer;
- `floor(x)` - rounding the number `x` to a smaller integer.

1.4. Operators of the language

The total list of operators of the language:

- *var* - operator for variable initialise;
- *if* - operator of the condition "If";
- *else* - operator of the condition "ELSE";
- *while* - description of the loop while;
- *for* - description of the loop for;
- *in* - for-cycle separator for object's properties scan;
- *break* - interrupt of the execution of the loop;
- *continue* - continue the execution of the loop from the beginning;
- *using* - allows to establish scope of functions of often used library (using `Special.FLibSYS`;) for future reference only by means of the function name;
- *return* - interruption of the function and return of the result, the result is copied to the attribute with the flag return (return 123;);
- *new* - object creation, realized object "Object" and massif "Array".

1.4.1. Conditional operators

The language of module supports two types of conditions. First – this is the operation of condition for use within the expression, the second – a global, based on the conditional operators.

Conditions inside the expression is based on the operations of «?» And «:». As an example we'll write the following practical expression `<st_open=(pos>=100)?true:false;>`, which reads as «If the variable `<pos>` greater than or equal to 100, the variable `st_open` is set to true, otherwise – to false.

The global condition is based on the conditional operators «if» and «else». An example is the same expression, but written by other means `<if(pos>100) st_open=true; else st_open=false;>`. As shown, the expression is written in a different way, but is read in the same way.

1.4.2. Loops

Two types of loops are supported: while, for and for-in. The syntax of the loops corresponds to programming languages: C++, Java, and JavaScript.

Loop **while** generally written as follows: `while(<condition>) <body of the loop>;`

Loop **for** is written as follows: `for(<pre-initialization>;<condition>;<post-calculation>) <body of the loop>;`

Loop **for-in** is written as follows: `for(<variable> in <object>) <body of the loop>;`

Where:

- `<condition>` - expression, determining the condition;
- `<body of the loop>` - the body of the loop of multiple execution;
- `<pre-initialization>` - expression of pre-initialization of variable of the loop;
- `<post-calculation>` - expression of modification of parameters of the loop after the next iteration;
- `<variable>` - variable, which will contain object's properties name at scan;
- `<object>` - object for which properties scan gone.

1.4.3. Special characters of string variables

The language supports the following special characters of string variables:

- "\n" - line feed;
- "\t" - tabulation symbol;
- "\b" - culling;
- "\f" - page feed;
- "\r" - carriage return;
- "\" - the character itself \".

1.5. Object

The language provides the data type "Object" support. The data type "Object" is associated container of properties and functions. The properties can support data of fourth basic types and other objects. The access to properties is doing through the dot to object <obj.prop> and also by property placement into the rectangle brackets <obj["prop"]>. It is obvious that the first mechanism is static, while the second lets you to specify the name of the property through a variable. Creating an object is carried by the keyword <new>: <varO = new Object()>. The basic definition of the object does not contain functions. Copying of an object is actually makes the reference to the original object. When you delete an object is carried out the reducing of the reference count, and when a reference count is set to zero then object is removed physically.

Different components can define basic object with special properties and functions. The standard extension of the object is an array "Array", which is created by the command <varO = new Array(prm1,prm2,prm3,...,prmN)>. Comma-separated parameters are placed in the array in the original order. If the parameter is the only one the array is initiated by the specified number of empty elements. Peculiarity of the array is that it works with the properties as the indexes and their complete naming is meaningless, and therefore the mechanism of addressing only by the placing the index into square brackets <arr[1]> is accessible. Array stores the properties in its own container of the one-dimensional array.

The array provides a special property of "length" to get the size of the array <var = arr.length>. Also, the array provides the following special functions:

- *string join(string sep = ","), string toString(string sep = ","), string valueOf(string sep = ",")* - Returns a string array elements separated by <sep> or symbol ' '.
- *Array concat(Array arr);* - Adds to the original array of array elements <arr>. Returns the original array with the changes.
- *int push(ElTp var, ...);* - Puts the item(s) <var> to the end of the array as a stack. Returns a new array size.
- *ElTp pop();* - Removing the last element of the array and returns its value as from the stack.
- *Array reverse();* - Changing the order of the elements of the array. Returns the original array with the changes.
- *ElTp shift();* - The shift of the array in the top. This first element is removed and its value is returned.
- *int unshift(ElTp var, ...);* - Unshift the item(s) <var> in the array. The first element to the 0, the second one to the 1, etc.
- *Array slice(int beg, int end);* - Returns an array of fragment <beg> to <end>. If the beginning or end is negative, then the count is done from the end of the array. If the end is not specified, then the end is the end of the array.
- *Array splice(int beg, int remN, ElTp val1, ElTp val2, ...);* - Insert, delete or replace the elements of array. Returns the original array with the changes. First of all the removing of the items from the position <beg> and number <remN> is done, and then the values <val1> etc. are inserted from the position <beg>.
- *Array sort();* - Sort the array elements in lexicographical order.

The basic types have the partial properties of the object. Properties and functions of the basic types are listed below:

- Logical type, functions:
 - *bool isEval();* - Check value to "EVAL".
 - *string toString();* - Performs the value as the string "true" or "false".

- Integer and real number:

Properties:

- *MAX_VALUE* - maximum value;
- *MIN_VALUE* - minimum value;
- *NaN* - error value.

Functions:

- *bool isEval()*; - Check value to "EVAL".
- *string toExponential(int numbs)*; - Return the string of the number, formatted in exponential notation, and with the number of significant digits <numbs>. If <numbs> is missing the number of digits will have as much as needed.
- *string toFixed(int numbs)*; - Return the string of the number, formatted in the notation of fixed-point, and with the number of significant digits after the decimal point <numbs>. If <numbs> is missing the number of digits after the decimal point is equal to zero.
- *string toPrecision(int prec)*; - Return the string of the formatted number with the number of significant digits <prec>.
- *string toString(int base)*; - Return the string of the formatted number of integer type with the following representation base: octal , decimal, hex.

- String:

Properties:

- *int length* - string length.

Functions:

- *bool isEval()*; - Check value to "EVAL".
- *string charAt(int symb)*; - Extracts from the string the symbol <symb>.
- *int charCodeAt(int symb)*; - Extracts from the string the symbol code <symb>.
- *string concat(string val1, string val2, ...)*; - Returns a new string formed by joining the values <val1> etc. to the original one.
- *int indexOf(string substr, int start)*; - Returns the position of the required string <substr> in the original row from the position <start>. If the initial position is not specified then the search starts from the beginning. If the search string is not found then -1 is returned.
- *int lastIndexOf(string substr, int start)*; - Returns the position of the search string <substr> in the original one beginning from the position of <start> when searching from the end. If the initial position is not specified then the search begins from the end. If the search string is not found then -1 is returned.
- *string slice(int beg, int end)*; *string substring(int beg, int end)*; - Return the string extracted from the original one starting from the <beg> position and ending be the <end>. If the beginning or end is negative, then the count is conducted from the end of the line. If the end is not specified, then the end is the end of the line.
- *Array split(string sep, int limit)*; - Return the array of strings separated by <sep> with the limit of the number of elements <limit>.
- *string insert(int pos, string substr)*; - Insert substring <substr> into this string's position <pos>.
- *string replace(int pos, int n, string substr)*; - Replace substring into position <pos> and length <n> to string <substr>.
- *real toReal()*; - Convert this string to real number.
- *int toInt(int base = 0)*; - Convert this string to integer number in accordance with the base <base> (from 2 to 36). If base is 0, then the prefix will be considered a record for determining the base (123-decimal; 0123-octal; 0x123-hex).
- *string parse(int pos, string sep = ".", int off = 0)*; - Get token with numbet <pos> from the string when separated by <sep> and from offset <off>. Result offset is returned to back <off>.
- *string parsePath(int pos, int off = 0)*; - Get path token with numbet <pos> from the string and from offset <off>. Result offset is returned to back <off>.
- *string path2sep(string sep = ".")*; - Convert path into this string to separated by <sep> string.

For access to system objects (nodes) of the OpenSCADA the corresponding object is provided which is

created simply by specifying the enter point "SYS" of the root object OpenSCADA, and then with the point separator the sub-objects in accordance with the hierarchy are specified. For example, the call of the request function over the outgoing transport is carried out as follows:
SYS.Transport.Sockets.out_testModBus.messIO(strEnc2Bin("15 01 00 00 00 06 01 03 00 00 00 05"));

1.6. Examples of programs on the language

Here are some examples of programs on Java-like language:

```
//Model of the course of the executive machinery of ball valve
if( !(st_close && !com) && !(st_open && com) )
{
    tmp_up=(pos>0&&pos<100)?0:(tmp_up>0&&lst_com==com)?tmp_up-1./frq:t_up;
    pos+=(tmp_up>0)?0:(100.*(com?1.: -1.))/(t_full*frq);
    pos=(pos>100)?100:(pos<0)?0:pos;
    st_open=(pos>=100)?true:false;
    st_close=(pos<=0)?true:false; lst_com=com;
}
//Valve model
Qr=Q0+Q0*Kpr*(Pi-1)+0.01;
Sr=(S_kl1*l_kl1+S_kl2*l_kl2)/100.;
Ftmp=(Pi>2.*Po)?Pi*pow(Q0*0.75/Ti,0.5):
    (Po>2.*Pi)?Po*pow(Q0*0.75/To,0.5):
    pow(abs(Q0*(pow(Pi,2)-pow(Po,2))/Ti),0.5);
Fi-=(Fi-7260.*Sr*sign(Pi-Po)*Ftmp)/(0.01*lo*frq);
Po+=0.27*(Fi-Fo)/(So*lo*Q0*frq);
Po=(Po<0)?0:(Po>100)?100:Po;
To+=(abs(Fi)*(Ti*pow(Po/Pi,0.02)-To)+
    (Fwind+1)*(Twind-To)/Riz)/(Ct*So*lo*Qr*frq);
```


2. Controller and its configuration

The controller of the module connects with the functions of libraries, built with his help, to provide immediate calculations. In order to provide calculated data in the system OpenSCADA parameters can be created in the controller. Example of the configuration tab of the controller of the given type depicted in Figure 2.

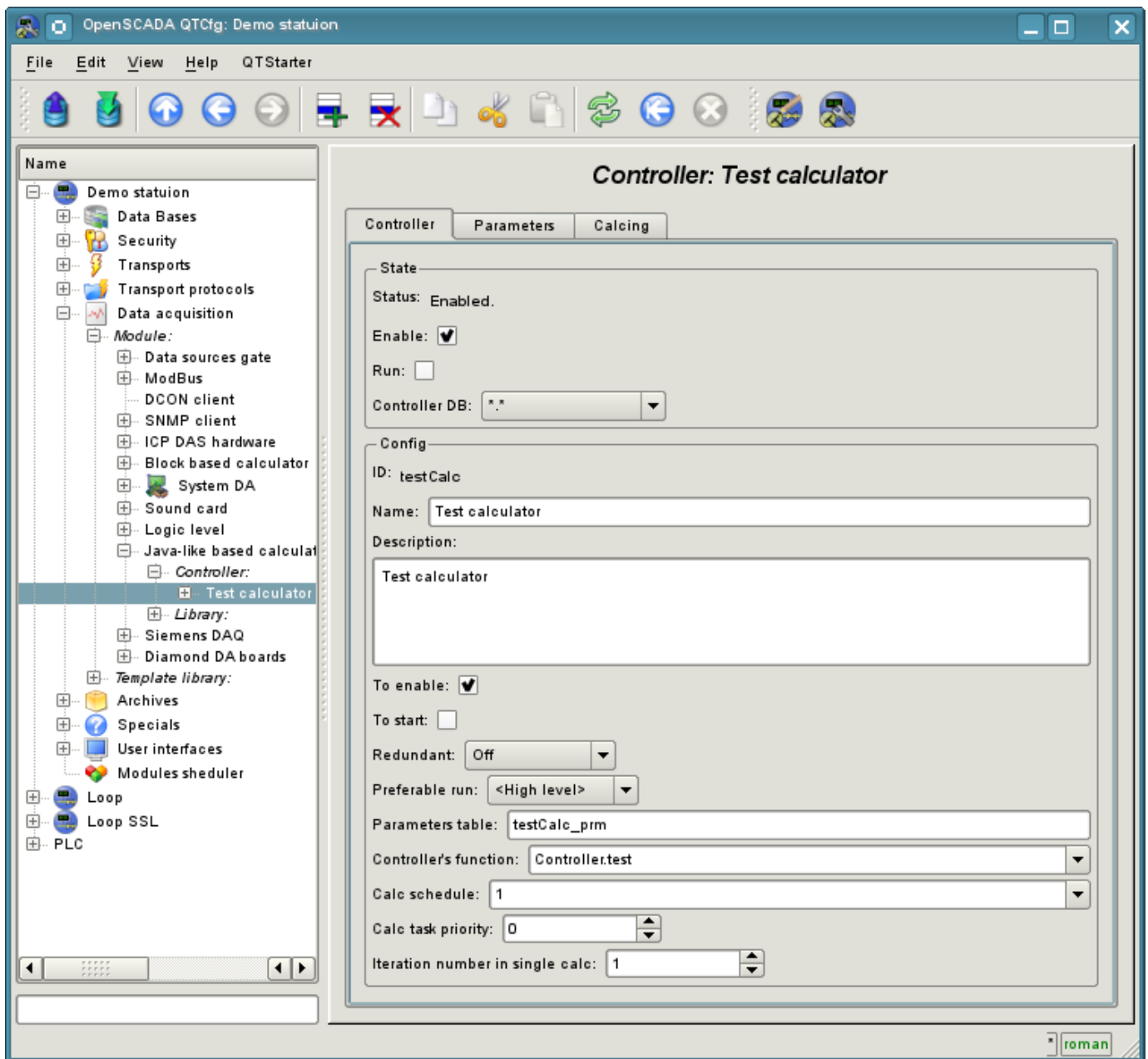


Fig.2. Configuration tab of the controller.

From this tab you can set:

- The state of the controller, as follows: Status, «Enabled», «Running» and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «Enabled» and «Running».
- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the settings.
- Address of the computational function.
- Period, priority and number of iterations in one cycle of computing task.
- Automatic synchronization period of blocks with the database.
- Save/load controller to/from the database.

Tab “Calculations” of the controller (Fig. 3) contains the parameters and the text of the program, directly performed by the controller. Also for monitoring of execution the time of calculating of the program is shown.

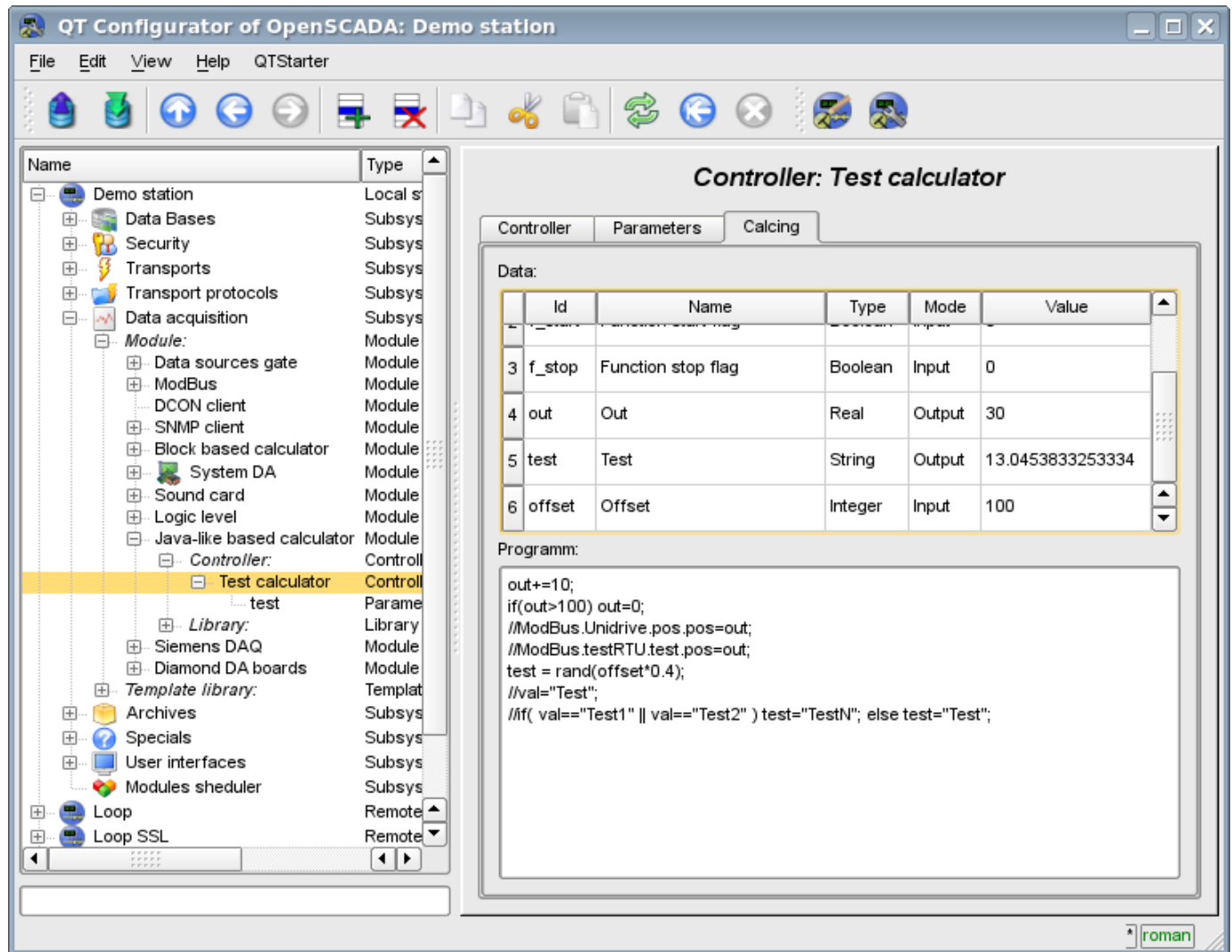


Fig.3. Tab “Calculations” of the controller.

3. The parameter of the controller and its configuration

Parameter of the controller of the module executes the function of providing the access to the results of computation of the controller to the system OpenSCADA by attributes if the parameters. Configuration tab contains only one specific field of the, set the controller only contains a field of listing the parameters of calculated function, which should be reflected.

4. Libraries of functions of module

The module provides a mechanism to create libraries of user functions on Java-like language. Example of the configuration tab of the library is depicted in Figure 4. The tab contains the basic fields: status, identifier, name and description, and also address of the table, in which the library is kept. In the “Functions” tab of the library besides the list of functions the form of copying functions is contained.

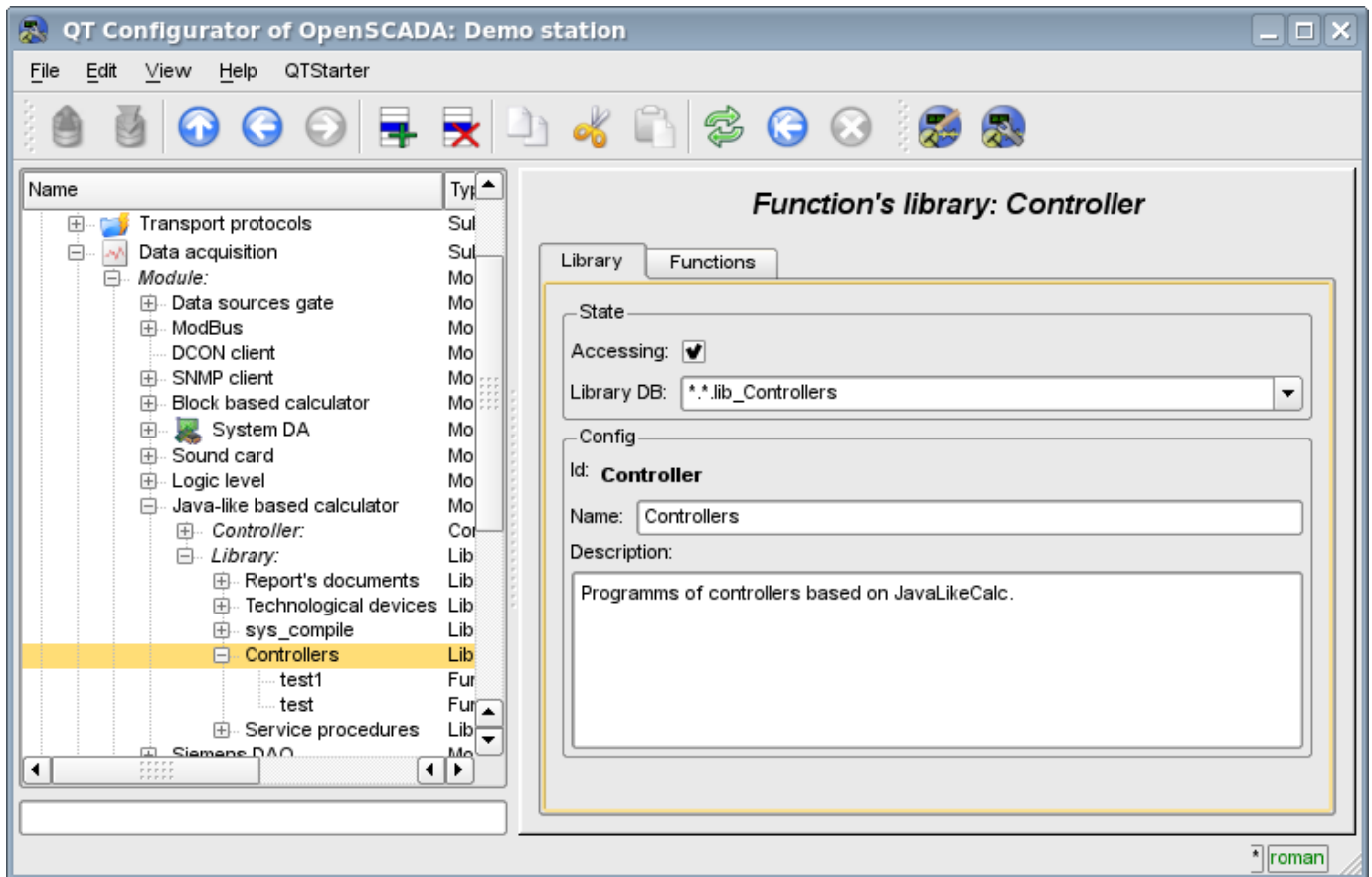


Fig.4. Tab of the configuration of the library.

5. User functions of the module

Function, as well as the library, contains the basic configuration tab, tab of the formation of the program and the parameters of function (Fig. 1), as well as the performance tab of the created function.

The module of subsystem “Data acquisition” <LogicLev>

<i>Module:</i>	LogicLev
<i>Name:</i>	Logic level
<i>Type:</i>	DAQ
<i>Source:</i>	daq_LogicLev.so
<i>Version:</i>	1.1.2
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the logical level of parameters.
<i>License:</i>	GPL

The module is a pure logic-level implementation mechanism, based on the templates of parameters of the subsystem “Data acquisition – DAQ”. The implementation of the module is based on the “Logical level of the parameters of the system OpenSCADA” <http://diyaorg.dp.ua/oscadawiki/Doc/LogParmUrov>. Practically, this module is an implementation of the subsystem “Options” of the project without templates and putting it into the module.

The module provides a mechanism for the formation of the parameters of subsystem “DAQ”, based on other sources of the subsystem at the level of the user. Actually, the module uses templates of subsystem “DAQ” and the specific format for the description of references to the attributes of the parameters of subsystem “DAQ”.

Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level. In addition to the synchronization of the archives of values and archives of attributes of parameters the module implements synchronization of computational templates, in order to shockless catch of the algorithms.

1. Data controller

For addition of the data source of parameters of the logical level the controller in the system OpenSCADA is created and configured. Example of the configuration tab of the controller of the type is depicted in Figure 1.

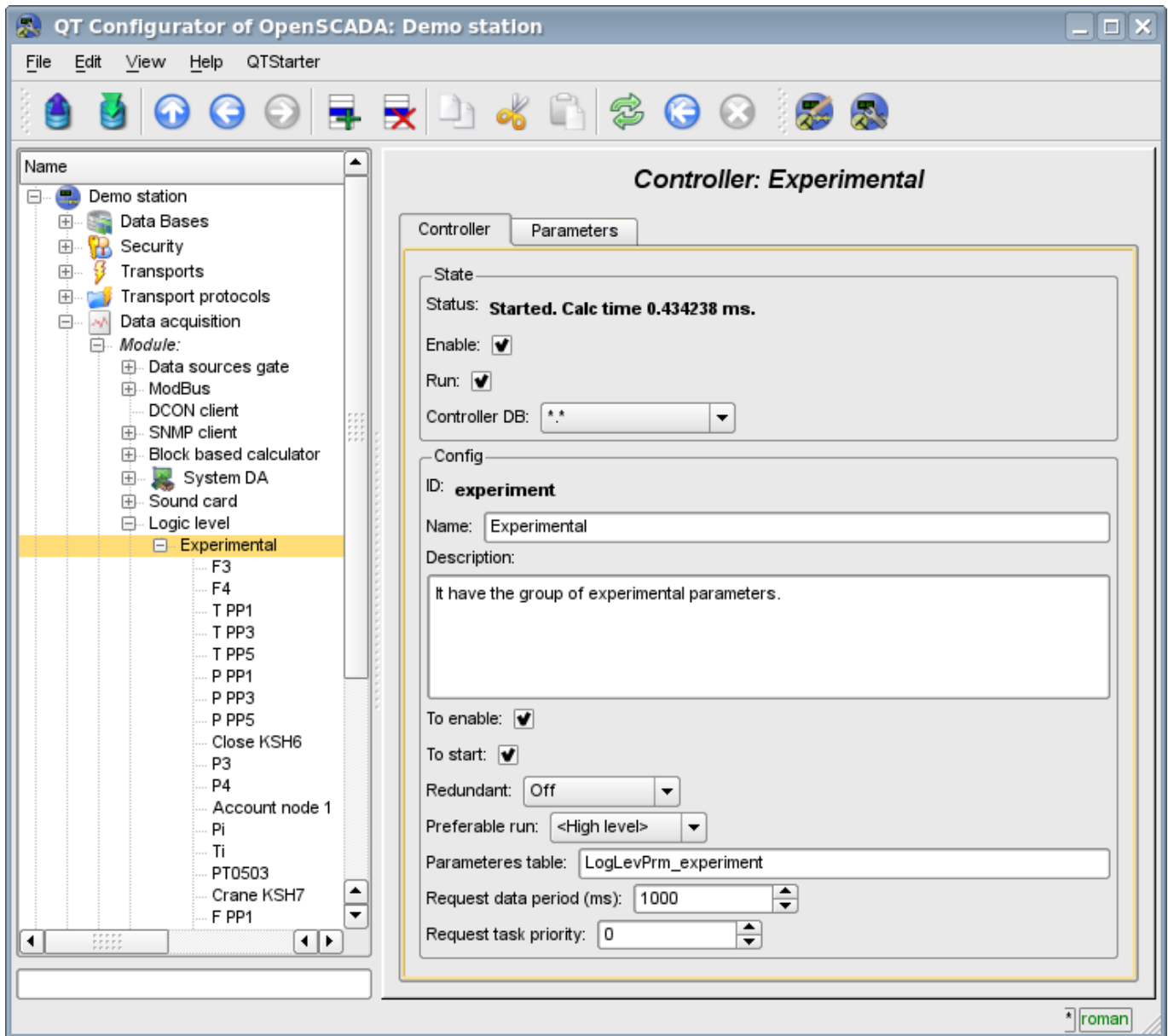


Fig.1. Configuration tab of the controller.

From this tab you can set:

- The state of the controller, as follows: Status, «Enable», «Run» and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».
- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the settings.
- The period and the priority of the task of the interrogation of data sources.

2. Parameters

Module *LogicLev* provides only one type of parameters – “Standard”. Additional configuration fields of the parameters of the module (Fig. 2) are:

- mode of the parameter;
- address; in the event of the template – this is the address of the template, and in the case of direct reflection – this is the address of the parameter.

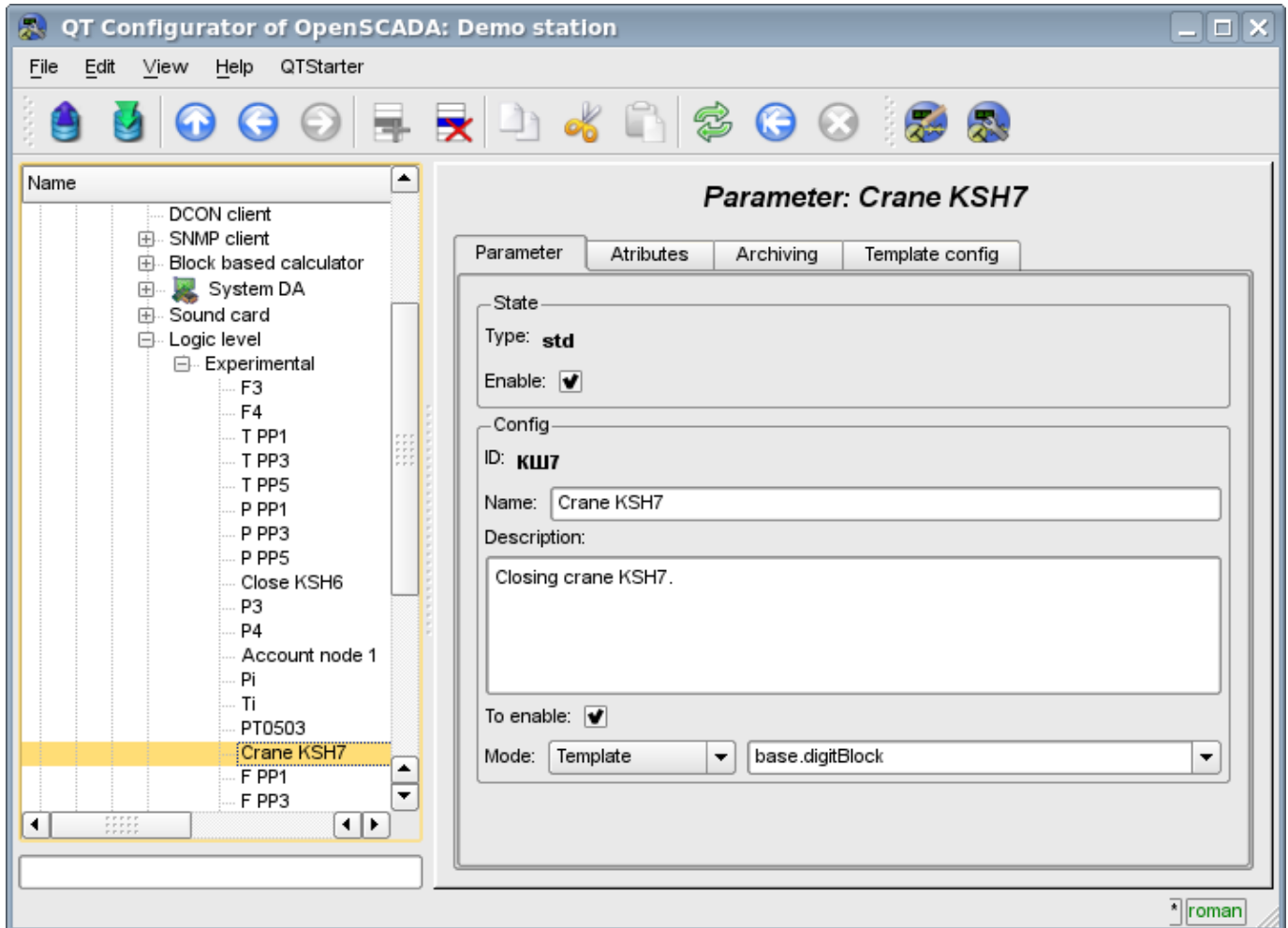


Fig.2. Configuration tab of the parameter.

When building a template for the controller the peculiarity of the link format of the template must take into account. Reference should be written in the form: **<Parameter>|<identifier>**, where:

- <Parameter>** - line, characterizing the parameter;
- <Identifier>** - id of the attribute of parameter.

This record allows to group multiple attributes of a source parameter and assign them only by the choice of the parameter. In the configuration dialog of the template (Fig. 3) it will be shown only parameter. This does not preclude the possibility to assign the attributes of the parameter each separately, in addition, if you miss in the configuration of the template the description of the links in the specified format, it will be assigned an attribute of the parameter (Fig.4).

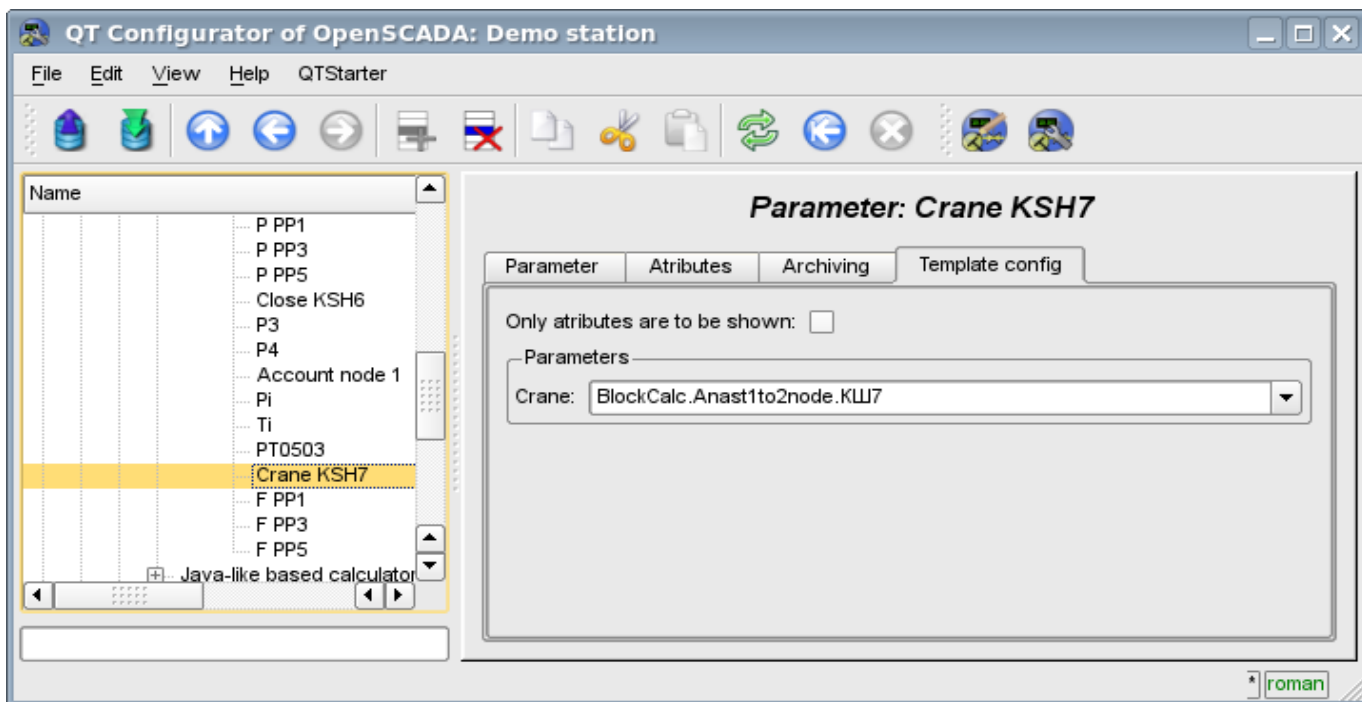


Fig.3. Configuration tab of the template of parameter.

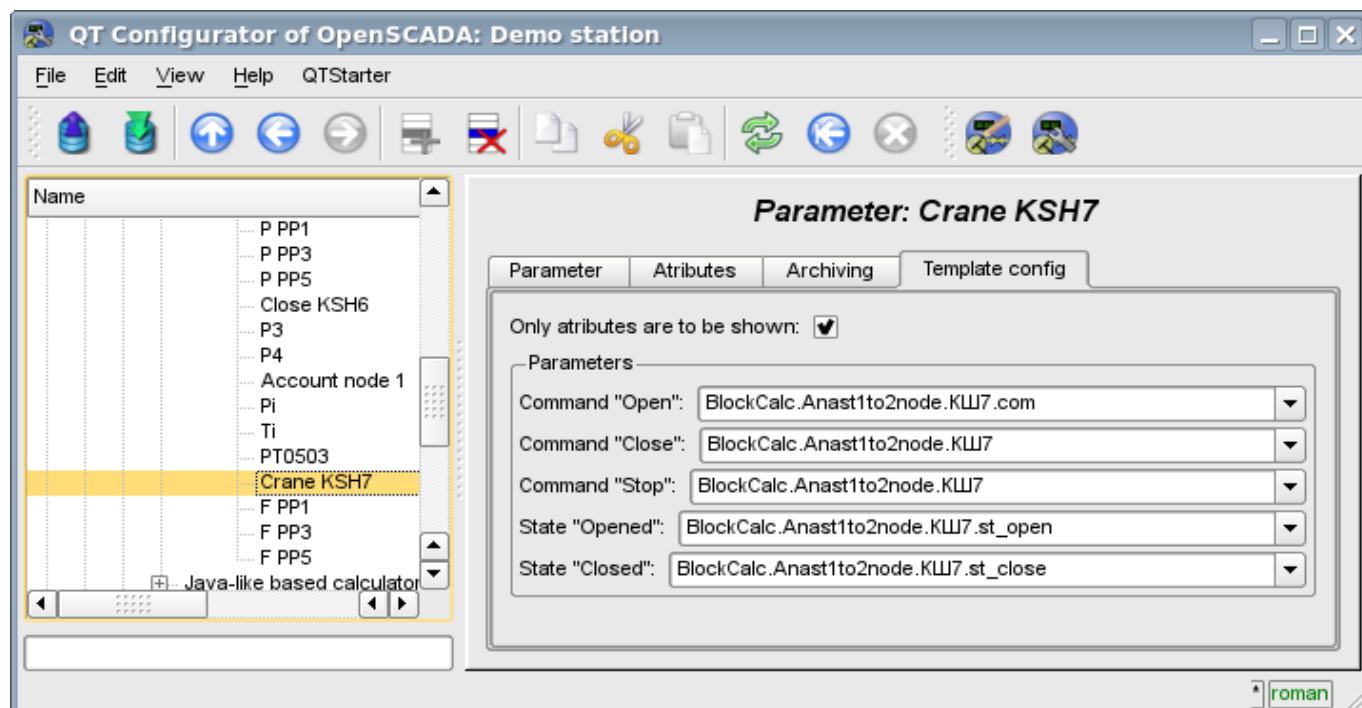


Fig.4. Configuration tab of the template of parameter. Show only attributes.

In accordance with the template underlying the parameter, we get the set of attributes of the parameter Fig.5.

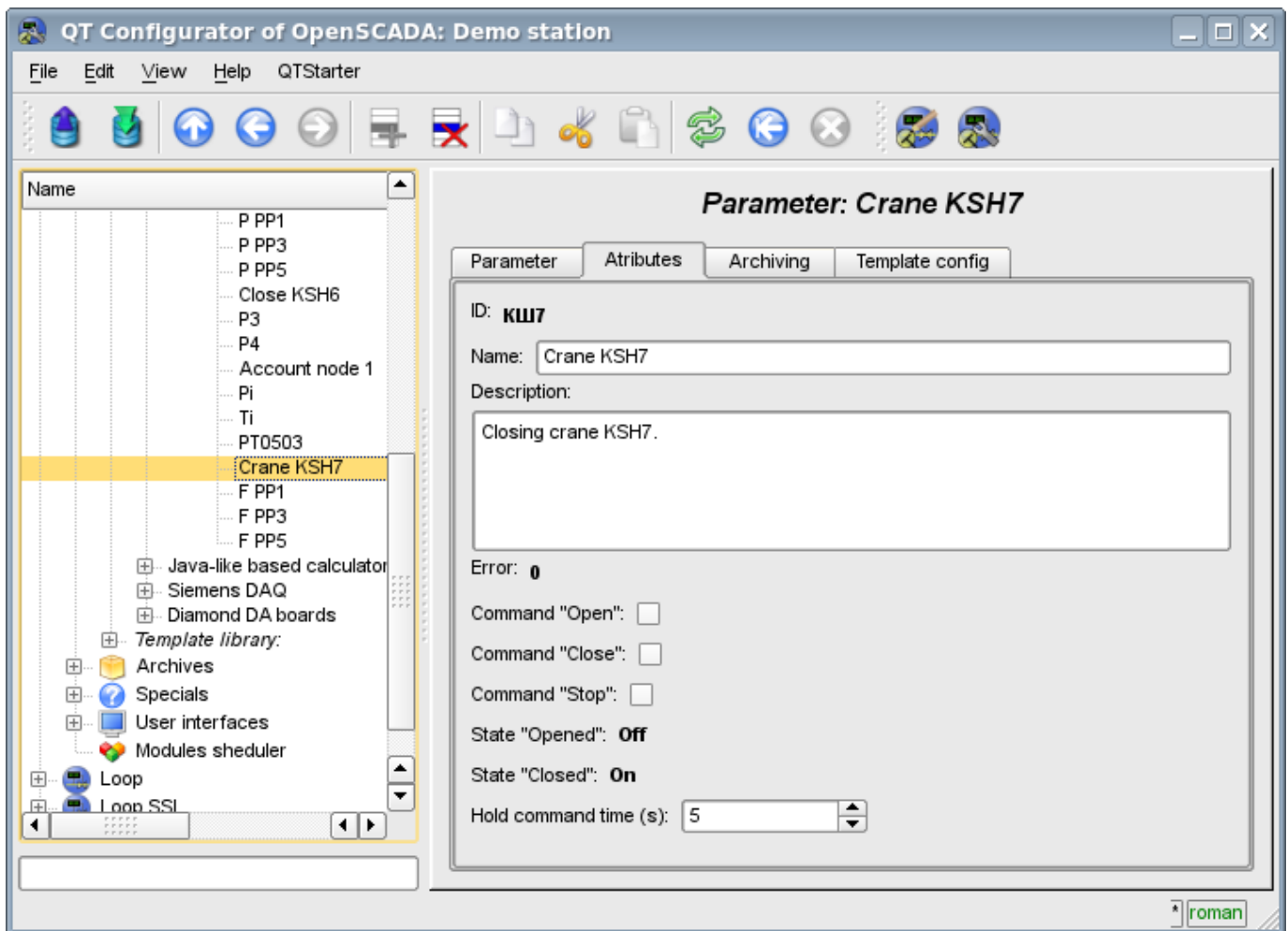


Fig.5. Tab of the attributes of the parameter.

The module of subsystem “Data acquisition” <SNMP>

<i>Module:</i>	SNMP
<i>Name:</i>	SNMP client
<i>Type:</i>	DAQ
<i>Source:</i>	daq_SNMP.so
<i>Version:</i>	0.4.1
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides an implementation of the client of SNMP-service.
<i>License:</i>	GPL

SNMP protocol was designed to verify the operation of network routers and bridges in 1988. Subsequently, the scope of the protocol coverage and other network devices such as hubs, gateways, terminal servers, and even devices that are remotely related to the network: printer, uninterruptible power supplies, PLC, etc. In addition, the protocol allows the possibility of changes in the functioning of these devices. At this time, SNMP protocol is standardized as RFC-1157, -1215, -1187, -1089.

This module provides the ability to gather information from various devices on the SNMP protocol. Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level.

1. SNMP

The main interacting “individuals” of the protocols are the agents and management systems. If we consider these two concepts in the language of «client – server», then the server role is played by agents, that is the same devices for the survey of the state of which the protocol has been developed. Accordingly, the role of the clients is played by the management systems – network applications which are necessary to gather the information about the functioning of agents. In addition to these two entities in the model of the protocol it can be identified as two more: control information and the protocol for data exchange.

All information about the objects of system-agent is contained in the so-called MIB (management information base) – the base of control information, in other words, MIB is the totality of objects (MIB-variables) accessible to the reading-writing operations.

For this time there are four base of MIB:

1. Internet MIB – database of objects for providing the diagnosis of errors and configurations. It includes 171 objects (including objects of MIB I).
2. LAN manager MIB – database of 90 objects – passwords, sessions, users, shared resources.
3. WINS MIB – database of objects required for the operation of a WINS server.
4. DHCP MIB – base of objects required for the operation of the DHCP server that serves for dynamic allocation of IP addresses on the network.

1.1. MIB

All names of MIB have a hierarchical structure. There are ten root aliases:

1. System – the group of MIB II contains the seven objects, each of which serves to store information about the system (OS version, time, etc.).
2. Interfaces – contains 23 objects necessary for the conduct of network interfaces of agents (the number of interfaces, the size of MTU, the rate of transmission, physical addresses, etc.).

3. AT (3 objects) – are responsible for the broadcast address. No longer used. Was included in the MIB I. In SNMP v2 this information was transferred to the MIB for the relevant protocols.
4. IP (42 objects) – data on the passing IP packets (number of requests, responses, offcast packages).
5. ICMP (26 objects) – information about control messages (incoming/outgoing messages, errors, etc.).
6. TCP (19) – all that relates to the same-name transport protocol (algorithms, constants, connections, open ports, etc.).
7. UDP (6) – the same one for UDP protocol (incoming/outgoing datagram, ports, errors).
8. EGP (20) – data about the traffic Exterior Gateway Protocol (used by routers, object stores information about the received/sent/ offcast frames).
9. Transmission – is reserved for specific MIB.
10. SNMP (29) – statistics on SNMP – incoming/outgoing packets, limiting package size, errors, data on the process request, and much more.

1.2. Addressing

Each of the root alias appears in the form of tree growing down. For example, to the address of the administrator you can contact by the means of the way: `system.sysContact.0`, to the time of the system: `system.sysUpTime.0`, to the description of the system (version, kernel and other information about the OS): `system.sysDescr.0`. On the other hand, the same data can be specified in the point notation. So, `system.sysUpTime.0` value corresponds to `1.3.0`, because the system has an index “1” in groups of MIB II, and `sysUpTime` – 3 in the hierarchy of the group system. Zero at the end of the path indicates the scalar type of data storage. During the work symbolic names of the objects are not used, that is, if the manager asks the agent the contents of the parameter `system.sysDescr.0`, then in the query string the link to the object will be converted to “1.1.0”, and will not be handed over «as is».

In general, there are several ways to write the addresses of MIB-variable:

- 1 Direct code addressing – “1.3.6.1.2.1.1” (root alias System). With this addressing each MIB variable is coded by the identifier, and the full address is written in the form of a sequence of identifiers separated by point, from left to right. This record of the address is the main and all other ways of recording are given to it.
- 2 Full character, in accordance with the previous code – “.iso.org.dod.internet.mgmt.mib-2.system”.
- 3 Addressing from the root alias – “system.sysDescr”.
- 4 Addresses of the MIB base – “SNMPv2-MIB:: sysDescr”.

1.3. Interaction

In the SNMP client interacts with a server on a request-response principle. On its own, the agent is able to initiate only one action, called a trap interrupt. In addition, all the actions of agents are to respond to requests sent by managers.

There are 3 main versions of the protocol SNMP (v1 & v2 & v3), which are not compatible. SNMP v3 supports encryption of traffic, which, depending on implementation, uses the algorithms DES, MD5. This leads to the fact that while transfer the most critical and important data is unavailable for listening. As a transport protocol the UDP protocol is usually used in the SNMP. Although, in fact, SNMP supports a variety of other lower-level transport protocols.

1.4. Authorization

One of the key concepts of SNMP is the notion of group. Procedure of the authorization of the manager is a simple test for membership of a particular group from the list, which belongs to the agent. If the agent does not find a group of the manager in its list, their further interaction is impossible. By default, the group used: private and public.

2. Module

This module supports the work with the SNMP protocol of version 1. Currently it is only supported the reading of MIB-parameters. In addition, a list of types of MIB-parameters is restricted by the list: ASN_OCTET_STR, ASN_INTEGER and ASN_COUNTER. Support of the other types and recording are scheduled in the following versions of the module.

2.1. Controller of data

For addition of the SNMP data source the controller is created and configured in the system OpenSCADA. Example of the configuration tab of the controller is depicted in Figure 1.

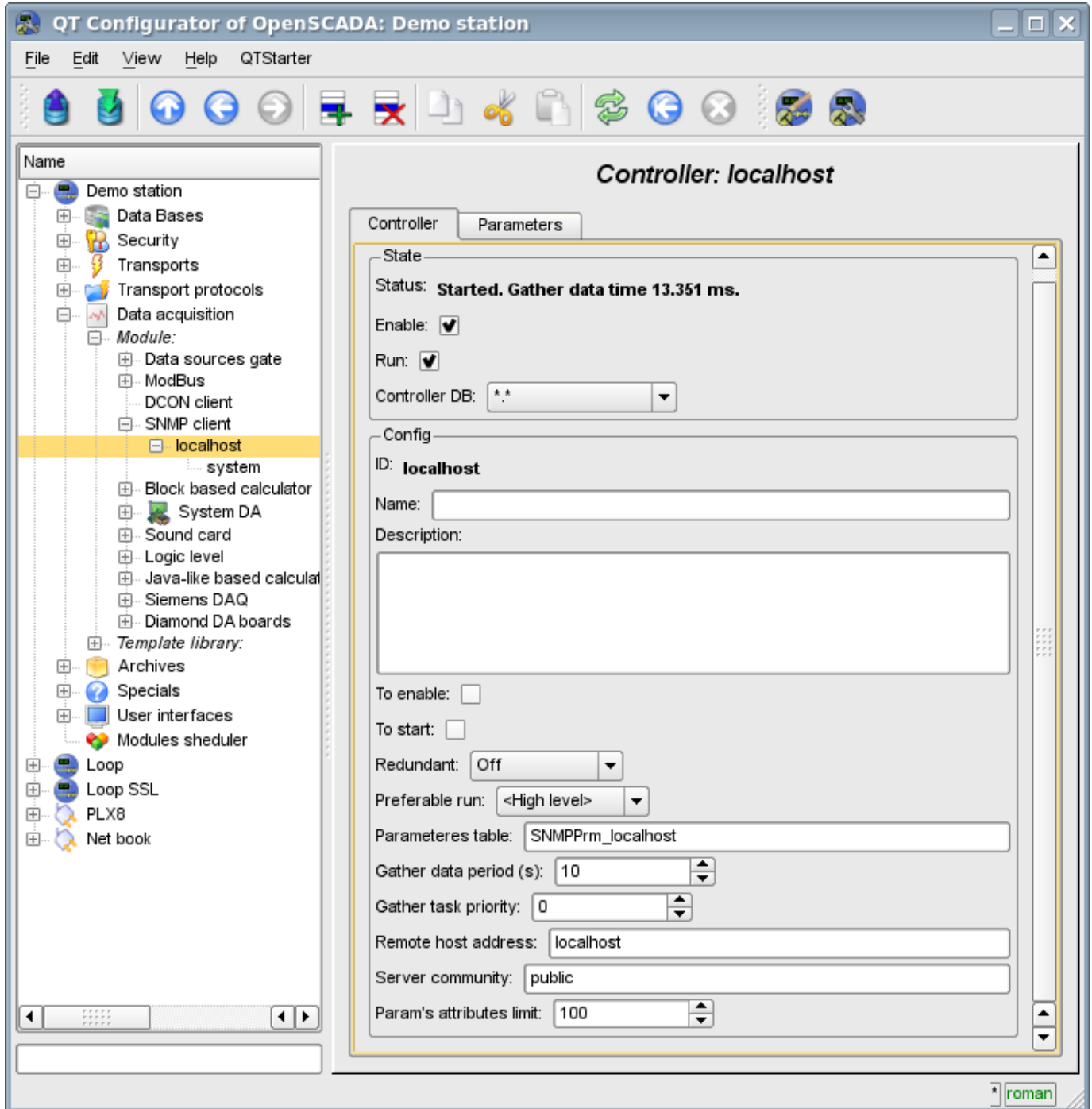


Fig.1. Configuration tab of the controller.

From this tab you can set:

- The state of the controller, as follows: state, «Enable», «Run» and the name of the database containing the configuration.

- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start»
- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the configuration of the parameters of the controller.
- The period and the priority of the task of data acquisition.
- Address of remote host, a group of access and restriction on the number of attributes in the one parameter.

2.2. Parameters

Module *SNMP* provides only one type of parameters – “Standard”. An additional configuration field of the parameter of the module(Fig. 2) is a list of MIB-parameters, the branches of which are to be read.

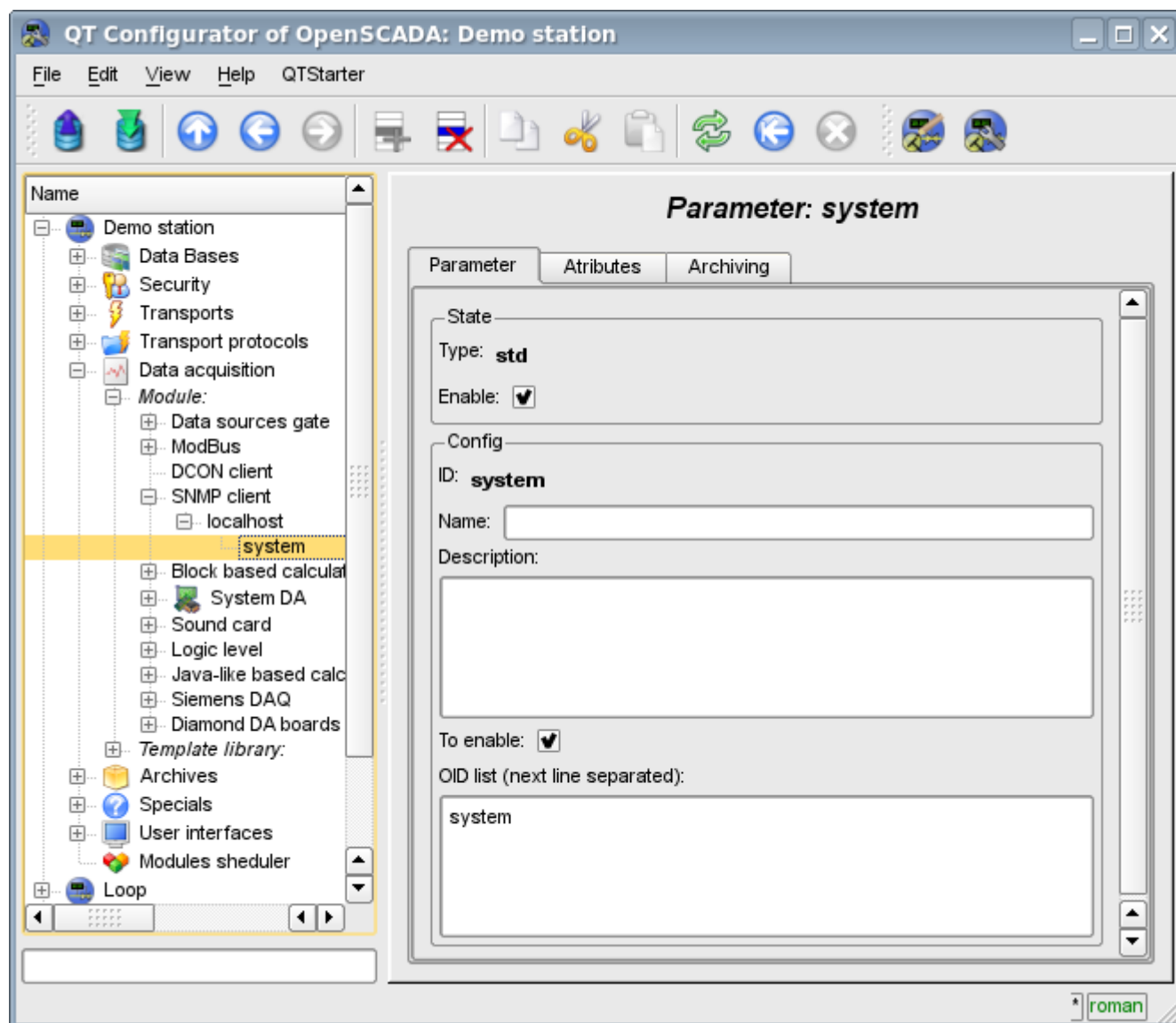


Fig.2. Configuration tab of the parameter.

In accordance with a specified list of MIB-parameters is carried out a survey of their branches and the creation of the attributes of the parameter. Further, updating of the values of parameters is carried out. Attributes are named in accordance with the code addressing of MIB-parameters, as the ID, and the addressing from the base of MIB objects in the name of the attribute(Figure 3).

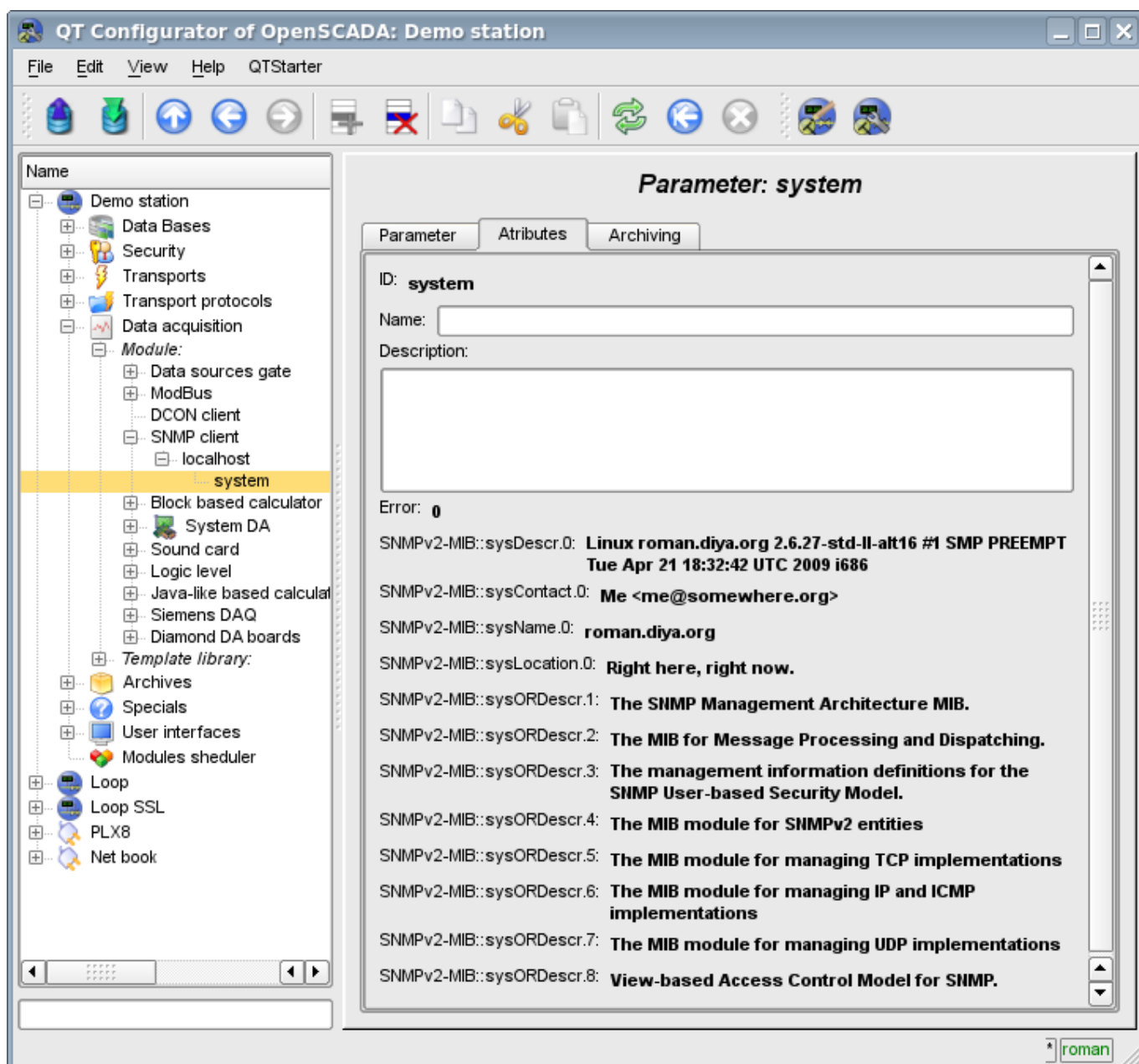


Fig.3. Tab of the attributes of the parameter.

The module of subsystem “Data acquisition”

<Siemens>

<i>Module:</i>	Siemens
<i>Name:</i>	Siemens DAQ
<i>Type:</i>	DAQ
<i>Source:</i>	daq_Siemens.so
<i>Version:</i>	1.2.3
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides a data source PLC Siemens by means of Hilscher CIF cards by using the MPI protocol and Libnodave library for the rest.
<i>License:</i>	GPL

The primary aim of creating the module is to provide support for industrial controllers of firms Siemens of series S7(S7–300, S7–400). Historically, access to the controllers of the firm in the Profibus network is made only through its own communication processor (CP5412, CP5613, etc.) and the protocol S7. These communications processors and API to the protocol S7 are rather expensive, in addition to the drivers for the communication processors and S7 API are closed and are only available for the platform Intel + Windows (I met the information on opportunities to buy for Linux).

As an alternative to these decisions of the company Siemens, which allows you to fully work with the controllers of Siemens, is the range of communication products of firm Hilscher (<http://hilscher.com>), through the communications processors CIF of series PB(Profibus) and the library Libnodave (<http://libnodave.sourceforge.net>).

Feature of Hilscher products is completely open specification of the protocol of exchange with the communication processor, the unified driver for all cards CIF, the availability of drivers for many common operating systems(OS) and openness of the driver for OS Linux(GPL).

The basis of the module is the driver of version 2.621 of Hilsher, kindly provided by Hilsher in the face of [Devid Tsaava](#) for the 2.6 series kernels of OS Linux. Everything needed files to building are included in the module and it is don't needed to satisfy any special dependencies. The driver version 2.621 for the CIF cards is available for download [cif2621.tgz](#).

The range of boards of CIF family of firm Hilsher and unified driver supports the widest range of equipment. To lay support all these features in this module without having all the equipment on hand, it is not possible. Therefore, the support of the equipment will be added on demand and availability of equipment. As of version 1.1.0 module provides support for data sources on the network via Profibus or MPI by means of MPI protocol at the network speed of 9600Bod to 12MBod. In particular, supported and it is carried out check on the controllers of the Siemens company of family S7 (S7–300, S7–400).

Library Libnodave is an implementation of the MPI, S7, ISO-TSAP and others protocols by means of revers-engineering, that are used in interaction with the controllers of Siemens. Library supports many MPI and USB adapters, as well as ProfiNet. Communication processors firm Siemens, on platforms other than Windows, the library doesn't support. At this stage, module support the protocol ISO-TSAP (ProfiNet) through the library Libnodave. Library Libnodave fully incorporated in this module and does not require a special permit of any dependencies during building and in the performance.

1. Communication controllers CIF

CIF family card driver supports the ability to install up to 4 CIF boards. In order to control the availability of cards in the system and their possible configurations, the module provides a form of control and configuration of the CIF-cards(Fig. 1).

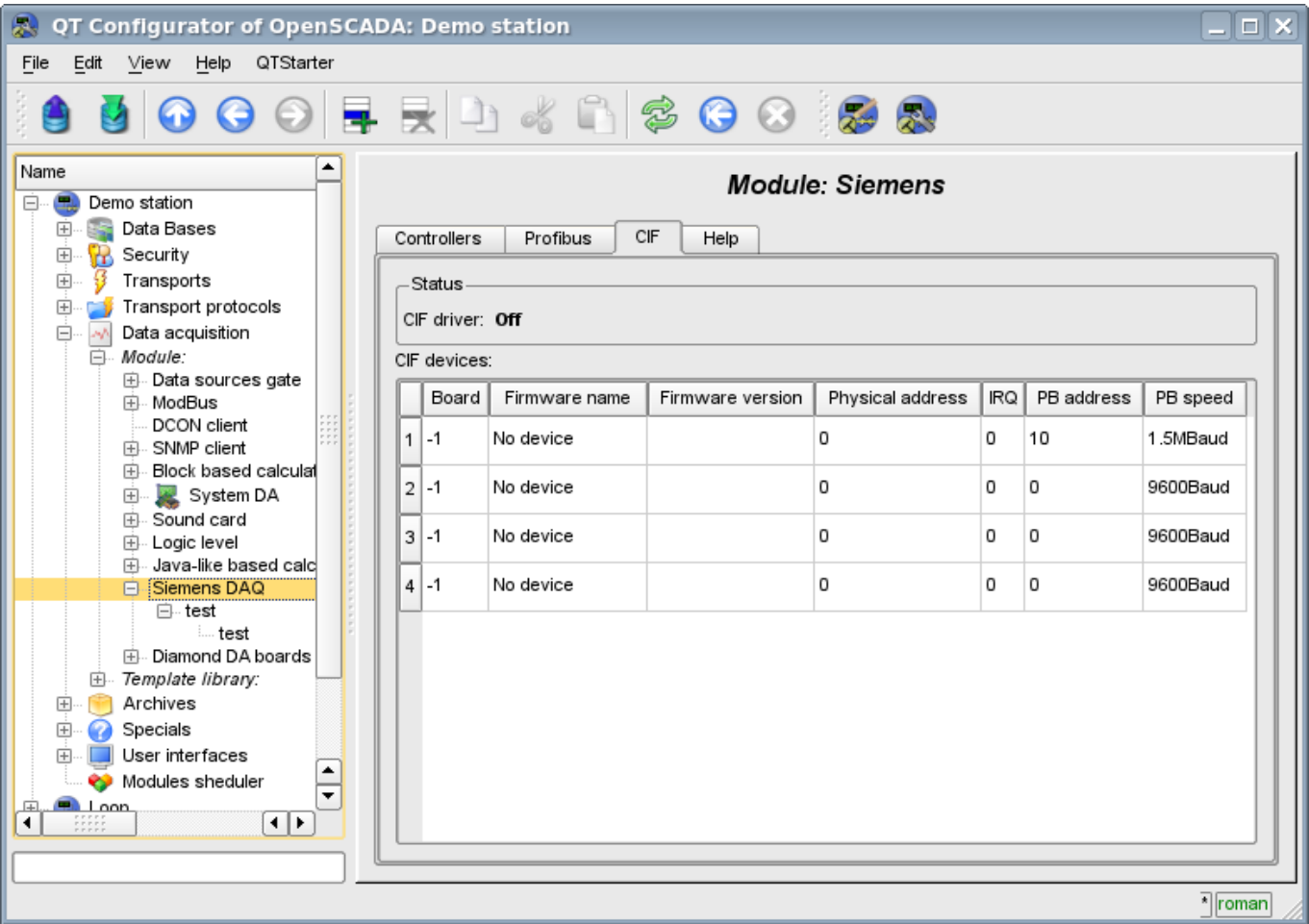


Fig.1. Configuration tab of CIF-boards.

Use this form you can verify the existence of communication processors and their configuration, and configure the network settings of PB Profibus in the form of addresses of communication processor and speeds of bus Profibus. In the other tab of the module(Fig.2) you can verify the presence of various stations in the network Profibus.

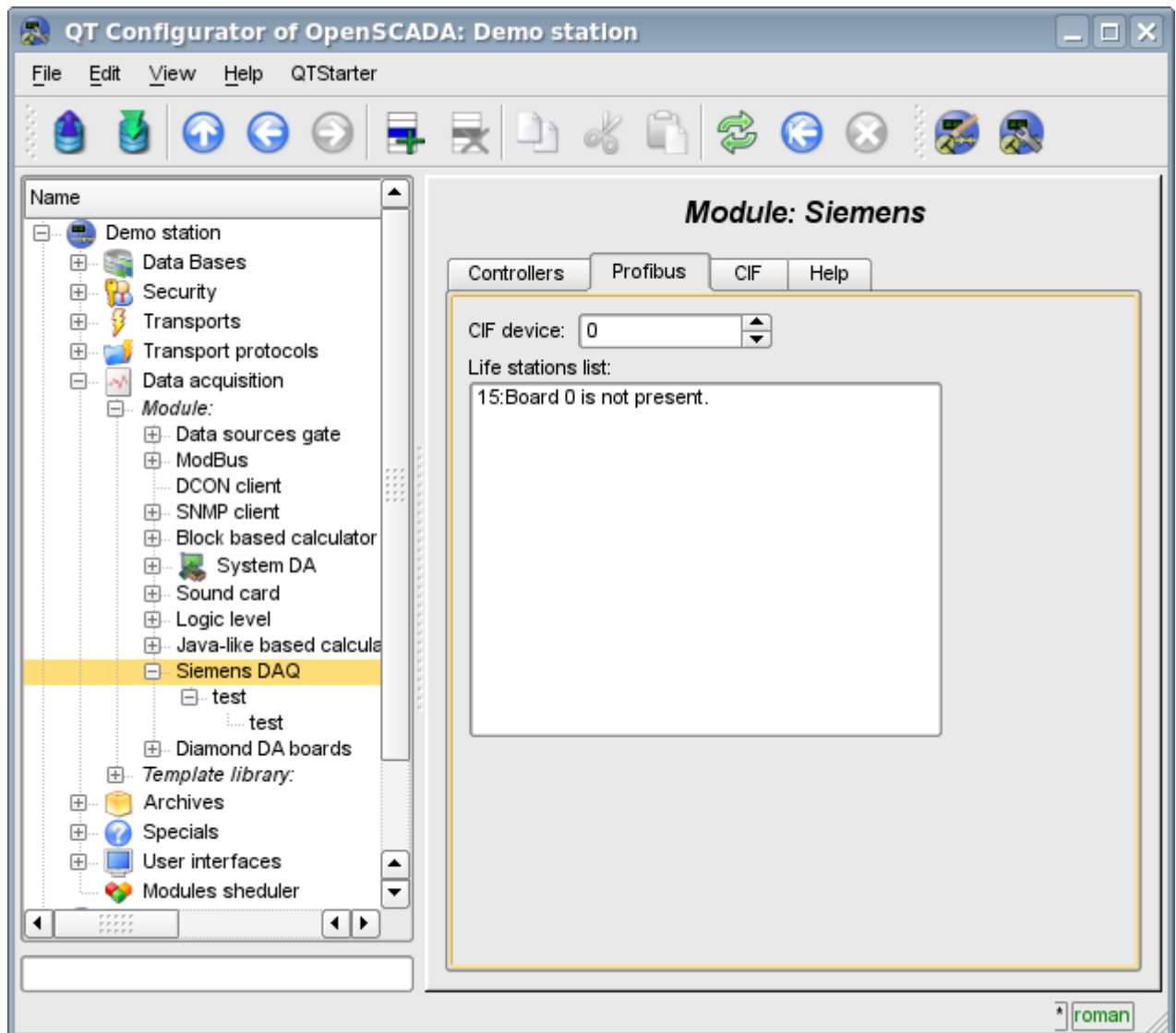


Fig.2. Monitoring tab of Profibus network.

2. The controller of the data source

To add a data source it is created and configured the controller in the system OpenSCADA. Example of the configuration tab of the controller of this type is depicted in Figure 3.

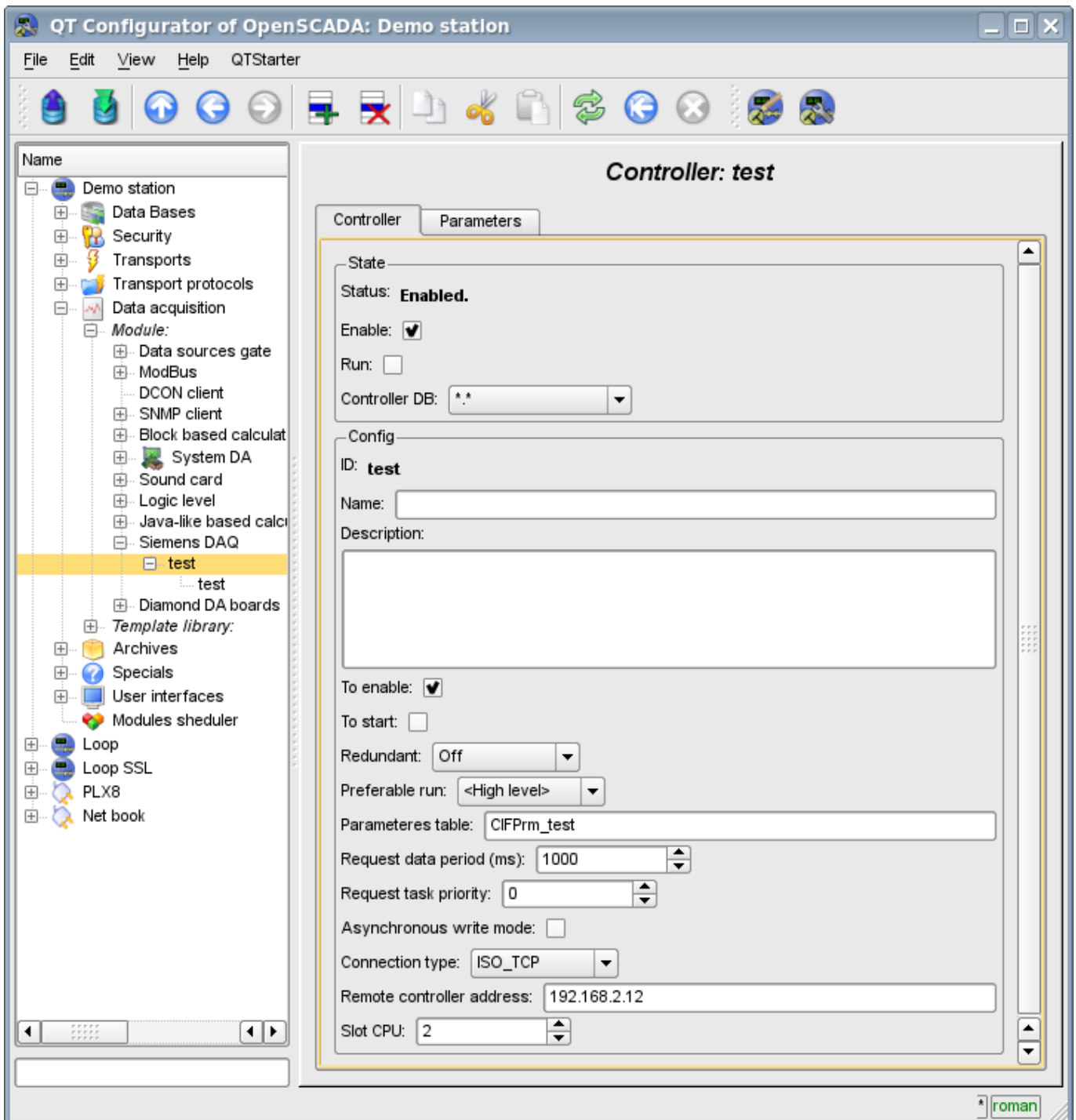


Fig.3. Configuration tab of the controller.

Using this tab you can set:

- The state of the controller, as follows: State, «Enable», «Run» and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».
- Name of table to store the configuration of the parameters of the controller.
- The period and the priority of the task of data acquisition.
- The mode of the asynchronous recording in the remote controllers.
- Connection type. Supported CIF_PB and ISO_TCP

- Address of the industrial controller. If the connection type CIF it is the address of the network Profibus, and in the case of ISO_TCP it is the IP-address in the Ethernet.
- Slot CPU in which the central processor of the controller is placed.
- CIF card used for access to the industrial controller through CIF communication processors.

3. The parameters of the data source

Given the high intellectuality of data sources in the face of industrial controllers of Siemens S7–300 and S7–400, the options are executed on the basis of [templates](#). This approach allows us to go beyond a rigid list of types of parameters, which limits the possibilities of the controllers, and provide users with the ability to build the necessary types of parameters independently or use the library of already been developed types of parameters (templates).

Accordingly, the module provides only one type of parameters – “Logical”. Additional configuration fields of the parameters of the module(Figure 4) is the field of selection of template of the parameter.

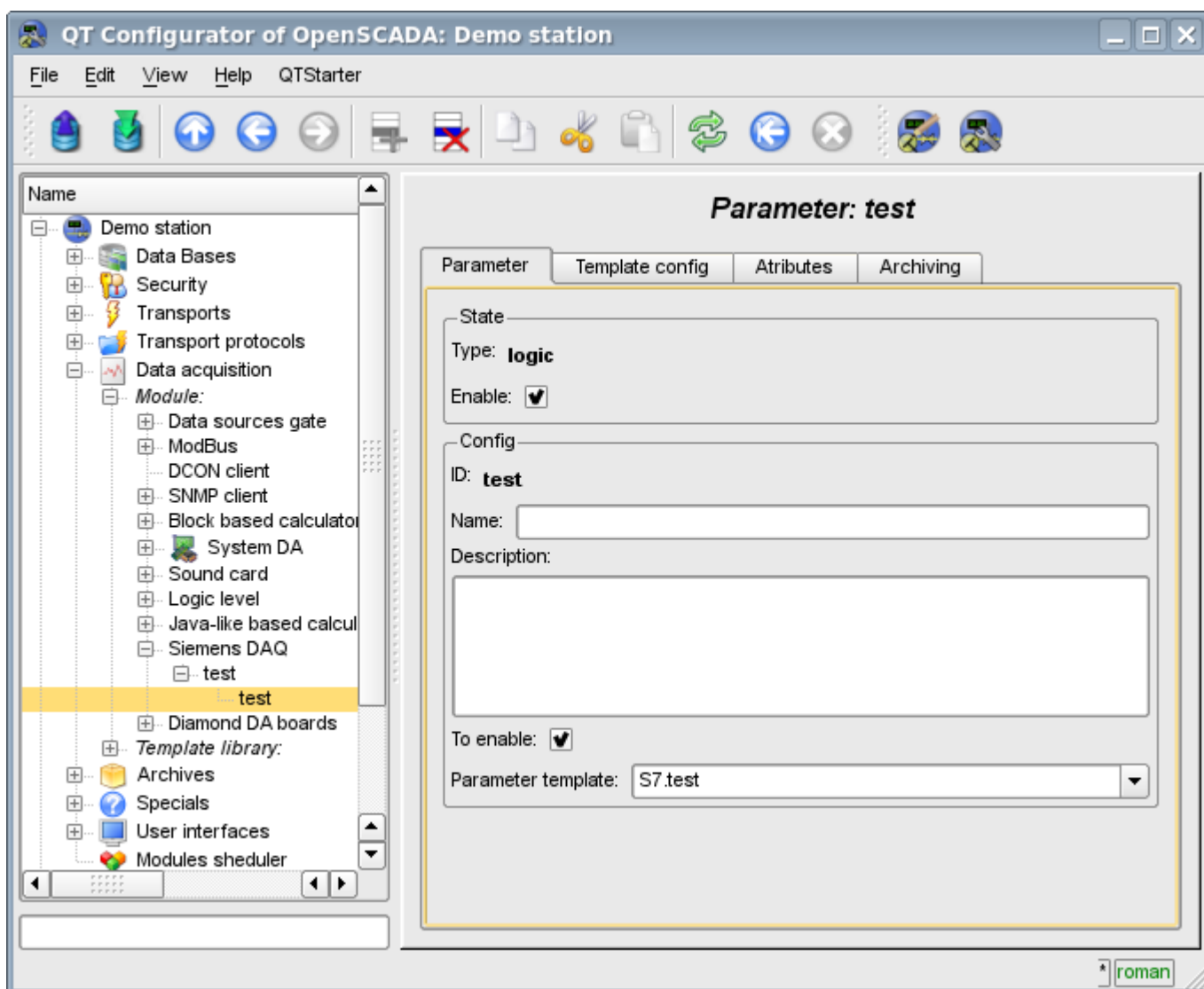


Fig.4. Configuration tab of the parameter.

To configure a template of parameter it is made the appropriate tab. The contents of this tab is defined by the configuration of the template that is the corresponding link fields and fields of setting the constants are formed.

Types of links depend on the type of parameter in the template (boolean, integer, real and string) and the definition of link value(for the group of links). Definition of the group link in the template is written in the format: "<Name of the link>|<The offset in the database>|<The size of the value>", where:

- <Name of the link> — Name of the group link. All links with the same name are grouped and

shown as a link to the database or database with the specified offset.

- *<The offset in the database>* — Name of the offset in the data block (DB). If the only database in the configuration of the template is specified the offset will be specified for the parameter, but if in the configuration of the template the offset will be specified too, the both offsets are summarized together. This approach allows to access a variety of structures in the single data block.
- *<the size of the value>* — Optional field that specifies a custom size of the value in the controller. The following sizes of types of values are provided:

- *Integer*: — 1 byte(signed), 2 byte(signed by default) and 4 byte(signed).
- *Real*: — 4 byte(float by default), 8 byte(double).
- *Boolean*: — always one byte (with a bit through the point – DB1.10.1).
- *String*: — 10 byte(by default) and 1–200 can be set.

An illustrative example of the overall process of the configuration of parameter form the template and to the values is shown in Figures from 5 to 8.

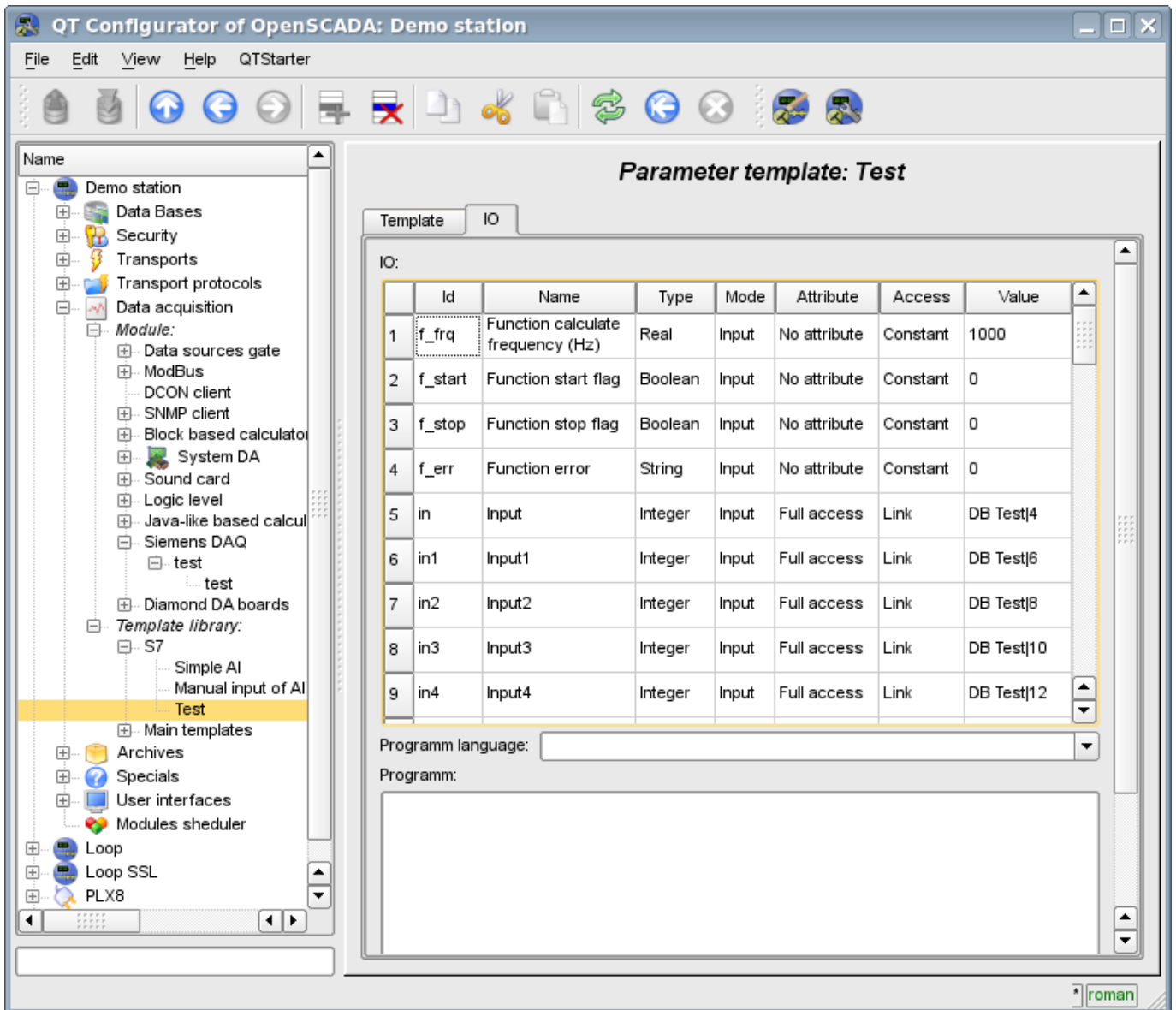


Fig.5. Example of the template with grouping.

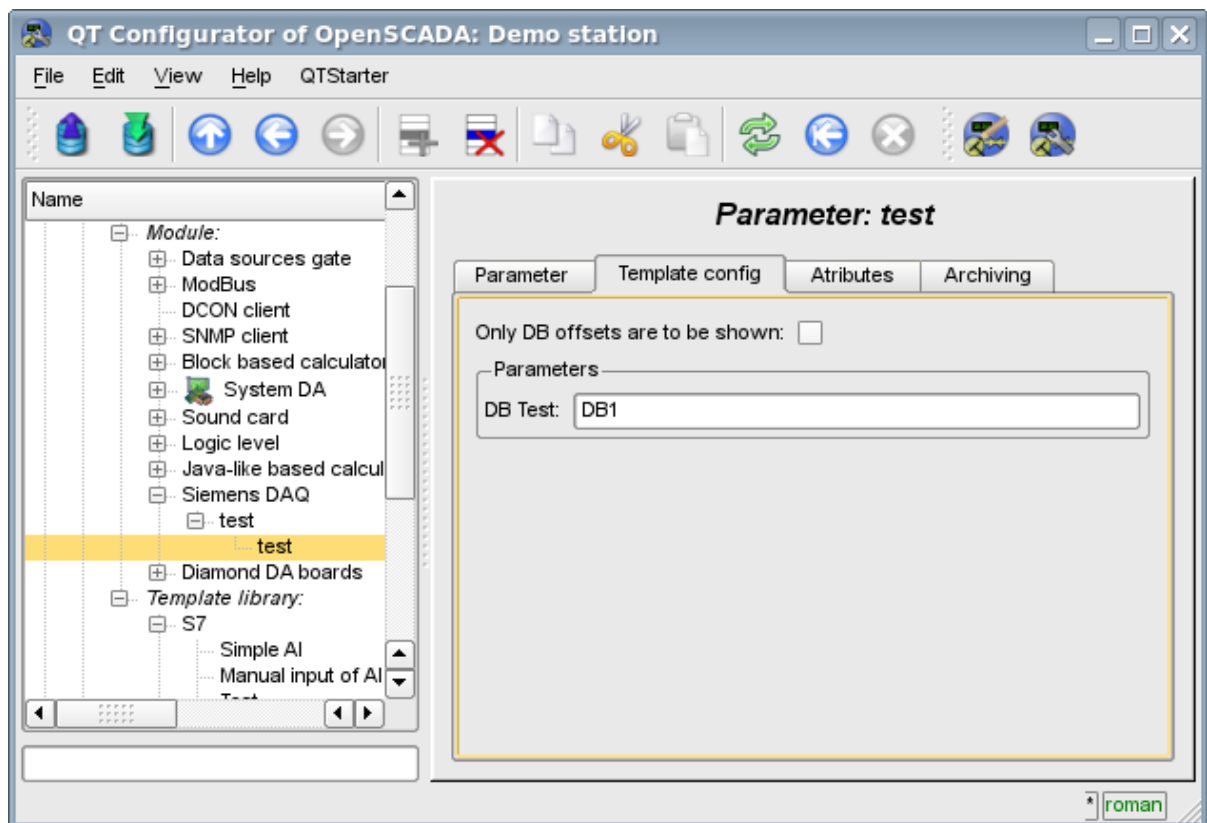


Fig.6. Configuration tab of the template of parameter

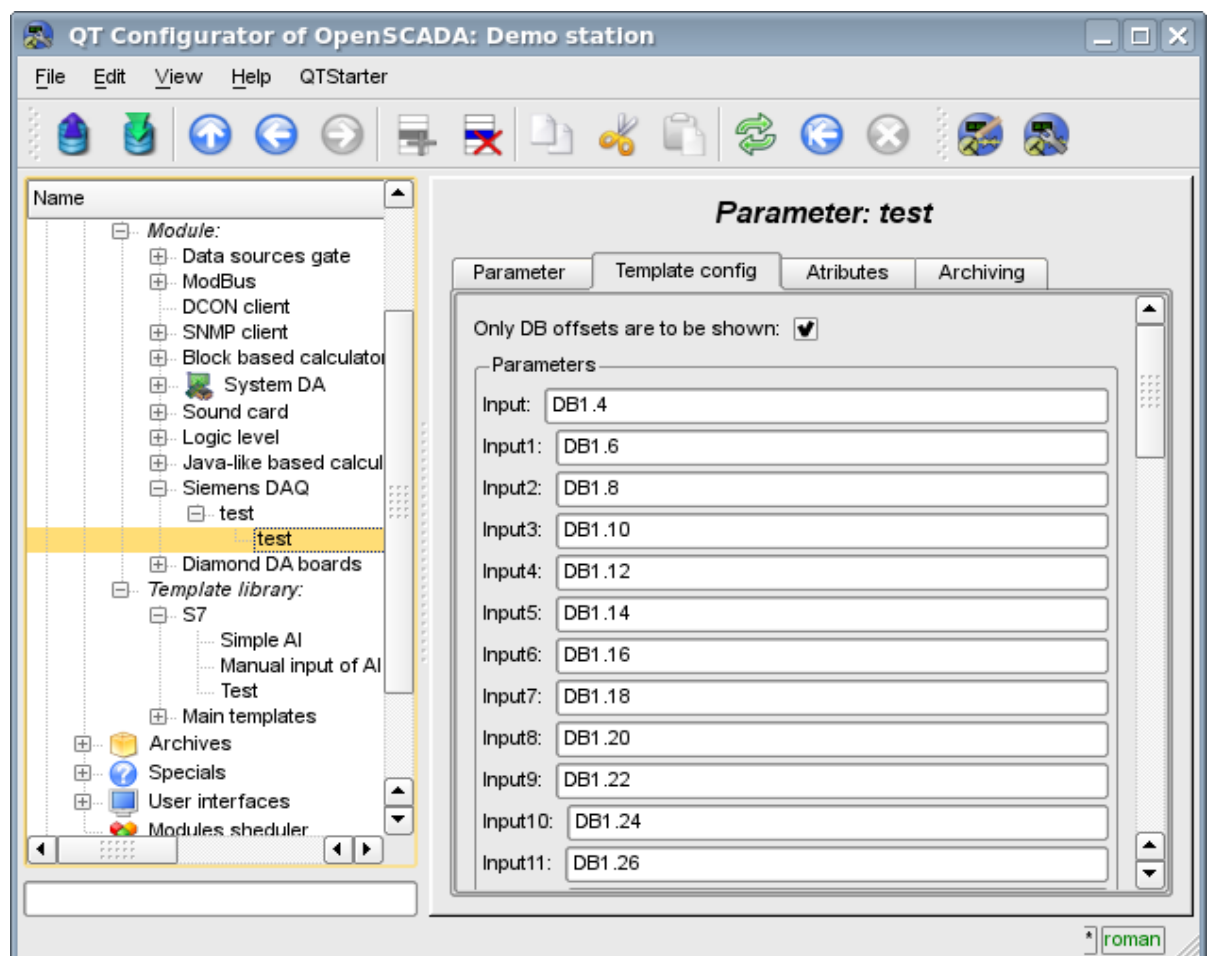


Fig.7. Configuration tab of template of the parameter with an indication of the parameters separately.

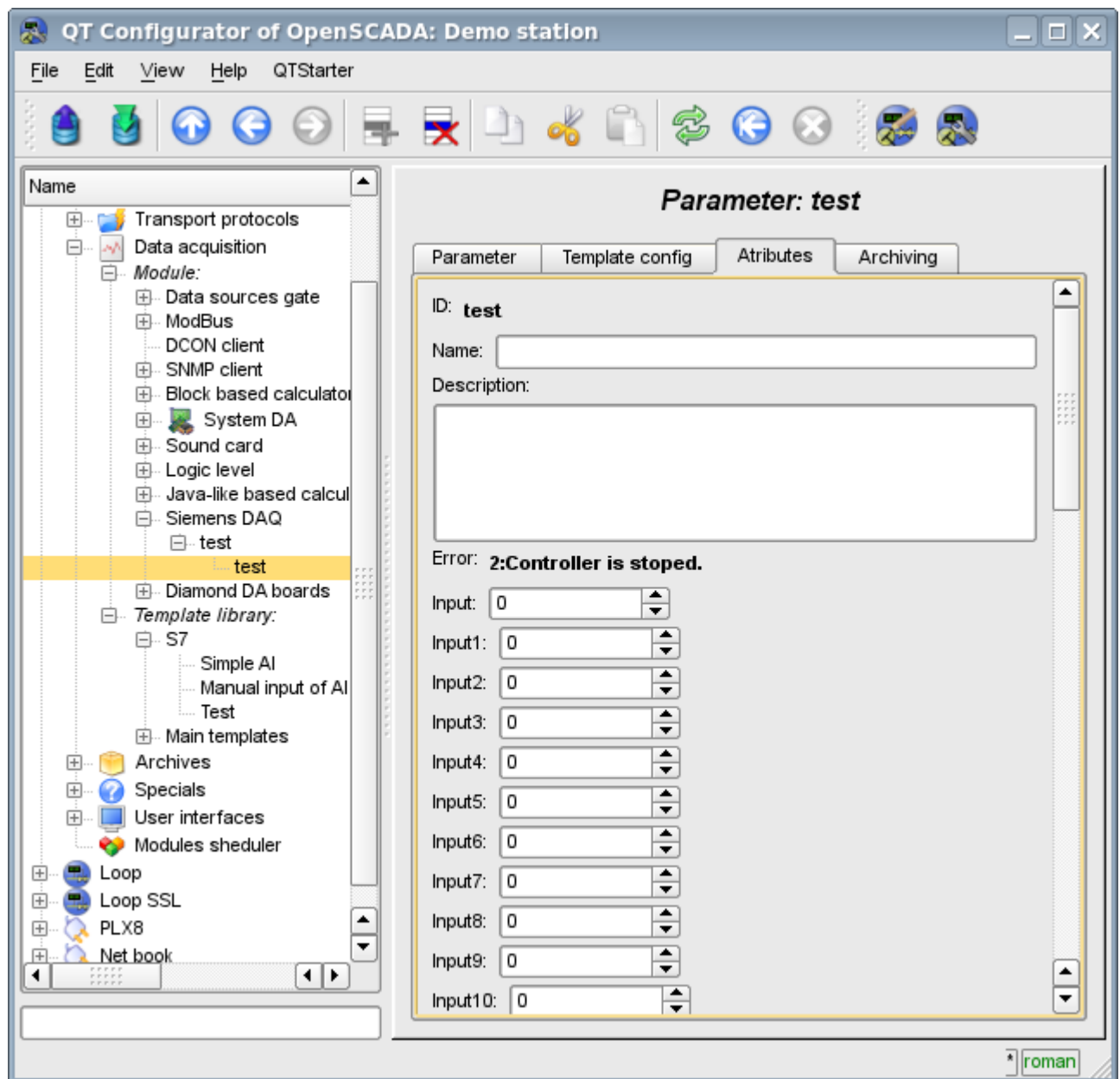


Fig.8. The values of the parameter.

Module supports only the data blocks(DB) of the controllers addressing!

4. Asynchronous recording mode

The standard recording mode for SCADA-systems interacting with the PLC, is the synchronous, because it allows to control the correctness of the conclusion of the record operation. However, in cases of recording multiple parameters at once and often, this approach is not justified in view of sending a multitude of small requests to the controller that overrides the PLC and has a large time interval. The solution is asynchronous recording of an adjacent values by means of the single block. This is supported by this module and allows you to record all parameters immediately by the adjacent blocks of 240 byte. Read and write in this mode is performed by the adjacent blocks with the periodicity of survey of the controller.

5. Comments

After a targeted search was found a few solutions of the problem of communication with industrial controllers of firm Siemens through various communication interfaces:

- Found a lot of solutions from the company Siemens, which supplied with solutions that support an open operating system "Linux"

(http://www.automation.siemens.com/net/html_76/produkte/040_cp_1616.htm, ...).

Links

Firm's Hilscher driver for boards family CIF: [cif2621.tgz](#)

The patch for build driver for kernel Linux 2.6.29: [lastkernels.patch](#)

The modules <ModBus> of subsystem “Data acquisition” and subsystem “Transport protocols”

Parameter	Module 1	Module 2
<i>ID:</i>	ModBus	ModBus
<i>Name:</i>	ModBus	ModBus
<i>Type:</i>	DAQ	Protocol
<i>Source:</i>	daq_ModBus.so	daq_ModBus.so
<i>Version:</i>	1.1.1	0.6.1
<i>Author:</i>	Roman Savochenko	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko	
<i>Description:</i>	Provides implementation of client service of the protocol ModBus. Modbus/TCP, Modbus/RTU and Modbus/ASCII protocols are supported.	Provides implementation of protocols ModBus. Modbus/TCP, Modbus/RTU and Modbus/ASCII protocols are supported.
<i>License:</i>	GPL	GPL

ModBus – communication protocol based on the client-server architecture. It was developed by Modicon for using in the programmable logic controllers (PLC). It became the de facto standard in the industry and is widely used for the connection of industrial electronic equipment. Used to transfer data via serial line RS-485, RS-422, RS-232, and network TCP/IP. Currently supported non-profit organization ModBus-IDA.

There are three modes of the protocol: ModBus/RTU, ModBus/ASCII and ModBus/TCP. The first two use the serial communication lines (mostly RS-485, less RS-422/RS-232), the last uses TCP/IP network to transfer data.

Module of the data acquisition provides an opportunity to gather the information from various devices by means of the protocol ModBus in all modes. Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level. At the same time, the module of the protocol allows you to create and issue data by means of the protocol ModBus in various modes, and through interfaces that are supported by modules of subsystem “Transports”.

1. General description of the ModBus protocol

Protocol ModBus/RTU requires one lead(requesting) device in the line(master), which can send commands to one or more driven devices(slave), referring to them by a unique in the line address. Syntax of the commands of the protocol allows to address 247 devices on the one connection line of standard RS-485(less RS-422 or RS-232). In the case of TCP addressing mode is excluded from the protocol, as it is implemented in the TCP/IP stack.

Initiative of exchange always comes from the leading device. Slave devices listen the line. Master request (package, the sequence of bytes) in the line and turns into a listening line. Slave device responds to the request, which came to him. The end of the response package the master determines by the time interval between the end of the reception the previous byte and the beginning of the reception of the following. If this interval exceeds the time required to receive one and a half of the byte on the given speed, receiving of the response frame is considered complete. Frames of the request and reply in the ModBus protocol have the fixed format.

1.1. Addressing

All data operations are tied to zero, each type of data (register, bit, register of input or bit of input) addresses begin with 0000 and ends 65535.

1.2. Standard codes of functions

In ModBus protocol it can be divided into several subsets of commands (Table 1).

Table 1: The subset of commands of ModBus protocol

Subset	Range of codes
Standard	1–21
Reserve for advanced features	22–64
Custom	65–119
Reserve for own needs	120–255

Module of the protocol process the requests by the commands 0x03 and 0x06 for reading and writing registers, 0x01 and 0x05 for reading and writing bits.

2. Module of the implementation of the protocol

ModBus protocol module contains the code implementing of the protocol part of ModBus, namely particular variants of protocols ModBus/TCP, ModBus/RTU and ModBus/ASCII. Module of the protocol in conjunction with the selected transport is actively used by the data acquisition module for direct queries implementation. Because of the module of the protocol is independent, by using of it you can create additional modules for data acquisition by non-standard functions of the expansion of ModBus of various automation equipment.

2.1. API functions of outgoing requests

API functions for outgoing queries operate with the exchange of blocks PDU, XML-wrapped in packages with the following structure:

```
<prt id="sId" reqTm="reqTm" node="node" reqTry="reqTry">[pdu]</prt>
```

Where:

- *prt* - name of the tag with the name of the used variant of the protocol (TCP, RTU or ASCII).
- *sId* - identifier of the source of the query. Used for placing to the protocol the output protocol.
- *reqTm* - time of the request, namely the time during which the answer is expected.
- *node* - number of the destination node or the identifier of the unit ModBus/TCP.
- *reqTry* - number of attempts of repeating the request with the error in the answer. Only for ModBus/RTU and ModBus/ASCII.
- *pdu* - directly block of the unit of the protocol data (PDU) ModBus.

The resulting pdu replaces the request pdu in the XML-package, and set the attribute "err" with the code and text of the errors, if it is took place.

2.2. Servicing of the requests for ModBus protocol

Input part of servicing of the requests to the module of the protocol realizes verification and processing of the requests through objects of the nodes, provided by the module (Fig. 1). Actually, the mechanism is implemented, that allow the system OpenSCADA to perform the role of the ModBus/TCP server or the slave device of ModBus/RTU and ModBus/ASCII. Thus the system OpenSCADA gets an opportunity to serve the role of any participant of the ModBus networks.

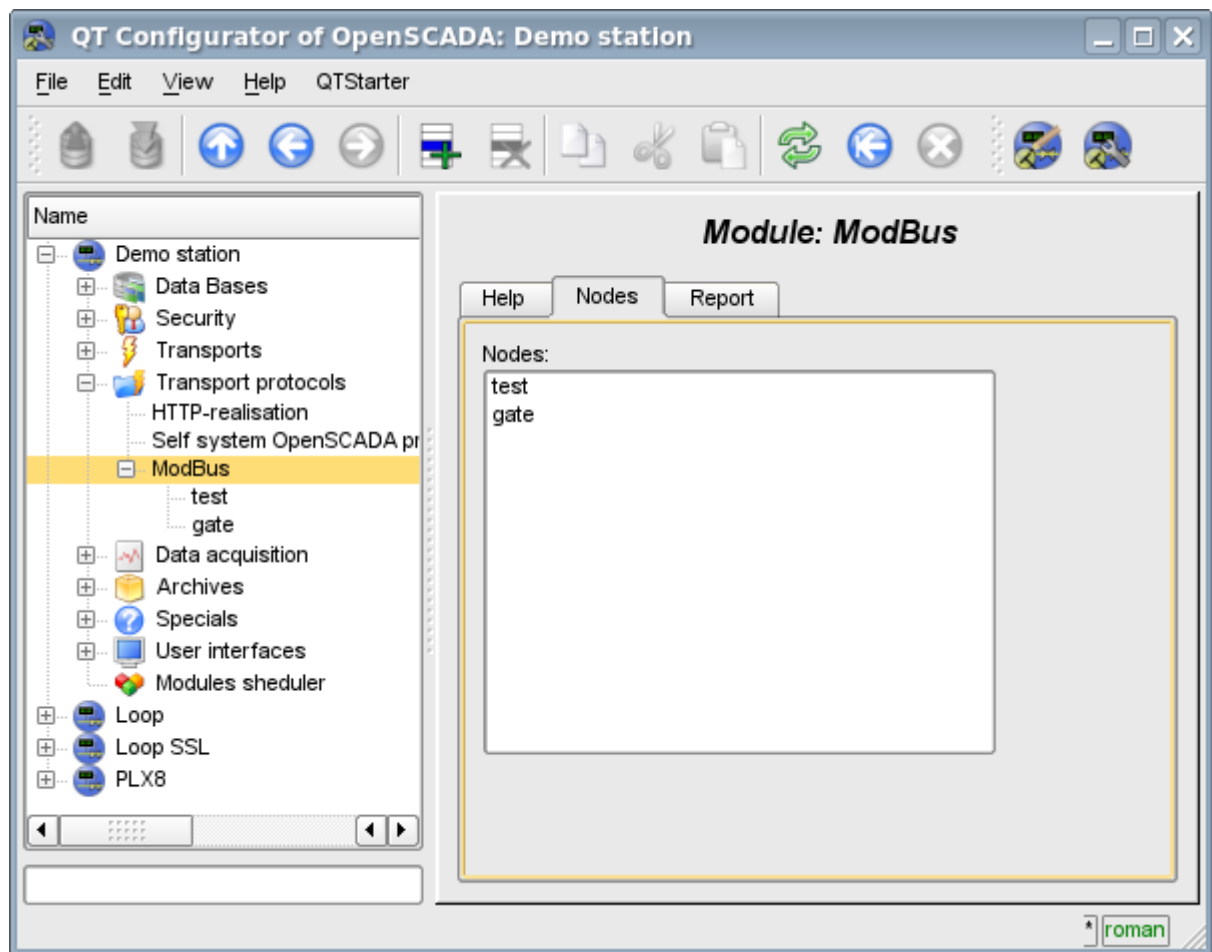


Fig.1. Tab of the list of the nodes of servicing incoming requests of the protocol.

The node of the protocol is equivalent to the physical node of the device of ModBus network. Node of the protocol can operate in three modes:

- "Data" - mode of the reflection of data of OpenSCADA on arrays of registers and bits of ModBus to transfer them at the request of the client node or master.
- "Gateway of the node" - mode of the redirecting of the requests to the node of the another ModBus network through this node.
- "Gateway of the network" - mode of the redirecting of the requests to any node in another ModBus network, actually carrying out the integration of several ModBus networks into one.

Since the protocol nodes can be created a great number, it turns out that on the one interface, ie in the one network, one station on the basis of OpenSCADA can clear provide multiple nodes of ModBus network with different data.

Lets consider particular configuration of the node of the protocol in various modes.

The mode of the node of the protocol “Data”

Mode is used to reflect the data of OpenSCADA on arrays of registers and bits of ModBus. The overall configuration of the node is made in the tab “Node”(Fig. 2) by the parameters:

- The state of the node, as follows: «Enable» and the name of the database containing the configuration.
- Id, name and description of the node.
- The state, in which the node must be translated at boot: «To enable».
- Address of the node in the ModBus network from 1 to 247.
- Inbound traffic, to the network of which the node is belonged to. It is selected from the list of input transport of the subsystem “Transports” of OpenSCADA. Specifying as the transport the symbol "*" makes this node a participant of any ModBus network with the processing of requests from any transport.
- Variant of the ModBus protocol, requests in which must be processed by the node from the list: All, RTU, ASCII, TCP/IP.
- The choice of the mode, in this case the mode “Data”.
- Period of calculation of data in seconds. Specifies the frequency of processing of forming for the requests data, namely, data tables of ModBus, calculation of data programs and servicing of links to the data of OpenSCADA.

Node in this mode process the following standards commands of the ModBus protocol:

- 01 - reading of the group of bits;
- 03 - reading of the group of registers;
- 05 - setting of the single bit;
- 06 - setting of the single register.

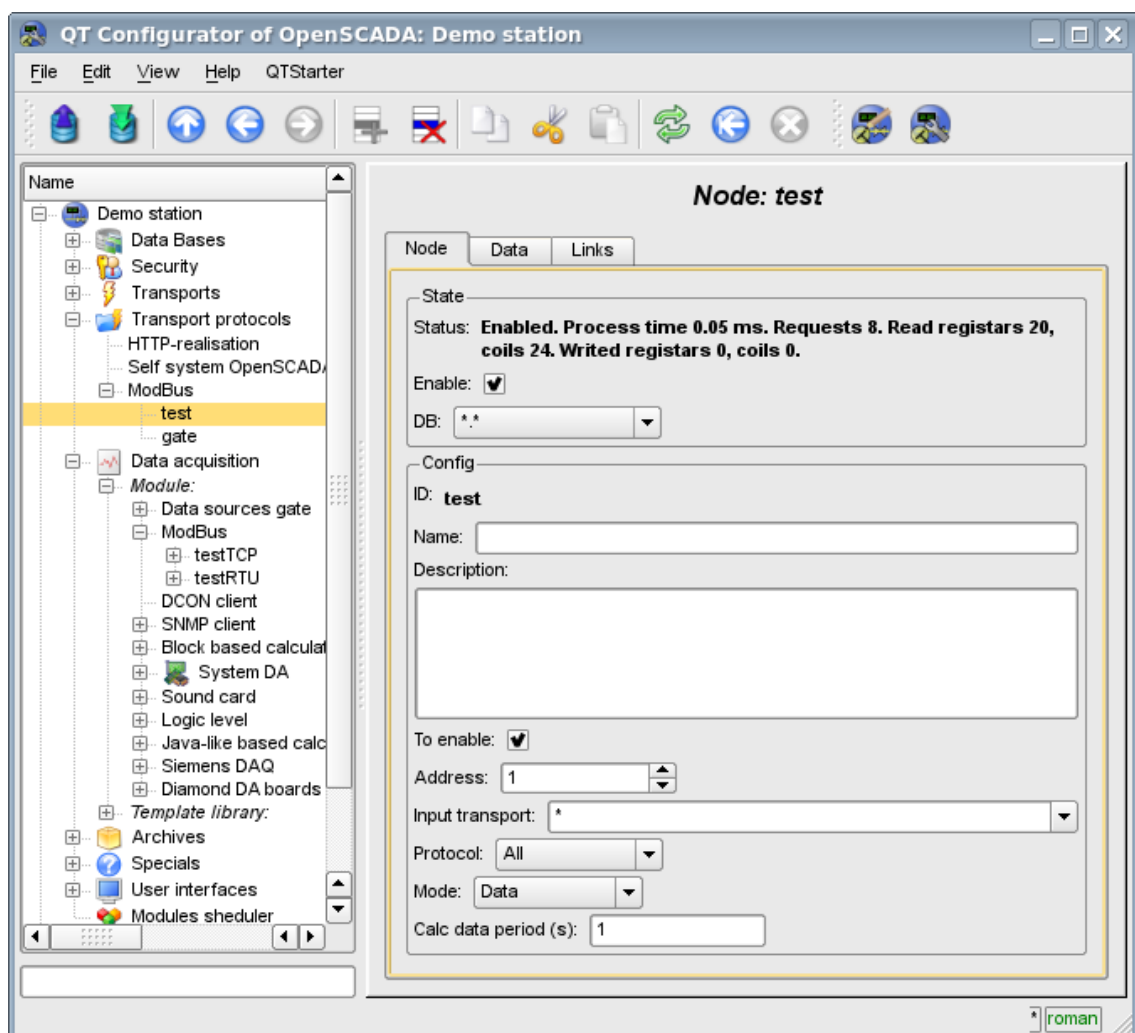


Fig.2. The tab “Node” of the configuration page of the node of the protocol in the “Data” mode.

To form the table of the reflection of the data of ModBus network, namely, registers and bits the tab "Data" is provided(Fig.3). The tab "Data" contains a table of parameters and program for processing of the parameters with the specified programming language, which is available in the system OpenSCADA. Table contains the parameters with the properties:

- *Id* - ID of the parameter. It is the key for the formation of the tables of registers and bits of ModBus. To specify that this parameter is the register of the ModBus, identifier must be written as "R[N]w", where N - number of the register's number from 0 to 65535, and w - optional character indicating the possibility of setting of it by the ModBus request eg: R23, R456, R239w. For the ModBus bit specifying, ID must be written as "C[N]w", where N - number of bits from 0 to 65535, and w - optional character indicating the possibility of setting of it by the ModBus request, eg: C437, C0, C39w. All other parameters are internal and are used for a variety of intermediate calculations, processing and conversion.
- *Name* - The name of the parameter is used for the naming of the connection.
- *Type* - Type of the parameter from the list: "Real", "Integer", "Boolean" and "String". For the registers and bits of ModBus it makes sense to set "Integer" and "Boolean" type, respectively.
- *Connection* - Sign that this option should be to connect with the attribute of the parameter of subsystem "Data acquisition". These connections are set in the "Connection" tab.
- *Value* - The original or current, if the node is switched on, the value of the parameter.

The table by default identifies several parameters of special significance:

- *f_frq* - frequency of computing the table by the program;
- *f_start* - sign of the first computing, the start up of the program.
- *f_stop* - sign of the last execution, the stop of the program.

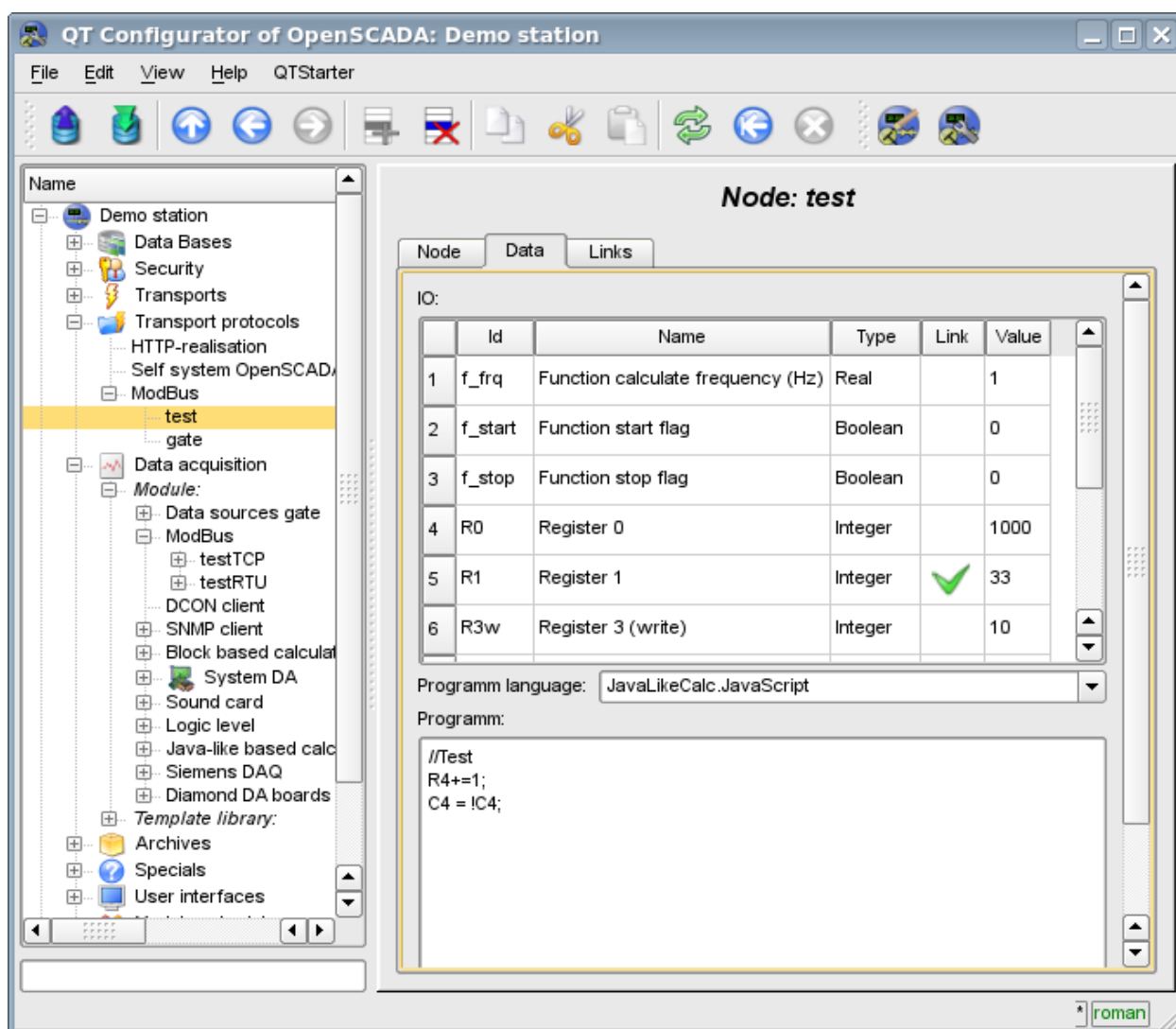


Fig.3. The tab "Data" of the configuration page of the node of the protocol in the "Data" mode.

For the parameter which are signed as links above it can be set the links only to switched off node of the protocol in the tab “Connections”(Figure 4).

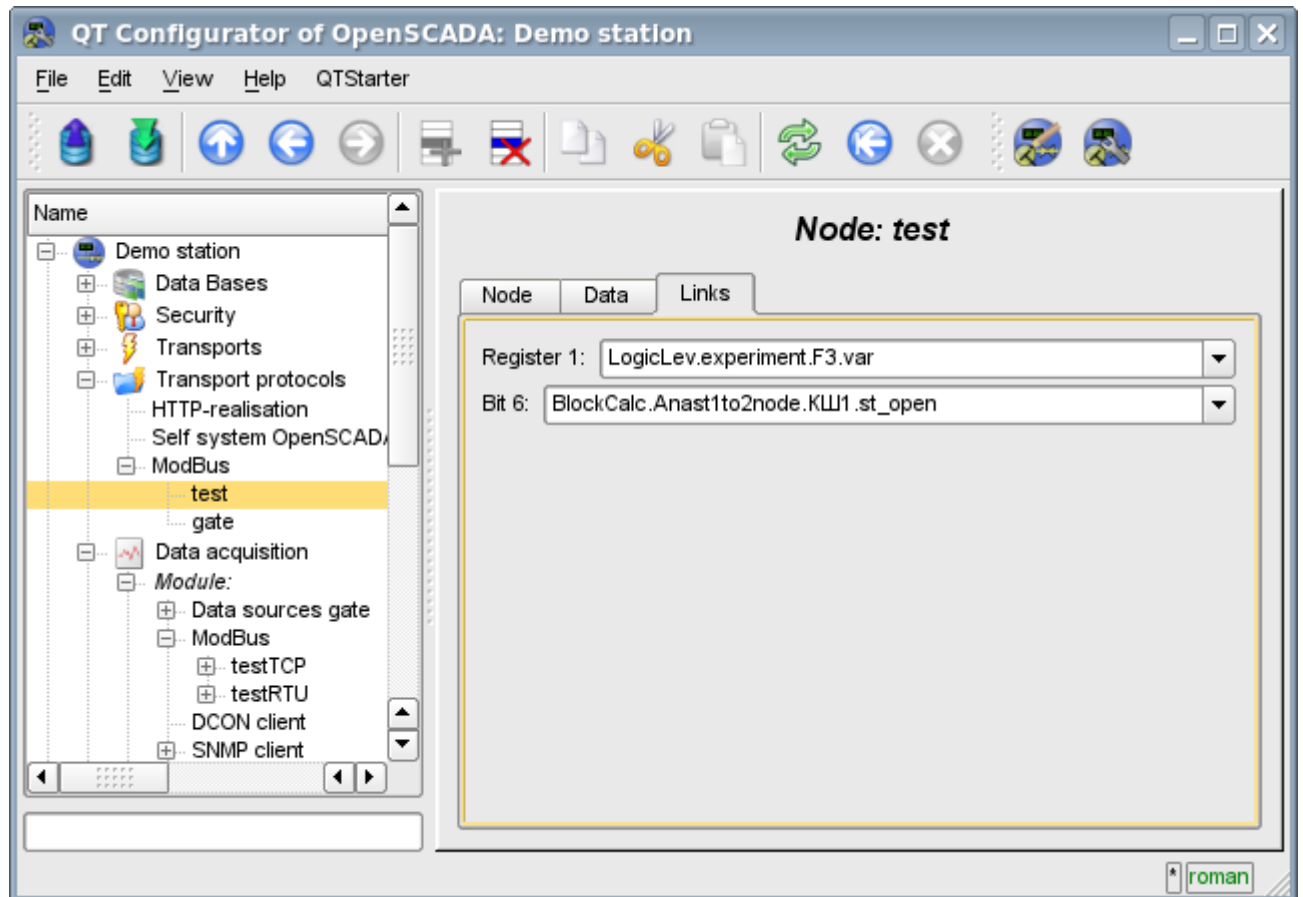


Fig.4. The tab “Links” of the configuration page of the node of the protocol in the “Data” mode.

The mode of the node of the protocol “Gateway of the node”

Mode is used to carryover the requests to a separate device in the other ModBus network to the ModBus network, in which this node is configured. The overall configuration of the node is made in the tab “Node”(Fig. 2) by the parameters:

- The state of the node, as follows: Status, «Enable» and the name of the database containing the configuration.
- Id, name and description of the node.
- The state, in which the node must be translated at boot: «To enable».
- Address of the node in the ModBus network from 1 to 247.
- Inbound traffic, to the network of which the node is belonged to. It is selected from the list of input transport of the subsystem “Transports” of OpenSCADA. Specifying as the transport the symbol "*" makes this node a participant of any ModBus network with the processing of requests from any transport.
- Variant of the ModBus protocol, requests in which must be processed by the node from the list: All, RTU, ASCII, TCP/IP.
- The choice of the mode, in this case the mode “Gateway of the node”.
- Transport, in which the request must be redirected, from the list of outgoing transports of subsystem “Transports”.
- Protocol in which to redirect the request.
- Address of the node of ModBus network from 1 to 247, in which the request is forwarded to.

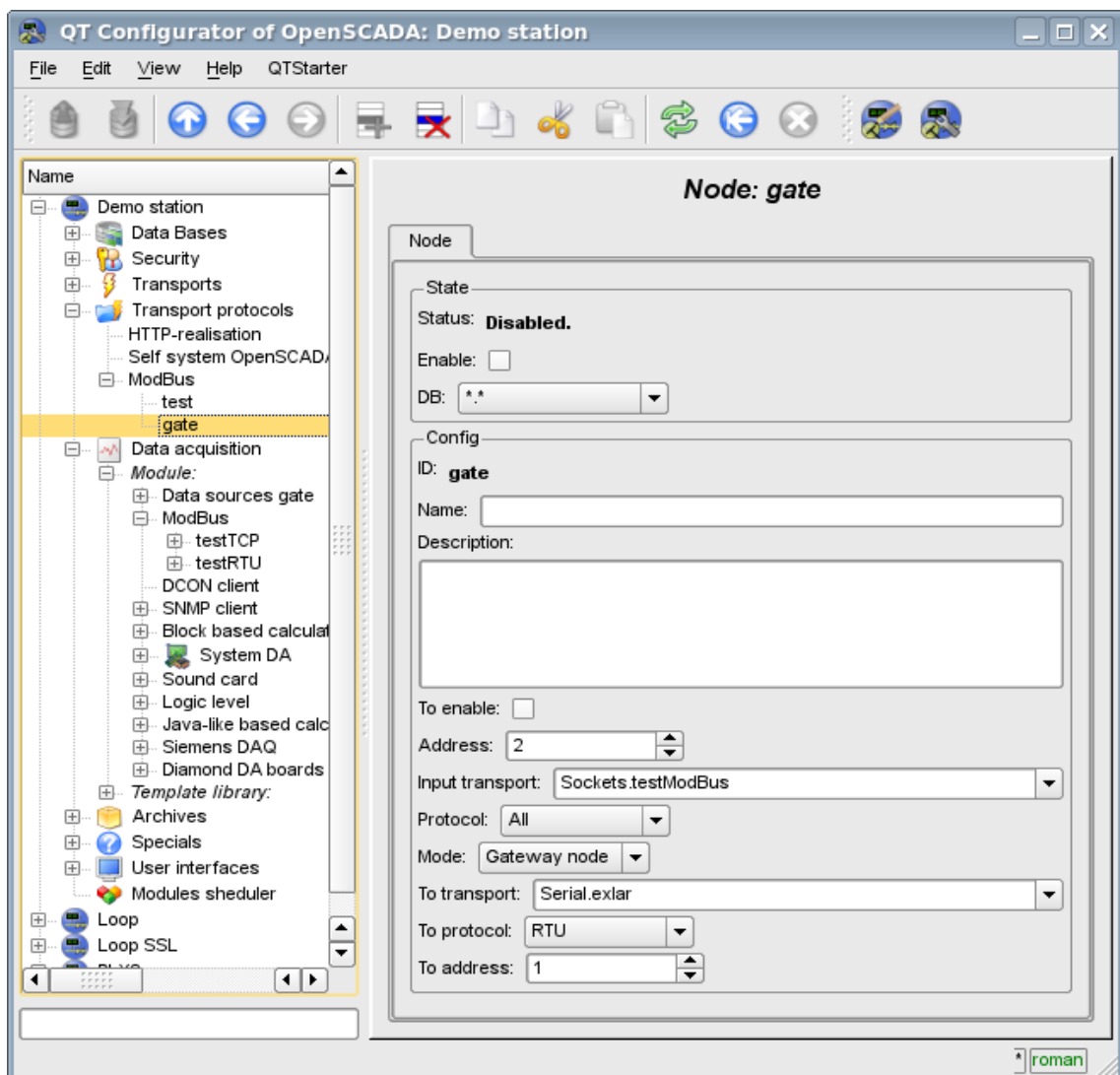


Fig.5. The tab “Node” of the configuration page of the node of the protocol in the “Gateway of the node” mode.

The mode of the node of the protocol “Gateway of the network”

Mode is used to carryover the requests of the network at whole to the other ModBus network from the ModBus network, in which this node is configured. The overall configuration of the node is made in the tab “Node”(Fig. 2) by the parameters:

- The state of the node, as follows: «Enable» and the name of the database containing the configuration.
- Id, name and description of the node.
- The state, in which the node must be translated at boot: «To enable».
- Incoming transport of the network, from which the requests are transfered. It is selected from the list of input transport of the subsystem “Transports” of OpenSCADA.
- Variant of the ModBus protocol, requests in which must be processed by the node from the list: All, RTU, ASCII, TCP/IP.
- The choice of the mode, in this case the mode “Gateway of the network”.
- Transport, in which the request must be redirected, from the list of outgoing transports of subsystem “Transports”.
- Protocol in which to redirect the request.

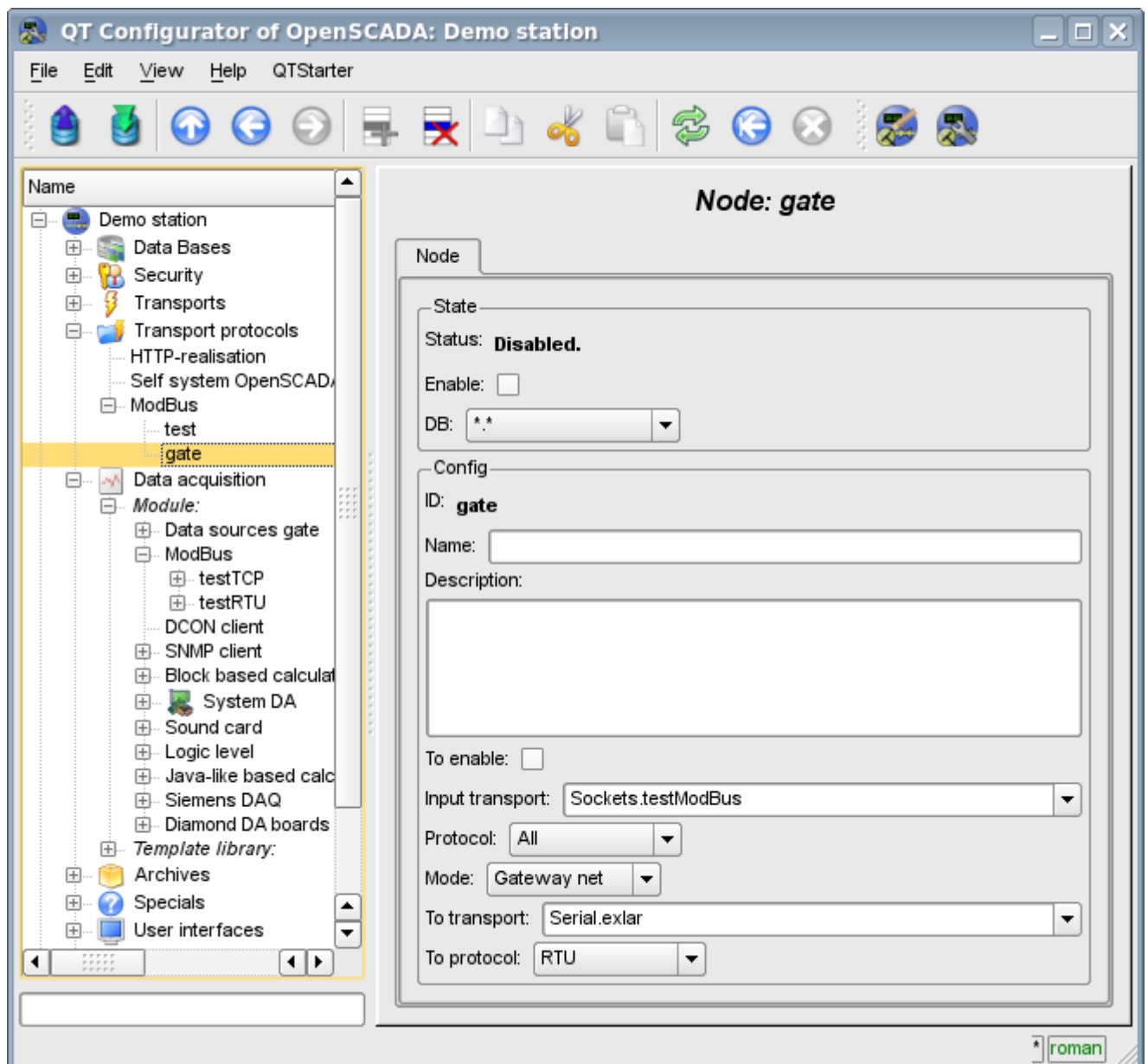


Fig.6. The tab “Node” of the configuration page of the node of the protocol in the “Gateway of the network” mode.

2.3 Report of the ModBus requests

To be able to monitor the correct implementation of requests to the various components the a module provides an opportunity to incorporate the report of the requests that pass through the protocol module. The report included by indication of non zero number of entries in the tab “Report” of the page of the module of the protocol(Fig.7).

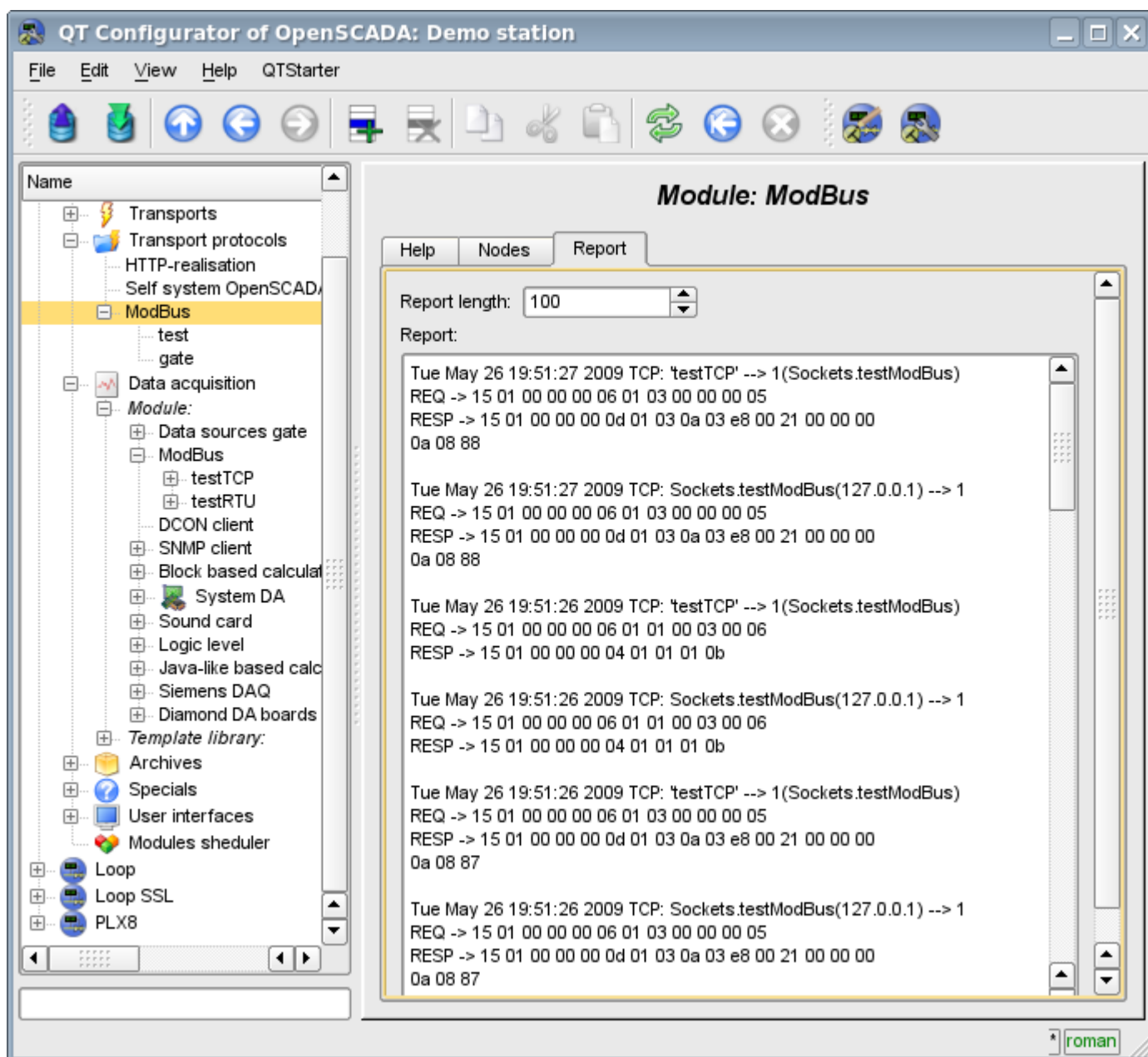


Fig.7. “Report” tab of the page of the module of the protocol.

3. Data acquisition module

Module of the data acquisition provides an opportunity to interrogate and write registers and bits of devices through protocol modes TCP, RTU, ASCII and commands of request 0x01 – 0x06.

3.1. Controller of data

For addition of a ModBus data source the controller is created and configured in the system OpenSCADA. Example of the configuration tab of the controller is depicted in Fig.8.

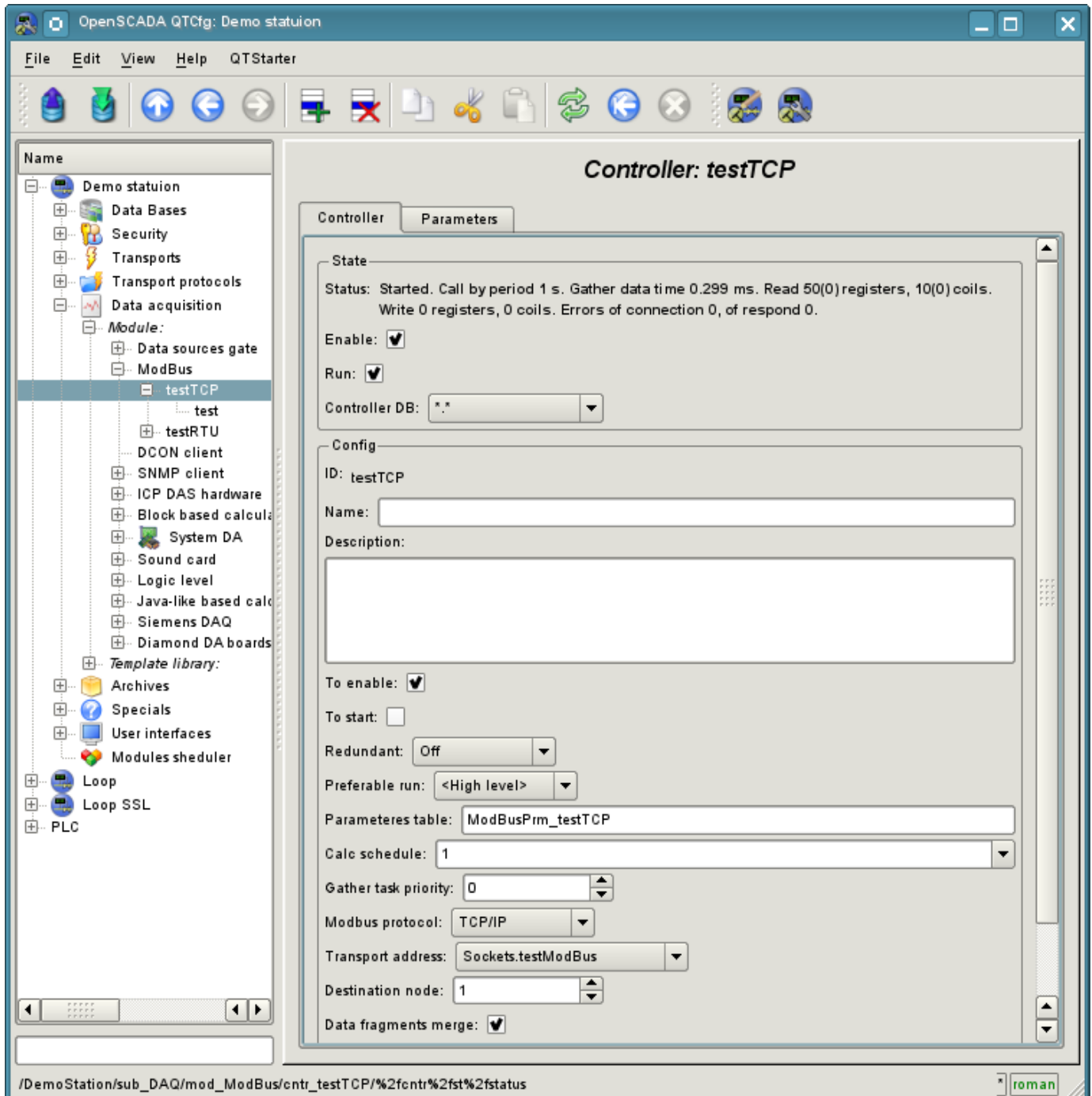


Fig.8. Configuration tab of the controller.

Using this tab you can set:

- The state of the controller, as follows: Status, «Enable», «Run» and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».

- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the configuration of the parameters of the controller.
- The period and the priority of the task of data acquisition.
- ModBus protocol, used for request to real device (TCP/IP, RTU or ASCII).
- Address of outbound transport from the list of configured outbound transports in the subsystem “Transports” of OpenSCADA.
- ModBus destination node. In the case of protocols RTU and ASCII – this is the unique address of the physical device, and when TCP/IP – the identifier of the unity.
- Combining fragments of registers. Standard functions 01–04 let to request at once multiple adjacent registers or bits. This strategy often allows to optimize the traffic. However, the required registers are not always located adjacent to each other, this option allows you to collect them in blocks of up to 100 registers, or 1600 bits. The installing of this parameter must be approached with caution, since not all devices support access to registers between fragments.
- Connection timeout in milliseconds. Specifies the time interval during which the answer is expected. If there is zero waiting time by default the transport waiting time is used.
- Time of connection recovery in seconds. Specifies the time interval after which the re-attempt of the request to previously inaccessible device is done.
- Attempts of request for the protocols RTU and ASCII. Indicates the number of attempts by the repetition of the request in case of incomplete or damaged answer.

3.2. Parameters

Module of data acquisition provides only one type of parameters - "Standard". An additional configuration parameter field of the module(Fig.9) is a list of processed attributes(registers ModBus). Attribute in the list is written as follows: **<dt>:<numb>:<wr>:<id>:<name>**.

Where:

dt - Type of ModBus data (R-register, C-bit, RI- input register, CI-input bit). R and RI can expanded by suffixes: i2-Int16, i4-Int32, f-Float, b5-Bit5.

numb - number of register or bit of ModBus device (decimal, octal or hexadecimal);

wr - read-write mode (r-read, w-write, rw-read and write);

id - ID of the attribute OpenSCADA;

name - name of the attribute OpenSCADA.

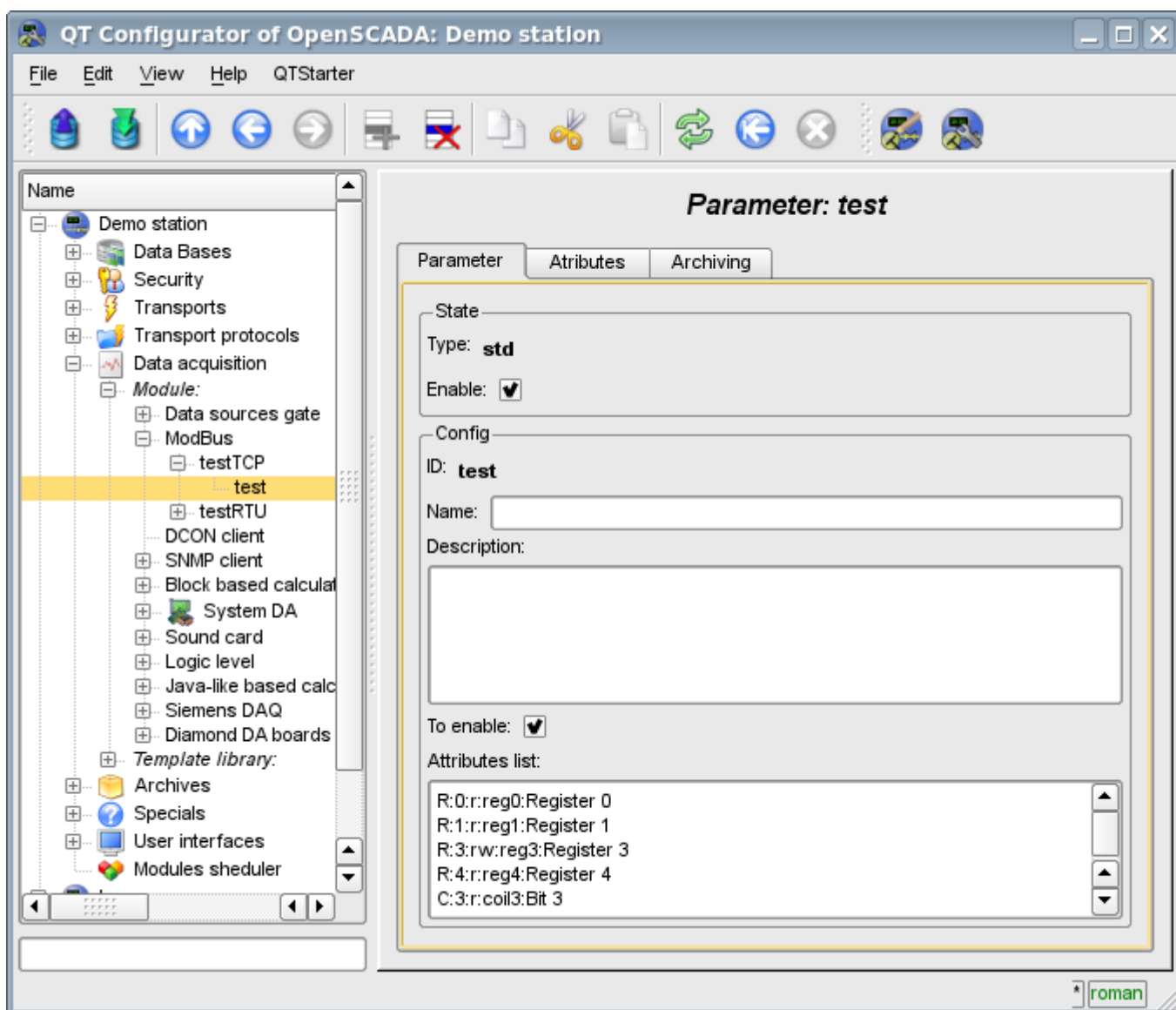


Fig.9. Configuration tab of the parameter.

In accordance with a specified list of attributes interrogation and the creation of the attributes of the parameter is carried out(Figure 10).

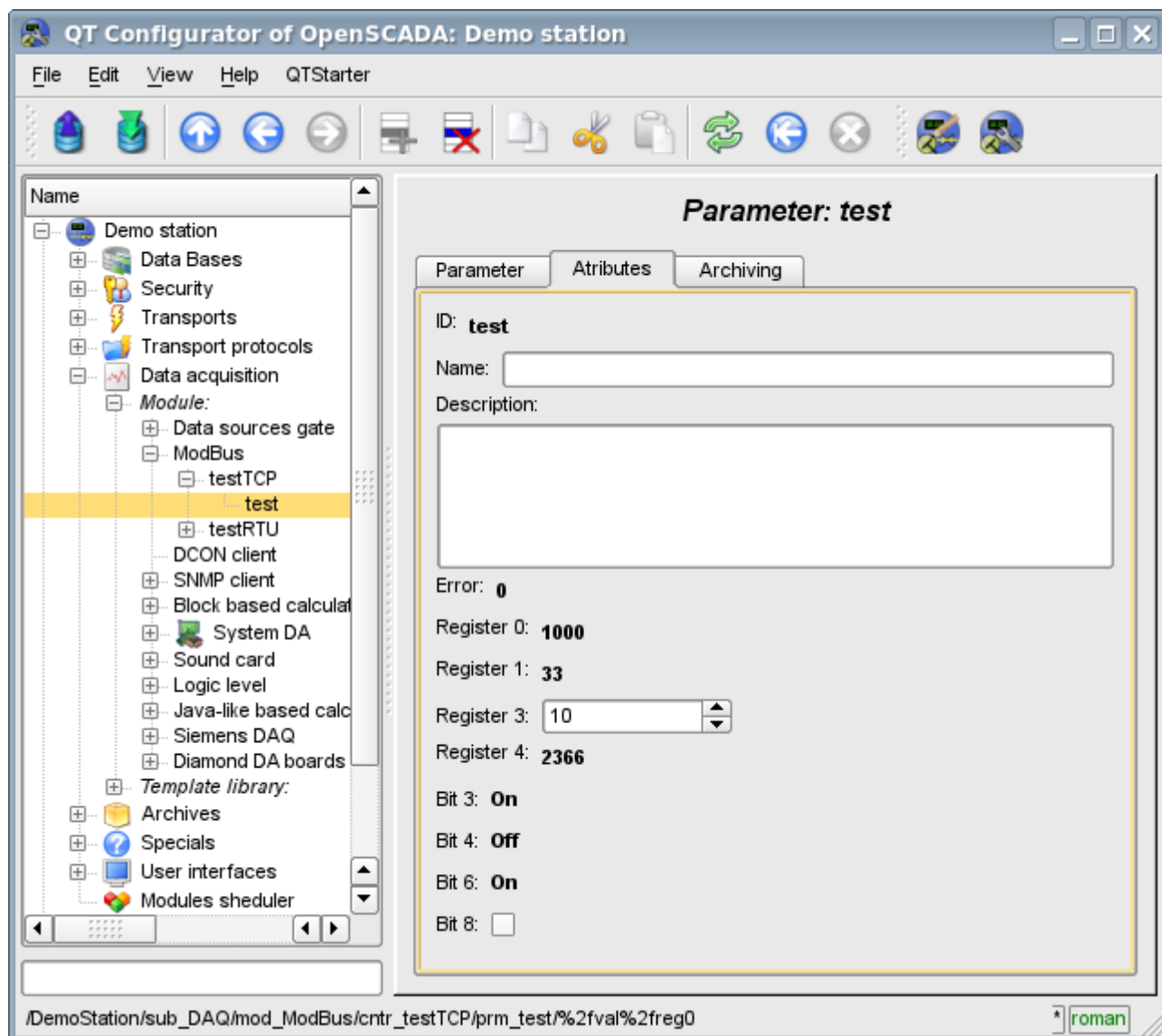


Fig.10. Tab of the attributes of the parameter.

The module of subsystem “Data acquisition”<DCON>

<i>Module:</i>	DCON
<i>Name:</i>	DCON client
<i>Type:</i>	DAQ
<i>Source:</i>	daq_DCON.so
<i>Version:</i>	0.3.3
<i>Author:</i>	Almaz Kharimov
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides an implementation of DCON-client protocol. Supports I-7000 DCON protocol.
<i>License:</i>	GPL

DCON – the protocol of controllers' family ADAM(<http://www.advantech.com/>, <http://ipc2u.ru/>), ICP DAS(<http://www.icpdas.com/>, <http://ipc2u.ru/>), RealLab(<http://www.RLDA.ru/>) and the like ones. It uses serial lines RS-485 to transfer data.

This module provides the ability of input/output of information from various devices on the protocol DCON. Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level.

1. General description of the protocol DCON

DCON protocol requires one lead(requesting) device in the line (master), which can send commands to one or more driven devices (slave), referring to them by a unique address in the line. Syntax of the commands of the protocol allows the address 255 devices at one line of standard RS-485.

Initiative to exchange always comes from the leading device. Slave devices listen the line. Master request (package, the sequence of bytes) in the line and turns into a listening the line. Slave device responds to the request, which came to him.

2. Module

This module provides the ability of clear interrogation and record of input-output ports of devices that are compatible with ICP DAS I-7000. On the settings tabs of DCON module the necessary settings are inserted, and on the attributes tabs the corresponding to the given parameters variables of input-output appear.

2.1. Data controller

For addition of the DCON data source the controller is created and configured in the system OpenSCADA. Example of the configuration tab of the controller of the type is depicted in Figure 1.

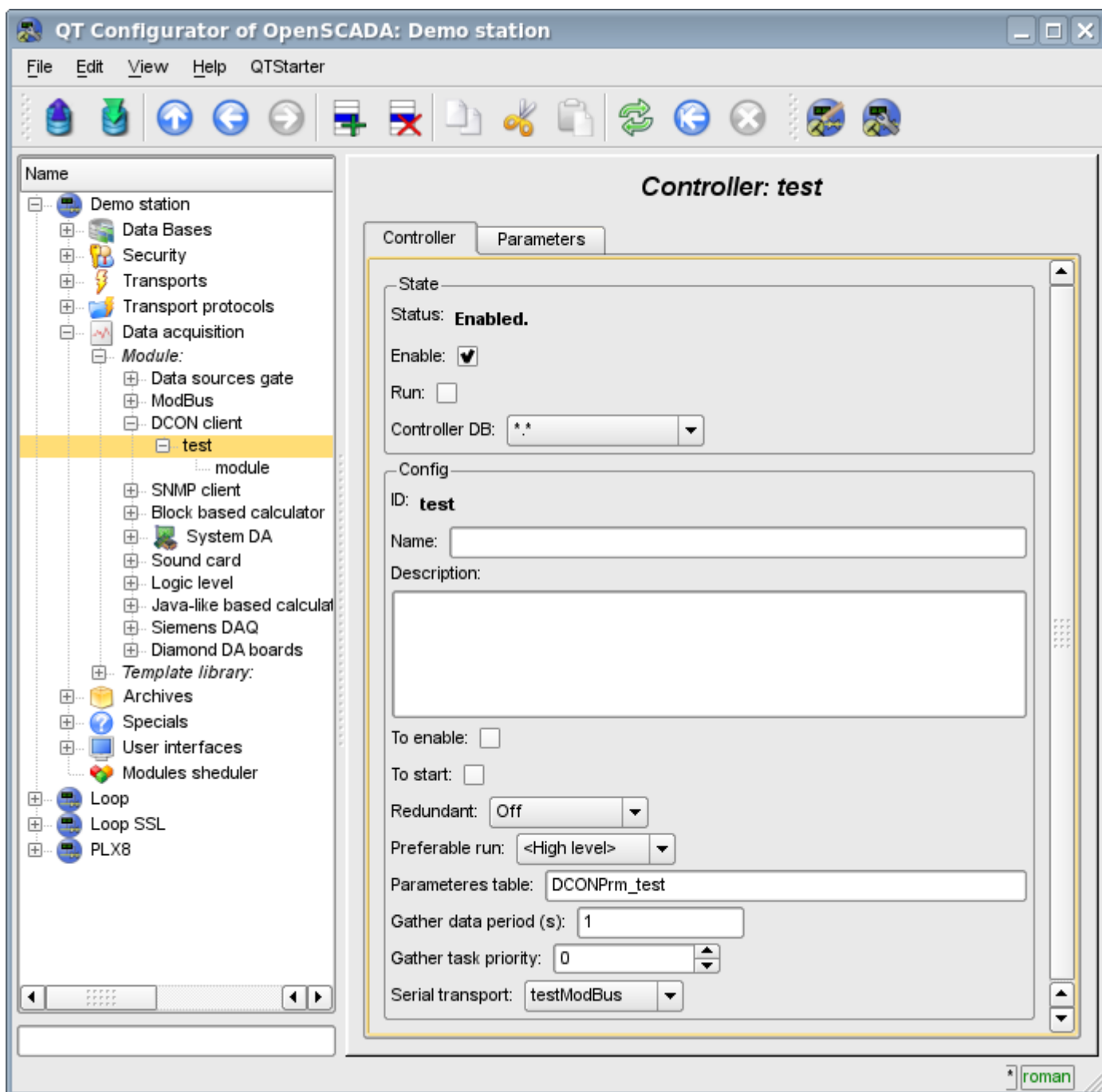


Fig.1. Configuration tab of the controller.

From this tab you can set:

- The state of the controller, as follows: Status, «Enable», «Run» and the name of the database containing the configuration.

- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».
- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the configuration of the parameters of the controller.
- The period and the priority of the task of data acquisition.
- Name of the outgoing transport of serial interface configured in the module of transport “Serial”.

2.2. Parameters

Module *DCON* provides only one type of parameters – “Standard”. On the parameters tab you can set:

- The state of the parameter "Enable": requires disabling-enabling for the changes on this tab take effect.
- Id, name and description of the parameter.
- The state, in which the parameter must be translated at boot: "To enable".
- Type of the input-output module I-7000.
- Address of I-7000 device in the network RS-485. In decimal form from 0 to 255.
- Flag of checking the checksum. Must correspond to the one of the input/output module I-7000.

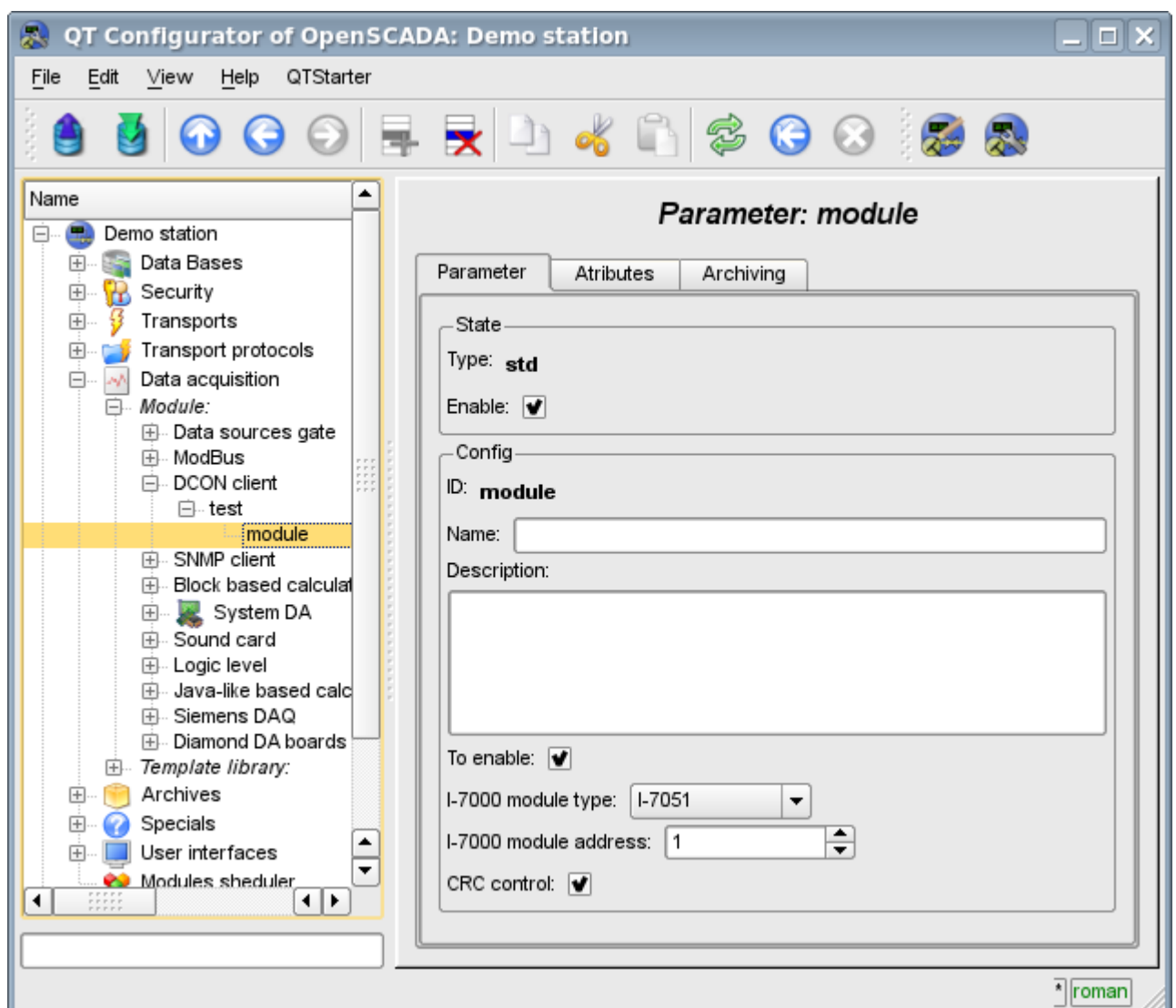


Fig.2. Configuration tab of the parameter.

In accordance with the settings of the parameter and the interrogation and creation of the attributes is carried out(Fig. 3).

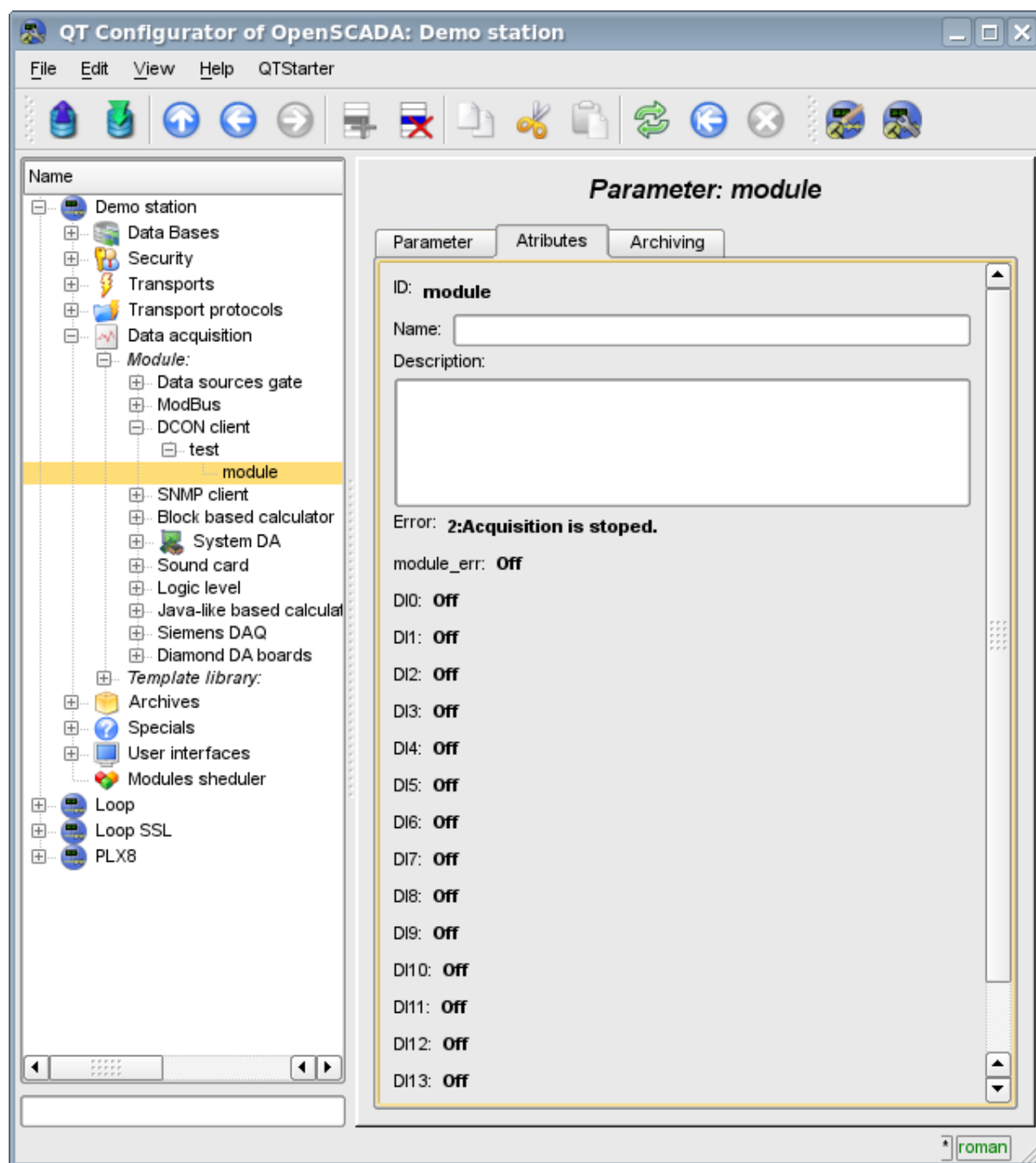


Fig.3. Tab of the attributes of the parameter.

3. Compatibility table of input/output modules of different manufacturers

№№	Module	ICP DAS	ADAM	RealLab
1	I-7051	I-7051, I-7053*	ADAM-4051*, ADAM-4053*	NL-16DI*, NL-16HV*
2	I-7045	I-7045, I-7043*		NL-16DO*
3	I-7063	I-7063		
4	I-7017	I-7017, I-7018*, I-7019*, I-7005*	ADAM-4017*, ADAM-4018*, ADAM-4019*	NL-8TI*, NL-8AI*
5	I-7024	I-7024	ADAM-4024	NL-4AO

* – not tested.

The module of subsystem “Data acquisition” <ICP_DAS>

<i>Module:</i>	ICP_DAS
<i>Name:</i>	ICP_DAS equipment
<i>Type:</i>	DAQ
<i>Source:</i>	daq_ICP_DAS.so
<i>Version:</i>	0.7.2
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides support for ICP DAS hardware. The support of I-87000 and I-7000 DCON modules and I-8000 fast modules is included.
<i>License:</i>	GPL

The module provides the OpenSCADA system with the support of various equipment of ICP DAS company (<http://www.icpdas.com/>, <http://ipc2u.ru/>) through the API library of company *libi8k.a*. Most of the equipment of the ICP DAS company is working under the DCON protocol, but some new equipment such as I-8000 Series operates on a parallel bus, while another part is set into the parallel bus slots of I-8000 which are available under the serial interface and DCON protocol, they are not addressed directly and require call of the specialized command of the slot selection. Access to equipment that uses direct requests under the DCON protocol, can be implemented by the module *DAQ.DCON*. Support for the rest of the equipment is not added to the module *DAQ.DCON*, but it was implemented in this module due to the availability of API library of the ICP_DAS company only for the x86_32 platform, which brings restrictions on access to the equipment of the ICP DAS company and other equipment under the DCON protocol on the other hardware platforms.

The reason for creating this module was the works with the controller LP-8781 of LinPAC series of ICP_DAS company with the purpose to implement runtime PLC based on the OpenSCADA system.

API library of the ICP_DAS company (*libi8k.a*) is available with source code of the module and does not require separate installation.

1. Data controller

To add the ICP DAS data source the controller is created and configured in the OpenSCADA system. Example of the configuration tab of the controller of this type is shown in Figure 1.

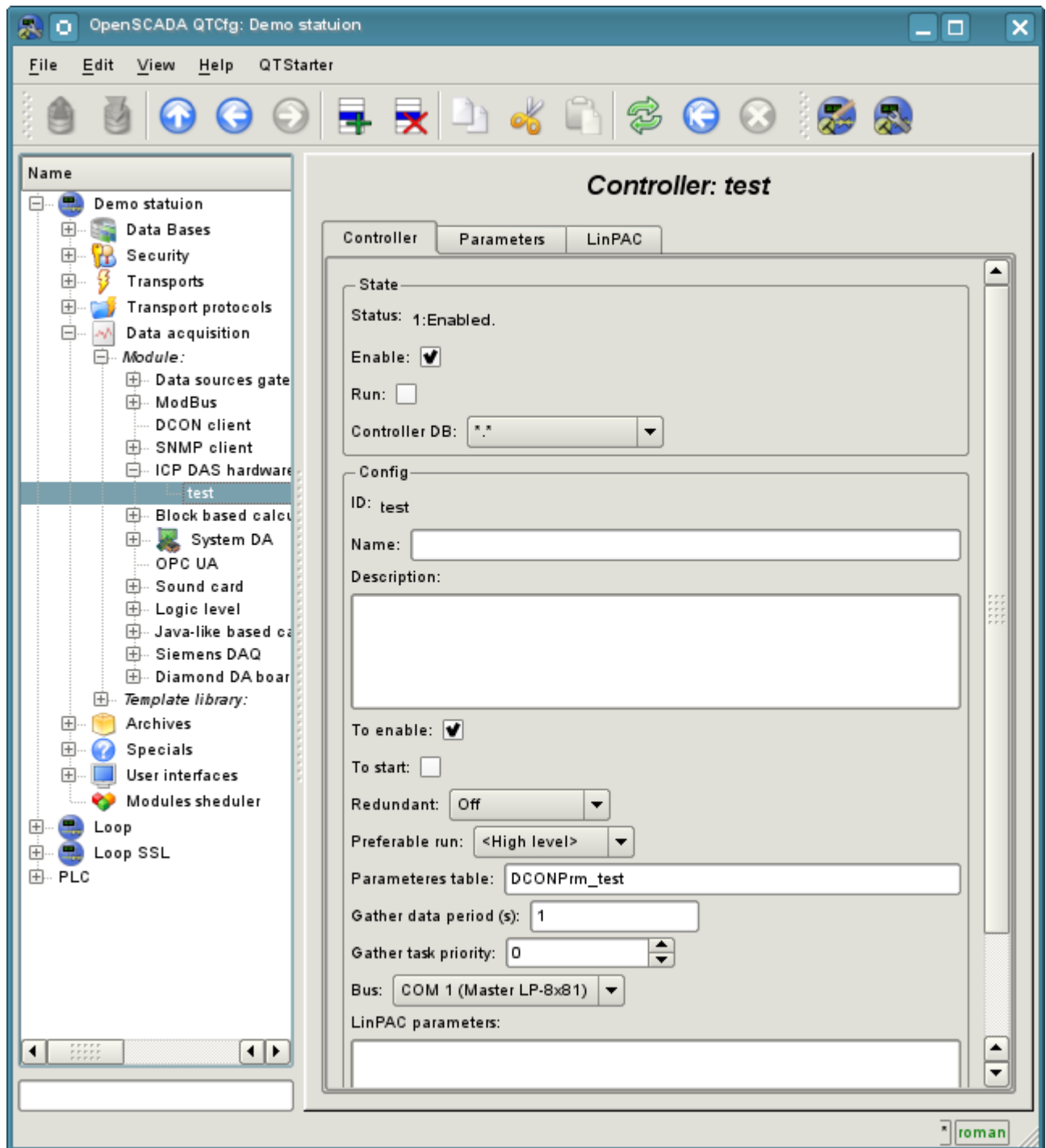


Fig.1. Configuration tab of the controller.

From this tab you can set:

- State of the controller, namely: the status, "Enable" and "Run" and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».
- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the configuration of the parameters of the controller.
- The period and the priority of the task of data acquisition.
- Bus, on which the modules are placed. If you specify a serial interface (COMx), then access is

made under the protocol DCON. If the main controller bus is LP-8x81 the access is made through the parallel bus API or mixed.

- Parameters LinPAC. Wrapped to XML generic parameters of PLC family LinPAC. In generic cases this field don't edited manual and edited into.
- Data transfer rate for the serial interface. It is indicated for the not main bus.
- Serial request's tries.

2. Parameters

Module provides only one type of parameters - "Standard". On the parameters tab you can set:

- The state of the parameter, namely the type and the status "Enable"
- Id, name and description of the parameter.
- The state, in which the parameter must be translated at boot: "To enable".
- Type of the input-output module.
- Address of the I/O module, in the case of work not on the main bus - in the decimal value from 0 to 255.
- Slot of the module in the case of work with a series of devices I-8000.
- More options of the module. It is used not by all the modules and contains the text in XML. Not intended for manual editing, and is formed on the Configuration tab, which is usually specific to the each type of modules.

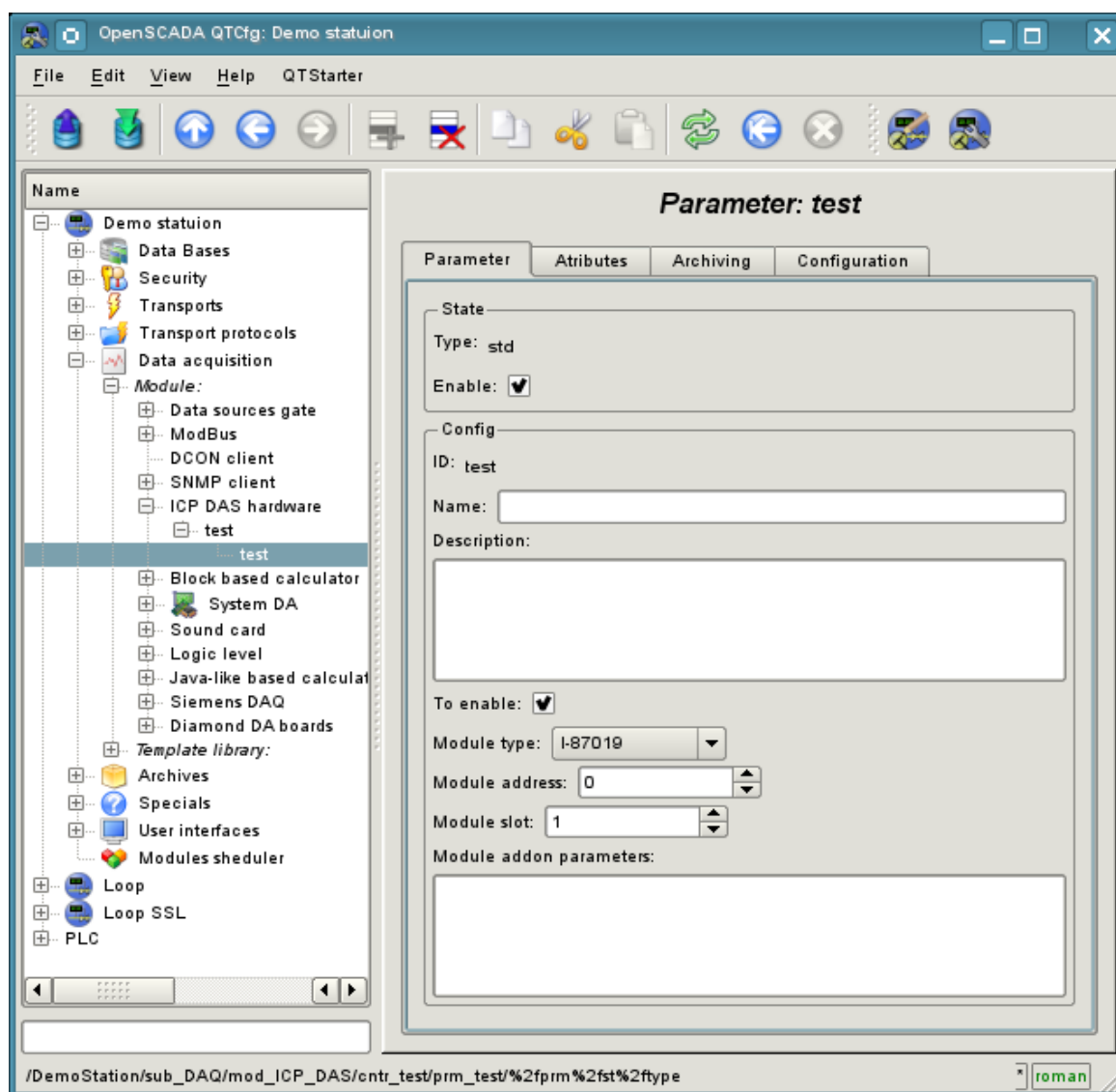


Fig.2. Configuration tab of the parameter.

In accordance with the parameter settings the poll and the creation of attributes is made (Fig. 3).

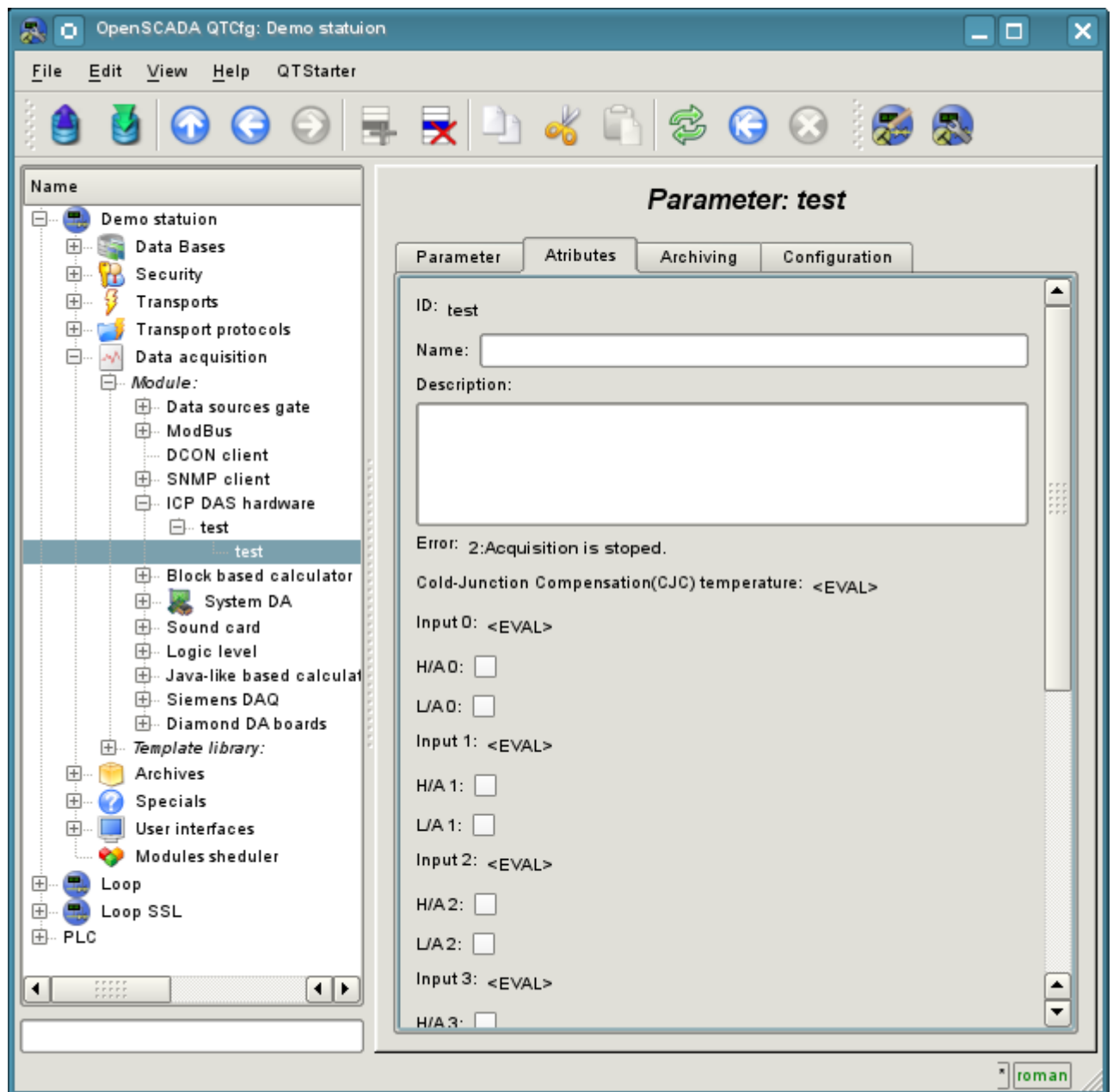


Fig.3. Tab of the attributes of the parameter.

2.1 Module I-8017

Fast analog input module that runs on a parallel bus. Provides speed access to data on one channel at 130 kHz. However, because of the pledged hardware limitations it does not allow to reach speed over 33 kHz per channel when scanning multiple channels. Data expectation is in the blind cycle, which leads to great losses of the CPU at high frequencies of the acquisition.

Module provides eight analog input attributes $i\{x\}$ and eight signs of violation of the upper $ha\{x\}$ and the lower $la\{x\}$ boundaries. Also the configuration tab is available with advanced configuration:

- *Number of processed parameters* - indicates how many inputs to process. It is characteristic for the mode of fast data acquisition and used to limit the number of processed channels, commensurate with used resources of the CPU.
- *Frequency of the fast data acquisition (seconds)* - indicates how often to carry out fast data acquisition for the number of channels listed above. Fast mode of data acquisition is turned off by indicating zero period.
- Modes of the gain for each input define a the following gains: +1.25V, +2.5V, +5V, +10V and +20mA.

2.2 Module I-8042

Fast digital input/output module works on a parallel bus. Provides 16 attributes for input $i\{x\}$ and 16 for the output $o\{x\}$.

2.3 Module I-87019

The module of the analog input for the eight channels works on the serial bus and accessible under the DCON protocol. Provides eight analog input attributes $i\{x\}$ and eight signs of violation of the upper $ha\{x\}$ and the lower $la\{x\}$ boundaries. The module provides temperature measurement of cold junctions of thermocouples.

Module provides the tab "Configuration" with the advanced configuration of modes of inputs: $\pm 15\text{mV}$, $\pm 50\text{mV}$, $\pm 100\text{mV}$, $\pm 150\text{mV}$, $\pm 500\text{mV}$, $\pm 1\text{V}$, $\pm 2.5\text{V}$, $\pm 5\text{V}$, $\pm 10\text{V}$ $\pm 20\text{mA}$, J type, K type, T type, E type, R type, S type, B type, N type, C type, L type, M type, L type (DIN43710C).

2.4 Module I-87024

Analog output module for the four channels working on the serial bus and accessible under DCON protocol. Provides four analog output attributes $o\{x\}$.

In addition it include tab "Configuration" with configuration host watchdog and output values wich set at enable and reset by watchdog.

2.5 Module I-87057

Digital output at 16 channels working on the serial bus and accessible under DCON protocol. Provides 16 diskret outputs $o\{x\}$.

In addition it include tab "Configuration" with configuration host watchdog and output values wich set at enable and reset by watchdog.

3. LP-8x81 series controllers configuration

For common properties the controllers series LP-8x81 configuration allowed accordingly tab on module's page, where you can get information about controller's serial number, SDK version and DIP-switch value, and also set value for controller's watchdog timer. The watchdog timer is disabled by set it to zero value. Watchdog timer's value updated into controller's task and with it period. The acquisition task hang consequently follow controller's restart!

Links

Special modules for Linux kernel 2.6.29 for controllers LP-8x81: [lp8x81_2629.tgz](#)

The driver from VIA for controllers LP-8x81 network: [rhinefet20070212111037.tgz](#)

On standard Linux network driver the speed is dropped significant after days work

The patch for build network driver for Linux 2.6.29: [build_2.6.29.patch](#)

The module of subsystem “Data acquisition” <DAQGate>

<i>Module:</i>	DAQGate
<i>Name:</i>	Gateway of the data sources
<i>Type:</i>	DAQ
<i>Source:</i>	daq_DAQGate.so
<i>Version:</i>	0.9.1
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Allows you to perform the locking of the data sources of the remote OpenSCADA stations in the local ones.
<i>License:</i>	GPL

The main function of this module is the reflection of the data of the “Data acquisition” subsystem of the remote OpenSCADA stations on the local ones. In its work, the module uses the self protocol of the OpenSCADA system ([Self System](#)) and service functions of the subsystem “Data acquisition”.

Module realizes the following functions:

- The reflection of the structure of the parameters of the subsystem “Data acquisition” of the remote station. The structure is periodically synchronized while working.
- Access to the configuration of the parameters. Configuration of the parameters of the controllers of remote stations is transparently reflected that lets you to change it remotely.
- Access to the current value of the attributes of the parameters and the possibility of their modification. The values of the attributes of the parameters are updated at a frequency of execution of the local controller. Requests for modification of the attributes are transmitted to the remote station.
- Reflection of the archives of values of individual attributes parameters. The reflection of the archives is realized in two ways. The first method includes creating of the local archive for the attribute and its synchronization with the remote, the restoration of the archive at the stop of the station is provided. The second method is the translation of the requests of the local archive file to the one of the remote station.
- Provides the implementation of the mechanism of the vertical redundancy as an opportunity to reflect data from the multiple stations at the same level.
- Realization of the functions of horizontal redundancy, namely, working in the conjunction with the remote station of the same level.

Using of the available redundancy schemes is graphically represented in Figure 1.

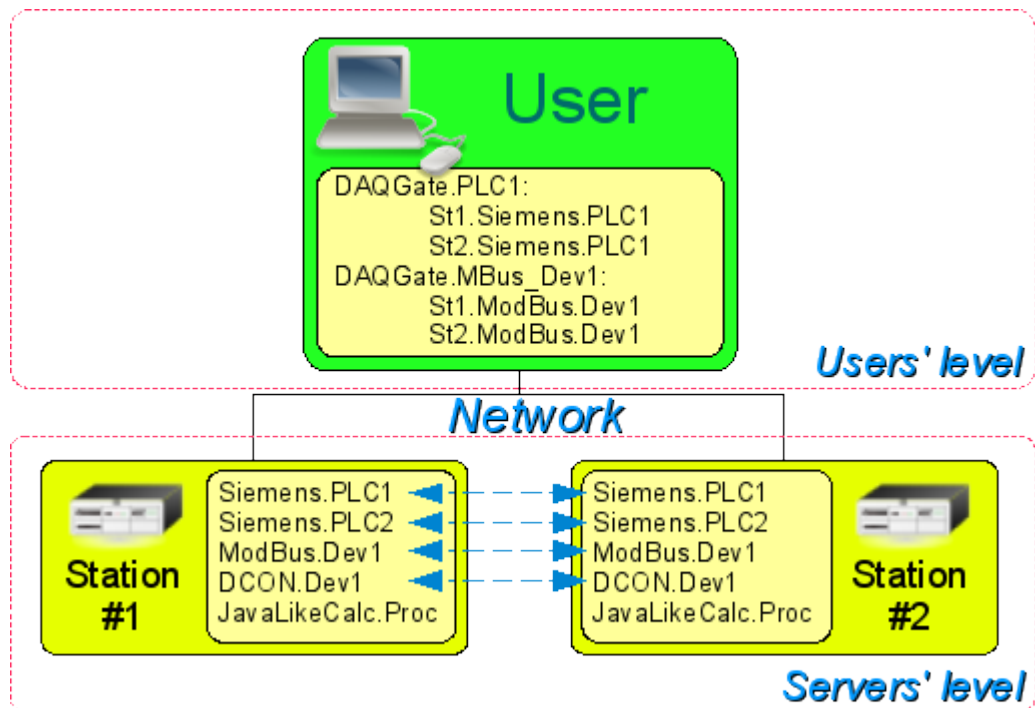


Fig.1. Horizontal and vertical redundancy.

1. Controller of data

For addition of the data source the controller is created and configured in the system OpenSCADA. Example of the configuration tab of the controller is depicted in Figure 2.

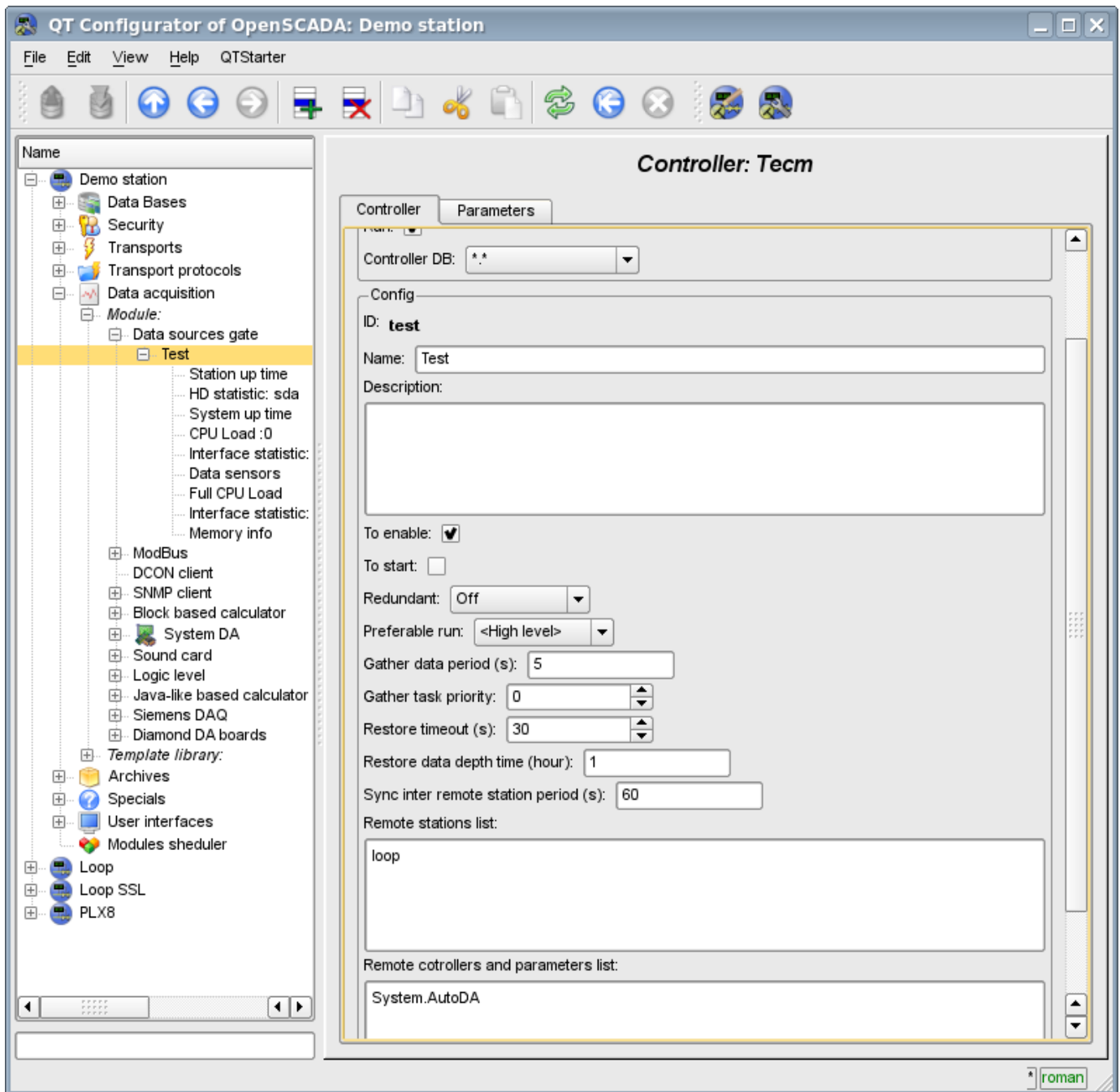


Fig.2. Configuration tab of the controller.

From this tab you can set:

- The state of the controller, as follows: «Enable», «Run» and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».
- Horizontal mode for redundancy and performance mode of the controller.
- The period in seconds, and the priority of the date acquisition task.
- Recurrence interval of time of the attempting to restore a lost connection with the station in seconds.
- Maximum depth of data of the archive to restore when start in the hours.
- The period of synchronization with a remote station in seconds.

- List of the reflected remote stations. Several stations in the list include a mechanism of vertical redundancy.
- The list of the reflected controllers and parameters. The list can be used as for controllers for the reflection of all their parameters, and for individual parameters too.
- The commands to go to the configuration of remote stations.

2. Parameters

The module does not provide the possibility of setting up the parameters manually, all parameters are created automatically, taking into account the list of reflected controllers and parameters. Example of the reflected parameter is shown in Fig. 3.

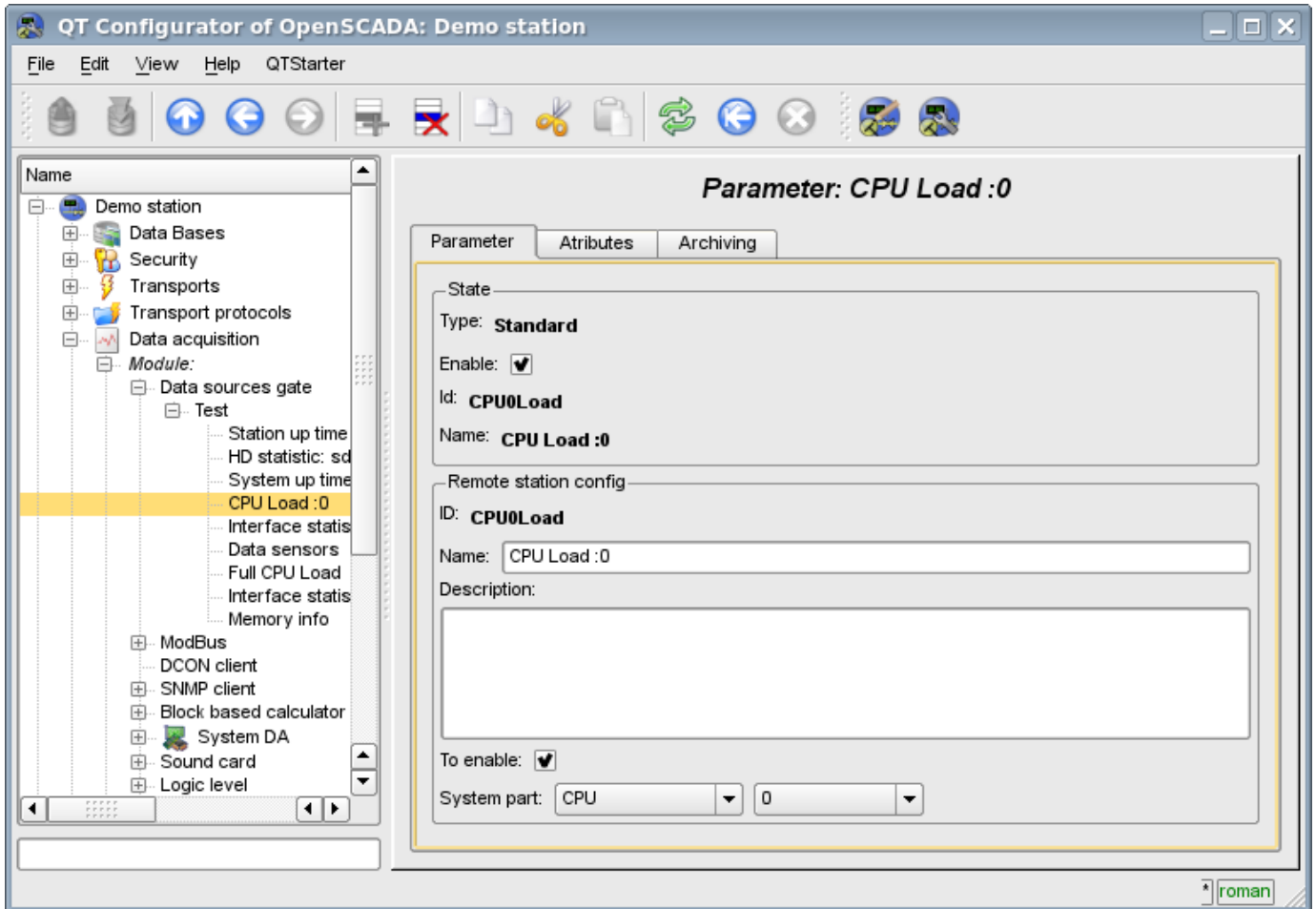


Fig.3. Configuration tab of the reflected parameter.

The module of subsystem “Data acquisition”<SoundCard>

<i>Module:</i>	SoundCard
<i>Name:</i>	Sound card
<i>Type:</i>	DAQ
<i>Source:</i>	daq_SoundCard.so
<i>Version:</i>	0.6.1
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides an access to the sound card.
<i>License:</i>	GPL

This module is designed to provide data from the inputs of sound cards of the system. The module is based on the multi-platform library of work with sound PortAudio (<http://www.portaudio.com>). The feature of this library is the unified API, which allows you to easily adapt this module to work on different platforms and even different audio subsystems on a single platform.

Structure of the module is the reflection of the object “Controller” of subsystem “Data acquisition” on a separate audio input device available in the system. The object “Parameter” of the subsystem “Data acquisition” reflects a separate channel available from the sound input device to the attribute “val”. The most functional is to use the attribute “val” in conjunction with the archive, or at least with its buffer. In the case of the archiving enabling data of the channel of audio input are placed in the buffer of the archive by the packages with the frequency of data fetch of input device that allows you to perform further operations on that data. In addition, the last package value is installed as the current value of the attribute. In the case of archive absence operation of the last package value placing as the current value of the attribute is performed only.

Also, the module implements the functions of the horizontal reservation, namely, working in conjunction with the remote station of the same level.

1. Controller of the data

To add an audio input device the controller is created and configured in the system OpenSCADA. Example of the configuration tab of the controller is depicted in Figure 1.

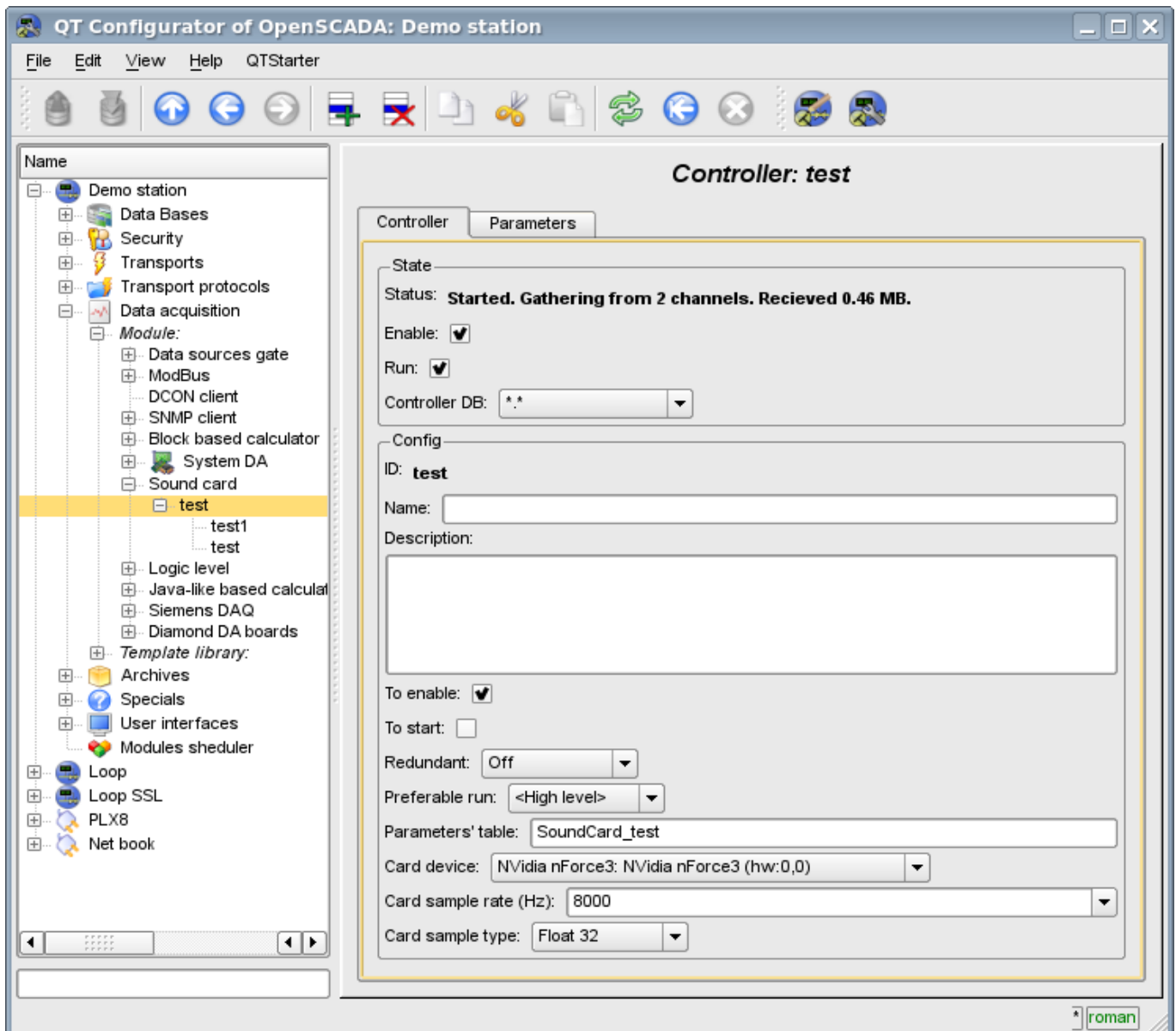


Fig.1. Configuration tab of the controller.

From this tab you can set:

- The state of the controller, as follows: Status, «Enable», «Run» and the name of the database containing the configuration.
- Id, name and description of the controller.
- The state, in which the controller must be translated at boot: «To enable» and «To start».
- Horizontal mode of redundancy and performance preference of the controller.
- Name of table to store the configuration of the parameters of the controller.
- Card device from the list of available ones.
- Frequency of the fetch of values of cards in hertz.
- Type of values of fetch from the list: Real 32, Integer 32 and Integer 16.

2. Parameters

To add a channel of input sound device the parameter of controller is created and configured in the system OpenSCADA. Example of the configuration tab of the parameter is depicted in Figure 2.

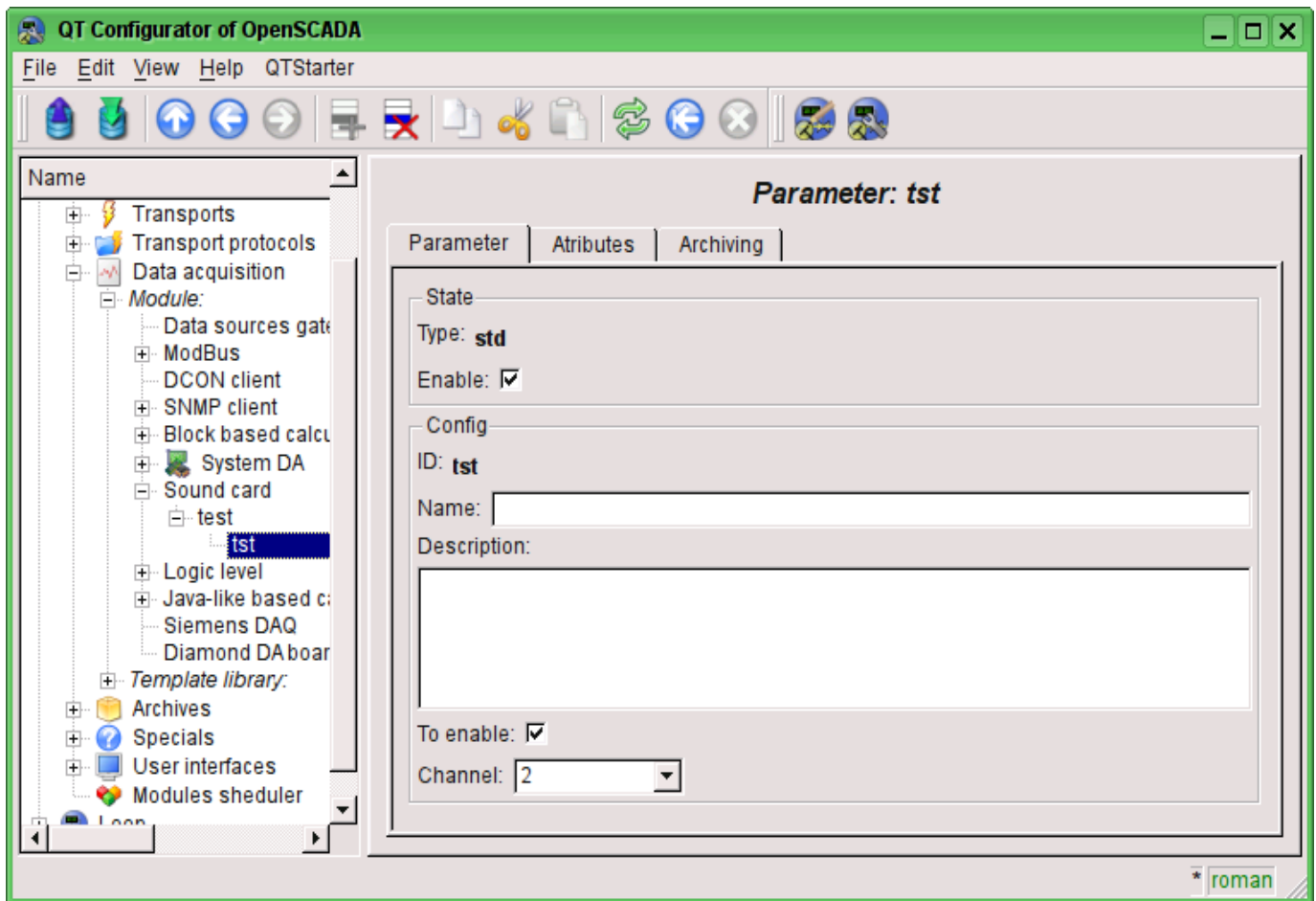


Fig.2. Configuration tab of the parameter.

From this tab you can set:

- Type of the parameter and indicate the status "Enable".
- Id, name and description of the parameter.
- The state, in which the parameter must be translated at boot: «To enable».
- Channel of the audio input device from the list of available channels.

Tab of attributes of the parameter has the form presented in Figure 3, the tab of the values of the attribute's archive "val" is presented in Fig.4.

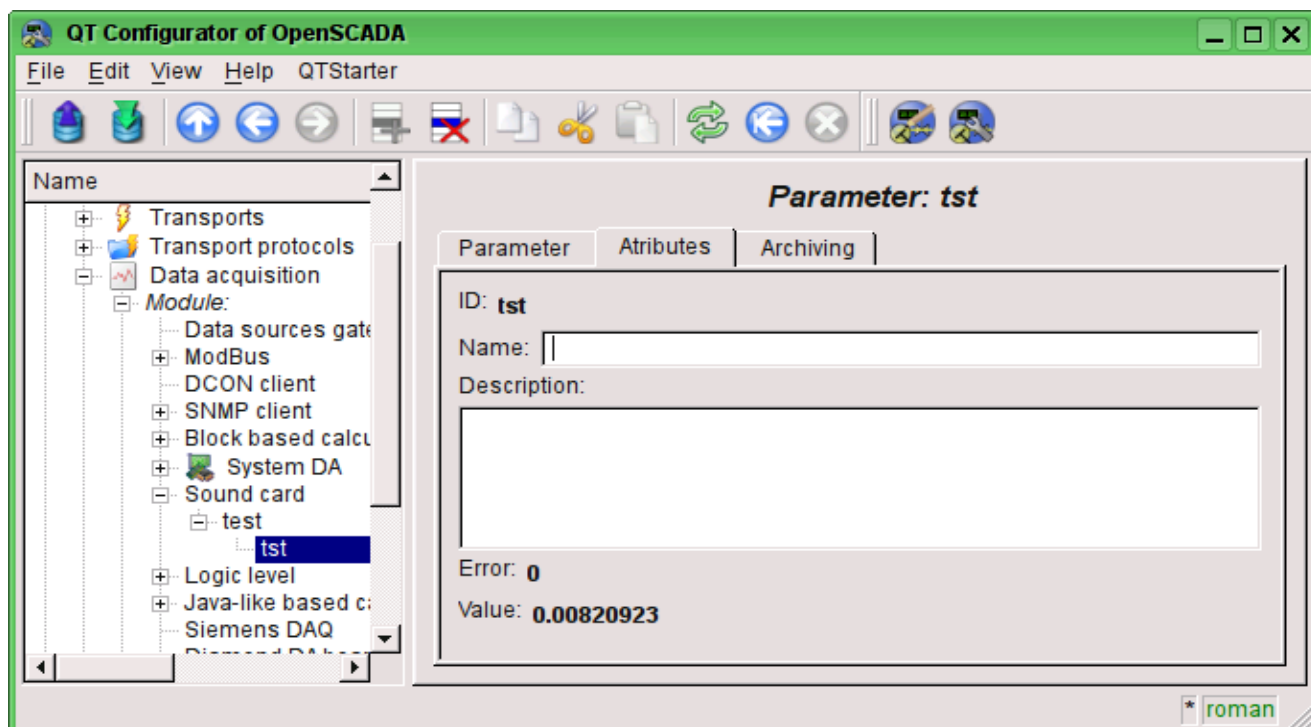


Fig.3. Tab of the attributes of the parameter.

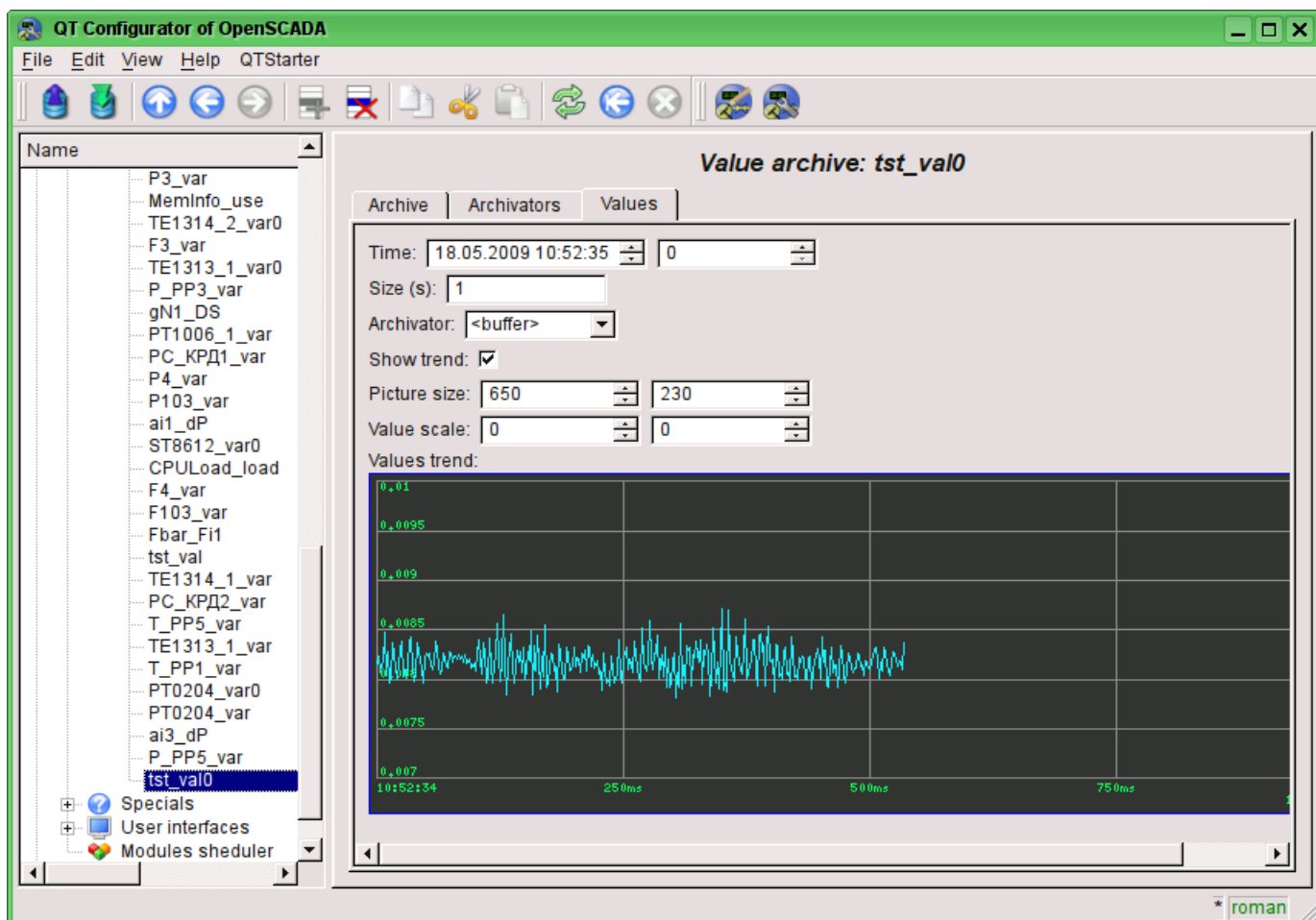


Fig.4. Tab of the values of the archive of the attribute "val".

The <OPC-UA> module of “Data acquisition” and “Transport protocols” subsystems

Parameter	Module 1	Module 2
<i>ID:</i>	OPC-UA	OPC-UA
<i>Name:</i>	OPC-UA	OPC-UA
<i>Type:</i>	DAQ	Protocol
<i>Source:</i>	daq_daq OPC-UA.so	
<i>Version:</i>	0.6.0	0.6.0
<i>Author:</i>	Roman Savochenko	
<i>Translated:</i>	Maxim Lysenko	
<i>Description:</i>	Provides the implementation of client service of OPC UA.	Provides the implementation of the OPC UA protocol.
<i>License:</i>	GPL	

OPC (OLE for Process Control) - it is the family of protocols and technologies that provide the single interface to control the objects of automation and technological processes. The creating and support of specifications of OPC coordinates an international nonprofit organization OPC Foundation, established in 1994 by the leading manufacturers of industrial automation.

In view of the fact that a significant influence in the OPC Foundation organization has the Microsoft corporation, OPC protocols, until recently, was single platform and closed, due to binding to the closed technologies of MS Windows. However, more recently, the OPC Foundation organization has created multi-platform interfaces such as OPC XML DA and OPC UA. The most interesting of them is the OPC UA interface, as unifying all the earlier interfaces in an open and multi-platform technology.

This module implements the interface and protocol support for OPC UA in the form of client service, and as the OPC UA server. Client service of OPC UA is implemented by the same name module of the subsystem "Data acquisition", and the server is implemented by the subsystem's "Protocols" module.

In the current version of these modules it is implemented the binary part of the protocol and basic services in unsafe mode and safe mode of policies "Base128Rsa15" и "Base256". Later it is planned to extend the module to work via HTTP/SOAP and implementation of other OPC UA services.

Although the OPC UA protocol is multi-platform, its specification and SDK are not freely available, but are provided only to members of the OPC Foundation organization. For this reason, the implementation of these modules has faced significant obstacles and problems.

First, the protocol OPC UA is complex and its realization in general without specification an extremely laborious. For this reason, the work on these modules for a long time was not started, and only thanks to sponsorship by an organization-member of OPC Foundation the OpenSCADA project received documentation of the specification. The SDK and source code ANSIC-API of the OPC-UA protocol have not been received due to their incompatibility with the GPL license and as a consequence, the potential threat of violation of the license when working with source code, which could lead to subsequent legal problems with the free distribution of these modules.

Secondly, even the presence of specification does not allow to solve some technical question without an example of implementation and the possibility of test the working prototype of the client and server of OPC UA. For example, it is the technical features of the implementation of symmetric encryption algorithms and the keys for them do not allowed to make the implementation of support for security policy at once.

To debug the operation of modules the demonstration software of company Unified Automation consisting of the OPC UA client - UA Expert and Server - OPC UA Demo Server, from SDK package, was used.

1. OPC UA protocol

OPC UA - is the platform-independent standard by the means of which the systems and devices of various types can interact by sending messages between the client and the server through various types of networks. The protocol supports secure communication through the validation of clients and servers, as well as the counteraction to attacks. OPC UA defines the concept *Services* that the servers can provide, as well as services that the server supports for the client. Information is transmitted as the data types defined by OPC UA and producer, in addition the servers define object model, for which the clients can implement the dynamic review.

OPC UA provides the combination of integrated address space with service model. This allows the server to integrate data alarms and events, the history in this address space, as well as provide access to them through integrated services. Services also provide an integrated security model.

OPC UA allows servers to provide for clients the definitions of types for access to the objects of the address space. OPC UA supports the provision of data in various formats, including binary structures and XML-documents. Through the address space clients can request the server metadata that describe the data format.

OPC UA adds the support for multiple connections between nodes instead of a simple hierarchy. Such flexibility in combination with types' definition allows to use OPC UA for solving problems in the wide problem area.

OPC-UA is designed to provide the reliable output of data. The main feature of all OPC servers - the ability to issue the data and events.

OPC-UA is designed to support the wide range of servers, from simple PLC to industrial servers. These servers are characterized by the wide range of sizes, performance, platforms and functional capacity. Consequently, the OPC UA defines the comprehensive set of possibilities, and the server can implement the subset of these possibilities. To ensure the interoperability between OPC UA defines the subsets, named the *Profiles* that the server can indicate for agreement Clients may subsequently make the review of server's profiles and make the interaction with the server, based on the profiles.

OPC UA specification is designed as the core in the layer, isolated from the underlying computer technologies and network transports. This allows OPC UA if necessary to expand on the future technologies without exclusion the framework of design. Currently, the specification defines two ways to data encode: XML/text and UA Binary. In addition, the two types of transport layer are defined: TCP and HTTP/SOAP.

OPC UA is designed as the solution for migration from OPC clients and servers, which are based on Microsoft COM technologies. OPC COM servers (DA, HDA and A&E) can be easily reflected in the OPC UA. Producers can independently make such migration or recommend users to use wrappers and converters between these protocols. OPC UA unifies the previous models in the single address space with the single set of services.

2. The module of the protocol implementation

Protocol module contains the implementation code for the protocol part of the OPC UA for both the client and for server services. To construct the OPC UA server it is enough to create an incoming transport, for ordinarily this TCP-transport of module Sockets, and select in it the module of the protocol, and also configure although one endpoint node of protocol module, about it bellow.

2.1. Service the requests on the OPC UA protocol

Incoming requests to the module-protocol are processed by the module in accordance with configured end points of OPC UA (EndPoints) (Fig. 1).

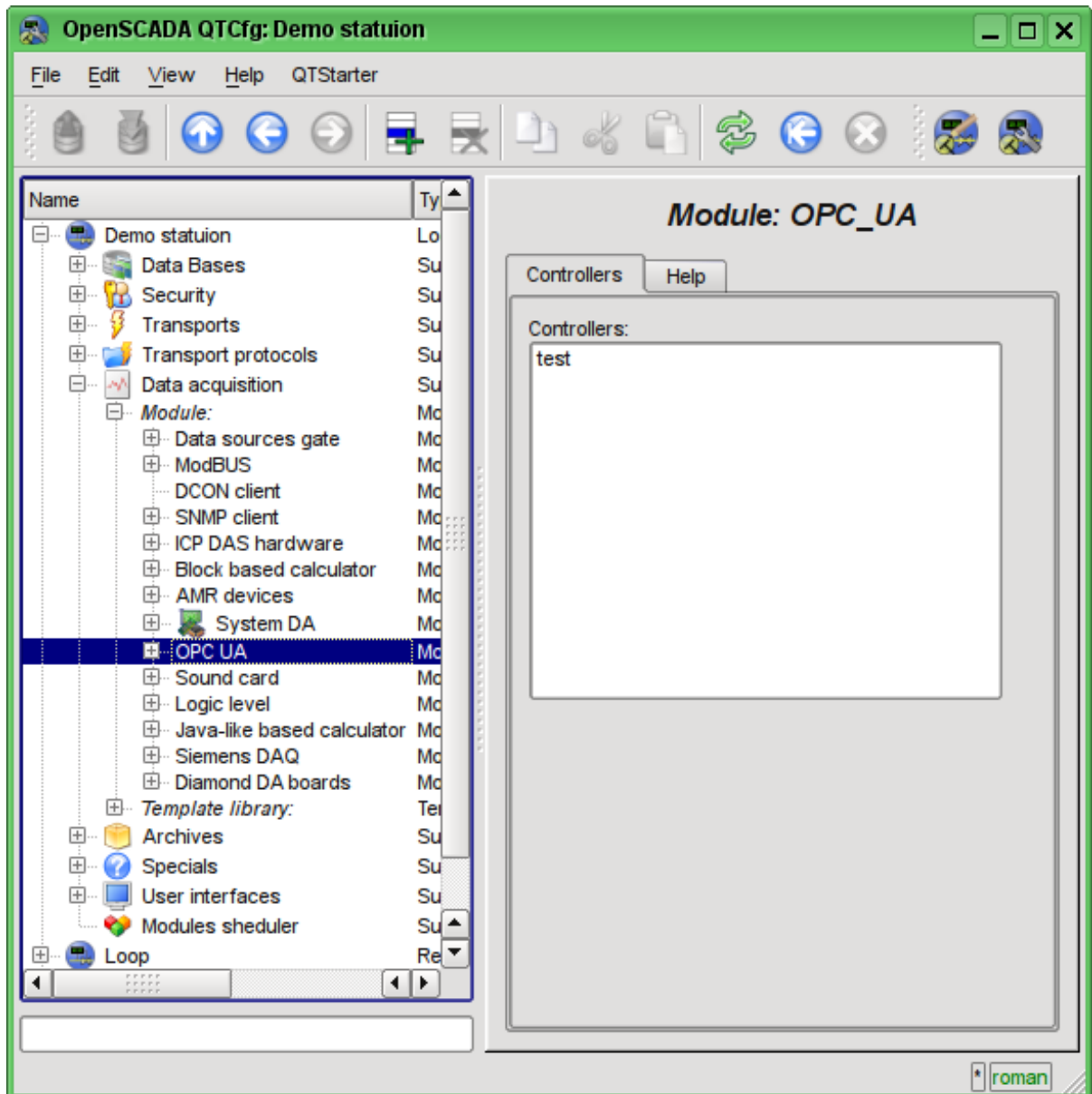


Fig.1. End points of the protocol.

Endpoint of the OPC UA protocol is actually the server object of OPC UA. End points in OPC UA can be either local or remote. The local one is designed to provide the resources of OpenSCADA station to protocol OPC UA, while the remote end points are both for the service and review of available OPC-UA units, and for locking requests to remote stations. In this version of the module is only supported the configuration of the local endpoints.

The general configuration of the endpoint is made on the main tab of the endpoint page (Fig. 2) with the parameters:

- Node status, namely: the state, "Enable" and the name of the database containing the configuration.
- ID, name and description of the node.
- The state, in which to transfer the node at start: "Enable".
- Type of the protocol coding. At the moment it is only "binary".
- URL of the end point.
- The server certificate in the PEM format.
- The private key in the PEM format.
- Security Server Policy.

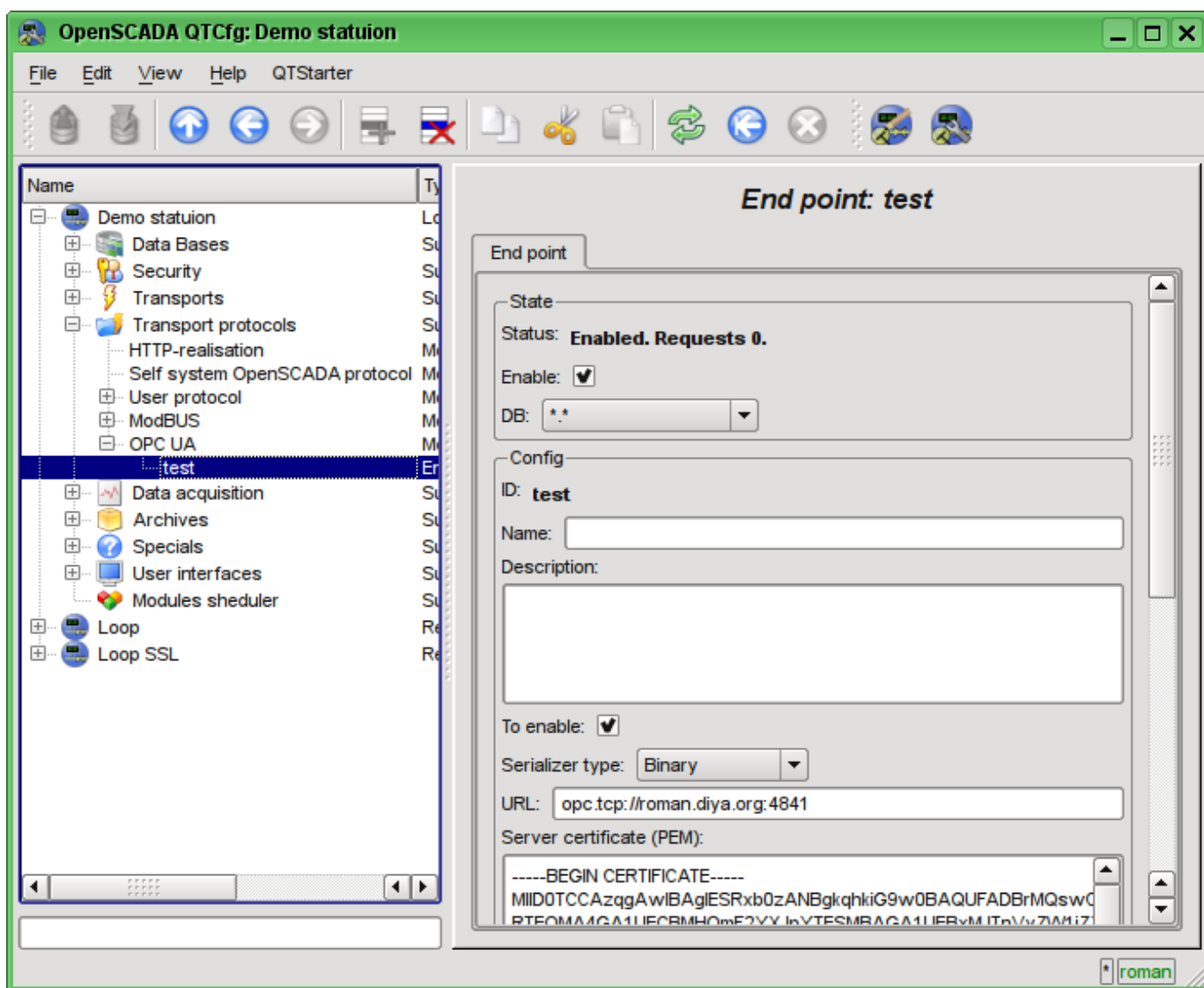


Fig.2. The main tab of the end node.

3. Data acquisition module

Data acquisition module provides the ability of inquiry and recording the value's attributes (13) of points with the "Variable" type.

3.1. Data controller

To add the OPC UA data source controller in the OpenSCADA system is created and configured. An example of the configuration tab of the controller is shown in Figure 3.

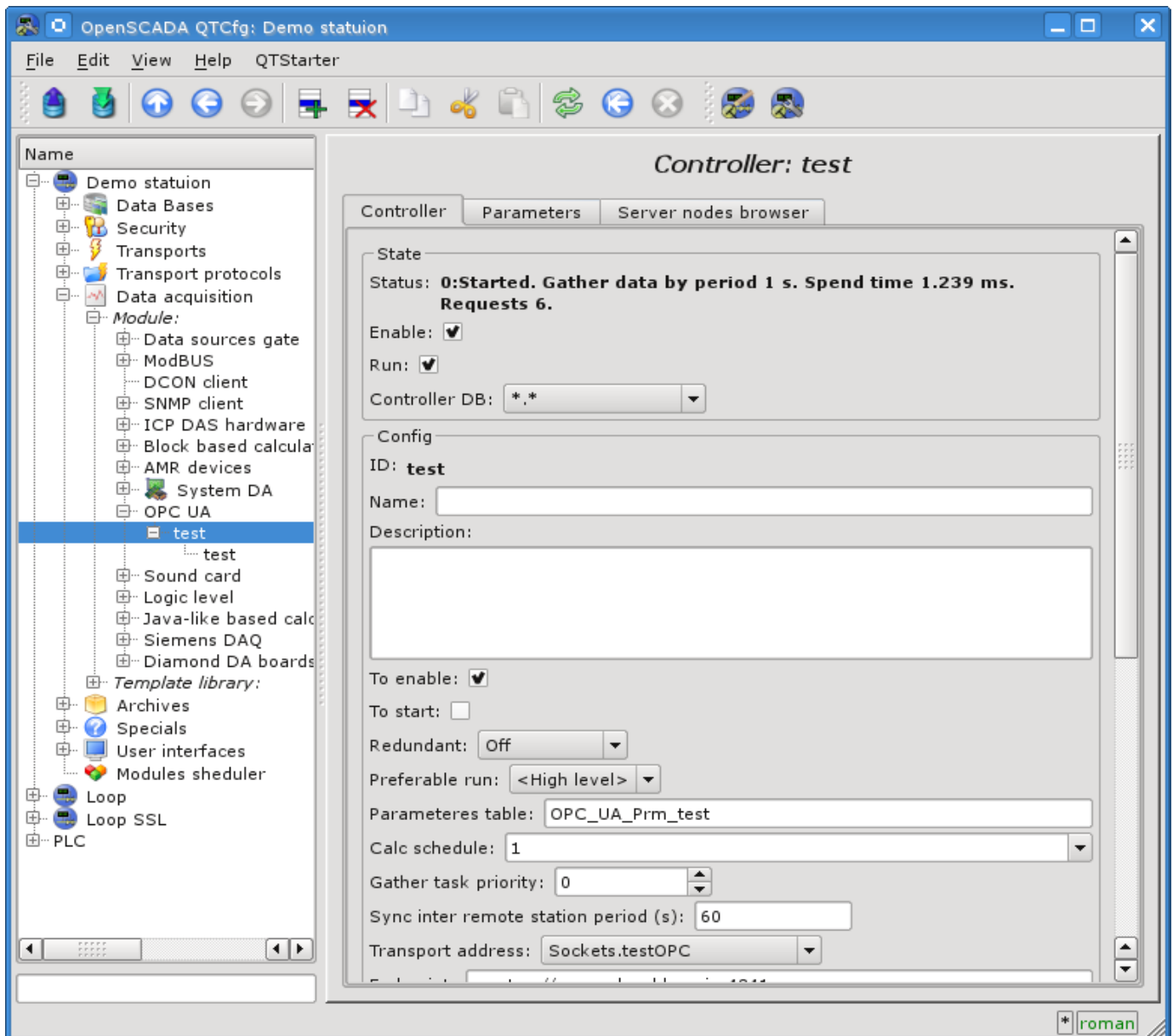


Fig.3. Controller's configuration tab.

From this tab you can set:

- The state controller, namely: Status, "Enable", "Run" and the name of the database containing the configuration.
- ID, name and description of the controller.
- The state, in which to transfer the controller at start: "Enable", "Run".
- The mode of horizontal redundancy and preference for the execution of the controller.
- The name of the table to store the configuration of parameters of the controller.
- The parameters of the execution the acquisition task scheduler and its priority.
- The period of the synchronization of the configuration of attributes of the parameters with the remote station, and try time for connection restore.

- The address of the outgoing transport from the list of configured outgoing transports in the subsystem "Transports" of OpenSCADA.
- The URL of the endpoint of remote station.
- Security policy and the mode of messaging security.
- The client certificate and private key in PEM format.
- The limit of the number of attributes in the parameter for the import mode of all the attributes belonging to the object.

To facilitate the identification of nodes on the remote station, as well as their choice to be inserted in the parameter of the controller in the controller's object it is provided the navigation on the remote station's nodes tab, where you can walk through the tree of objects and familiar with their attributes (Figure 4).

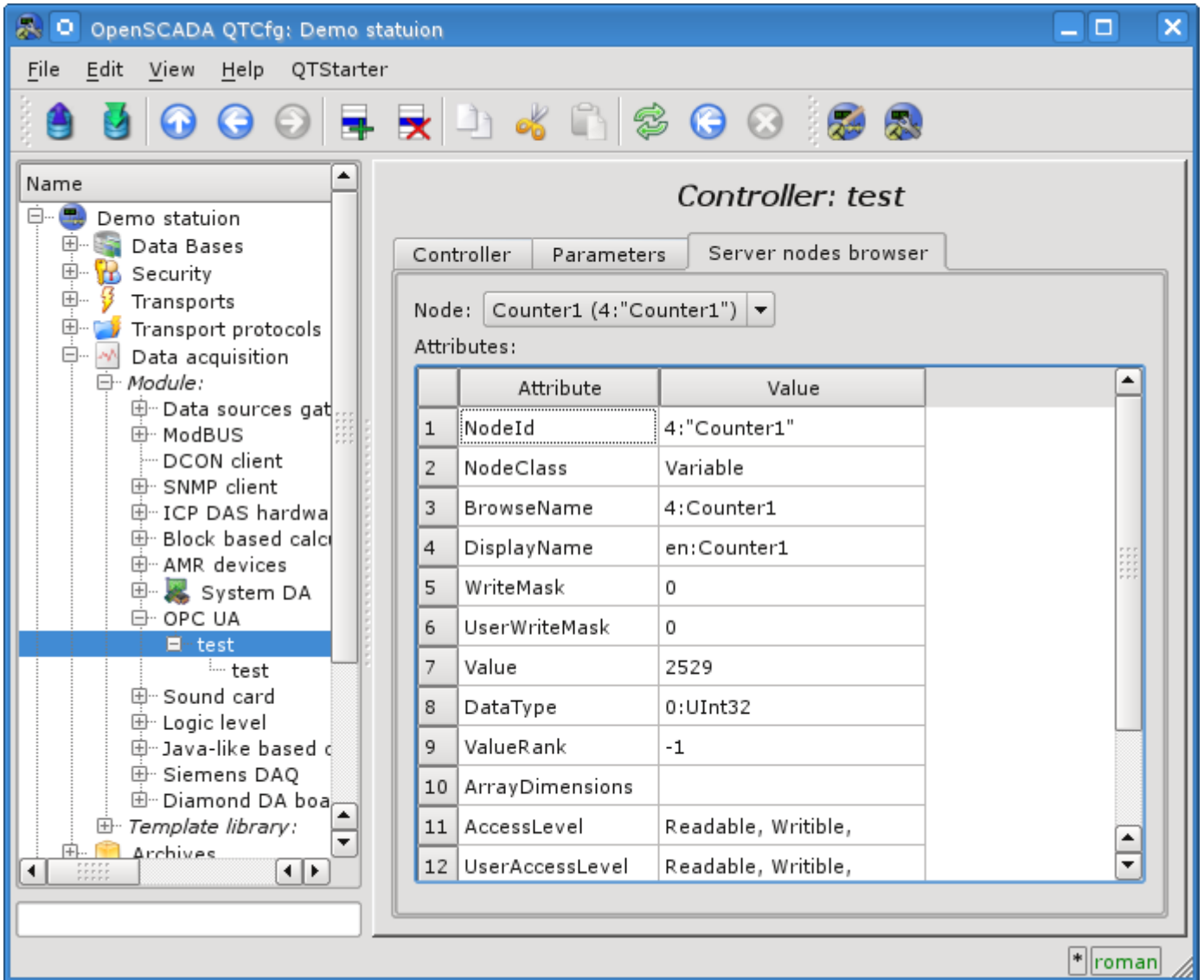


Fig.4. The "Server nodes browser" tab of the controller's page.

3.2. Parameters

Data acquisition module provides only one type of parameters - "Standard". Additional configuration field of the parameter of the module (Fig. 5) is the list of OPC UA nodes. Attribute in this list is written as follows: [ns:id].

Where:

- ns - names scope, number, zero value can missed;
- id - node identifier, number, string, bytes string and GUID.

Example:

- 84 - root directory;
- 3:"BasicDevices2" - basic devices node in the names scope 3 and string view;
- 4:"61626364" - node in the names scope 4 and byte string view;
- 4:{40d95ab0-50d6-46d3-bffd-f55639b853d4} - node in the names scope 4 and GUID view.

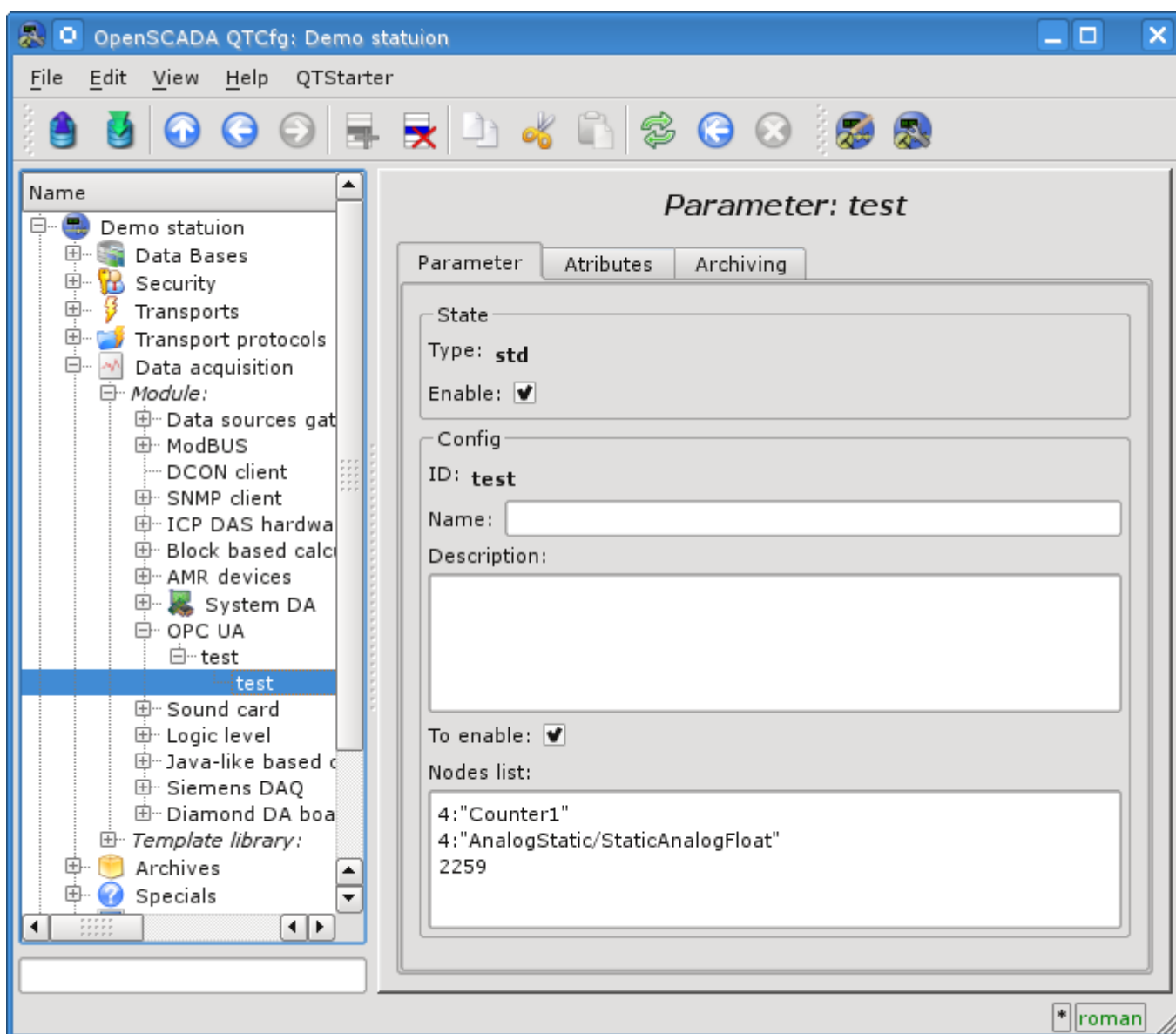


Fig.5. The configuration tab of the parameter.

In accordance with the specified list of nodes the inquiry and the creation of the parameter's attributes is made (Fig. 6).

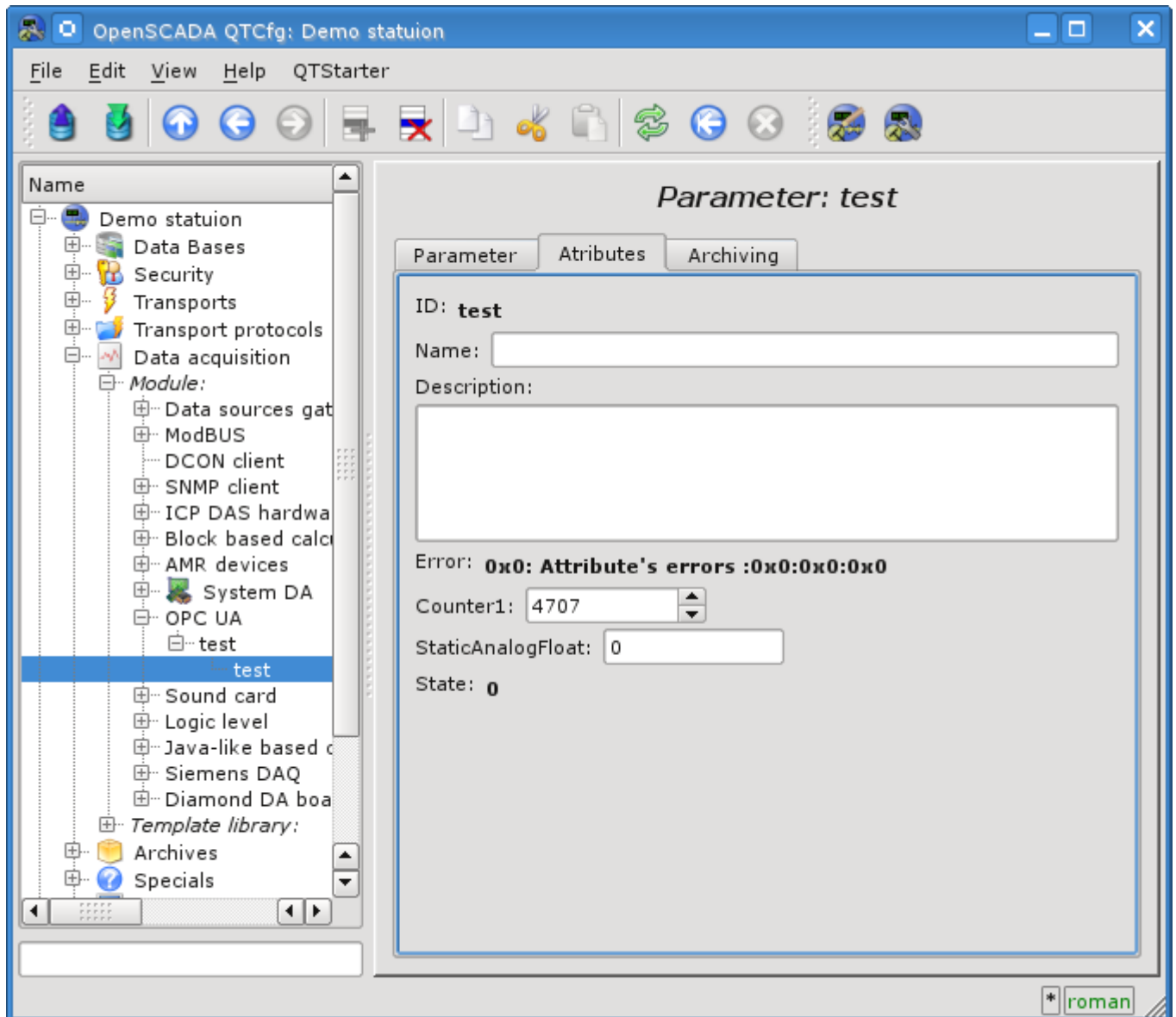


Fig.6. The parameter's attributes tab.

4. Notes

During the implementation of modules supporting OPC UA was detected several inconsistencies with the official SDK specification OPC UA:

- OPC UA Part 6 on page 27 contains an image of a handshake to establish a secure channel. The message of session create is signed by the client symmetric key and encrypted by server. In fact, both signature and encryption of the server key made.
- OPC UA Part 4 on page 141 contains a description of data structure signatures, which are the first data signature, and then the string algorithm. In fact, the reverse order is implemented.

Module <Sockets> of subsystem “Transports”

<i>Module:</i>	Sockets
<i>Name:</i>	Sockets
<i>Type:</i>	Transport
<i>Source:</i>	tr_Sockets.so
<i>Version:</i>	1.4.5
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides transport, based on the socket. It supports internet and unix sockets. Internet socket uses TCP and UDP protocols.
<i>License:</i>	GPL

Transport module Sockets provides support of transport based on the socket to the system. incoming and outgoing transport, based on internet sockets: TCP, UDP and UNIX sockets are supported. Addition of the new incoming and outgoing sockets can be done through the configuration of the transport subsystem in any system configurator of OpenSCADA.

1. Incoming transports

Configured and running incoming transport opens the server socket for the expectation of connection of the clients. In the case of the UNIX socket, the UNIX socket file is created. TCP and UNIX sockets are multi-stream, ie when the client connects to a socket of these type, the client socket and the new stream in which the client is served are created. Server socket in this moment switches to the waiting for the request from the new client. Thus the parallel service of the clients is achieved.

Each incoming socket is necessarily associated with one of the available transport protocols, to which incoming messages are transmitted. In conjunction with the transport protocol is supported by a mechanism of the combining of pieces of requests, disparate while transferring.

Configuration dialog of the incoming socket is depicted in Figure 1.

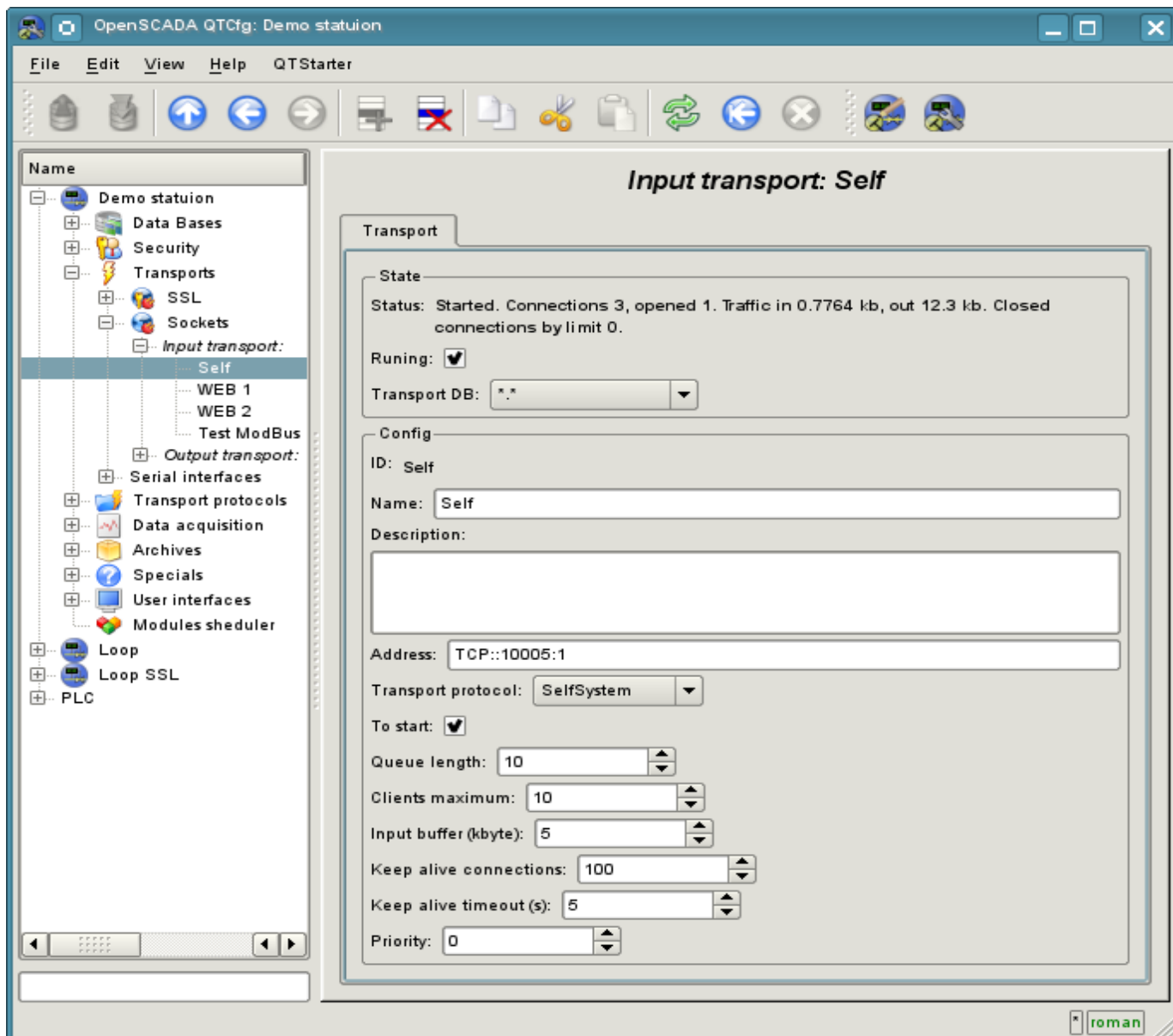


Fig.1. Configuration dialog of the incoming socket.

Using this dialog you can set:

- The state of transport, namely: “Status”, “Running” and the name of the database, containing the configuration.
- Id, name and description of transport.
- Address of the transport. The format of the address is listed in the table below.
- The choice of transport protocol.
- The state, in which the controller must be translated at boot: «Running».

- The length of the queue of sockets, the maximum number of clients to serve and the size of the input buffer.
- The limits the mode "Keep-alive" by requests counter and timeout.
- Transport's tasks priority.

Features of the formation of addresses of incoming sockets are shown in the table below:

Socket's type	Address
TCP	<p><i>TCP:[address]:[port]:[mode]</i> where:</p> <ul style="list-style-type: none"> • address – Address, on which the socket is opened. It must be one of the addresses of the host. If nothing is specified, the socket will be available in all the host interfaces. There may be as symbolic as well as IP presentation of address. • port – Network port, on which the socket is opened. Indication of the character name of the port (according to /etc/services) is available. • mode – mode of working of the incoming socket (0 – close the connection after the session reception-response, 1 – do not close). <p>Example: <i><TCP::10001:1></i> – TCP-socket is available on all interfaces, is opened on port 10001 and doesn't close the connection.</p>
UDP	<p><i>UDP:[address]:[port]</i> where:</p> <ul style="list-style-type: none"> • address – the same as in the TCP; • port – the same as in the TCP. <p>Example: <i><UDP:localhost:10001></i> – UDP-socket is only available on the “localhost” interface and is opened on the port 10001.</p>
UNIX	<p><i>UNIX:[name]:[mode]</i> where:</p> <ul style="list-style-type: none"> • name – UNIX socket file name; • mode – the same as in the TCP. <p>Example: <i><UNIX:/tmp/oscada:1></i> – UNIX-socket is available through the file /tmp/oscada and it doesn't close the connection.</p>

2. Outgoing transports

Configured and running outgoing transport opens a connection to the specified server. In the case of destroying of the connection, outgoing transport is disconnected. In order to resume the connection transport must be re-run.

Main tab of the configuration page of outgoing socket is shown in Fig.2.

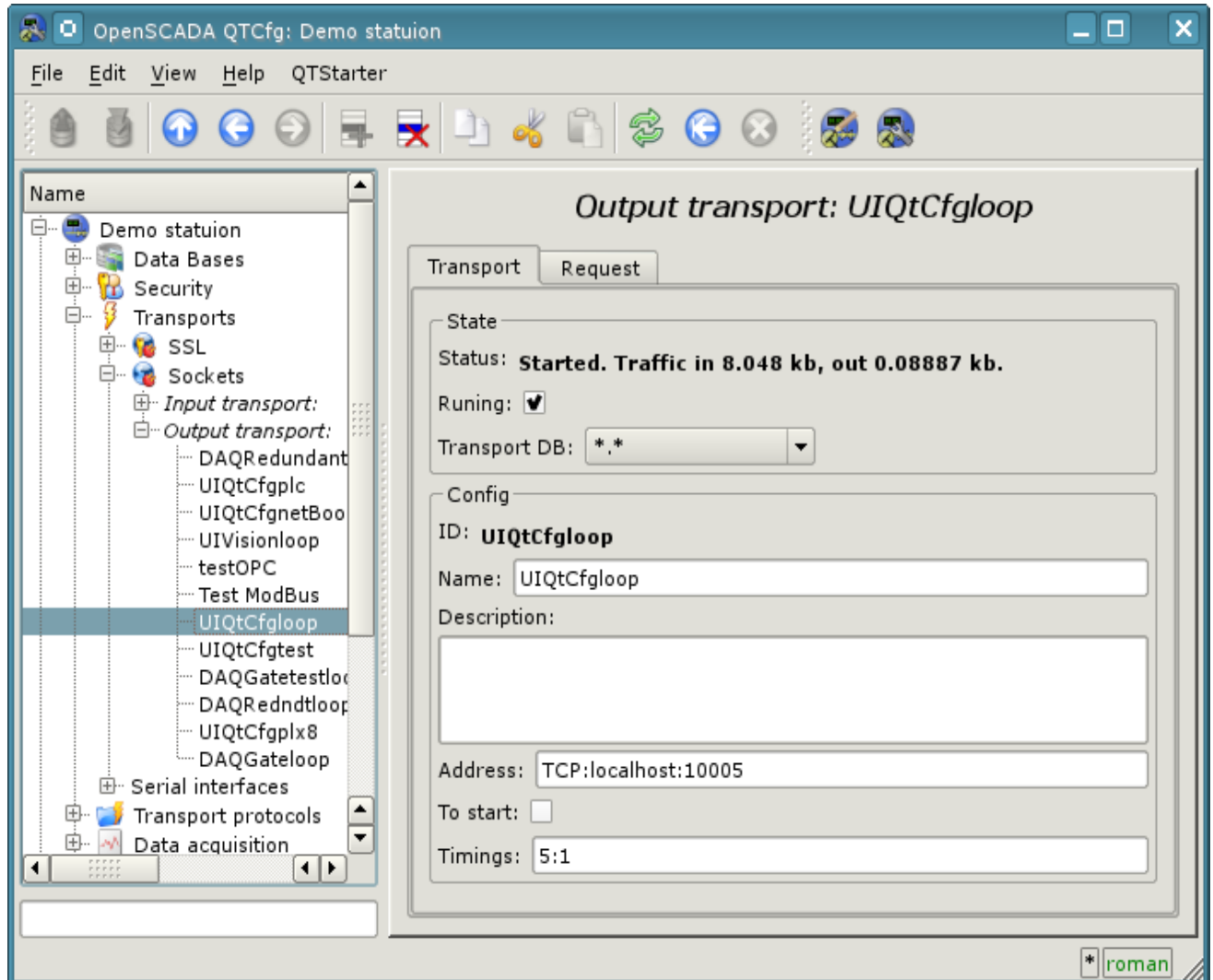


Fig.2. Main tab of the configuration page of the outgoing socket.

Using this dialog you can set:

- The state of transport, namely: "Status", "Running" and the name of the database, containing the configuration.
- Id, name and description of transport.
- Address of the transport. The format of the addresses is listed in the table below.
- The state, in which the controller must be translated at boot: «To start».
- Default timeout for connection and respond wait, separated.

The addresses of outgoing sockets of different types are formed as follows:

Socket's type	Address
TCP/UDP	<i>TCP:[address]:[port] UDP:[address]:[port]</i> where: <ul style="list-style-type: none">• address – Address to which the connection is performed. There may be as the symbolic representation as well as IP one of the address.• port – Network port, with which the connection is made. Indication of the character name of the port is available(according to /etc/services). Example: <i><TCP:127.0.0.1:7634></i> – To connect to the port 7634 on the host 127.0.0.1.
UNIX	<i>UNIX:[name]</i> where: <ul style="list-style-type: none">• name – UNIX socket file name. Example: <i><UNIX:/tmp/oscada></i> – to connect to the UNIX-socket through the file /tmp/oscada.

Module <SSL> of subsystem “Transports”

<i>Module:</i>	SSL
<i>Name:</i>	SSL
<i>Type:</i>	Транспорт
<i>Source:</i>	tr_SSL.so
<i>Version:</i>	0.9.5
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides transport based on the secure sockets' layer. OpenSSL is used and SSLv2, SSLv3 and TLSv1 are supported.
<i>License:</i>	GPL

The module SSL of the transport provides the support of transport based on secure sockets layer (SSL) into the system. In the basis of the module there is the library [OpenSSL](#). Incoming and outgoing transports of protocols SSLv2, SSLv3 and TLSv1 are supported.

It is possible to add new incoming and outgoing transports through the transport subsystem configuration in any configurator of OpenSCADA system.

1. Incoming transports

The configured and running incoming transport opens server SSL-socket for the expectation of connection of the clients. SSL-socket is a multi-stream, ie when the client connects, the client SSL-connection and a new stream in which the client is served are created. Server SSL-socket in this moment switches to the waiting for the request from the new client. Thus the parallel service of the clients is achieved.

Each incoming transport is necessarily associated with one of the available transport protocols, to which incoming messages are transmitted. In conjunction with the transport protocol is supported by a mechanism of the combining of pieces of requests, disparate while transferring.

Configuration dialog of the incoming SSL-transport is depicted in Figure 1.

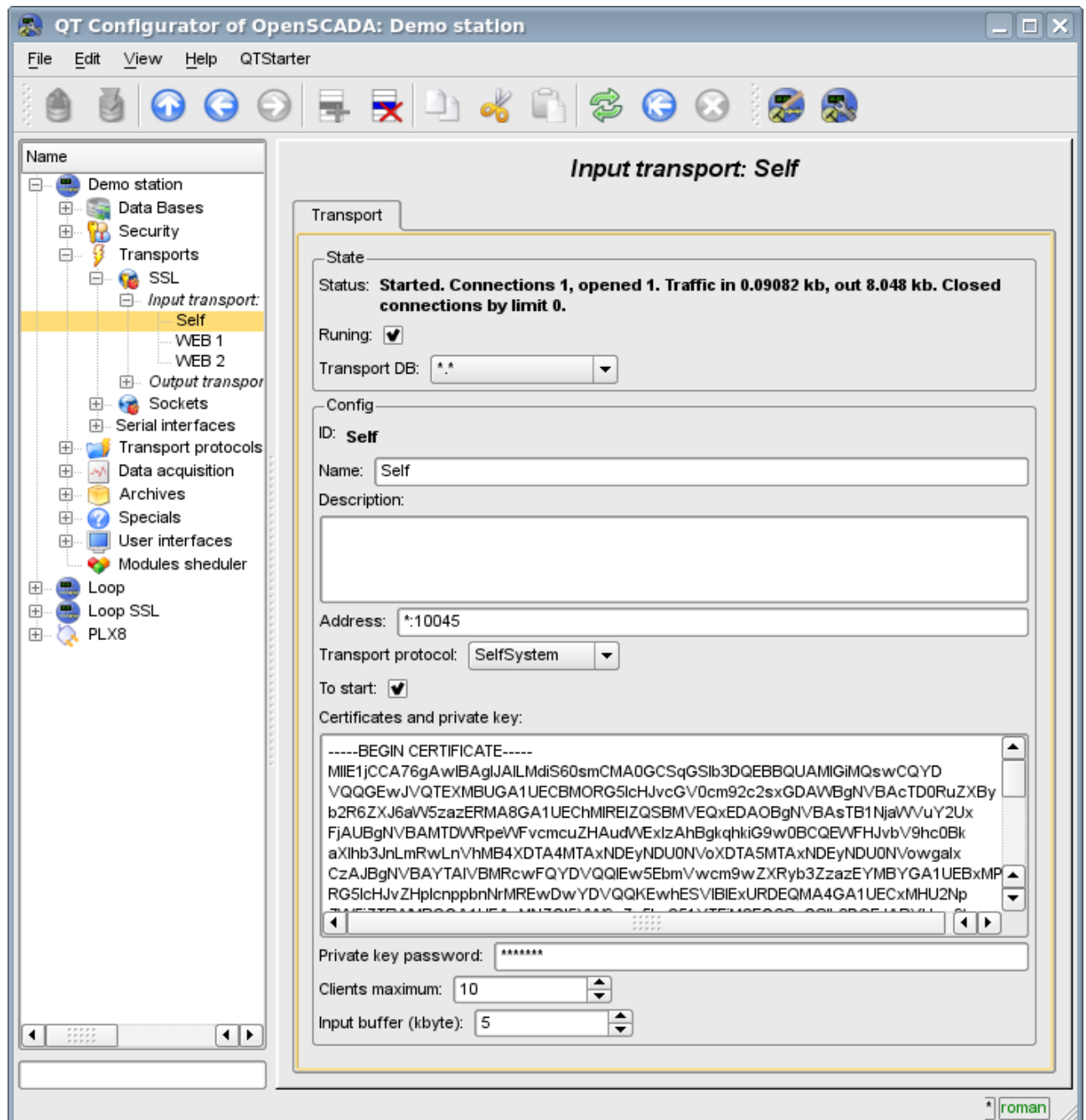


Fig.1. Configuration dialog of the incoming SSL-transport.

Using this dialog you can set:

- The state of transport, namely: "Status", "Running" and the name of the database, containing the configuration.
- Id, name and description of transport.

- Address of the transport in the format: "[address]:[port]:[mode]", where:
 - address – Address, on which the SSL is opened. It must be one of the addresses of the host. If the "*" is indicated then SSL will be available in all the host's interfaces. There may be as the symbolic representation as well as IP one of the address.
 - port – Network port, on which the SSL is opened. Indication of the character name of the port (according to /etc/services) is available.
 - mode – SSL-mode and version (SSLv2, SSLv3, SSLv23, TLSv1). By default and in case of error the SSLv23 is used.
- The choice of transport protocol.
- The state, in which the transport must be translated at boot: «To start».
- Certificates, private SSL key and password of private SSL key.
- The maximum number of clients to serve and the size of the input buffer.
- The limits the mode "Keep-alive" by requests counter and timeout.
- Transport's tasks priority.

2. Outgoing transports

Configured and running outgoing transport opens the SSL connection to the specified server. In the case of destroying of the connection, outgoing transport is disconnected. In order to resume the connection transport must be re-run.

Main tab of the configuration page of outgoing SSL-transport is shown in Fig.2.

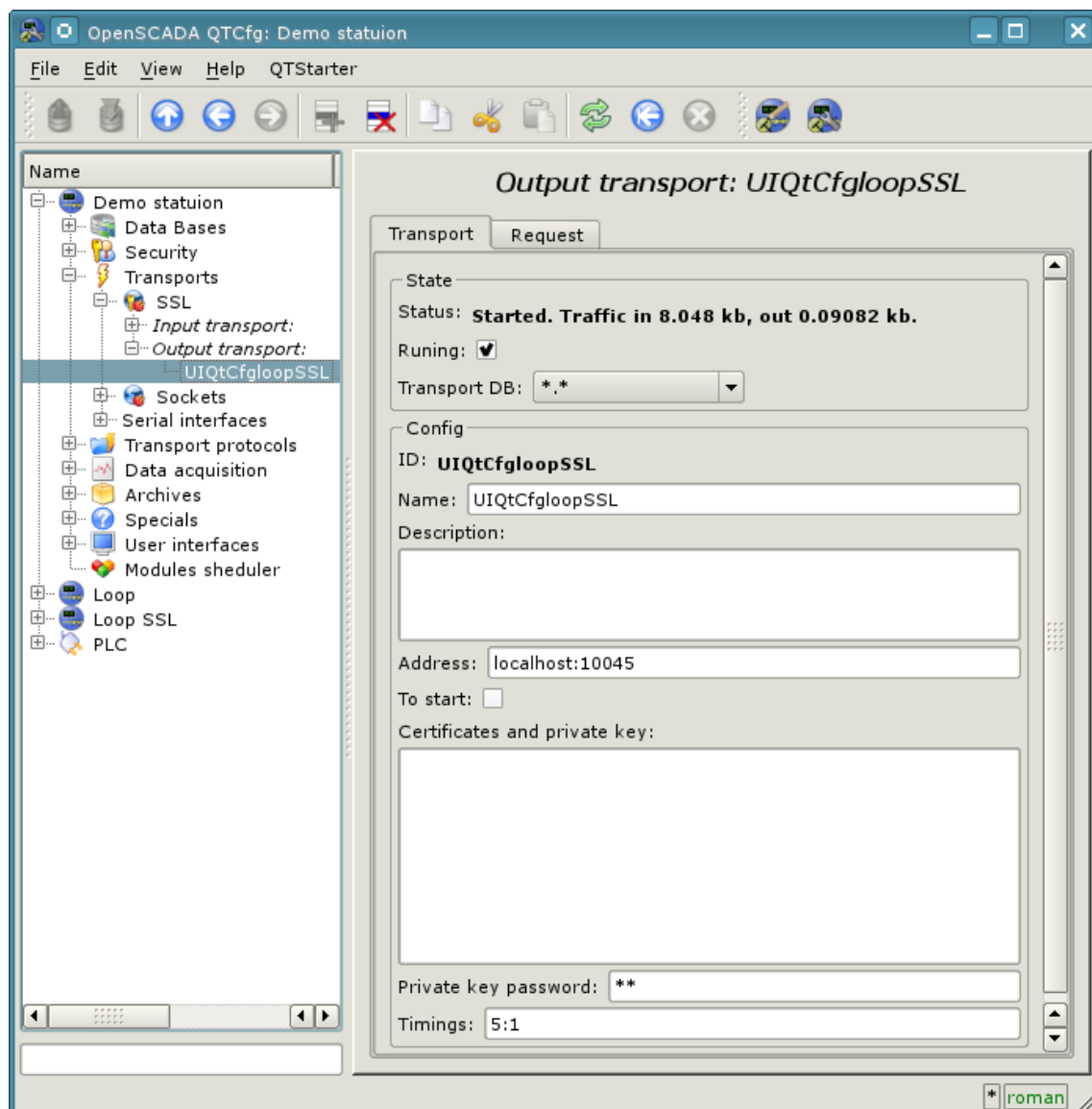


Fig.2. Main tab of the configuration page of the outgoing SSL-transport.

Using this dialog you can set:

- The state of transport, namely: “Status”, “Running” and the name of the database, containing the configuration.
- Id, name and description of transport.
- Address of the transport in the format: "[address]:[port]:[mode]", where:
 - address – Address with which the connection is made. There may be as the symbolic representation as well as IP one of the address.
 - port – Network port with which the connection is made. Indication of the character name of the port (according to /etc/services) is available.
 - mode – SSL-mode and version (SSLv2, SSLv3, SSLv23, TLSv1). By default and in case of error the SSLv23 is used.
- The state, in which the transport must be translated at boot: «To start».
- Certificates, private SSL key and password of private SSL key.
- Default timeout for connection and respond wait, separated.

3. Certificates and keys

For a valid module work certificates and private keys are required. In the case of the incoming SSL-transport (the server) they are compulsory. In the case of outgoing SSL-transport they can not be even installed though their using is desirable.

The simplest configuration of the certificate is self-subscription certificate and private key. The following describes how to create them using the tool openssl:

```
# Generation the secret key
$ openssl genrsa -out ./key.pem -des3 -rand /var/log/messages 2048
# Generation of self-subscription certificate
$ openssl req -x509 -new -key ./key.pem -out ./selfcert.pem -days 365
```

Next, the contents of the files key.pem and selfcert.pem is copied into the text field of the certificate and key. Password of the private key is installed in the appropriate field.

Module <Serial> of subsystem “Transports”

<i>Module:</i>	Serial
<i>Name:</i>	Serial Interface
<i>Type:</i>	Transport
<i>Source:</i>	tr_Serial.so
<i>Version:</i>	0.7.1
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides a serial interface. It is used to data exchange via the serial interfaces of type RS232, RS485, GSM and more.
<i>License:</i>	GPL

Module of transport Serial provides support of transports based on the type of serial interfaces RS232, RS485, GSM, and others to the system. Incoming and outgoing transports are supported. To add new incoming and outgoing interfaces is possible by means of configuration of the transport subsystem in the system configurator of OpenSCADA.

Into modem mode by the module support misc work mode. Misc mode mean an input transport allow, which wait ingoing connections, and also an output transport allow at idem device. That is the input transport will ignore all requests while the output transport's established connection allow, in idem time the output transport will not try make connection while the input transport have connection or other an output transport connected to other telephone, for example.

1. Incoming transports

The configured and running incoming transport opens port of serial interface for the expectation of the requests of the clients. Each incoming interface is necessarily associated with one of the available transport protocols, to which the incoming messages are transmitted.

Configuration dialog of the incoming serial interface is depicted in Figure 1.

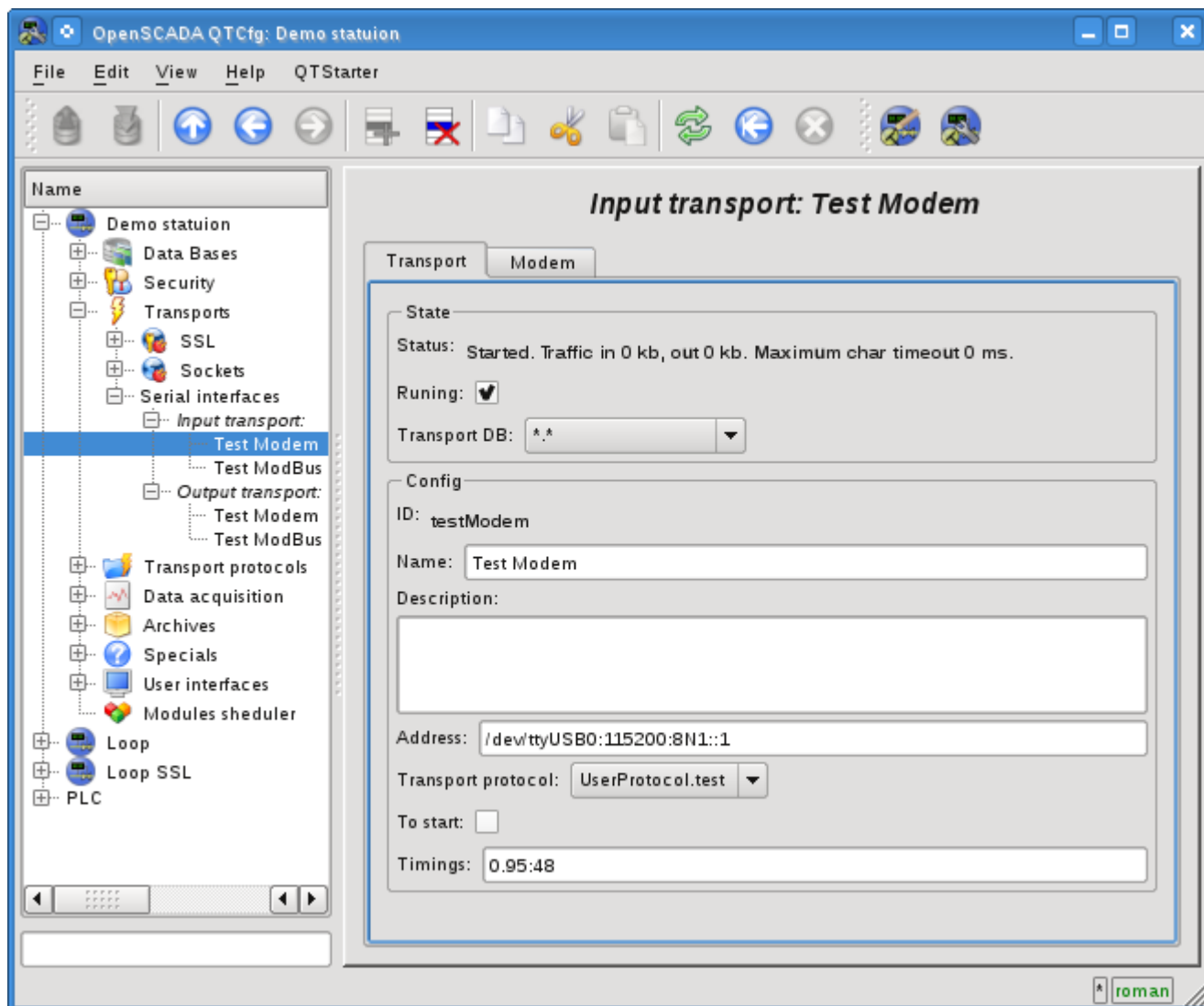


Fig.1. Configuration dialog of the incoming serial interface.

Using this dialog you can set:

- The state of transport, namely: "Status", "Running" and the name of the database, containing the configuration.
- Id, name and description of transport.
- Address of the transport in the format: "*[dev]:[spd]:[format]:[fc]:[mdm]*". Where:
 - *dev* - address of the serial device (/dev/ttyS0);
 - *spd* - speed of the serial devices from a number of: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000, 576000 or 921600;
 - *format* - asynchronous data format "<size><parity><stop>" (8N1, 7E1, 5O2, ...);
 - *fc* - flow control: 'h' - hardware (CRTSCTS), 's' - software (IXON|IXOFF);
 - *mdm* - modem mode, listen for 'RING'.
- The choice of transport protocol.
- The state, in which the transport must be translated at boot: "To start".
- Time intervals of the interface in the format of string: "*[symbol]:[frm]*". Where:
 - *symbol* - symbol time, in milliseconds. Used for control of the end of the frame;

- *frm* - the maximum time of the frame in milliseconds. Used to limit the maximum size of the package of the request (frame).

Transport supports the ability to work as a modem. This mode is activated by the fifth parameter of the address and includes call waiting from the remote modem (request "RING"), answering the call (command "ATA") and the subsequent transfer the requests from the remote station to the transport's protocol. Turning off the communication session is made by the initiator of the connection and leads to the reconnect of the modem-receiver for the waiting for new calls.

To configure the modem of the incoming transport the special tab "Modem" is provided (Fig. 2).

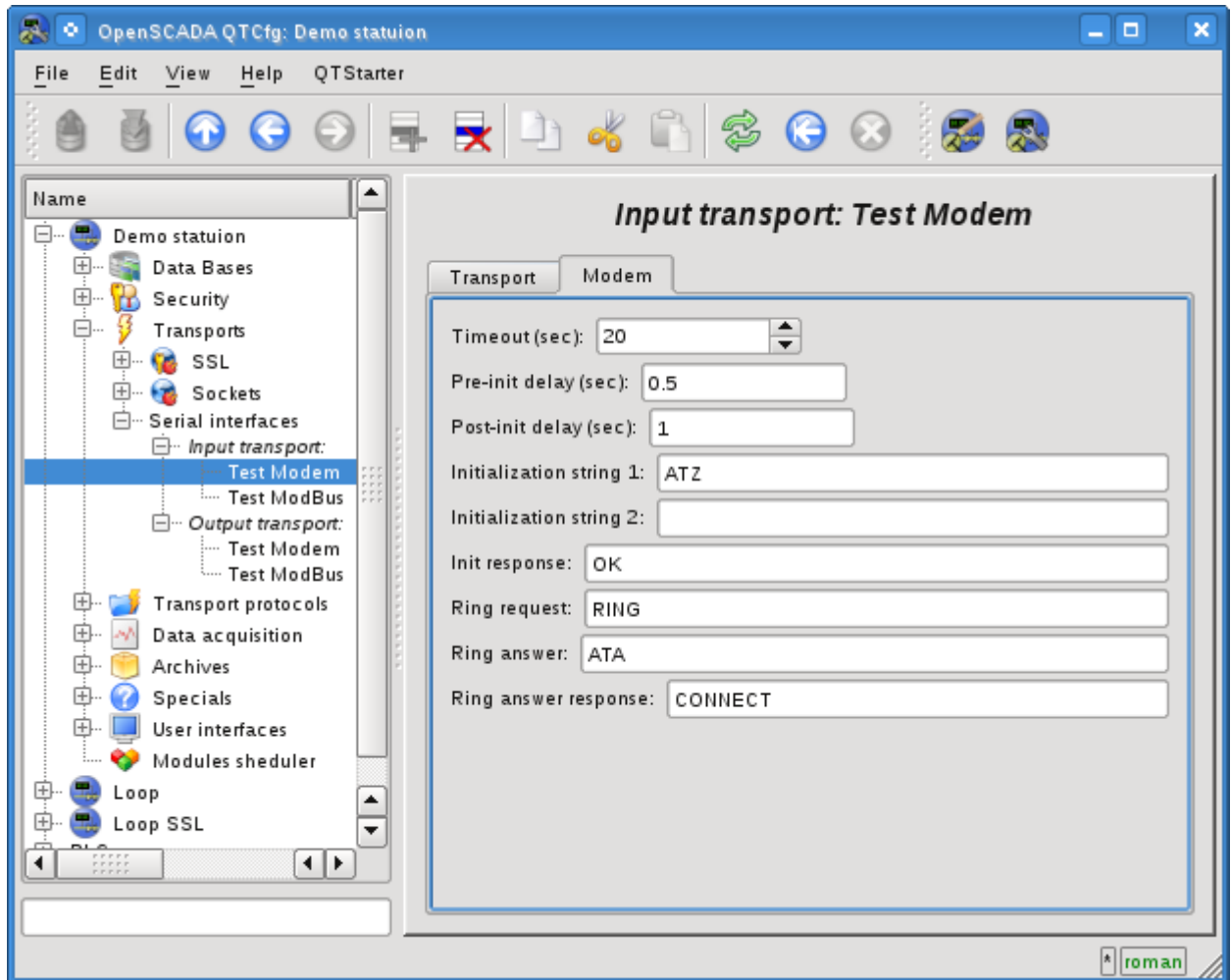


Fig.2. "Modem" tab of the modem's configuration of the incoming serial interface.

With this dialog you can set the following properties of working with modem:

- Requests timeout of the modem in seconds.
- The time delay before initializing the modem in seconds.
- The time delay after initializing the modem in seconds.
- The first initialization string typically contains the reset command of the modem "ATZ".
- The second initialization string.
- The result string of the modem's initialization, usually "OK", with which the modem answers for initializing and which must be expected.
- The call's request, usually is "RING", which is sent by the modem in the case of an outgoing call.
- The answer to the call, usually is "ATA", which is sent to the modem to answer the call.
- String result of the answer the call, usually is "CONNECT", with which the modem answers to the answer command, and that is to be expected.

2. Outgoing transports

Configured and running outgoing transport opens port of the serial interface for the sending the requests through it.

Main tab of the configuration page of outgoing serial interface is shown in Fig.2.

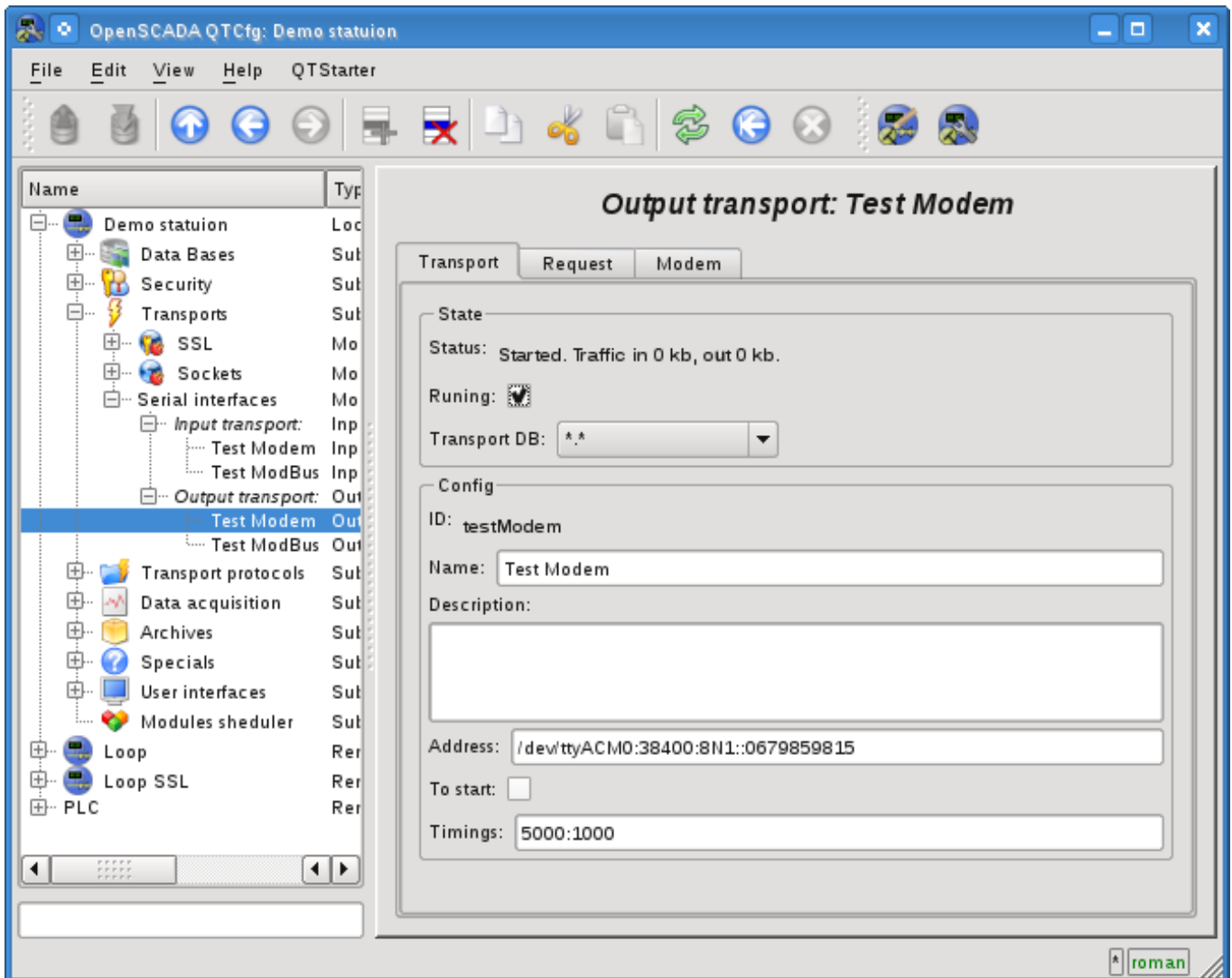


Fig.2. Main tab of the configuration page of outgoing serial interface.

Using this dialog you can set:

- The state of transport, namely: "Status", "Running" and the name of the database, containing the configuration.
- Id, name and description of transport.
- Address of the transport in the format: "*[dev]:[spd]:[format]:[fc]:[modTel]*". Where:
 - *dev* - address of the serial device (/dev/ttyS0);
 - *spd* - speed of the serial devices from a number of: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000, 576000 or 921600;
 - *format* - asynchronous data format "<size><parity><stop>" (8N1, 7E1, 5O2, ...);
 - *fc* - flow control: 'h' - hardware (CRTSCTS), 's' - software (IXON|IXOFF);
 - *modTel* - modem telephone, the field presence do switch transport to work with modem mode.
- The state, in which the transport must be translated at boot: "To start".
- Time intervals of the interface in the format of string: "*[conn]:[symbol]*". Where:
 - *conn* - waiting time of the connection i.e. response from the remote device.
 - *symbol* - symbol time, in milliseconds. Used for control of the end of the frame.

Transport supports the ability to work as a modem. This mode is activated by the fifth parameter of the address, and implies the phone call making at the number, specified in the fifth parameter, at the moment of transport's start. After installation the connection with the remote modem all requests are sent to the station behind the remote modem. Turning off the communication session at the transport's stop is made using the activity timeout.

To configure the modem of the outgoing transport the special tab "Modem" is provided (Fig. 4).

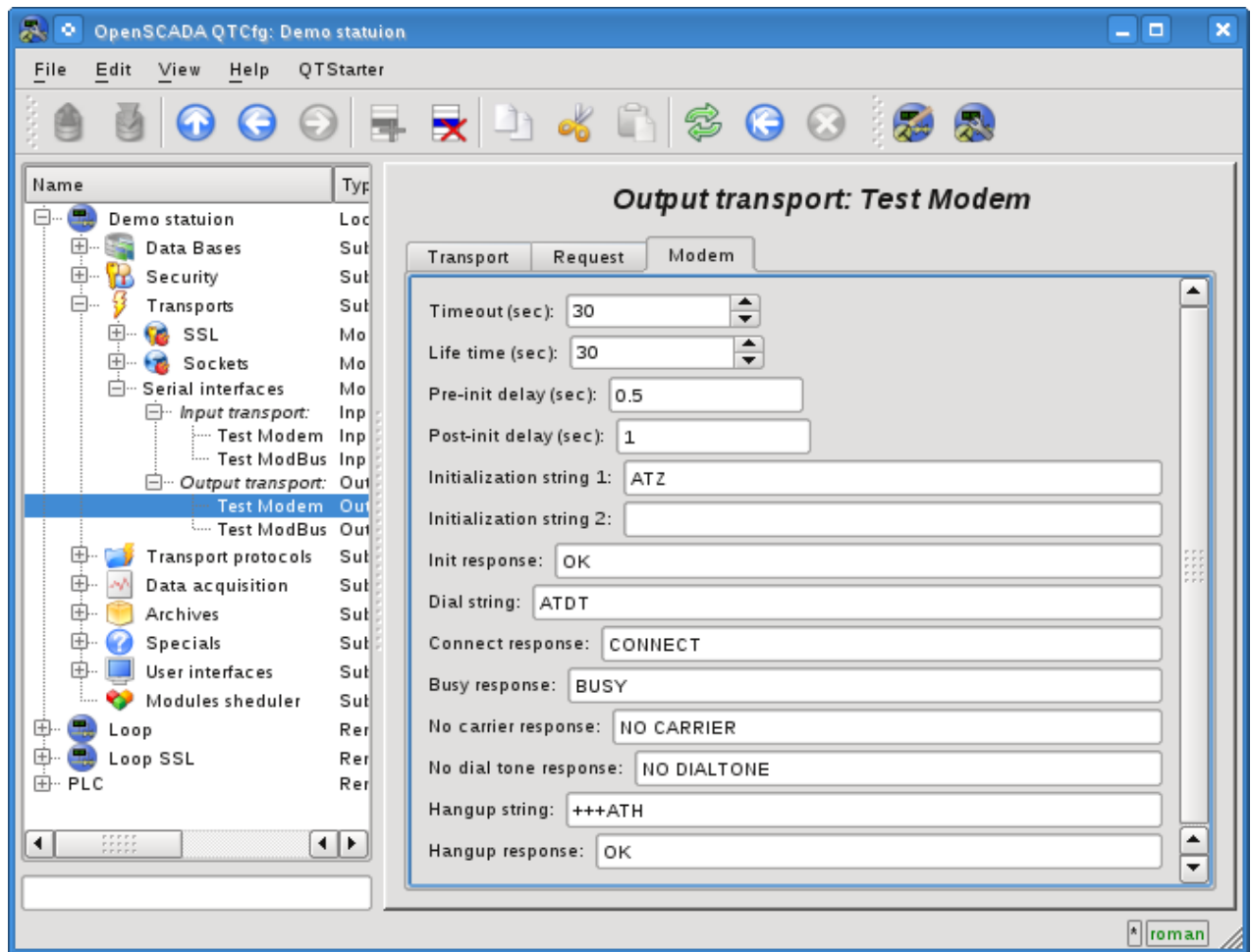


Fig.4. "Modem" tab of the configuration of modem of outgoing serial interface.

With this dialog you can set the following properties of working with modem:

- Requests timeout of the modem in seconds.
- Lifetime of the connection in seconds. If during this time there will be no data transmission over the transport the connection will be aborted.
- The time delay before initializing the modem in seconds.
- The time delay after initializing the modem in seconds.
- The first initialization string typically contains the reset command of the modem "ATZ".
- The second initialization string.
- The result string of the modem's initialization, usually "OK", with which the modem answers for initializing and which must be expected.
- Dialing string to the remote modem, usually is "ATDT". When you dial the phone number is appended to this prefix.
- The string result of the successful connection, typically is "CONNECT".
- The string result of the busy line, usually is "BUSY".
- The string result of the absence of the carrier in line, usually is "NO CARRIER".
- The string result of the lack of dial tone in the line, typically is "NO DIALTONE".
- The command hang up, is usually "+++ATH". This command is called whenever there is need to break the connection.
- The string result of the hang up command, usually is "OK", with which the modem answers to the command and which must be expected.

Module <HTTP> of subsystem “Protocols”

<i>Module:</i>	HTTP
<i>Name:</i>	HTTP
<i>Type:</i>	Protocol
<i>Source:</i>	prot_HTTP.so
<i>Version:</i>	1.5.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides support for the HTTP protocol for WWW-based user interfaces.
<i>License:</i>	GPL

Module of the transport protocol HTTP is designed to support the implementation of network protocol HTTP (Hypertext Transfer Protocol) in the system OpenSCADA.

HTTP Protocol is used to transfer the WWW contents. For example, via HTTP the following types of documents are transmitted: html, xhtml, png, java, and many others. Adding the HTTP support in OpenSCADA system together with the Sockets transport allows to implement various user functions based on the WWW interface. The module implements two main methods of the HTTP protocol: "GET" and "POST". "HTTP" module provides control of the integrity of HTTP-queries and, jointly with "Sockets" transport, allows to "collect" holistic requests of their fragments, as well as maintain the keeping of the connection alive (Keep-Alive).

For flexible connection of the user interfaces to the module the modular mechanism within the module HTTP is used. In the role of modules the modules of subsystem the "User interfaces" are used with the additional information field "SubType" with the value of "WWW".

In the requests for the Web resources the URL(Universal Resource Locator) are commonly used, hence the URL is passed as the main parameter via HTTP. The first element of the requested URL is used to identify the module UI. For example URL: <http://localhost:10002/WebCfg> means - address to module WebCfg on the host <http://localhost:10002>. In the case of an incorrect indication of the module ID, or when you address without identifier of the module at all, HTTP module generates the dialogue of the information on the input and with the choice of one of the available user interfaces. Example of a dialogue is shown in Figure 1.

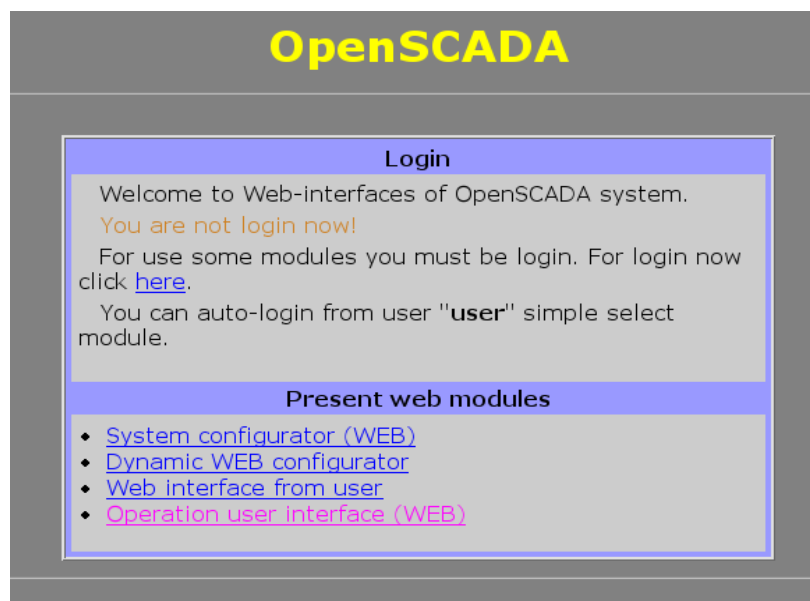


Fig.1. Dialog of the choice of WWW-interface module.

1. Authentication

Module supports authentication in the system OpenSCADA while providing access to the WEB-interface modules (Fig.2). Dialogue is formed in the language of XHTML 1.0 Transitional!

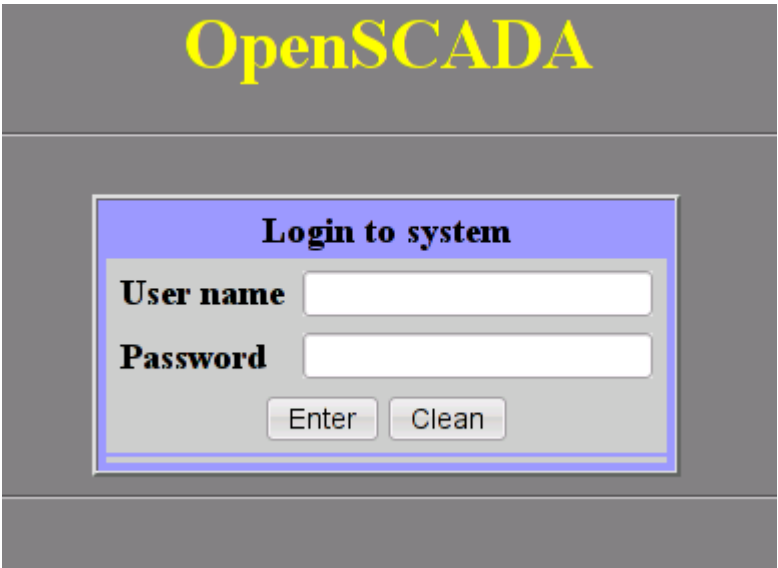


Fig.2. Authentication dialogue in the system OpenSCADA.

For ease of Web-based interface module provides the ability to automatically log on behalf of the specified user. Configuring automatic logon to make by the module settings page (Fig.3).

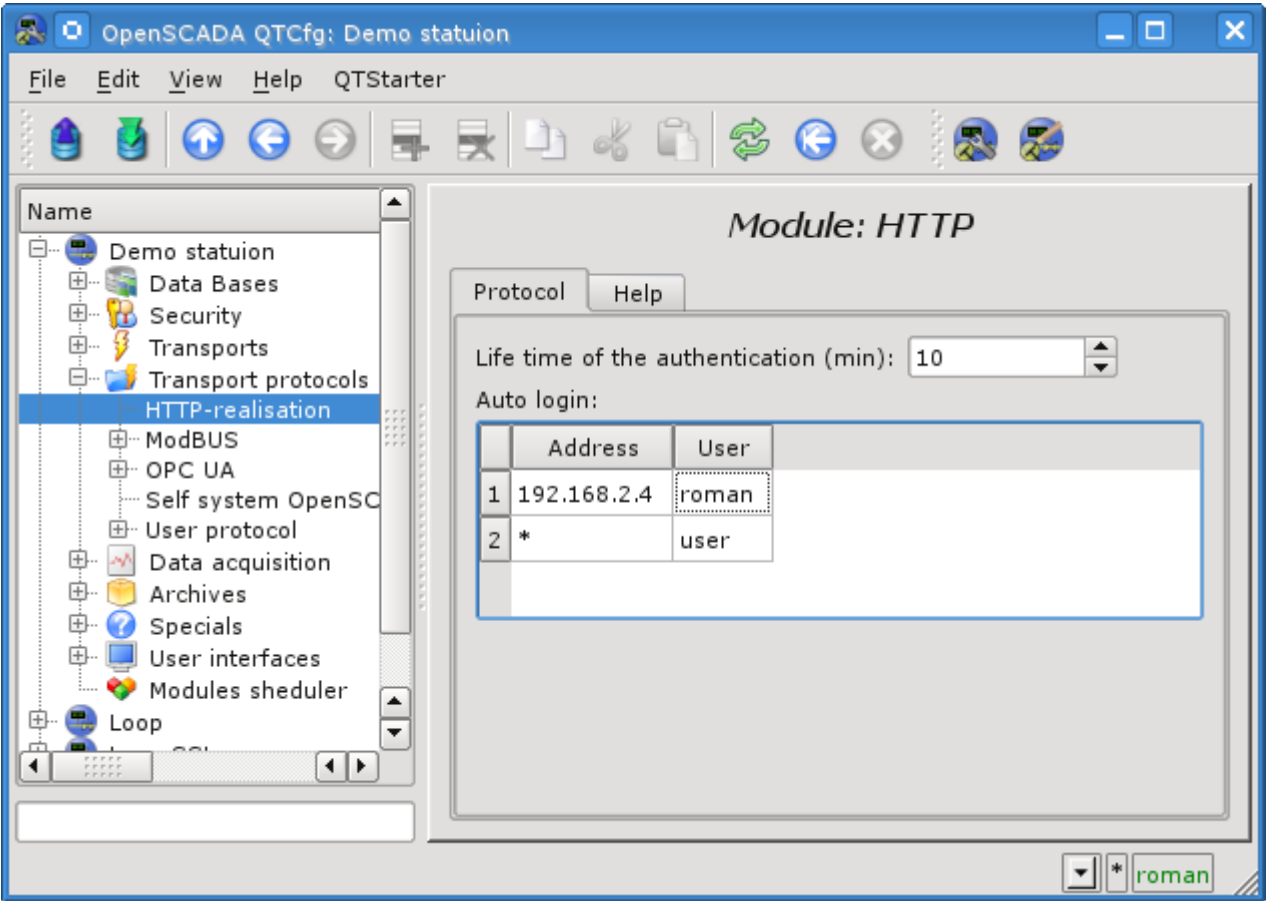


Fig.3. The module configuring page.

On the module settings you can specify the lifetime of the authentication and set up automatic login. Automatic login is carried out by matching the address indicated in the column "Address", on behalf of the user specified in the column "User".

2. The modules of user WEB-interface

Modules of the user interface (UI) designed to work with HTTP module, should indicate an information field "SubType" with the value "WWW" and "Auth" field with the value "1" if the module requires an authentication at login. For communication of HTTP module and UI modules an advanced communication mechanism is used. This mechanism involves the export of interface functions. In this case the UI modules must export the following function:

- `void HttpGet(const string &url, string &page, const string &sender, vector<string> &vars, const string &user);` - GET method with the parameters:
url - address of the request; *page* - page with the answer; *sender* - address of the sender; *vars* - request variables; *user* - user of the system.
- `void HttpPost(const string &url, string &page, const string &sender, vector<string> &vars, const string &user);` - POST method with the parameters:
url - address of the request; *page* - page with the answer and with the contents of the body of the POST request; *sender* - address of the sender; *vars* - request variables; *user* - user of the system.

Then, in the case of a HTTP GET request, the function `HttpGet` will be called, and in the case of the POST request, the function `HttpPost` will be called in the appropriate UI module.

3. Outgoing requests function's API

The outgoing function of API operate by HTTP-request's content which wrapped to XML-packages. The request structure is:

```
<req Host="host" URI="uri">
  <prm id="pId">pVal</prm>
  <cnt name="cName" filename="cFileName">
    <prmid="cpId">cpVal</prm>
    cVal
  </cnt>
</req>
```

Where:

- *req* - request method, supported methods "GET" and "POST".
- *host* - http-server address into format *[HostAddr]:[HostIp]*. If that field have been passed then used node address which set into address field of the transport.
- *uri* - resource address, file or direcorey, at http-server.
- *pId*, *pVal* - identifier and value of addition http-parameters. You can set multiply http-parameters by different *prm* tags set.
- *cName*, *cFileName*, *cVal* - name, file-name and value of content-element of POST-request. You can set multiply content-elements by different *cnt* tags set.
- *cpId*, *cpVal* - identifier and value of addition content-parameters. You can set multiply content-parameters by different *prm* tags set.

Request result's structure is:

```
<req Host="host" URI="uri" err="err" Protocol="prt" RezCod="rCod" RezStr="rStr">
  <prm id="pId">pVal</prm>
  respVal
</req>
```

Where:

- *req* - request method.
- *host* - http-server address.
- *uri* - resource address.
- *err* - the error wich appear in request time. For succeeded requests the field is empty.
- *RezCod*, *RezStr* - request result into view code and text.
- *pId*, *pVal* - identifier and value of addition http-parameters. Respond can set multiply http-parameters by different *prm* tags set.

- *respVal* - respond's content.

Into example role we accord using the function into users procedures for GET and POST requests making by language JavaLikeCalc.JavaScript:

```
//GET request
req = SYS.XMLNode("GET");
req.setAttr("URI", "/");
SYS.Transport.Sockets.out_testHTTP.messIO(req, "HTTP");
test = req.text();

//POST request
req = SYS.XMLNode("POST");
req.setAttr("URI", "/WebUser/FlowTec.txt");
cntNode = req.childAdd("cnt").setAttr("name", "pole0").setAttr("filename", "Object2-
k001-100309-17.txt");
cntNode.childAdd("prm").setAttr("id", "Content-Type").setText("text/plain");
cntText = "Object2-k001\r\n";
cntText += "\r\n";
cntText += "v002\r\n";
cntText += "  nl\r\n";
cntText += "  09.03.10 16    Polnyj      7155.25      216.0   32.000   17.5\r\n";
cntText += "v005\r\n";
cntText += "  nl\r\n";
cntText += "  09.03.10 16    Polnyj      188.81      350.0    4.000   40.0\r\n";
cntText += "\r\n";
cntNode.setText(cntText);
SYS.Transport.Sockets.out_testHTTP.messIO(req, "HTTP");
```


Module <SelfSystem> of subsystem “Protocols”

<i>Module:</i>	SelfSystem
<i>Name:</i>	OpenSCADA system own protocol
<i>Type:</i>	Protocol
<i>Source:</i>	prot_SelfSystem.so
<i>Version:</i>	0.9.3
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	OpenSCADA system own protocol, it supports the basic functions.
<i>License:</i>	GPL

The module of the transport protocol SelfSystem is designed to reflect the interface management of OpenSCADA system to the network, to provide an opportunity to the external systems to interact with the OpenSCADA system, as well as for the interaction of the stations constructed on the basis of OpenSCADA among themselves.

The first experience of using the functions of this module was the support of remote configuration of one OpenSCADA station from another through the network, by means of the module of configuration [QTCfg](#).

1. The syntax of the protocol

The protocol is built on the mechanism of request-response. Requests and their structure are summarized in Table 1.

Table 1 Structure of the request.

Requests
REQ: «SES_OPEN <user> <password>\n» REZ OK: «REZ 0 <ses_id>\n» REZ ERR: «REZ 1 Auth error. User or password error.\n» The request for the opening of the session on behalf of the user <user> with the password <password>. In case of success it will be received the session identifier, otherwise – the code and the error message.
REQ: «SES_CLOSE <ses_id>\n» REZ: «REZ 0\n» Closure of the session. The result is always successful.
REQ 1: «REQ <ses_id> <req_size> \n <control interface command>” REQ 2: «REQDIR <user> <password> <req_size> \n <control interface command>” REZ OK: «REZ 0 <rez_size> \n <control interface command result>” REZ ERR: «REZ 1 Auth error. Session is not valid.\n» REZ ERR: “REZ 2 <control interface err>” The main requests: the session and the direct are implemented by sending the standard command of OpenSCADA control interface to the field <control interface command>. As the result will it be received an answer from the management interface <control interface command result> or one of the errors.
REQ: “ERR REQUEST” REZ ERR: «REZ 3 Command format error.\n» Any invalid request.

Protocol supports the package of traffic. only the data of the management interface is to be packed <control interface command> and <control interface command result>. The fact of the arrival of packaged request or response is determined by the negative value of the size of the request <req_size> or response <rez_size>.

To control the parameters of the package the module provides the configuration form (Fig. 1).

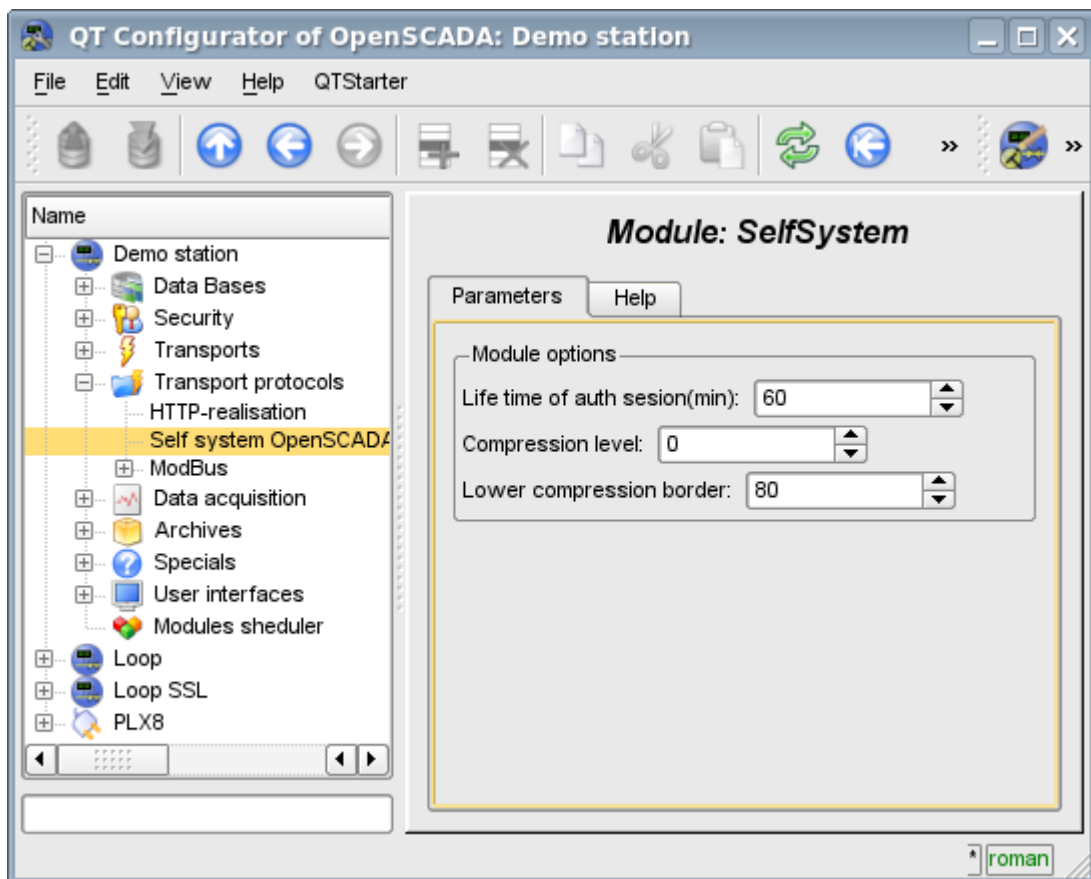


Fig.1. The form of the configuration of the package parameters.

On this form, you can specify:

- the lifetime of the authentication session;
- level of compression of the protocol, ranging from 0 to 9 (0-disable compression-1-optimal in performance and quality compression level);
- lower threshold for the compression using, turns off the compression of small requests.

2.The internal structure of an outgoing protocol

The internal structure if formed by means of the tree of XML requests of the language [OpenSCADA control interface](#) with the reservation of the redundant additional service attributes of the protocol in the root tag:

- *rqDir* — sign of the sending the message passing the procedure of the opening the session (0-open session, 1-send immediately);
- *rqUser* — user;
- *rqPass* — password.

The result of the request is the tree of XML language of the management interface of OpenSCADA.

Module <UserProtocol> of subsystem “Protocols”

<i>Module:</i>	UserProtocol
<i>Name:</i>	User protocol
<i>Type:</i>	Protocol
<i>Source:</i>	prot_UserProtocol.so
<i>Version:</i>	0.6.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Allows you to create your own user protocols on any OpenSCADA's language.
<i>License:</i>	GPL

Module UserProtocol of the transport protocol is made to provide the user with the possibility of creation the implementations of different protocols by himself at one of the internal languages of OpenSCADA, usually [JavaLikeCalc](#), without necessity of low-level programming of OpenSCADA.

The main purpose of the module is to simplify the task of connecting to the OpenSCADA system devices of data sources, that have limited distribution and/or provide access to their own data on a specific protocol that is usually fairly simple to implement in the internal language of OpenSCADA. For implementation of this the mechanism for the formation of the outgoing request protocol is provided.

In addition to the mechanism of the outgoing request protocol the mechanism for incoming request protocol is provided, which allows OpenSCADA to process the requests for data get on specific protocols, which simply can be implemented in the internal language of OpenSCADA.

The module provides the ability to create multiple implementations of different protocols in the object "User protocol" (Fig. 1).

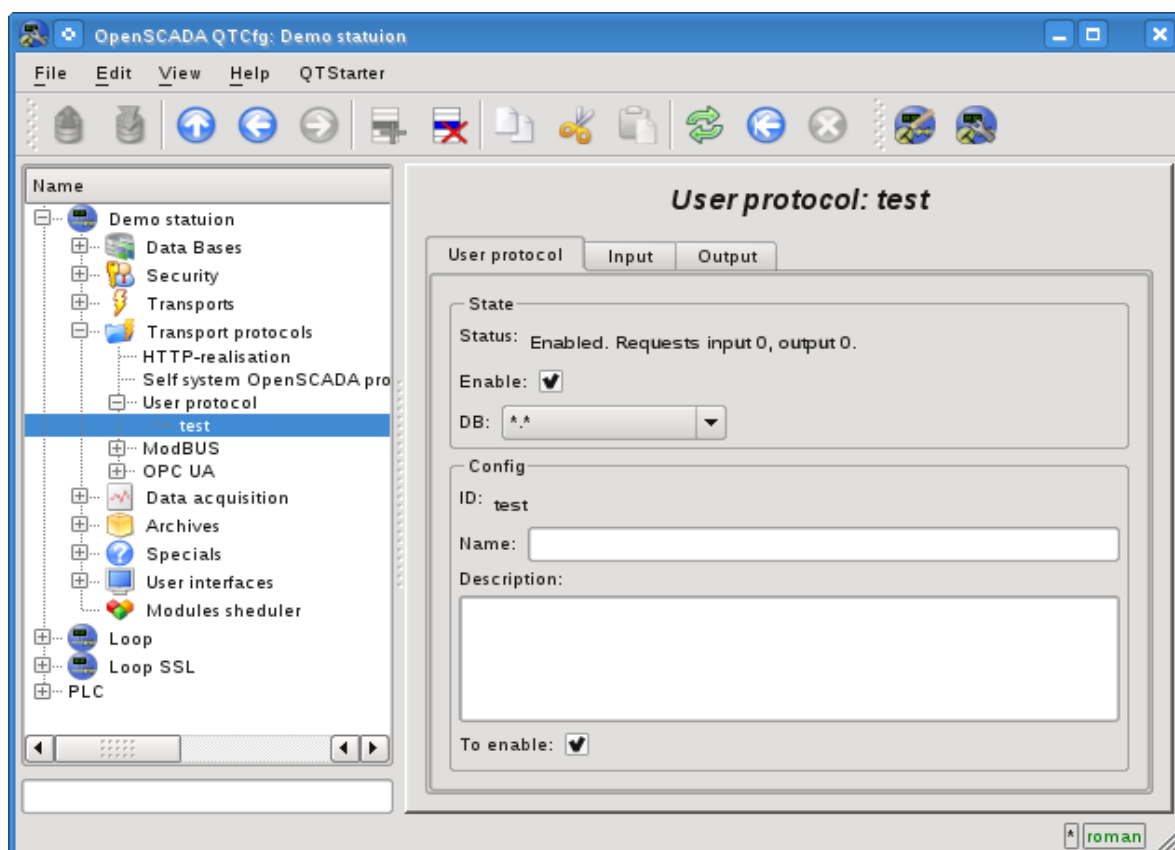


Fig.1. The main tab of the object "User protocol".

The main tab contains the basic settings of the user protocol:

- Section "Status" - contains properties that characterize the status of the protocol:
 - *Status* - current status of the protocol.
 - *Enable* - the protocol's status "Enabled".
 - *DB* - DB that stores configuration.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the protocol's identifier.
 - *Name* - specifies the name of the protocol.
 - *Description* - brief description of the protocol and its purpose.
 - *To enable* - indicates the status "Enable", in which to transfer the protocol at startup.

1. Part of the protocol for incoming requests

Protocol of incoming requests is working in cooperation with the incoming transport and the separate object "User Protocol" is set in the configuration field of transport protocol, together with the UserProtocol module's name. In the future, all requests to the transport will be sent to the processing procedure of the protocol's request (Fig. 2).

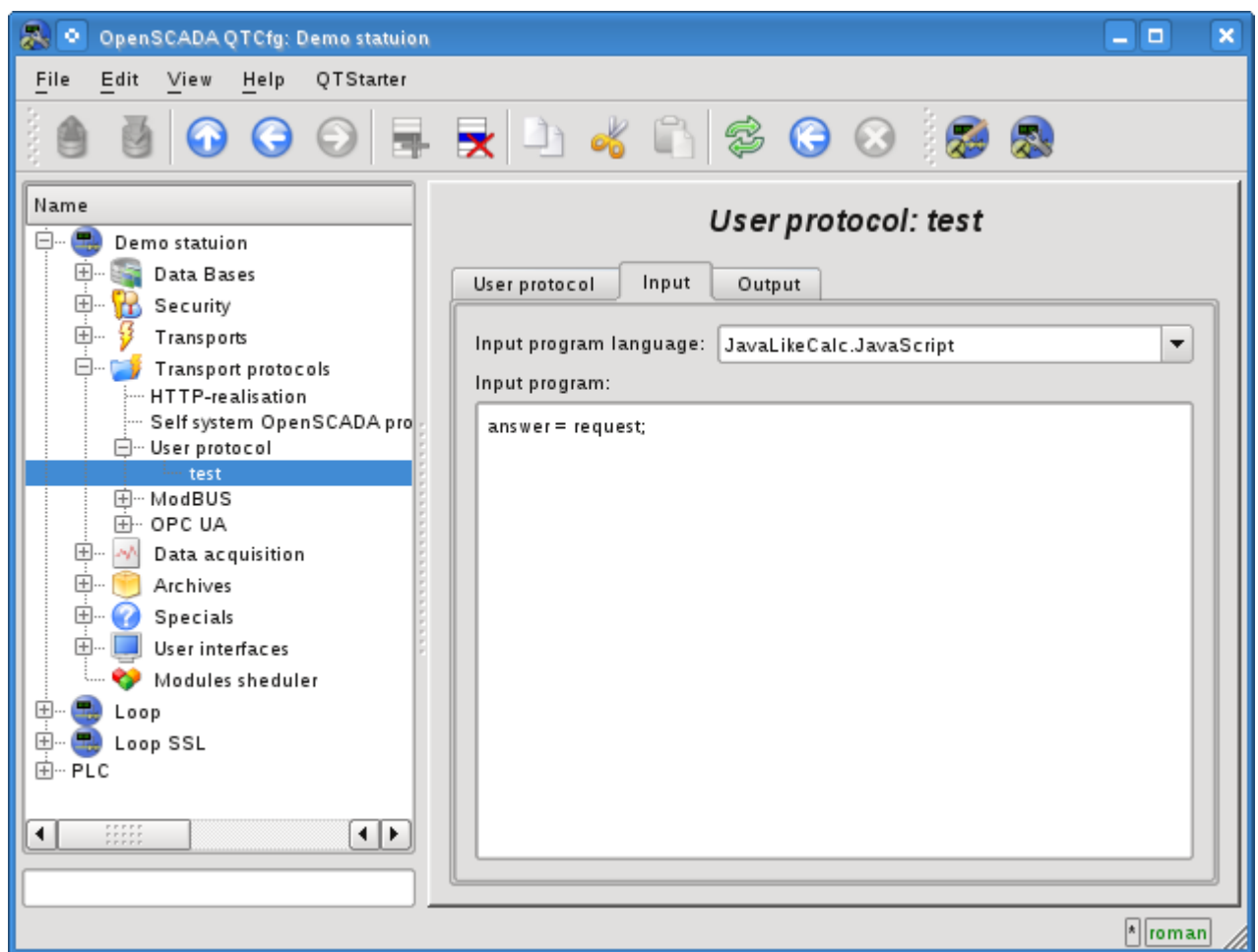


Fig.2. Tab of the processing procedures of the incoming requests.

Tab of the processing procedures of the incoming request contains the field for selecting the internal programming language of OpenSCADA and the text entry field for the typing the processing procedure.

For the processing procedure the following exchange variables with incoming traffic are predetermined:

- *rez* - processing result (false-full request;true-not full request);
- *request* - request message;
- *answer* - answer message;
- *sender* - request sender.

The overall scenario of processing of the incoming requests:

- Request is formed by the remote station and through the network it gets on the transport of OpenSCADA.
- OpenSCADA transport sends the request to the selected in the protocol's field UserProtocol module and to the objects of the user's protocol in the form of the variable's "request" values - for the block of the request and "sender" - for the sender address of the request.
- The execution of the the procedure of protocol of the incoming request is started, during which the contents of the variable "request" is analyzed and the response in the variable "answer" is formed. At the end of the procedure's execution the variable "rez" is formed, which indicates the transport to the fact of reception of full request and the formation of the correct answer (false) or to the necessity for the transport to expect for the remaining data (true).
- If the result of the processing procedure is the variable "rez" with the 'false' and the response in the variable "answer" is not zero, then the transport sends the response and reset the accumulation of "request".
- If the result of the processing procedure is the variable "rez" with 'true' then the transport continues to expect for the data. When it receives the next portion of data they are added to the variable "request" and this procedure is repeated.

2. Part of the protocol for outgoing requests

The protocol of outgoing requests is working in cooperation with the outgoing transport and with the separate object of the "User Protocol". The source of the request through the protocol may be a function of the system-wide API of the user programming of the outgoing transport *int messIO(XMLNodeObj req, string prt)*;, in the parameters of which it must be specified:

- *req* - request as an XML tree with the structure corresponding to the input format of the implemented protocol;
- *prt* - the name of the "UserProtocol" module.

The request which is sent with the aforesaid way is directed to the processing procedure of the protocol's request (Fig. 3) with the user protocol's ID which is specified in the attribute req.attr("ProtIt").

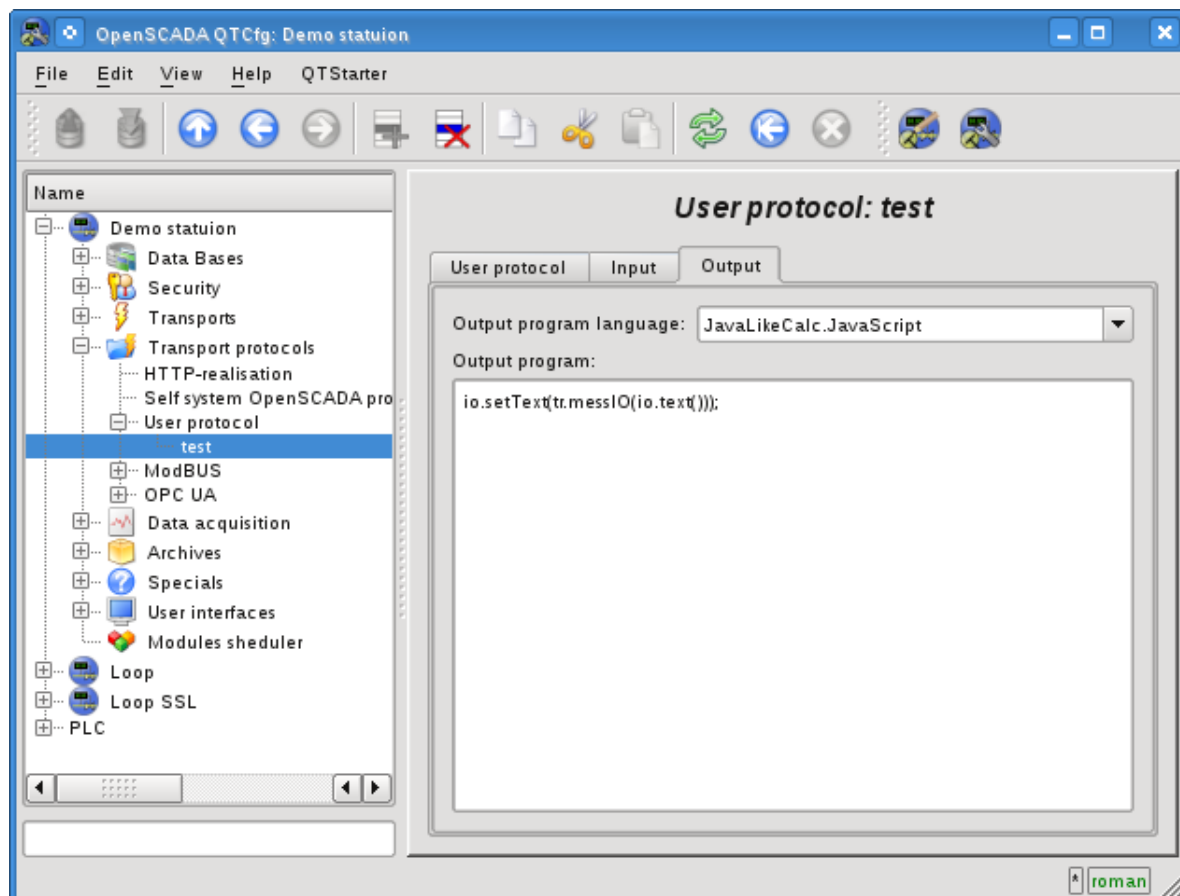


Fig.3. Tab of the processing procedures of the outgoing requests.

The tab of the processing procedure for outgoing requests includes the field to select the internal programming language of OpenSCADA and text field for typing the processing procedure.

For the processing procedure the following exchange variables are predetermined:

- *io* - XML node of the exchange with the client, through which the protocol gets the requests and into which it puts the result with the format implemented in the procedure;
- *tr* - The transport object is provided for the call the transport function *string messIO(string mess, real timeOut = 1000); "tr.messIO(req)".*

The overall scenario of the formation if the outgoing request:

- Building of the XML-tree in accordance with the structure implemented by the protocol and setting of the user protocol identifier in the attribute "ProtIt".
- Sending the request to transport through the protocol *SYS.Transport["Modul"] ["OutTransp"].messIO(req,"UserProtocol");*.
- Selection of the user interface in accordance with *req.attr("ProtIt")* and initialization of variables of outgoing transport *io* - respectively to the first argument *messIO()* and *tr* - object of the "OutTransp".
- Calling the procedure for execution which after the processing the "io" structure forms the direct request to the transport *tr.messIO(req);*, result of which is processed and put back in *io*.

The essence of the allocation the protocol part of the code to the procedure of the user protocol is to facilitate the interface of the client exchange for multiple use and assumes the formation of the structure of XML-node of the exchange as the attributes of the addresses of remote stations, addresses of the read and write variables and the values of the variables themselves. The entire work of direct coding of the request and decoding of the response is assigned to procedure of the user protocol.

The module <FLibComplex1> of the subsystem “Specials”

<i>Module:</i>	FLibComplex1
<i>Name:</i>	Library of functions compatible with SCADA Complex1.
<i>Tun:</i>	Specials
<i>Source:</i>	spec_FLibComplex1.so
<i>Version:</i>	1.0.6
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the library of functions compatible with SCADA Complex1 of the firm SIC “DIYA”.
<i>License:</i>	GPL

Special module FLibComplex1 provides the OpenSCADA system with the static library of functions compatible with SCADA Complex1 of firm SIC 'DIYA'. These functions are used in the SCADA system Complex1 in the form of algoblocks to create inner-computings on the virtual controller. Provision of the library of these functions lets to do the transfer of computational algorithms from the system Complex1.

To address the functions of this library must it is necessary to use the path: <Special.FLibComplex1.*>, where '*' – function identifier in the library.

Below is the description of each function of the librariy. For each function it was evaluated the execution time. Measurements were made on the system with the following parameters: Athlon 64 3000 + (ALTlinux 4.0 (32bit)) by measuring the total execution time of the function when you call it 1000 times. Sampling was carried out of the five calculations, rounded to integer. Time is in angle brackets and is measured in microseconds.

1. Alarm (alarm) <111>

Description: Set alarm sign in the case of going out of the variable for the specified boundary.

Formula:

```
out = if(val>max || val<min) true; else false;
```

2. Condition '<' (cond_lt) <239>

Description: Operation of branching in accordance with the condition “<”.

Formula:

```
out = if(in1<(in2_1*in2_2*in2_3*in2_4)) in3_1*in3_2*in3_3*in3_4;  
      else in4_1*in4_2*in4_3*in4_4;
```

3. Condition '>' (cond_gt) <240>

Description: Operation of branching in accordance with the condition “>”.

Formula:

```
out = if(in1>(in2_1*in2_2*in2_3*in2_4)) in3_1*in3_2*in3_3*in3_4;  
      else in4_1*in4_2*in4_3*in4_4;
```

4. Full condition (cond_full) <513>

Description: Full check of the conditions, including more, less and equal.

Formula:

```
out = if(in1<(in2_1*in2_2*in2_3*in2_4)) in3_1*in3_2*in3_3*in3_4;  
else if( in1>(in4_1*in4_2*in4_3*in4_4) in5_1*in5_2*in5_3*in5_4;  
else in6_1*in6_2*in6_3*in6_4;
```

5. Digital block (digitBlock) <252>

Description: Function contains the control algorithm of digital signals acquisition for valves and pumps that contain: signs of “Open”, “Close” and the command “Open”, “Close”, “Stop”. Supports work with pulse commands, i.e. can read the signal through the specified period of time.

Parameters:

ID	Parameter	Type	Mode
cmdOpen	Command “Open”	Bool	Out
cmdClose	Command “Close”	Bool	Out
cmdStop	Command “Stop”	Bool	Out
stOpen	Position “Opened”	Bool	In
stClose	Position “Closed”	Bool	In
tCmd	Command hold time (s)	Integer	In
frq	Frequency of calculation (milliseconds)	Integer	In

6. Division (div) <526>

Description: Makes division of the set of variables.

Formula:

```
out = (in1_1*in1_2*in1_3*in1_4*in1_5 +  
       in2_1*in2_2*in2_3*in2_4*in2_5 + in3)  
/ (in4_1*in4_2*in4_3*in4_4*in4_5 +  
   in5_1*in5_2*in5_3*in5_4*in5_5 + in6);
```

7. Exponent (exp) <476>

Description: Calculating the exponent under the group of variables.

Formula:

```
out = exp (in1_1*in1_2*in1_3*in1_4*in1_5 +  
           (in2_1*in2_2*in2_3*in2_4*in2_5+in3) /  
           (in4_1*in4_2*in4_3*in4_4*in4_5+in5) )
```

8. Flow (flow) <235>

Description: Calculation of the gas flow.

Formula:

```
f = K1* ((K3+K4*x) ^K2);
```

9. Iterator (increment) <181>

Description: Iterative calculation with the increment specifying. Gain ratio for different directions is different.

Formula:

```
out = if( in1 > in2 ) in2 + in3*(in1-in2);  
       else in2 - in4*(in2-in1);
```


10. Lag (lag) <121>

Description: Variation of the variable lag. Practice, this is the filter without reference to time.

Formula:

$$y = y - Klag * (y - x);$$

11. Simple multiplication(mult) <259>

Description: Simple multiplication with division.

Formula:

$$\text{out} = (\text{in1_1} * \text{in1_2} * \text{in1_3} * \text{in1_4} * \text{in1_5} * \text{in1_6}) / (\text{in2_1} * \text{in2_2} * \text{in2_3} * \text{in2_4});$$

12. Multiplication + Division(multDiv) <468>

Description: Branched multiplication + division.

Formula:

$$\text{out} = \text{in1_1} * \text{in1_2} * \text{in1_3} * \text{in1_4} * \text{in1_5} * (\text{in2_1} * \text{in2_2} * \text{in2_3} * \text{in2_4} * \text{in2_5} + (\text{in3_1} * \text{in3_2} * \text{in3_3} * \text{in3_4} * \text{in3_5}) / (\text{in4_1} * \text{in4_2} * \text{in4_3} * \text{in4_4} * \text{in4_5}));$$

13. PID regulator (pid) <745>

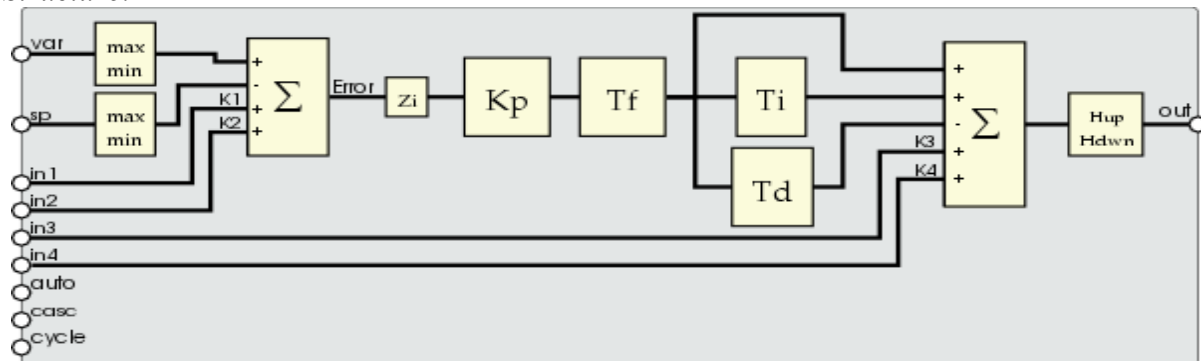
Description: Proportional-integral-differential regulator.

Parameters:

ID	Parameter	Type	Mode	By defaults
var	Variable	Real	In	0
sp	Set point	Real	Out	0
max	Maximum of scale	Real	In	100
min	Minimum of scale	Real	In	0
manIn	Manual input (%)	Real	In	0
out	Out (%)	Real	Return	0
auto	Auto	Bool	In	0
cas	Cascade	Bool	In	0
Kp	Kp	Real	In	1
Ti	Ti (ms)	Integer	In	1000
Kd	Kd	Real	In	1
Td	Td (ms)	Integer	In	0
Tzd	Td lag (ms)	Integer	In	0
Hup	Upper limit of the out (%)	Real	In	100
Hdwn	Lower limit of the out (%)	Real	In	0
Zi	Insensitivity (%)	Real	In	1
followSp	Follow sp from var on manual	Bool	In	1
K1	Koef. of the input 1	Real	In	0
in1	Input 1	Real	In	0
K2	Koef. of the input 2	Real	In	0
in2	Input 2	Real	In	0
K3	Koef. of the input 3	Real	In	0
in3	Input 3	Real	In	0

ID	Parameter	Type	Mode	By defaults
K4	Koef. of the input 4	Real	In	0
in4	Input 4	Real	In	0
f_frq	Frequency of calculation (Hz)	Real	In	1

Structure:



14. Power (pow) <564>

Description: Raising to the power

Formula:

```
out = (in1_1*in1_2*in1_3*in1_4*in1_5) ^
      (in2_1*in2_2*in2_3*in2_4*in2_5 +
       (in3_1*in3_2*in3_3*in3_4*in3_5) /
       (in4_1*in4_2*in4_3*in4_4*in4_5));
```

15. Selection (select) <156>

Description: Selection of the one from four options.

Formula:

```
out = if( sel = 1 ) in1_1*in1_2*in1_3*in1_4;
      if( sel = 2 ) in2_1*in2_2*in2_3*in2_4;
      if( sel = 3 ) in3_1*in3_2*in3_3*in3_4;
      if( sel = 4 ) in4_1*in4_2*in4_3*in4_4;
```

16. Simple integrator (sum) <404>

Description: A simple summation with the multiplication.

Formula:

```
out = in1_1*in1_2+in2_1*in2_2 + in3_1*in3_2+in4_1*in4_2 +
      in5_1*in5_2+in6_1*in6_2 + in7_1*in7_2+in8_1*in8_2;
```

17. Sum with the division (sum_div) <518>

Description: The summation the set of values with the division.

Formula:

```
out = in1_1*in1_2*(in1_3+in1_4/in1_5) +
      in2_1*in2_2*(in2_3+in2_4/in2_5) +
      in3_1*in3_2*(in3_3+in3_4/in3_5) +
      in4_1*in4_2*(in4_3+in4_4/in4_5);
```

18. Sum with the multiplication. (sum_mult) <483>

Description: The summation the set of values with the multiplication.

Formula:

$$\begin{aligned} \text{out} = & \text{in1_1} * \text{in1_2} * (\text{in1_3} * \text{in1_4} + \text{in1_5}) + \\ & \text{in2_1} * \text{in2_2} * (\text{in2_3} * \text{in2_4} + \text{in2_5}) + \\ & \text{in3_1} * \text{in3_2} * (\text{in3_3} * \text{in3_4} + \text{in3_5}) + \\ & \text{in4_1} * \text{in4_2} * (\text{in4_3} * \text{in4_4} + \text{in4_5}); \end{aligned}$$

The module <FLibMath> of the subsystem “Specials” <FLibMath>

<i>Module:</i>	FLibMath
<i>Name:</i>	The library of standard mathematical functions.
<i>Type:</i>	Specials
<i>Source:</i>	spec_FLibMath.so
<i>Version:</i>	0.5.2
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the library of standard mathematical functions.
<i>License:</i>	GPL

Special FLibMath module provides the library of standard mathematical functions into the system.

For addressing to the functions of the library you must use the path: <Special.FLibMath.*>. Where '*' – function's identifier in the library.

1. Functions

Table 1 provides a description of each function of the library. For each function the evaluation time of execution is measured. The measurement was made on a system with the following parameters: Athlon 64 3000 + (ALTLinux 3.0 (32bit)), by measuring the total time of execution of the function, while calling it 1000 times.

Table 1: The functions of the library of standard mathematical functions

Id	Name	Description	Time (micro-seconds)
abs	Module	Math. function – the number module.	81
acos	Anti-cosine	Math. function – anti-cosine.	149
asin	Anti-sine	Math. function – anti-sine.	140
atan	Anti-tangent	Math. function – anti-tangent.	109
ceil	Rounding up to a larger	Math. function – rounding up to a larger integer.	96
cos	Cosine	Math. function – cosine.	93
cosh	Hyperbolic cosine	Math. function – hyperbolic cosine.	121
exp	Exponent	Math. function – exponent.	145
floor	Rounding to the lower	Math. function – rounding to the lower integer	95
if	If Condition	Condition function – “If”.	92
lg	Common logarithm	Math. function – common logarithm.	168
ln	Natural logarithm	Math. function – natural logarithm.	185
pow	Power	Math. function – involution.	157
rand	Random number	Math. function – random number generator.	147
sin	Sine	Math. function – sine.	127
sinh	Hyperbolic sine	Math. function – hyperbolic sine.	199

Id	Name	Description	Time (micro-seconds)
sqrt	The square root	Math. function – the square root.	94
tan	Tangent	Math. function – tangent.	153
tanh	Hyperbolic tangent	Math. function – hyperbolic tangent.	177

The module <FLibSYS> of the subsystem “Specials”

<i>Module:</i>	FLibSYS
<i>Name:</i>	Library of system API functions.
<i>Type:</i>	Specials
<i>Source:</i>	spec_FLibSYS.so
<i>Version:</i>	0.9.2
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the library of system API of user programming area.
<i>License:</i>	GPL

Special module FLibSYS provides static library of functions for working with the OpenSCADA system at the level of its system API. These functions can be used in an user programming area of OpenSCADA system for the organization of not ordinary interaction algorithms.

To address the functions of this library must it is necessary to use the path: <Special.FLibSYS.*>, where '*' – function identifier in the library.

Below is the description of each function of the librariy. For each function it was evaluated the execution time. Measurements were made on the system with the following parameters: Athlon 64 3000 + (ALTLinux 4.0 (32bit)) by measuring the total execution time of the function when you call it 1000 times. Sampling was carried out of the five calculations, rounded to integer. Time is in angle brackets and is measured in microseconds.

1. System-wide functions

1.1. Calling the console commands and operating system utilities (sysCall)

Description: Call the console commands of the OS. The function offers great opportunities to the OpenSCADA user by calling any system software, utilities and scripts, as well as getting the access to the huge volume of system data by means of them. For example the command “ls-l” returns the detailed contents of the working directory.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
com	Command	String	In	

Example:

```
using Special.FLibSYS;  
test=sysCall("ls -l");  
messPut("Example",0,"Example: "+test);
```

1.2. SQL query (dbReqSQL)

Description: Formation of the SQL-query to DB.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	Object(Array)	Return	
addr	DB address	String	In	
req	SQL-query	String	In	

1.3. XML node (xmlNode)

Description: Creation of the XML node object.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	Object(XMLNodeObj)	Return	
name	Name	String	In	

Example:

```
using Special.FLibSYS;
//Creating the "get" object of the XML node.
Req = xmlNode("get");
//Creating the "get" object of the XML node with creating attributes.
//sub_DAQ/mod_ModBus/cntr_1/prm_1 - The path in accord of project structure.
Req = xmlNode("get").setAttr("path", "/sub_DAQ/mod_ModBus/cntr_1/prm_1/%2fprm%2fst%2fen");
```

1.4. Request of the management interface (xmlCntrReq)

Description: Request of the management interface to the system via XML. The usual request is written in the form `<get path="/OPat/%2felem"/>`. When we indicate the station the request to the external station is made.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
req	Request	Object(XMLNodeObj)	Out	
stat	Station	String	In	

Example:

```
using Special.FLibSYS;
//Getting status "Off/On" of the parameter "1" of the controller "1"
//of the module "ModBus".
//sub_DAQ/mod_ModBus/cntr_1/prm_1 - The path in accord of project structure.
req = xmlNode("get").setAttr("path", "/sub_DAQ/mod_ModBus/cntr_1/prm_1/%2fprm%2fst%2fen");
rez = xmlCntrReq(req);
messPut("test", 0, "Example: "+req.text());

//Setting status "On" of the parameter "1" of the controller "1"
//of the module "ModBus".
req = xmlNode("set").setAttr("path", "/sub_DAQ/mod_ModBus/cntr_1/prm_1/%2fprm%2fst%2fen").setText(1);
rez = xmlCntrReq(req);

//Setting status "Off" of the parameter "1" of the controller "1"
//of the module "ModBus".
req = xmlNode("set").setAttr("path", "/sub_DAQ/mod_ModBus/cntr_1/prm_1/%2fprm
```

```
%2fst%2fen").setText(0);
rez = xmlCntrReq(req);
```

1.5. Values archive (vArh)

Description: Getting the object of the values archive (VArchObj) by connecting to the archive using its address.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	Object(VArchObj)	Return	
name	Name and address to the attribute of the parameter with the archive or directly to the archive of values.	String	In	

VArchObj object

Functions:

- *begin(usec, archivor)* — Getting the start time of the archive through the return of seconds and microseconds *<usec>* for the archivor *<archivor>*.
- *end(usec, archivor)* — Getting the end time of the archive through the return of seconds and microseconds *<usec>* for the archivor *<archivor>*.
- *period(usec, archivor)* — Getting the periodicity of the archive through the return of seconds and microseconds *<usec>* for the archivor *<archivor>*.
- *get(sec, usec, upOrd, archivor)* — Getting the value from the archive at the time *<sec>:<usec>* linked to the top *<upOrd>* for the archivor *<archivor>*. Real time of the value obtained is set in *<sec>:<usec>*.
- *set(val, sec, usec)* — Writing of the value *<val>* in the archive buffer for the time *<sec>:<usec>*.
- *copy(src, begSec, begUSec, endSec, endUSec, archivor)* — Copying of the part of the source archive *<src>* or its buffer in the current beginning from *<begSec>:<begUSec>* and ending with *<endSec>:<endUSec>* for the archivor *<archivor>*.
- *FFT(tm, size, archivor, tm_usec)* -- Performs the Fast Fourier Transformation using the FFT algorithm. Returns an array of amplitudes of the frequencies for archive's values window for begin time *<tm>:<tm_usec>* (seconds:microseconds), depth to history *<size>* (seconds) and for archivor *<archivor>*.

Example:

```
using Special.FLibSYS;
val = vArh(strPath2Sep(addr)).get(time,uTime,0,archtor);
return val.isEval() ? "Empty" : real2str(val,prec);
```

1.6. Buffer of the values archive (vArhBuf)

Description: Getting the object of the buffer of the values archive (VArchObj) to perform the intermediate operations on frames of data.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	Object(VArchObj)	Return	
tp	Type of the values of the archive (0-Boolean, 1-Integer, 4-Real, 5-String)	Integer	In	1
sz	Maximum buffer size	Integer	In	100
per	periodicity of buffer (in microseconds)	Integer	In	1000000
hgrd	Mode “Hard time grid”	Boolean	In	0
hres	Mode «High time resolution (microseconds)”	Boolean	In	0

2. Functions for the astronomical time processing

2.1. Time string (tmFStr) <3047>

Description: Converts an absolute time in the string of the required format. Recording of the format corresponds to the POSIX-function strftime.

Parameters:

ID	Parameter	Type	Mode	By defaults
val	Full date string	String	Return	
sec	Seconds	Integer	In	0
form	Format	String	In	%Y-%m-%d %H:%M:%S

Example:

```
using Special.FLibSYS;  
test=tmFStr(SYS.time(), "%d %m %Y");  
messPut("Example", 0, "tmFStr(): "+test);
```

2.2. Full Date (tmDate) <973>

Description: Returns the full date in seconds, minutes, hours, etc., based on the absolute time in seconds from the epoch of 1/1/1970.

Parameters:

ID	Parameter	Type	Mode	By defaults
fullsec	Full seconds	Integer	In	0
sec	Seconds	Integer	Out	0
min	Minutes	Integer	Out	0
hour	Hours	Integer	Out	0
mday	Day of the month	Integer	Out	0
month	Month	Integer	Out	0
year	Year	Integer	Out	0
wday	Day of the week	Integer	Out	0
yday	Day of the year	Integer	Out	0
isdst	Daylight saving time	Integer	Out	0

Example:

```
using Special.FLibSYS;  
curMin=curHour=curDay=curMonth=curYear=0;  
tmDate(tmTime(), 0, curMin, curHour, curDay, curMonth, curYear);  
messPut("test", 0, "Current minute: "+curMin);  
messPut("test", 0, "Current hour: "+curHour);  
messPut("test", 0, "Current day: "+curDay);  
messPut("test", 0, "Current month: "+curMonth);  
messPut("test", 0, "Current Year: "+curYear);
```

2.3. Absolute time (tmTime) <220>

Description: Returns the absolute time in seconds from the epoch and in microseconds, if <usec> is installed in a non-negative value.

Parameters:

ID	Parameter	Type	Mode	By defaults
sec	Seconds	Integer	Return	0
usec	Microseconds	Integer	Out	-1

2.4. Conversion the time from the symbolic representation to the time in seconds from the epoch of 1/1/1970 (tmStrPTime) <2600>

Description: Returns the time in seconds from the epoch of 1/1/1970, based on the string record of time, in accordance with the specified template. For example, template "%Y-%m-%d %H:%M:%S" corresponds the time «2006–08–08 11:21:55». Description of the format of the template can be obtained from the documentation on POSIX-function “strptime”.

Parameters:

ID	Parameter	Type	Mode	By defaults
sec	Seconds	Integer	Return	0
str	Date string	String	In	
form	Date record format	String	In	%Y-%m-%d %H:%M:%S

Example:

```
using Special.FLibSYS;  
curMin=curHour=curDay=curMonth=curYear=0;  
tmDate(tmTime(),0,curMin,curHour,curDay,curMonth,curYear);  
test = tmStrPTime(""+curYear+"-"+(curMonth+1)+"-"+curDay+" 9:0:0","%Y-%m-%d %H:%M:%S");  
messPut("Example",0,"tmStrPTime(): "+test);
```

2.5. Planning of the time in the Cron format (tmCron)

Description: Returns the time planned in the format of the Cron standard beginning from the base time of from the current time, if the base is not specified.

Parameters:

ID	Parameter	Type	Mode	By defaults
res	Result	Integer	Return	0
str	Record in the Cron standard	String	In	* * * * *
base	Base time	Integer	In	0

3. Functions of the messages processing

3.1. Messages request (messGet)

Description: Request of the system messages.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	Object(Array)	Return	
btm	Start time	Integer	In	
etm	End time	Integer	In	
cat	Category of the message	String	In	
lev	Level of the message	Integer	In	
arch	Archivator	String	In	

3.2. Generation of the message (messPut)

Description: Formation of the system message.

Parameters:

ID	Parameter	Type	Mode	By defaults
cat	Category of the message	String	In	
lev	Level of the message	Integer	In	
mess	Text of the message	String	In	

Example:

```
rnd_sq_gr11_lineClr="red";  
Special.FLibSYS.messPut("Example",1,"Event: "+rnd_sq_gr12_leniClr);
```

4. Functions of the strings processing

4.1. Getting the size of the string (strSize) <114>

Description: It is used to get the size.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	Integer	Return	
str	String	String	In	

Example:

```
Special.FLibSYS.messPut("Example",1,"ReturnString: "+strSize("Example"));
```

4.2. Getting the part of the string (strSubstr) <413>

Description: It is used to det the part of the string.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
str	String	String	In	
pos	Position	Integer	In	0
n	Quantity	Integer	In	-1

Example:

```
using Special.FLibSYS;  
test=strSubstr("Example", 0, strSize("Example")-1);  
messPut("Example",1,"ReturnString: "+test);
```

4.3. Insert of the on string to the another (strInsert) <1200>

Description: It is used to insert of the on string to the another.

Parameters:

ID	Parameter	Type	Mode	By defaults
str	String	String	Out	
pos	Position	Integer	In	0
ins	Inserting string	String	In	

4.4. Change the part of the string with the another one (strReplace) <531>

Description: It is used to change the part of the string with the another one.

Parameters:

ID	Parameter	Type	Mode	By defaults
str	String	String	Out	
pos	Позиция	Integer	In	0
n	Quantity	Integer	In	-1
repl	Changing string	String	In	

4.5. Parsing the string on separator (strParse) <537>

Description: It is used to parse the string on separator.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
str	String	String	In	
lev	Level	Integer	In	
sep	Separator	String	In	."
off	Offset	Integer	Out	

Example:

```
using Special.FLibSYS;  
ExapleString="Example:123";  
test=strParse(ExapleString,1,":");  
messPut("Example",0,"strParse(): "+test);
```

4.6. Path parsing (strParsePath) <300>

Description: It is used for the parsing the path on the elements.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
path	Path	String	In	
lev	Level	Integer	In	
off	Offset	Integer	Out	

Example:

```
using Special.FLibSYS;  
test=strParsePath(path,0,"/");  
messPut("Example",1,"strParsePath(): "+test);
```

4.7. Path to the string with the separator (strPath2Sep)

Description: It is used to convert the path to the string with the separator.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
src	Source	String	In	
sep	Separator	String	In	."

Example:

```
//Converting value "/ses_AGLKS/pg_so" of the attribute "path"
```

```
//into value "ses_AGLKS.pg_so"
using Special.FLibSYS;
test = strPath2Sep(path);
messPut("Example",0,"path: "+path);
messPut("Example",0,"strPath2Sep(): "+test);
```

4.8. Coding of the string to HTML (strEnc2HTML)

Description: It is used to code the string for using in the HTML source.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
src	Source	String	In	

4.9. Encode text to bin (strEnc2Bin)

Description: Use for encode text to bin, from format <00 A0 FA DE>.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
src	Source	String	In	

4.10. Decode text from bin (strDec4Bin)

Description: Use for decode text from bin to format <00 A0 FA DE>.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
src	Source	String	In	

4.11. Convert real to string (real2str)

Description: It is used to convert real to string.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
val	Value	Real	In	
prc	Precision	Integer	In	4
tp	Type	String	In	"f"

4.12. Convert integer to string (int2str)

Description: It is used to convert integer to string.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	String	Return	
val	Value	Integer	In	
base	Base, supported: 8, 10, 16	Integer	In	10

4.13. Convert the string to real (str2real)

Description: It is used to convert string to real.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	Real	Return	
val	Value	String	In	

4.14. Convert the to integer (str2int)

Description: It is used to convert string to integer.

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	Integer	Return	
val	Value	String	In	
base	Base	Integer	In	0

5. Functions for the real processing

5.1. Splitting the float to the words (floatSplitWord) <56>

Description: Splitting the float (4 bites) to the words (2 bites).

Parameters:

ID	Parameter	Type	Mode	By defaults
val	Value	Real	In	
w1	Word 1	Integer	Out	
w2	Word 2	Integer	Out	

5.2. Merging the float from words (floatMergeWord) <70>

Description: Merging the float (4 bites) from words (2 bites).

Parameters:

ID	Parameter	Type	Mode	By defaults
rez	Result	Real	Return	
w1	Word 1	Integer	In	
w2	Word 2	Integer	In	

The module <SystemTests> of the subsystem "Specials"

<i>Module:</i>	SystemTests
<i>Name:</i>	OpenSCADA system tests.
<i>Type:</i>	Specials
<i>Source:</i>	spec_SystemTests.so
<i>Version:</i>	1.5.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the group of test to the OpenSCADA system.
<i>License:</i>	GPL

Special module SystemTests contains a set of tests designed to test various subsystems and components of the OpenSCADA system. Tests carried out in the form of user API functions. Hence the tests can be run as a one-time, in the "Execute" page of the function's object and from the procedures of the user as well, passing them the necessary arguments.

In addition to the usual mechanisms of user API execution an autonomous mechanism is provided. This mechanism is represented by the separate task, performed with the period of one second, which calls the functions of tests in accordance with the settings in the configuration file.

The configuration fields of the tests are placed in the section of the modulus SystemTests of subsystem "Special". The format of the configuration fields is: **<prm id="Test Id" on="1" per="10" />** Where:

- id - test ID;
- on - sign "Test is enabled";
- per - repetition period of the test (seconds).

In addition to the basic attributes the reflection of the input parameters of tests' functions on the same name attributes of tag "prm" is made. For example, the attribute "name" of function "Param", you can specify in the tag "prm".

It is allowed to indicate the set of tags "prm" for the same or different tests with the same or different parameters, thus indicating the separate test execution with the specified parameters. Here is an example of description of all available tests:

```
<?xml version="1.0" encoding="UTF-8" ?>
<OpenSCADA>
  <station id="DemoStation">
    <node id="sub_Special">
      <node id="mod_SystemTests">
        <prm id="Param" on="0" per="5" name="LogicLev.experiment.F3"/>
        <prm id="XML" on="0" per="10" file="/etc/oscada.xml"/>
        <prm id="Mess" on="0" per="10" categ="" arhtor="DBArch.test3"
          depth="10"/>
        <prm id="SOAttach" on="0" per="20"
          name="../../lib/openscada/daq_LogicLev.so" mode="0"
          full="1"/>
        <prm id="Val" on="0" per="1" name="LogicLev.experiment.F3.var"
          arch_len="5" arch_per="1000000"/>
        <prm id="Val" on="0" per="1" name="System.AutoDA.CPULoad.load"
          arch_len="10" arch_per="1000000"/>
        <prm id="DB" on="0" per="10" type="MySQL"
          addr="server.diya.org;roman;123456;oscadaTest" table="test"
          size="1000"/>
        <prm id="DB" on="0" per="10" type="DBF" addr="./DATA/DBF"
```

```

        table="test.dbf" size="1000"/>
<prm id="DB" on="0" per="10" type="SQLite" addr="./DATA/test.db"
    table="test" size="1000"/>
<prm id="DB" on="0" per="10" type="FireBird"
    addr="server.diya.org:/var/tmp/test.fdb;roman;123456"
    table="test" size="1000"/>
<prm id="TrOut" on="0" per="1" addr="TCP:127.0.0.1:10001"
    type="Sockets" req="time"/>
<prm id="TrOut" on="0" per="1" addr="UDP:127.0.0.1:10001"
    type="Sockets" req="time"/>
<prm id="TrOut" on="0" per="1" addr="UNIX:./oscada" type="Sockets"
    req="time"/>
<prm id="TrOut" on="0" per="1" addr="UDP:127.0.0.1:daytime"
    type="Sockets" req="time"/>
<prm id="SysContrLang" on="0" per="10"
    path="/Archive/FSArch/mess_StatErrors/%2fprm%2fst"/>
<prm id="ValBuf" on="0" per="5"/>
<prm id="Archive" on="0" per="30" arch="test1" period="1000000"/>
<prm id="Base64Code" on="0" per="10"/>
    </node>
</node>
</station>
</OpenSCADA>

```

1. Parameter (Param)

Description: Test of the DAQ parameters. Reads the attributes and configuration fields of the parameter.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
name	Address of the DAQ parameter	String	Input	System.AutoDA.CPULoad

2. XML parsing (XML)

Description: Test of the XML file parsing. Parses and displays the structure of the file.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
file	XML file	String	Input	

3. Messages (Mess)

Description: Test of the messages archive. Periodically reads new messages from the archive for the specified archiver.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
arhtor	Archiver	String	Input	FSArch.StatErrors
categ	The template of the messages category	String	Input	
depth	Message's depth (s)	Integer	Input	10

4. SO attaching (SOAttach)

Description: Test connection/disconnection of the modules.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
name	Path to the module	String	Input	
mode	Mode (1-connect;-1-disconnect;0-change)	Integer	Input	0
full	Full connection(when start)	Bool	Input	1

5. Attribute of the parameter (Val)

Description: Test the attribute values of the parameter. Performs periodic polling of the last value of the specified attribute, as well as a survey archive to the specified depth.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
name	Path to the attribute of the parameter	String	Input	System.AutoDA.CPULoad.load
arch_len	The depth of the query to the values' archive (s)	Integer	Input	10
arch_per	Period of query to the values' archive (mcs)	Integer	Input	1000000

6. DB test (DB)

Description: Complete database test. Includes:

- creating/opening of the DB;
- creating/opening of the table;
- creation of set of records (rows) of the predetermined structure;
- modification of the set of records;
- receiving and verifying the values of the set of records;
- modifying the structure of records and table;
- delete of the records;
- closing/deleting of the table;
- closing/deleting of the database.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
type	DB type	String	Input	SQLite
addr	DB address	String	Input	./DATA/test.db
table	DB table	String	Input	test
size	Number of records	Integer	Input	1000

7. Transport (TrOut)

Description: Test of the output and/or input transports. Performs testing of the output transport by sending the request to the specified input transport.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
addr	Address	String	Input	TCP:127.0.0.1:10001
type	Transport's module	String	Input	Sockets
req	Query text	String	Input	

8. Control system language (SysContrLang)

Description: Test of the control system language. Performs the query of the language elements through the full path. Full path to the element of language is of the form of system control </Archive/%2fbd%2fm_per>. Full path consists of two sub-paths. The first one </d_Archive/> is the path to the node of the control tree. The second one </bd/m_per> is the path to a particular element of the node.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
path	Path to the element of language	String	Input	/Archive/BaseArh/mess_StatErrors/%2fprm%2fst

9. Values buffer (ValBuf)

Description: Tests of the values' buffer. Contains 13 tests of the all aspects of the values' buffer (subsystem "Archives").

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	

10. Values archive (Archive)

Description: Tests of the placing of values in the archive. Contains 7 (8) tests of the values archiver to verify the correctness of the functioning of a coherent mechanism for packaging.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	
arch	Values archive	String	Input	
period	Values period (mcs)	Integer	Input	1000000

11. Base64 code (Base64Code)

Description: Tests of the Mime Base64 encoding algorithm.

Parameters:

ID	Name	Type	Mode	By defaults
rez	Result	String	Return	

The module of subsystems “User Interfaces” <QTStarter>

<i>Module:</i>	QTStarter
<i>Name:</i>	QT GUI starter
<i>Type:</i>	User Interfaces
<i>Source:</i>	ui_QTStarter.so
<i>Version:</i>	1.6.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the QT GUI starter. QT-starter is the only and compulsory component for all GUI modules based on the QT library.
<i>License:</i>	GPL

The module <QTStarter> provides the system OpenSCADA with the starter of QT GUI modules. A separate module of running the QT GUI modules is needed because of the need for single-flow execution of all components and centralized initialization of the main object of the QT-library - QApplication.

To run a QT GUI modules advanced interface of callback of functions of modules is used. This interface involves exporting of functions by the external modules. In our case, QT GUI modules must export the following functions:

- *QIcon icon()*; - Sends an object of icon of the called module.
- *QMainWindow *openWindow()*; - Creates an object of the main window of the QT GUI module, and passes it to the starter. It can return NULL in the case of the failure to create a new window.

For identification QT GUI module must identify the information item of the module "SubType" as "QT". Based on this feature "Starter" works with it.

After receiving the object of the main window "Starter" adds its own control panel and menu item in the window and runs it. Starter control panel contains icons to call all the available QT GUI modules. To except the addition of the control panel or the menu item, the module, which contains the window, can specify the properties of "QTStarterToolDis" or "QTStarterMenuDis" respectively.

For the specifying QT GUI modules that run at startup, the starter module contains the configuration field StartMod. In this field the identifiers of running modules are recorded via ';'. StartMod configuration field can be described in the configuration file, as well as in the system database table through dialog of configuration of the module (Fig. 1).

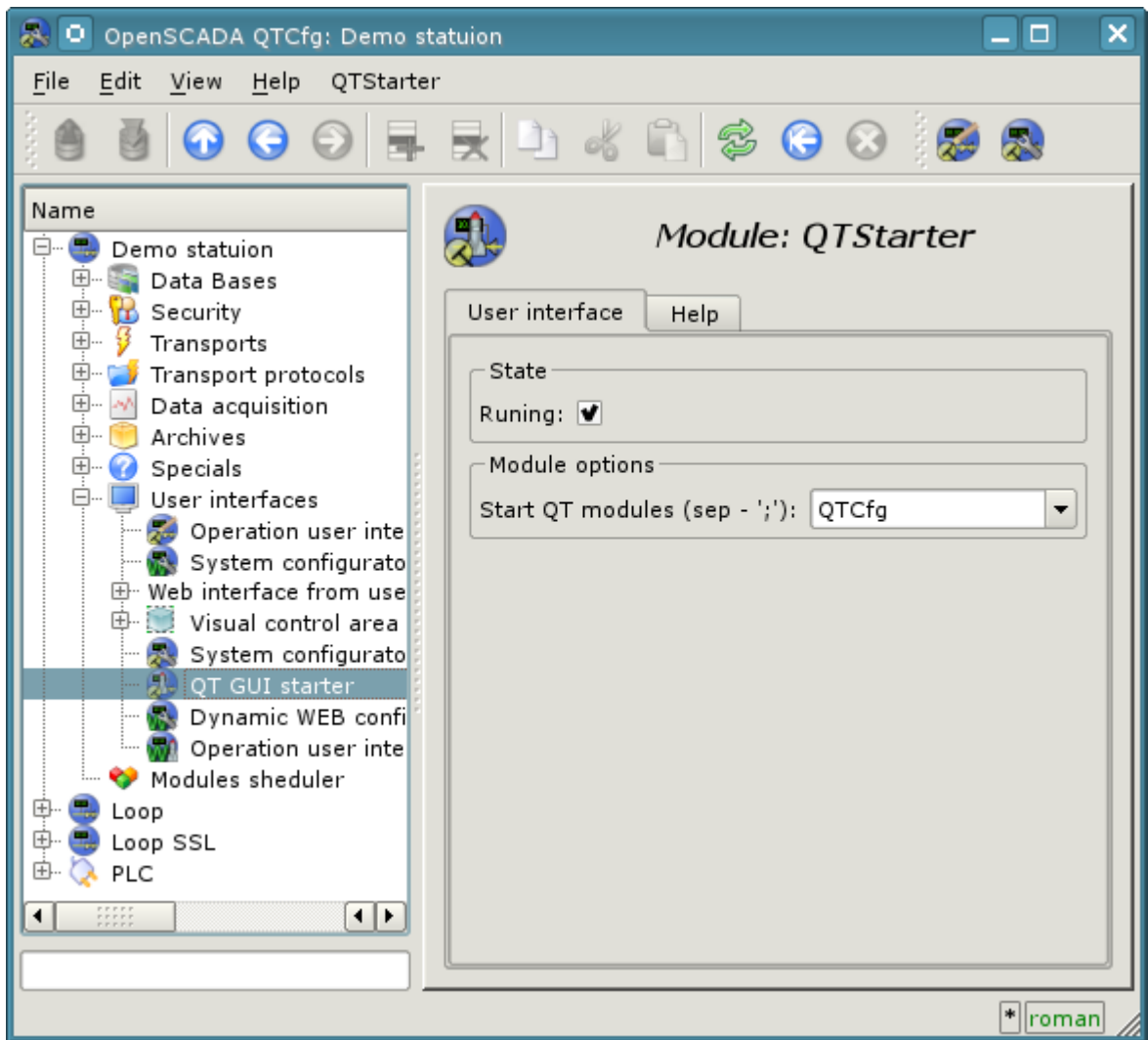


Fig.1. The module configuration page.

In the case of closing the windows of all QT GUI modules "Starter" creates a dialog box that offers to choose the available QT GUI modules, or shut down the system OpenSCADA. The view dialog box is given in the Figure 2.

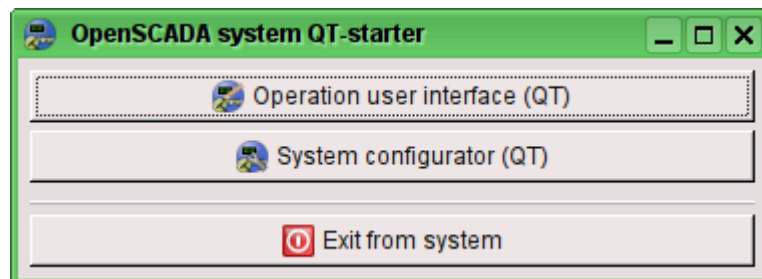


Fig.2. The dialog window of the "Starter".

The module <QTCfg> of subsystems “User Interfaces”

Module:	QTCfg
Name:	The system configurator (QT)
Type:	User Interfaces
Source:	ui_QTCfg.so
Version:	1.9.0
Author:	Roman Savochenko
Translated:	Maxim Lysenko
Description:	Provides the QT-based configurator of the OpenSCADA system.
License:	GPL

The <QTCfg> module provides the configurator of the OpenSCADA system. Configurator is based on multi-platform library of the graphical user interface (GUI) of the firm TrollTech – QT <<http://www.trolltech.com/qt/>>.

At the core of the module lies the management interface language of the OpenSCADA system, and thus provides a uniform configuration interface. Update of the module may be required only in the case of updating the specification of the language of the management interface.

Lets examine the working window of the configurator in Fig. 1.

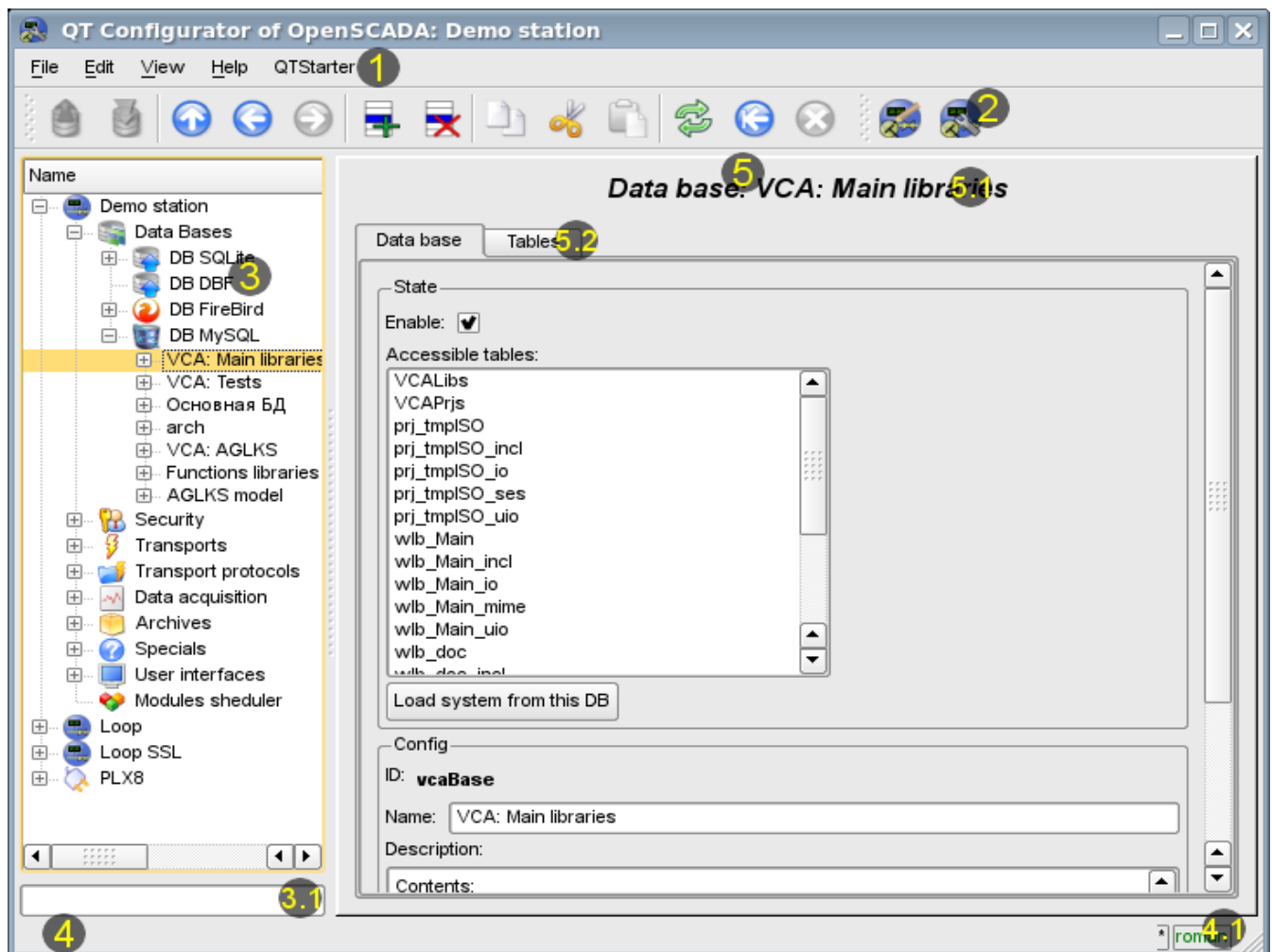


Fig.1. Working window of the configurator

Operating window of the configurator consists of the following parts:

- 1 – Menu — contains a drop-down configurator menu.
- 2 – Toolbar-- contains buttons of quickly control.
- 3 – Navigator — is intended for direct navigation of the control tree.
 - 3.1 – Text enter field for elements search into current branch of tree.
- 4 – Status line — indicating the status of the configurator.
 - 4.1 – Indicator/choice of the user — displays the current user. By double-clicking the user selection dialog opens. As well as an indicator of changes in configuration.
- 5 – Workplace field. It is divided into parts:
 - 5.1 – Node name – contains the name of the current node.
 - 5.2 – Tabulator of the working areas – the root page (management areas) of the node are placed into the tabulator. The management areas of the following levels are placed on the information panel.

Menu of the configurator contains the following items:

- *File* — the group of general commands:
 - *Load from DB* — downloads the selected object or branch of object from the database.
 - *Save to DB* — save the selected object or branch of object to the database.
 - *Close* — close the configurator window.
 - *Quit* — termination of the OpenSCADA system.
- *Edit* — editing commands:
 - *Add* — add a new object to the container.
 - *Delete* — delete the selected object.
 - *Copy item* — copy the selected object.
 - *Cut item* — cut of the selected object. The original object is removed after paste.
 - *Paste item* — paste of the copied or cut item.
- *View* — navigation and control of the view commands:
 - *Up* — climb up the tree.
 - *Previous* — open the previous page.
 - *Next* — open the following page.
 - *Refresh* — refresh the current page.
 - *Start* — run periodically update of the contents of the current page with an interval of one second.
 - *Stop* — stop periodically update of the contents of the current page with an interval of one second.
- *Help* — assistance call commands:
 - *About* — information about the module and the OpenSCADA system.
 - *About Qt* — information about the Qt library.
 - *What's this* — the command of the request the information about the elements of the interface.

The toolbar contains the following management buttons (from left to right):

- *Load from DB* — downloads the selected object or branch of object from the database.
- *Save to DB* — save the selected object or branch of object to the database.
- *Up* — climb up the tree.
- *Previous* — open the previous page.
- *Next* — open the following page.
- *Add* — add a new object to the container.
- *Delete* — delete the selected object.
- *Copy item* — copy the selected object.
- *Cut item* — cut of the selected object. The original object is removed after paste.
- *Paste item* — paste of the copied or cut item.
- *Refresh* — refresh the current page.
- *Start* — run periodically update of the contents of the current page with an interval of one second.
- *Stop* — stop periodically update of the contents of the current page with an interval of one second.

- Call buttons of the modules of the graphical interface based on the QT library

In the navigation tree the context menu of following contents is supported:

- *Load from DB* — downloads the selected object or branch of object from the database.
- *Save to DB* — save the selected object or branch of object to the database.
- *Add* — add a new object to the container.
- *Delete* — delete the selected object.
- *Copy item* — copy the selected object.
- *Cut item* — cut of the selected object. The original object is removed after paste.
- *Paste item* — paste of the copied or cut item.
- *Refresh the elements of a tree* — Performs the refreshing of the navigation tree contents.

The control tools are divided into basic, commands, lists, tables and images. All items are displayed in the sequence strictly appropriate to their location in the description of language of management interface.

1. Configuration

To adjust your own behavior in the not obvious situations module provides the ability to customize individual settings through the management interface of the OpenSCADA (Fig. 2). These parameters are:

- Initial path to the configurator — allows to determine what page to open when you start the configurator.
- Initial user of the configurator — points on behalf of the which user to open configuration without requiring a password.
- The link to the configuration page of the external OpenSCADA stations used to enable the remote configuration.

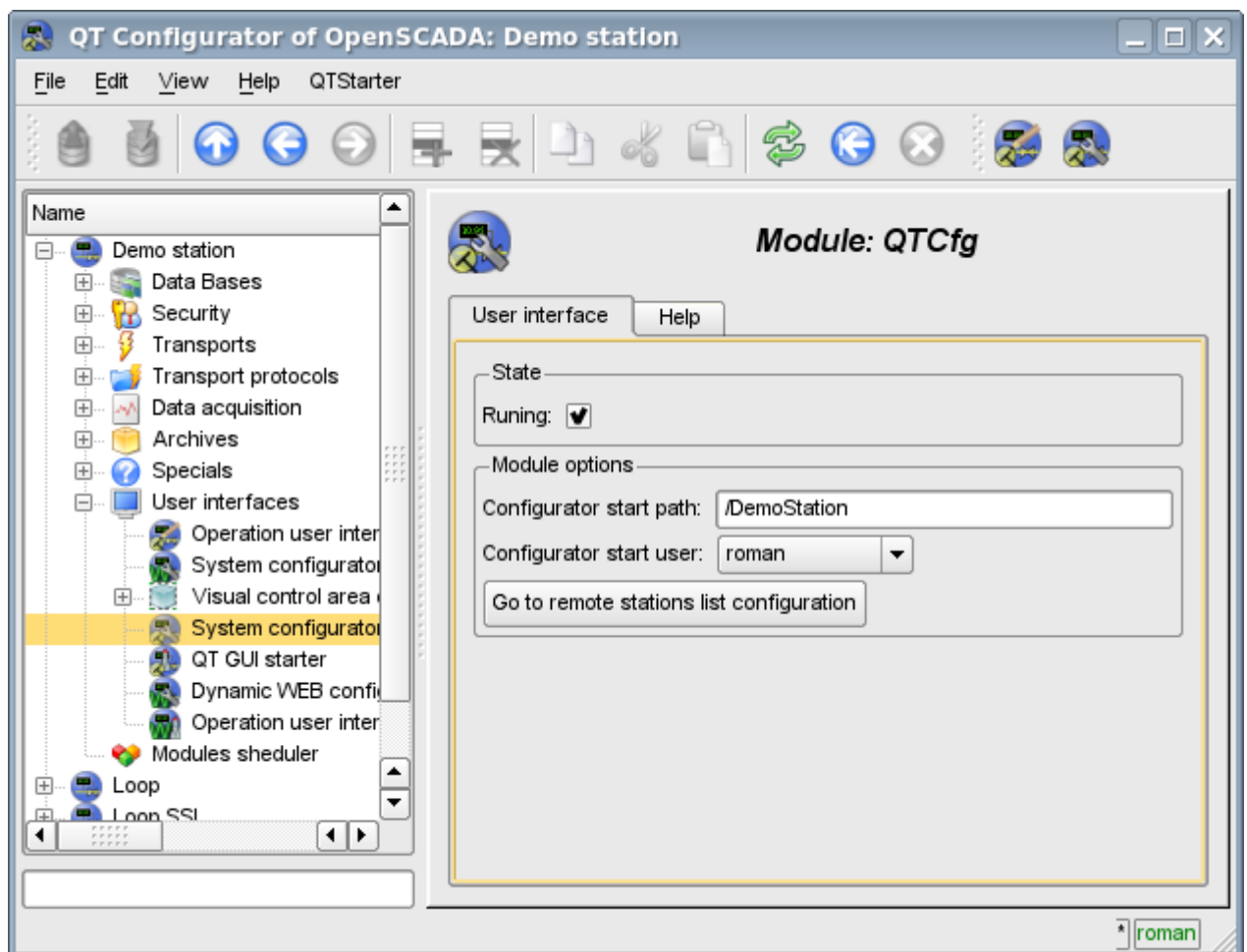


Fig.2. The configuration page of the configurator.

2. Basic elements

Into the number of the basic elements are included: information elements, the field to input values, the elements of combo box, flags. In the case of absence of an element name, the basic element connects to the previous basic element. Examples of basic elements with the connection is shown in Fig.3.

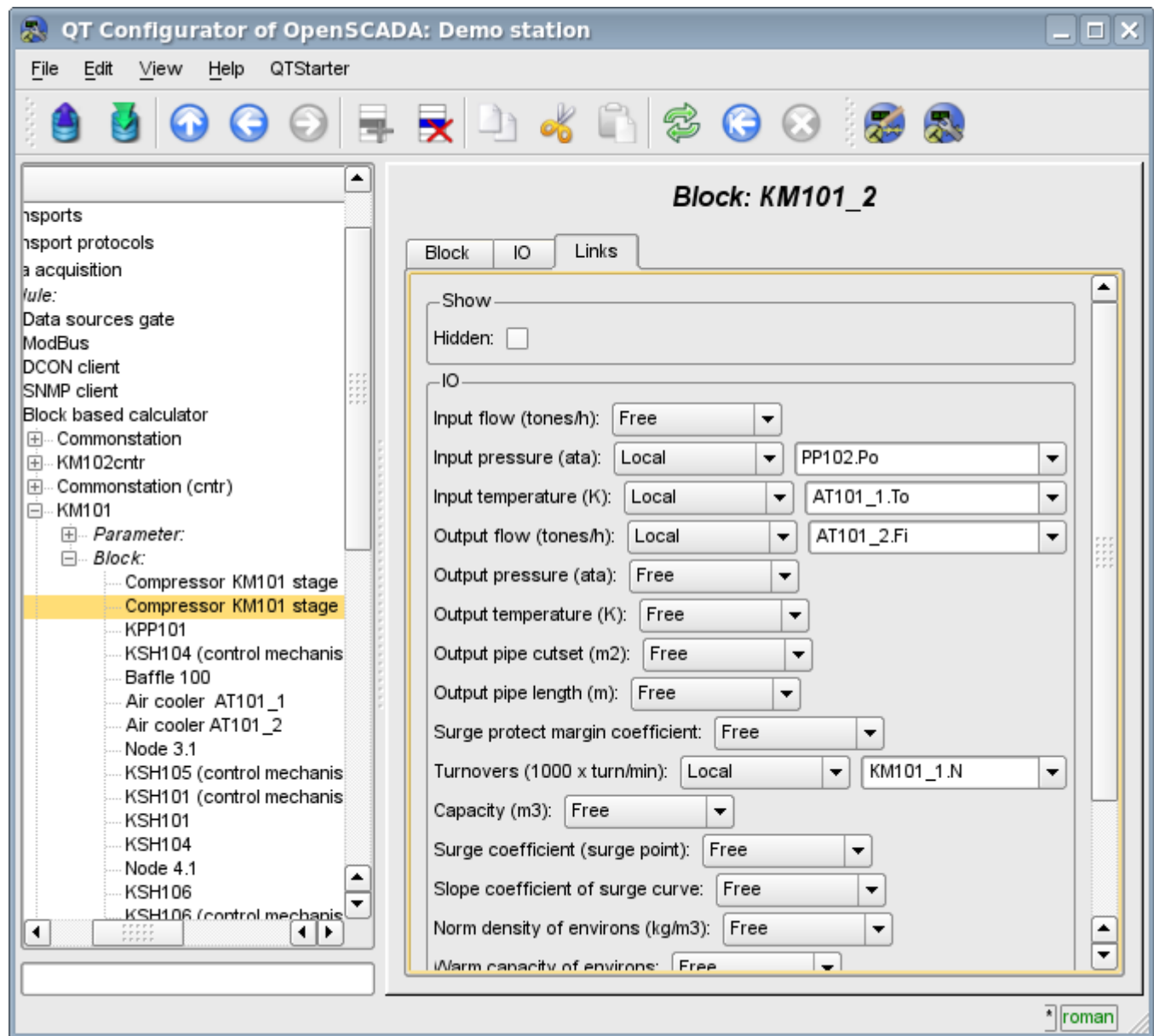


Fig.3. Connection of the basic elements.

3. Commands

Commands are the elements for the transfer of the certain instructions of the action to the node and for the organization of the links on the page. Commands may contain parameters. The parameters are formed from the basic elements. Example of the commands with the parameters is shown in Fig.4.

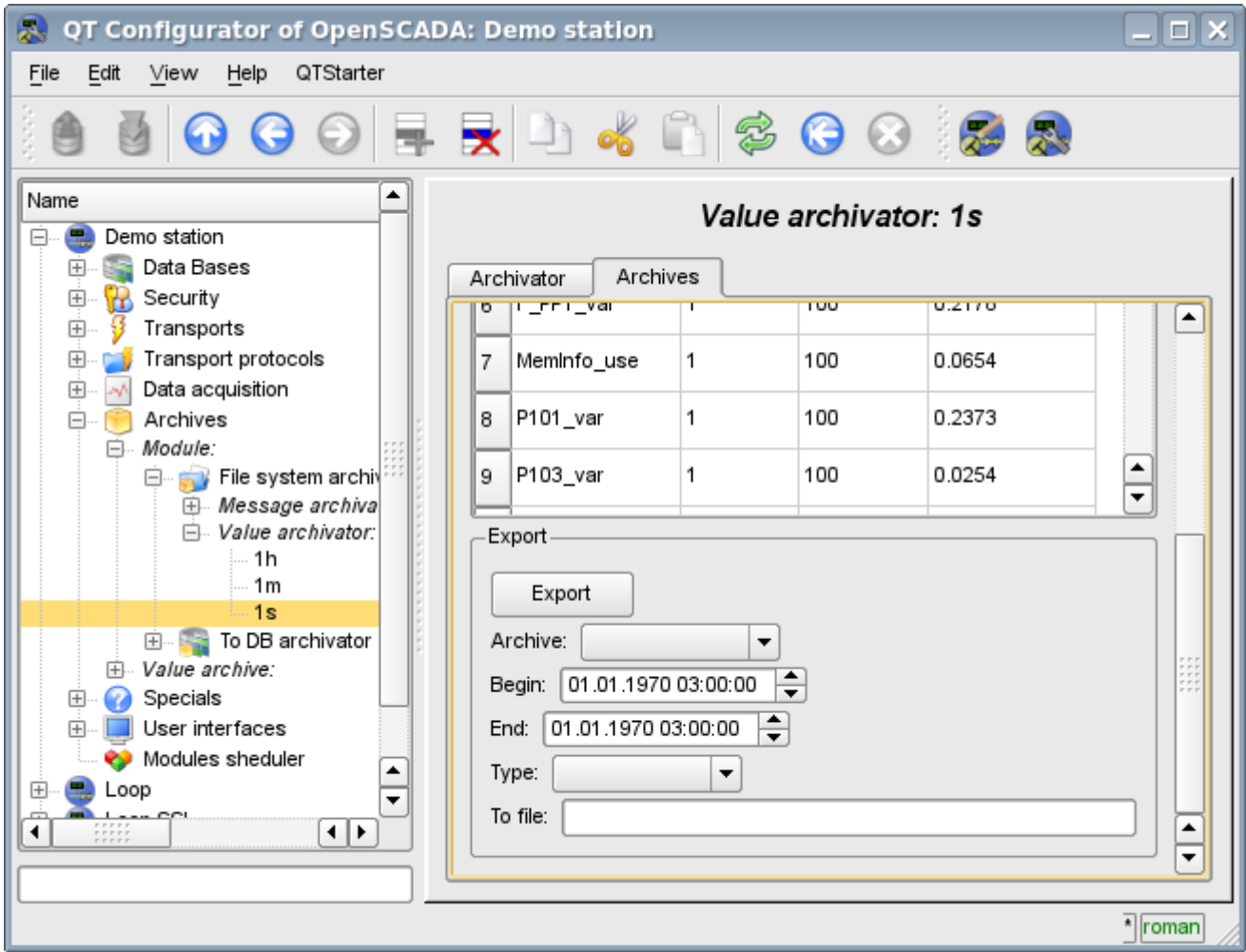


Fig.4. Command.

4. Lists

Lists contain a group of basic elements of the same type. Operations under the elements are accessible via the context menu of the list. Through the elements of the list can be performed the moving operations to other pages. The transition is implemented by double-clicking of the mouse on an element of the list. Lists can be indexed. An example of the list is shown in Fig. 5.

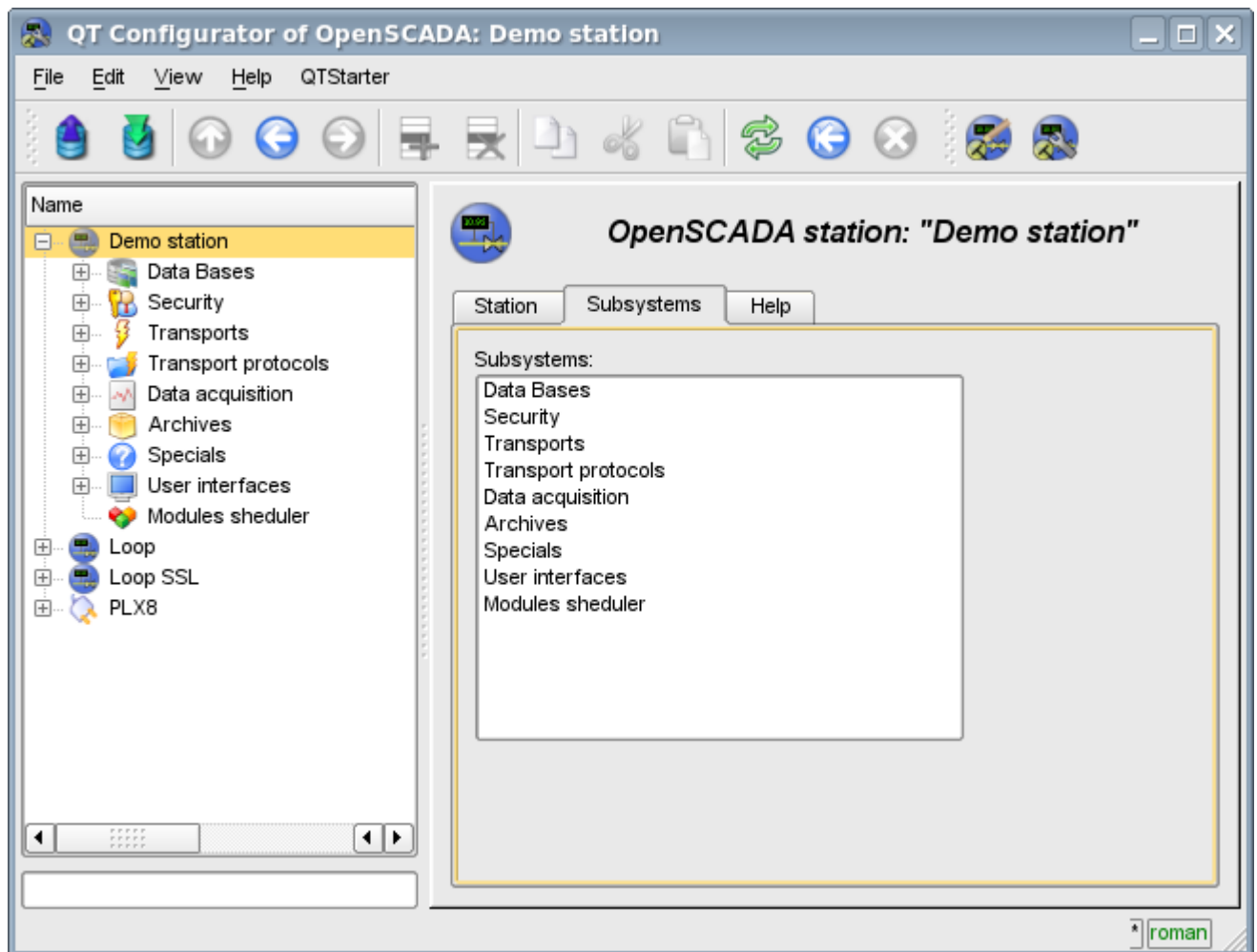


Fig.5. The list.

5. Tables

The tables contain values of the basic elements. Type of the basic element is an individual for each column. Example of the table is given in Fig. 6. Operations on the structure of the table for editable tables are accessible through the context menu. Editing of the table is done by double-clicking on the desired cell.

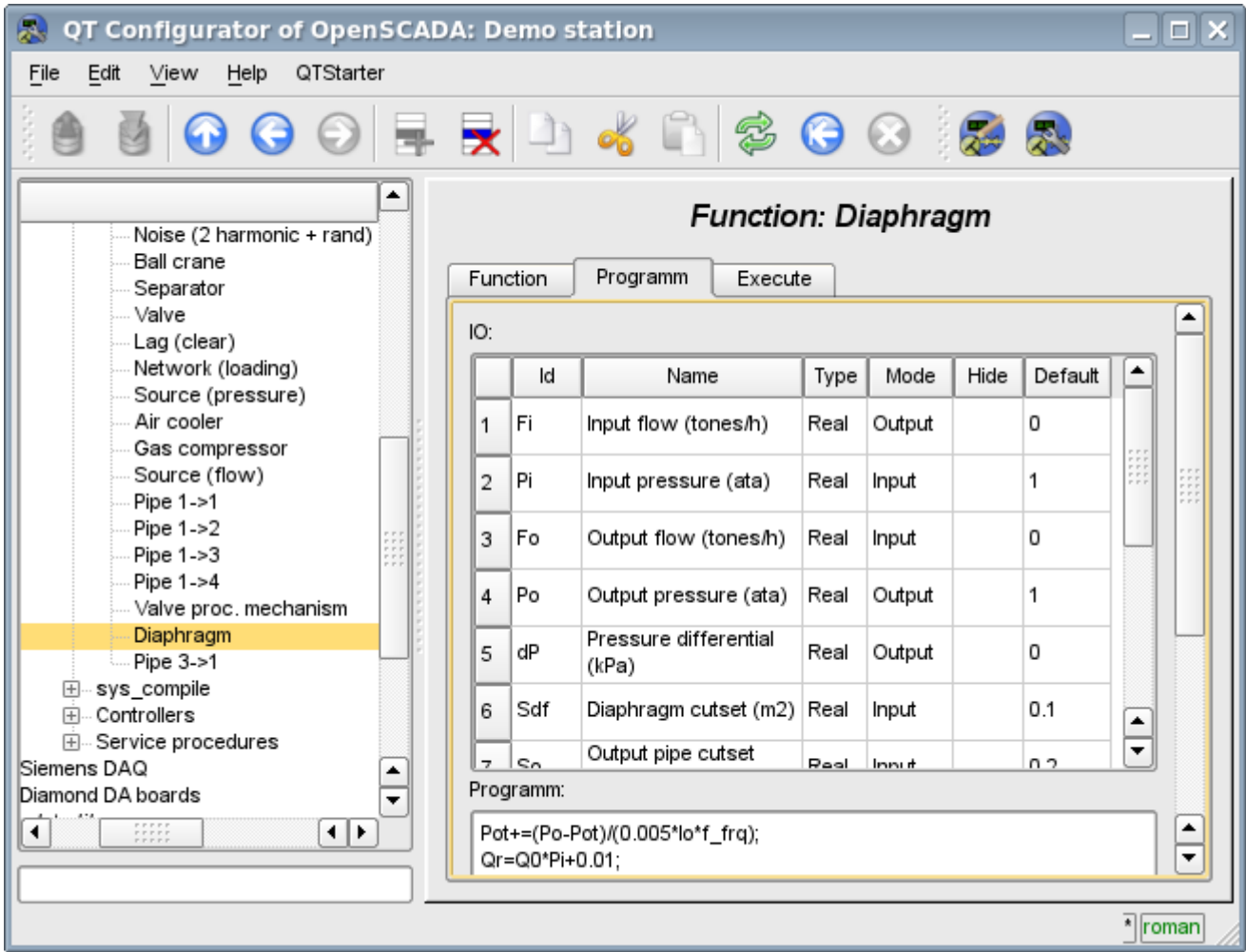


Fig.6. Table.

6. Images

The images are designed to transmit graphic information into the configurators. Example of the image is shown in Fig. 7.

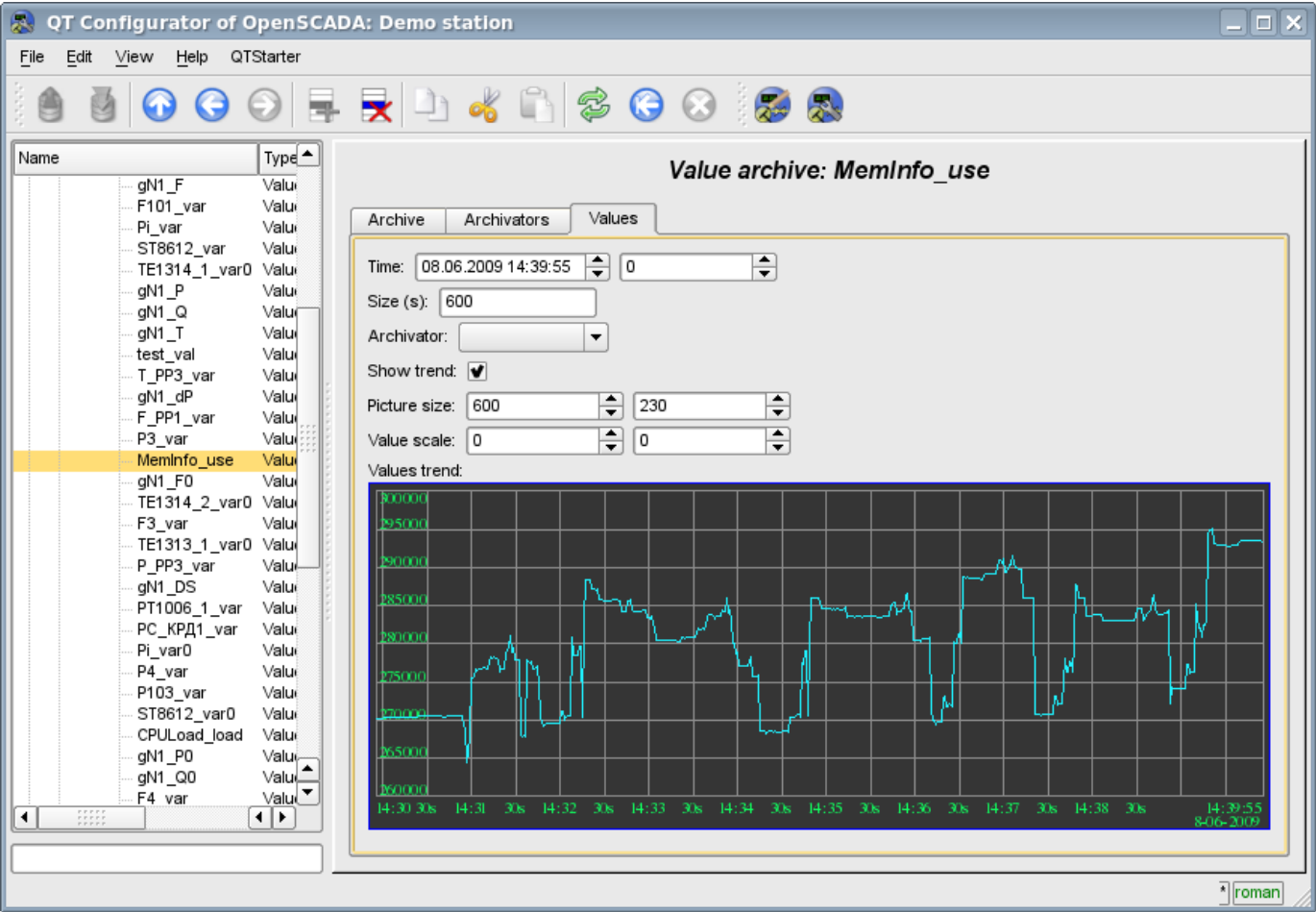


Fig.7. Image.

The module <WebCfg> of subsystems “User Interfaces”

<i>Module:</i>	WebCfg
<i>Name:</i>	The system configurator (Web)
<i>Type:</i>	User Interfaces
<i>Source:</i>	ui_WebCfg.so
<i>Version:</i>	1.5.4
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides the WEB-based configurator of the OpenSCADA system.
<i>License:</i>	GPL

The WebCfg module provides the configurator of the OpenSCADA system. Configurator is based on Web-technologies. For configurator working it is enough the usual WEB-Browser. The operability of the module <WebCfg> was tested in conjunction with modules <Transport.Sockets> and <Protocol.HTTP> on the following Web-browsers:

- Mozilla;
- Firefox;
- Konqueror;
- Opera;
- IE.

The module is based on the language of management interface of OpenSCADA system, and thus provides a uniform configuration interface. Updating of the module may be required only in the case of updating the specification of the language of management.

In addition to the belonging of the module to the OpenSCADA system, it also belongs, is a module, to the module transport protocol <HTTP>. Actually, the call ~ [Web Cfg](#) makes from of HTTP. The call is made through enhanced communication mechanism through exported in module ~ [Web Cfg](#) features: HttpGet and HttpSet.

The interface of the module is implemented by means of the language XHTML 1.0 Transitional with inclusions of the JavaScript.

Using the module starts with the opening session, the user authentication of the user module by the protocol HTTP (Protocol.HTTP) (Fig. 1). For the operation of the authentication and session saving mechanism the browser must allow Cookies.



Fig.1. User authentication.

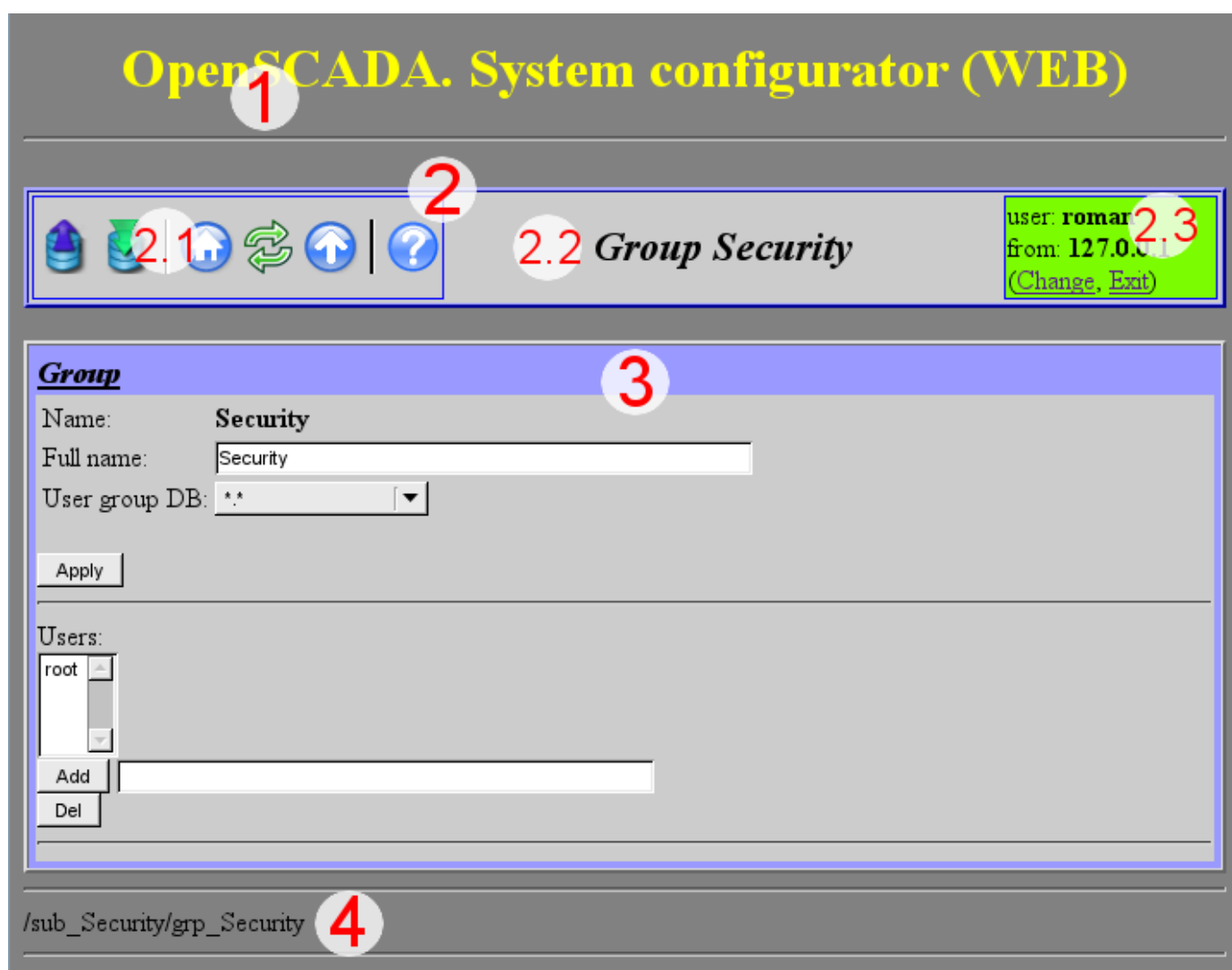


Fig.2. Structure of the operating window of the user.

After authenticating the user enters the operating window (Fig.2), which consists of the following parts:

- 1.Header – contains the name of the module.
- 2.Control Panel. Consists of:
 - 2.1.Navigator – serves the navigation functions through the tree of pages.
 - 2.2.The name of the node.
 - 2.3.User of the System – Displays the current user of the session, his address and lets you to change the user.
- 3.Workplace field – contains the configuration settings of language of management interface, starting with the root tabs to the end elements.
- 4.Footer – contains the address of current page.

Addressing of the pages begins with an element of second-level URL. This is due to the fact that the first-level element is used to identify the module of user Web-interface. For example URL: `<http://localhost.localdomain:10002/WebCfg//Functions>` can be deciphered as call of the first-level page “Functions” of the Web module `<WebCfg>` on the host `localhost.localdomain` through the port 10002.

The control tools are divided into: basic, commands, lists, tables and images. All four types are displayed by individual units not depending on their location in the description.

1. Basic elements

The basic elements include: information elements, the field for input of the values, the elements of combo box, flags. To set the new values of the basic elements the group method is used, for this there is a button `<Accept>` on the form. In the case of the absence of an element name, the basic element connects to the previous one. Examples of basic elements, with connections, is shown in Fig.3.

The screenshot shows a web interface titled "IO" with a list of configuration parameters. Each parameter has a dropdown menu for its value, and some have additional dropdown menus for connections. The parameters are:

Parameter	Value	Connection
Input pressure setpoint (ata):	Free	
Output flow (tones/h):	Local	KW1.Fi
Output pressure (ata):	Free	
Output pipe cutset (m2):	Free	
Output pipe length (m):	Free	
Input flow's noise:	Local	noisePP3.out
Norm density of environs (kg/m3):	Free	
Calc frequency (Hz):	Free	
Output pressure laged:	Free	
Input flow laged:	Free	

At the bottom left, there is an "Apply" button.

Fig.3. The basic elements and their connections.

2. Commands

Commands are the elements for the transfer of the certain instructions of the action to the node and for the organization of the links on the page. Commands may contain parameters. The parameters are formed from the basic elements. Example of the commands with the parameters is shown in Fig.4.

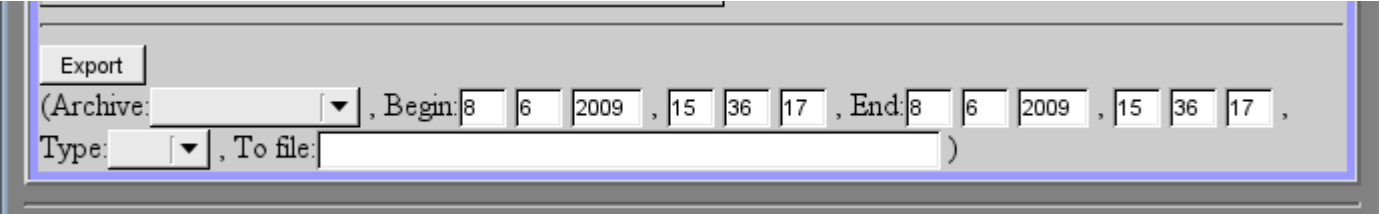


Fig.4. Command.

3. Lists

Lists contain a group of basic elements of the same type. For operations on elements of a list the additional buttons are added. In addition, through the elements of a list the moving operations to other pages are carried out. To move the button “Go” is added. Lists can be indexed. Example of the list with the moving is shown in Fig.5.



Fig.5. The list.

4. Tables

The tables contain values of basic elements. Type of the basic element is defined separately for each column. Example of the table is shown in Fig.6.

IO:

Id	Name	Type	Mode	Hide	Default	*
Fi	Input flow (tones/h)	Real ▼	Output ▼	<input type="checkbox"/>	0	<input type="checkbox"/>
Pi	Input pressure (ata)	Real ▼	Input ▼	<input type="checkbox"/>	1	<input type="checkbox"/>
Fo	Output flow (tones/h)	Real ▼	Input ▼	<input type="checkbox"/>	0	<input type="checkbox"/>
Po	Output pressure (ata)	Real ▼	Output ▼	<input type="checkbox"/>	1	<input type="checkbox"/>
dP	Pressure differential (kPa)	Real ▼	Output ▼	<input type="checkbox"/>	0	<input type="checkbox"/>
Sdf	Diaphragm cutset (m2)	Real ▼	Input ▼	<input type="checkbox"/>	0.1	<input type="checkbox"/>
So	Output pipe cutset (m2)	Real ▼	Input ▼	<input type="checkbox"/>	0.2	<input type="checkbox"/>
lo	Output pipe length (m)	Real ▼	Input ▼	<input type="checkbox"/>	10	<input type="checkbox"/>
Q0	Norm density of environs (kg/m3)	Real ▼	Input ▼	<input type="checkbox"/>	1	<input type="checkbox"/>
f_frq	Calc frequency (Hz)	Real ▼	Input ▼	<input checked="" type="checkbox"/>	100	<input type="checkbox"/>
Pot	Output pressure laged	Real ▼	Output ▼	<input checked="" type="checkbox"/>	1	<input type="checkbox"/>
Fit	Input flow laged	Real ▼	Output ▼	<input checked="" type="checkbox"/>	0	<input type="checkbox"/>

Apply

Fig.6. Table.

5. Images

The images are designed to transmit graphic information into the configurators. Example of the image is shown in Fig. 7.

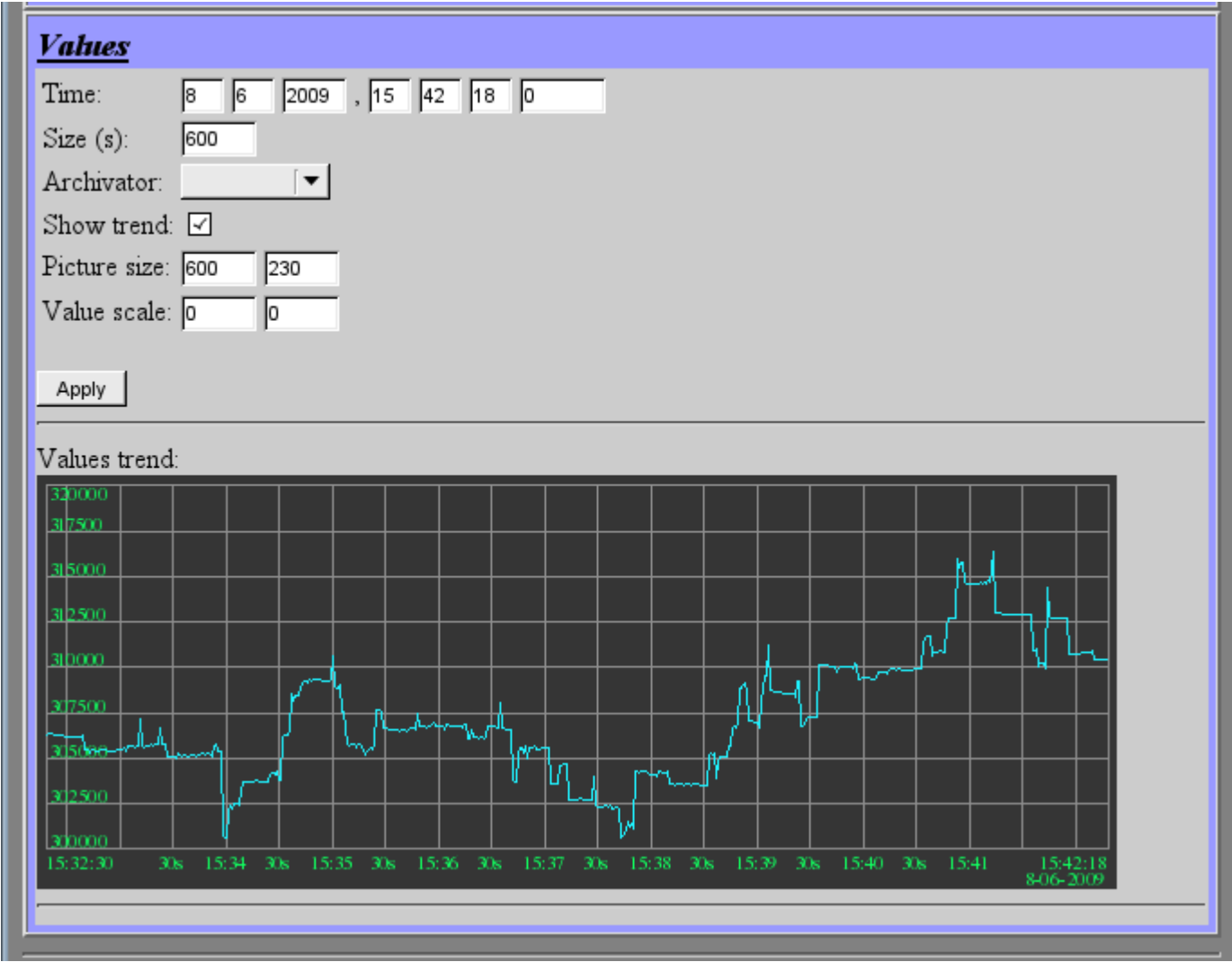


Fig.7. Image.

The module <WebCfgD> of subsystems “User Interfaces”

<i>Module:</i>	WebCfgD
<i>Name:</i>	Dynamic Web configurator
<i>Type:</i>	User Interfaces
<i>Source:</i>	ui_WebCfgD.so
<i>Version:</i>	0.6.5
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Provides dynamic WEB based configurator. Uses XHTML, CSS and JavaScript technology.
<i>License:</i>	GPL

The <WebCfgD> module provides the configurator of OpenSCADA system. Configurator is implemented on the basis of Web-technologies:

- *HTTP* — hypertext transfer protocol;
- *XHTML* — extended language of markup of the hypertext documents;
- *CSS* — cascading style sheets of hypertext documents;
- *JavaScript* — built-in into the hypertext document the browser programming language;
- *DOM* — document object model of the internal structure of the browser;
- *AJAX* — arrangement of asynchronous and synchronous requests from the JavaScript to the server;
- *XML* — eXtensible Markup Language.

Interface of the configurator is formed in the WEB-browser by reference to the WEB-server and getting from it the XHTML-document over HTTP. In this case there is the OpenSCADA system in the role of the WEB-server, which supports standard communication mechanisms of TCP-networks (module Transport.Sockets), hypertext transfer protocol (module Protocol.HTTP), as well as encryption of traffic between the browser and the server (Transport.SSL). Based on this to gain access to the interface configuration of the OpenSCADA, provided by this module, you need to configure the transport in the OpenSCADA (Transport.Sockets or Transport.SSL) in conjunction with the protocol HTTP (Protocol.HTTP). In the delivery of the OpenSCADA system there are configuration files containing settings of Transport.Sockets for ports 10002 and 10004. Hence the interface of the module in the configuration of the OpenSCADA by default will be available at URL: <http://localhost:10002> or <http://localhost:10004>.

After receiving the document XHTML the JavaScript program runs to create dynamic interface configurator.

At the core of the module there is the language of the management interface of the OpenSCADA system, and thus provides the uniform interface of configuration. Update of module may be required only in the case of updating the specification of the language of management.

The module was implemented and tested on three WEB-browsers, representatives of the three types of WEB-engines, as follows:

- Mozilla Firefox 3.0.4
- Opera 9.6.2
- Konqueror 3.5.10

Using the module starts with the opening of the session, the user authentication by the module of the protocol HTTP (Protocol.HTTP). For the operation of the authentication and the mechanism of

saving of the session the browser must allow Cookies.



Fig.1. User authentication.

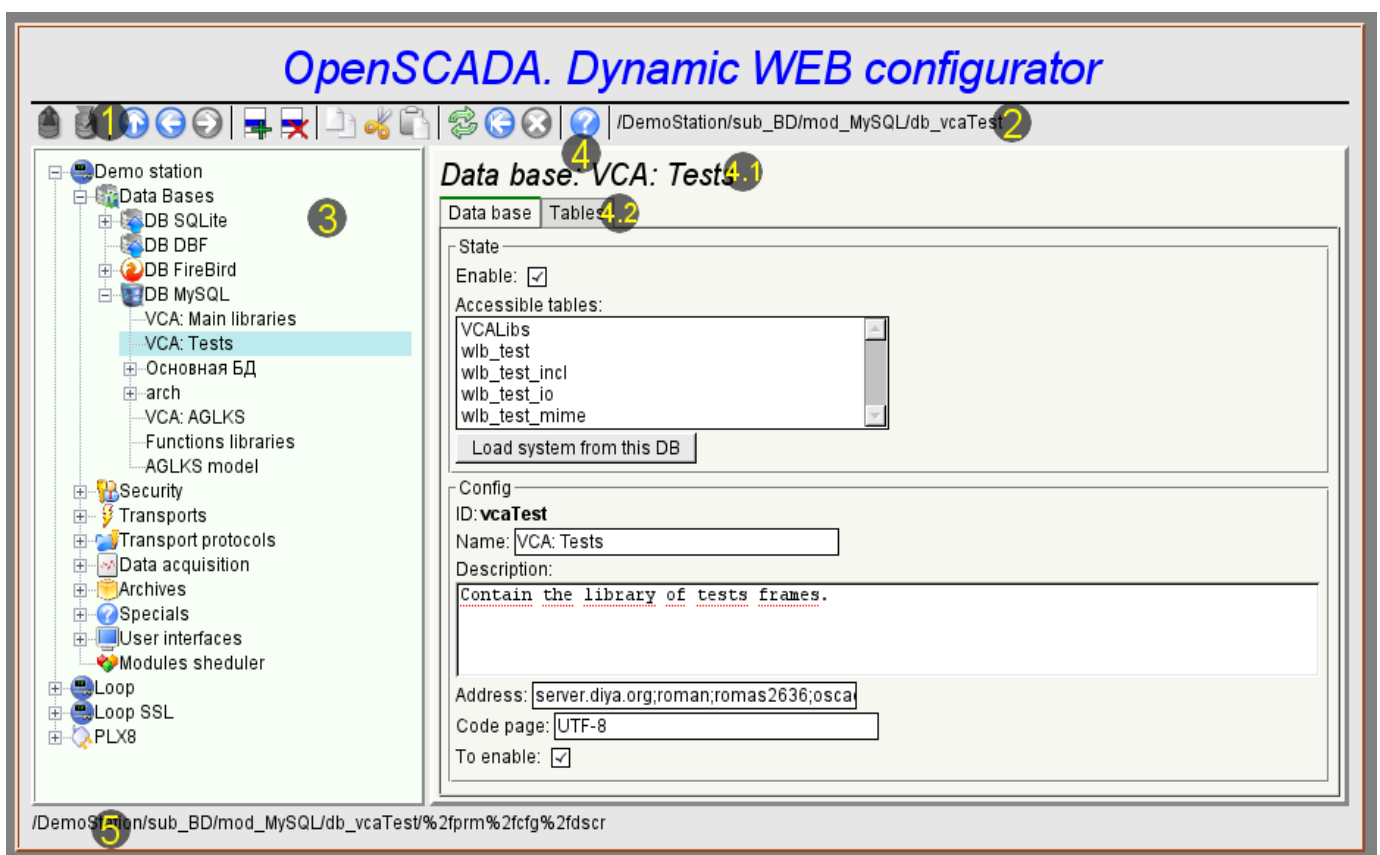


Fig.2. Working window of the configurator

Lets examine the working window of the configurator in Fig. 2.

Working window of the configurator consists of the following parts:

- 1 – Toolbar — contains the control buttons.
- 2 – Address of the open node — displays the current selected node.
- 3 – Navigator — intended for direct navigation through the control tree.
- 4 – Working field. Divided into parts:
 - 4.1 – The name of the node – contains the name of the current node.
 - 4.2 – Tabulator of the working areas -the root pages (control areas) of the node are placed into the tabulator. The control areas of the following levels are placed on the information panel.
- 5 – Status line — displaying the states of the configurator.

The toolbar contains the following control buttons (from left to right):

- *Load* — downloads the selected object or branch of object from the database.
- *Save* — save the selected object or branch of object to the database.
- *Up* — climb up the tree.
- *Previous* — open the previous page.
- *Next* — open the following page.
- *Add item* — add a new object to the container.
- *Delete item* — delete the selected object.
- *Copy item* — copy the selected object.
- *Cut item* — cut of the selected object. The original object is removed after paste.
- *Paste item* — paste of the copied or cut item.
- *Refresh item and the tree* — refresh the current page.
- *Start periodic update* — run periodically update of the contents of the current page with an interval of 5 second.
- *Stop periodic update* — stop periodically update of the contents of the current page with an interval of one second.
- *About* --the information about the module.

The control tools are divided into basic, commands, lists, tables and images. All items are displayed in the sequence strictly appropriate to their location in the description of language of management interface.

1. Configuration

To adjust your own behavior in the not obvious situations module provides the ability to customize individual settings through the management interface of the Open SCADA² (Fig. 3). These parameters are:

- The lifetime of the authentication session (min) – points during which time interval of user inactivity his session will be saved.
- The link to the configuration page of the external OpenSCADA stations used to enable remote configuration.

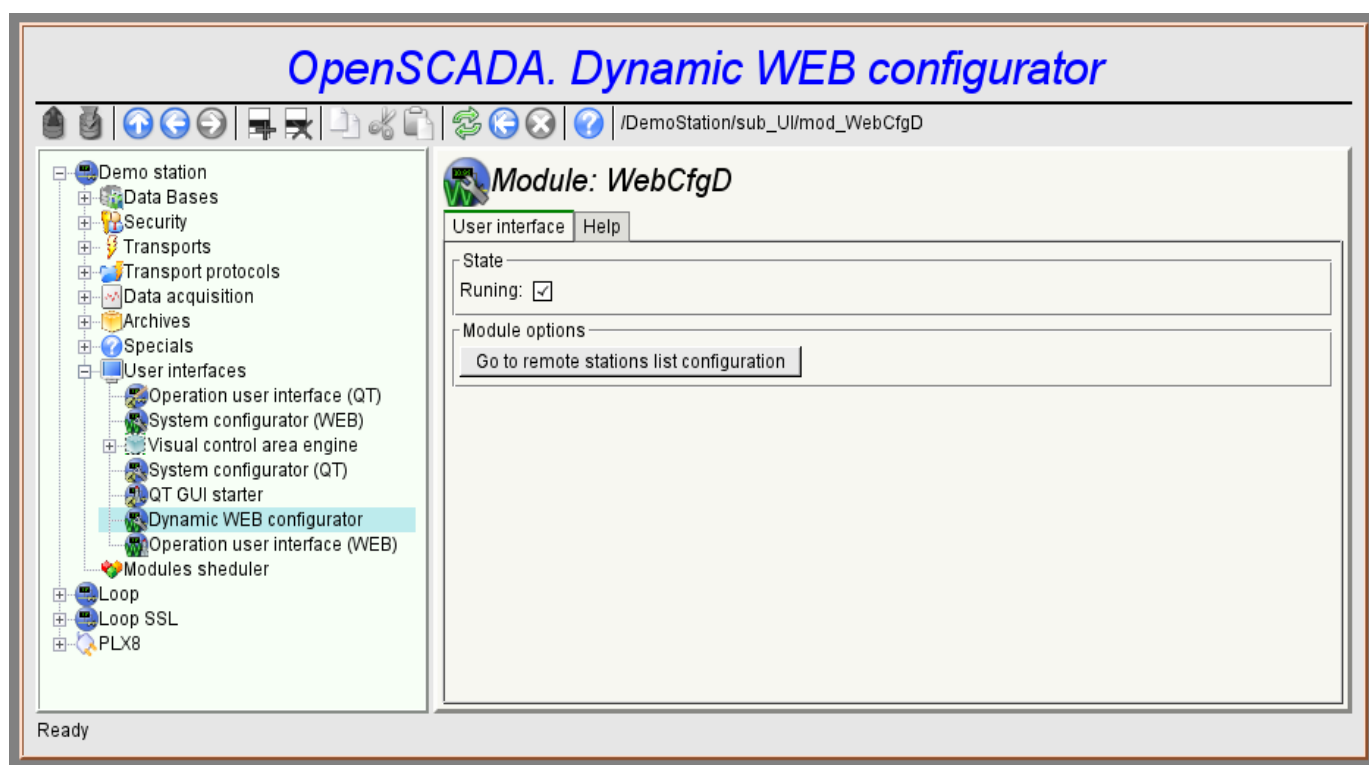


Fig.3. The configuration page of the configurator.

2. Basic elements

Into the number of the basic elements are included: information elements, the field to input values, the elements of choice from the list, flags. In the case of absence of an element name, the basic element connects to the previous basic element. Example of the group of the basic elements with the connection is shown in Fig.4.

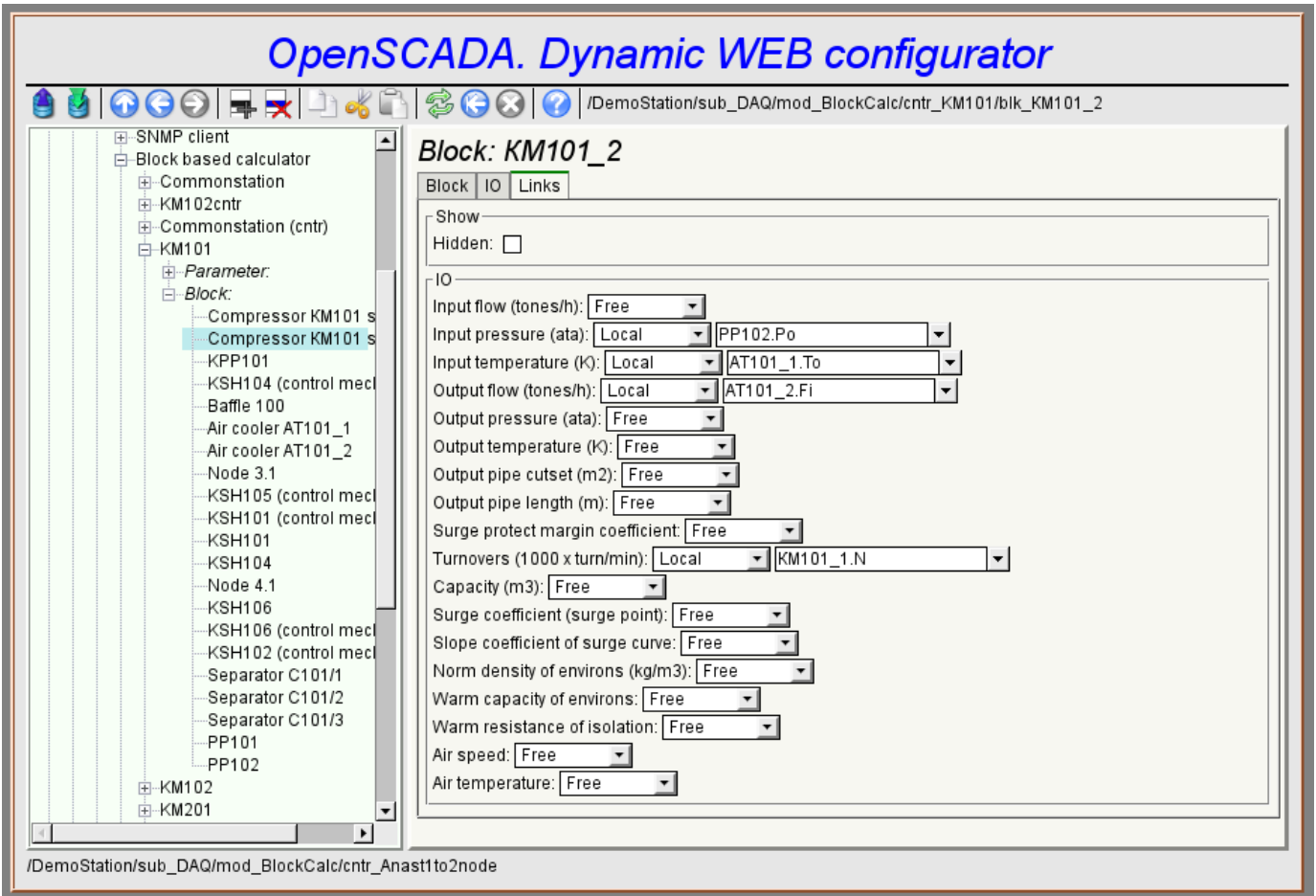


Fig.4. Connection of the basic elements.

3. Commands

Commands are the elements for the transfer of the certain instructions of the action to the node and for the organization of the links on the page. Commands may contain parameters. The parameters are formed from the basic elements. Example of the commands with the parameters is shown in Fig.5.

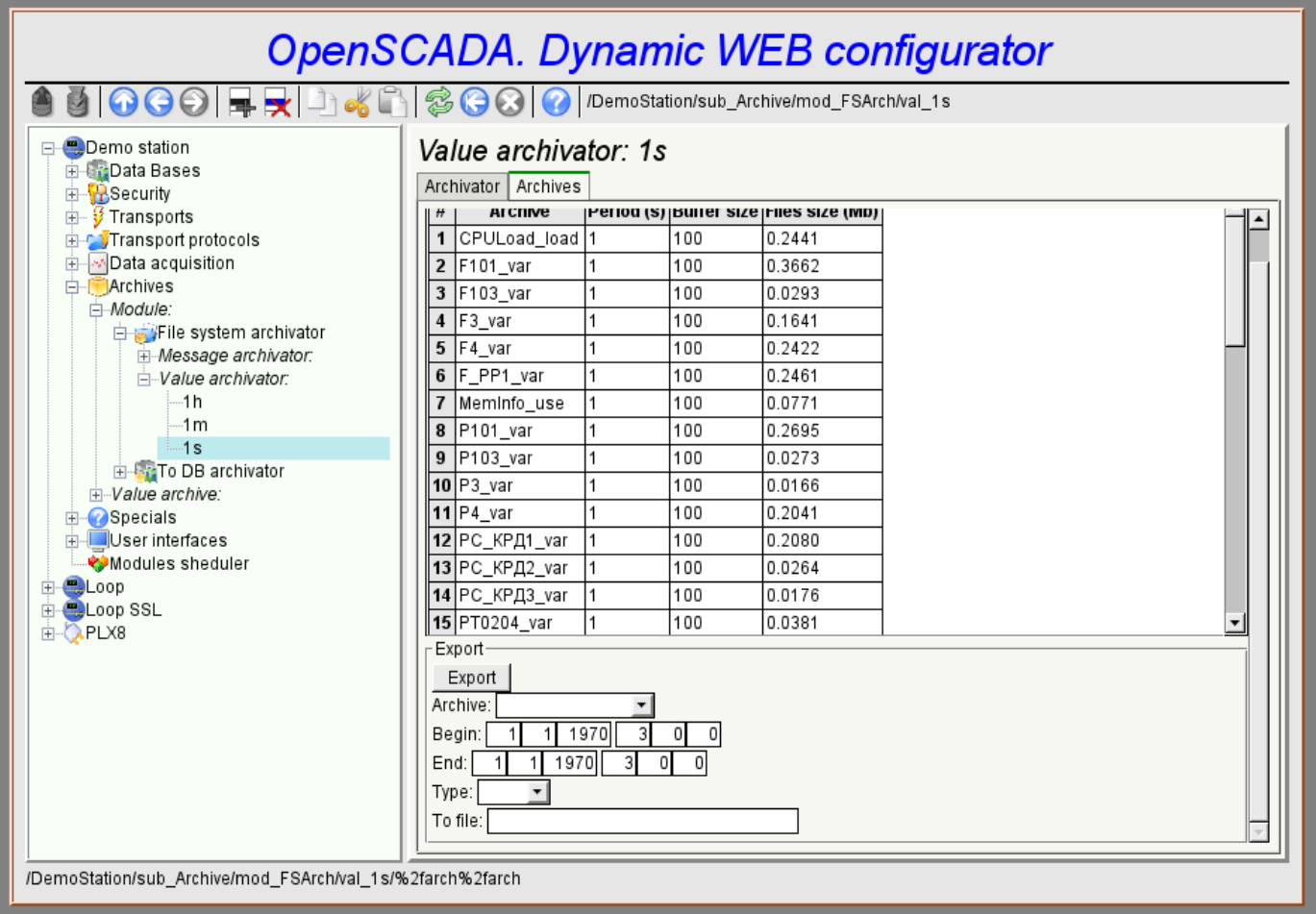


Fig.5. The command.

4. Lists

Lists contain a group of basic elements of the same type. Operations under the elements are accessible via the context menu by the mouse click on the list. Through the elements of the list can be performed the moving operations to other pages. Lists can be indexed. An example of the list is shown in Fig. 6.

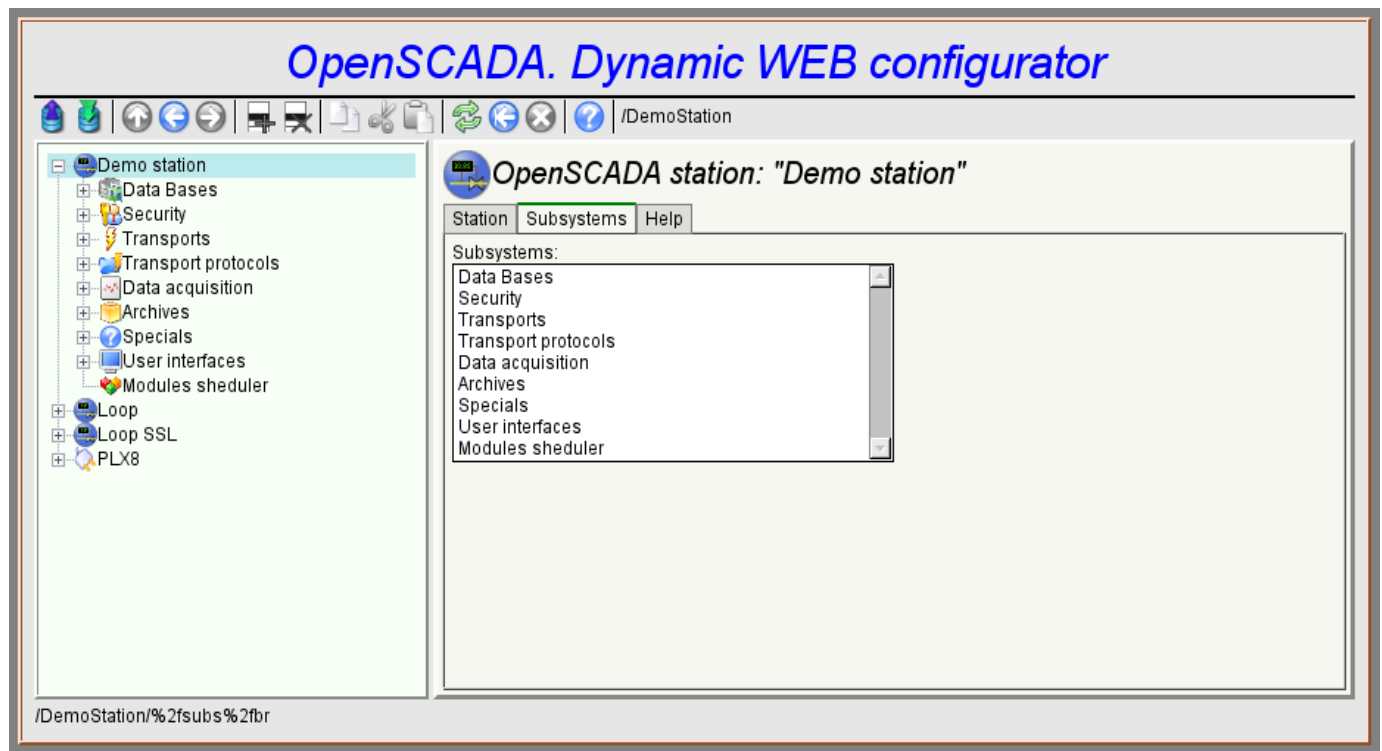


Fig.6. The list.

5. Tables

The tables contain values of the basic elements. Type of the basic element is an individual for each column. Example of the table is given in Fig. 7. Operations on the structure of the table for editable tables are accessible through the context menu by the mouse clicking on the service button with the line number. Editing of the table is done by double-clicking on the desired cell.

OpenSCADA. Dynamic WEB configurator

Function: Diaphragm

Function

Programm

Execute

IO:

#	Id	Name	Type	Mode	Hide	Default
1	Fi	Input flow (tones/h)	Real	Output	Off	0
2	Pi	Input pressure (ata)	Real	Input	Off	1
3	Fo	Output flow (tones/h)	Real	Input	Off	0
4	Po	Output pressure (ata)	Real	Output	Off	1
5	dP	Pressure differential (kPa)	Real	Output	Off	0
6	Sdf	Diaphragm cutset (m2)	Real	Input	Off	0.1
7	So	Output pipe cutset (m2)	Real	Input	Off	0.2
8	lo	Output pipe length (m)	Real	Input	Off	10
9	Q0	Norm density of environs (kg/m3)	Real	Input	Off	1
10	f_freq	Calc frequency (Hz)	Real	Input	On	100
11	Pot	Output pressure laged	Real	Output	On	1
12	Fit	Input flow laged	Real	Output	On	0

Programm:

Pot+= (Po-Pot)/(0.005*lo*f_freq);
Qr=Q0*Pi+0.01;
Fi=4e3*Sdf*sign(Pi-Pot)*pow(Q0*abs(pow(Pi,2)-pow(Pot,2))/293,0.5);
Fit+= (Fi-Fit)/(0.005*lo*f_freq);
Po+=0.27*(Fit-Fo)/(So*lo*Q0*f_freq);
Po= (Po<0)?0: (Po>200)?200:Po;
dP-= (dP-100.*(Pi-Po))/f_freq;

Sound card

Logic level

Java-like based calculator

Controller:

Library:

Report's documents

Technological devices

Lag

Noise (2 harmonic + r

Ball crane

Separator

Valve

Lag (clear)

Network (loading)

Source (pressure)

Air cooler

Gas compressor

Source (flow)

Pipe 1->1

Pipe 1->2

Pipe 1->3

Pipe 1->4

Valve proc. mechanis

Diaphragm

Pipe 3->1

sys_compile

Controllers

Service procedures

Siemens DAQ

Diamond DA boards

Template library

/DemoStation/sub_DAQ/mod_JavaLikeCalc/lib_techApp/fnc_diafragma

Fig.7. The table.

The module <WebCfgD> of subsystems “User Interfaces” 349

6. Images

The images are designed to transmit graphic information into the configurators. Example of the image is shown in Fig. 8.

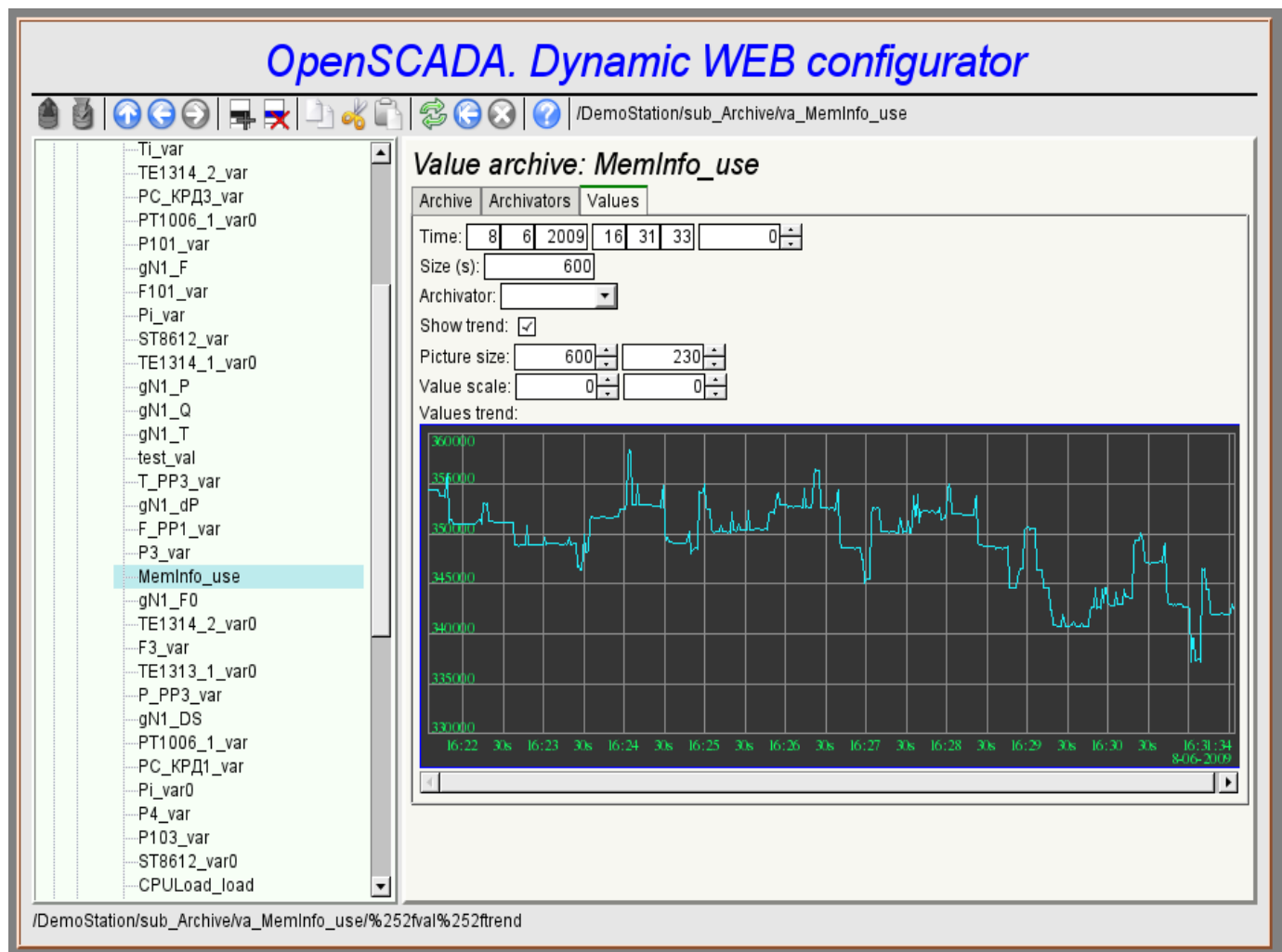


Fig.8. The image.

7. Errors

Performance of the configurator may differ for different types of browsers. This is due to the fact that the basis of this module is quite a lot of complex technologies, as well as differences between them on different types of WEB-engines.

In addition, each Web-browser has its own problems. Some errors were outflanked in the process of implementation, but part of them has stayed in sight of the significant difficulties in their outflanking and also of actual impossibility to do so.

This section contains a table listing the detected errors of the WEB-browsers, which are appeared in the configurator.

Error	Description	Correction
<i>Mozilla FireFox 3.0.4</i> (stable, few errors)		
Offset of the popup window of the editable combo-box at 5 pixels up and left.	The problem lies in the fact that the calculation of the absolute position of the element of the document doesn't grab exactly 5 pixels. Error of 5 pixels is visible in relation to the coordinates of the mouse pointer and the position of the newly-created entirely-positioned window. The algorithm for computing the position: <i>for(; obj != null; obj = obj.offsetParent) posX += obj.offsetLeft;</i>	To correction of this error to the estimated value on this browser the 5 pixels are added.
In the element of the list (<select size="10"/>) the vertical scroller is always shows and never turned on the horizontal one.	This element is actively used for building the context menu and drop-down list of the editable combo-box.	To outflank the browser's error I had to include the list in the block with scroll of the block itself.
The image field is not updating.	In order to eliminate the need for restructuring of the configurable page while updating the values of fields in the tree of the structure objects of the pages which was get from the XMLHttpRequest, the properties are created with the links to the object of the tags of the fields (addr_lab, addr_val_w). In objects with the name of the tag "img", these properties are not created by the browser.	The problem is not solved.
<i>Opera</i> (stable, few errors)		
Scroller of the unit of the page does not turn on. For example when displaying large images of the trend.	The block is fixed with the parameters { overflow: auto; width: 600px; } however, in the case of the exceeding the size of the interior elements the scroller is not turned on.	The problem is not solved.
<i>Konqueror</i> (very unstable on the dynamic resources and contains many errors)		
Stable browser crashing.	Browser repeatedly and consistently crashes while the computation of JavaScript and when dealing with external windows.	The problem is not solved.
The skroll of the navigation tree doesn't returned.	If the navigation tree to expand until the vertical scroll is appeared, then scroll down it, then fold a large branch, the vertical scroll disappears, and a part of the tree remains invisible in the upper part of the block. Ie the contents of the block is not updated.	The problem is not solved.

Error	Description	Correction
The images do not update.	In the fields of images, to update the image from the server, the property "src" is to be changed. Browser does not feel it, or even updates the size of the frame, and the image is not updated. Methods to prevent caching of the images are used, but do not help.	The problem is not solved.
Capture of the images of the buttons	JavaScript modules use non asynchronous, but synchronous requests to a server to save the sequence of actions. In moments of such a request if it was caused by the event from the image (the image is a button), the image is captured as if to move, even for short mouse clicks.	The problem is not solved.
Impossible to insert a new element to the tree of objects obtained as a result of XMLHttpRequest	To monitor the modification of the configuration page the current tree structure to a new one, just received from XMLHttpRequest, reduction was used. When inserting a new element to the tree of the structure through the insertBefore() an error occurs «DOM error 4". If the paste is made to the tree created from zero (not from XMLHttpRequest), this error does not occur. It seems the problem lies in the contrast of object "document" – the owner of these trees. In such a tree it is not possible to add a node as document.createElement (). Only created as a mytree.ownerDocument.createElement() are inserted.	The procedure for verifying the structure was simplified to reduced to the determining the fact of changes.
The mechanisms of the formation of the context menu in the Konqueror 4 doesn't work.	Typically, to form a context menu handler oncontextmenu is used in Firefox and IE, or onmousedown in processing the right keys on the remaining browsers. In the Konqueror generally oncontextmenu does not work, but onmousedown only works in Konqueror 3.5.	The problem is not solved.

The module <VCAEngine> of subsystems "User Interfaces"

<i>Module:</i>	VCAEngine
<i>Name:</i>	Visual control area engine
<i>Type:</i>	User Interfaces
<i>Source:</i>	ui_VCAEngine.so
<i>Version:</i>	1.0.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	The main visual control area engine.
<i>License:</i>	GPL

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.

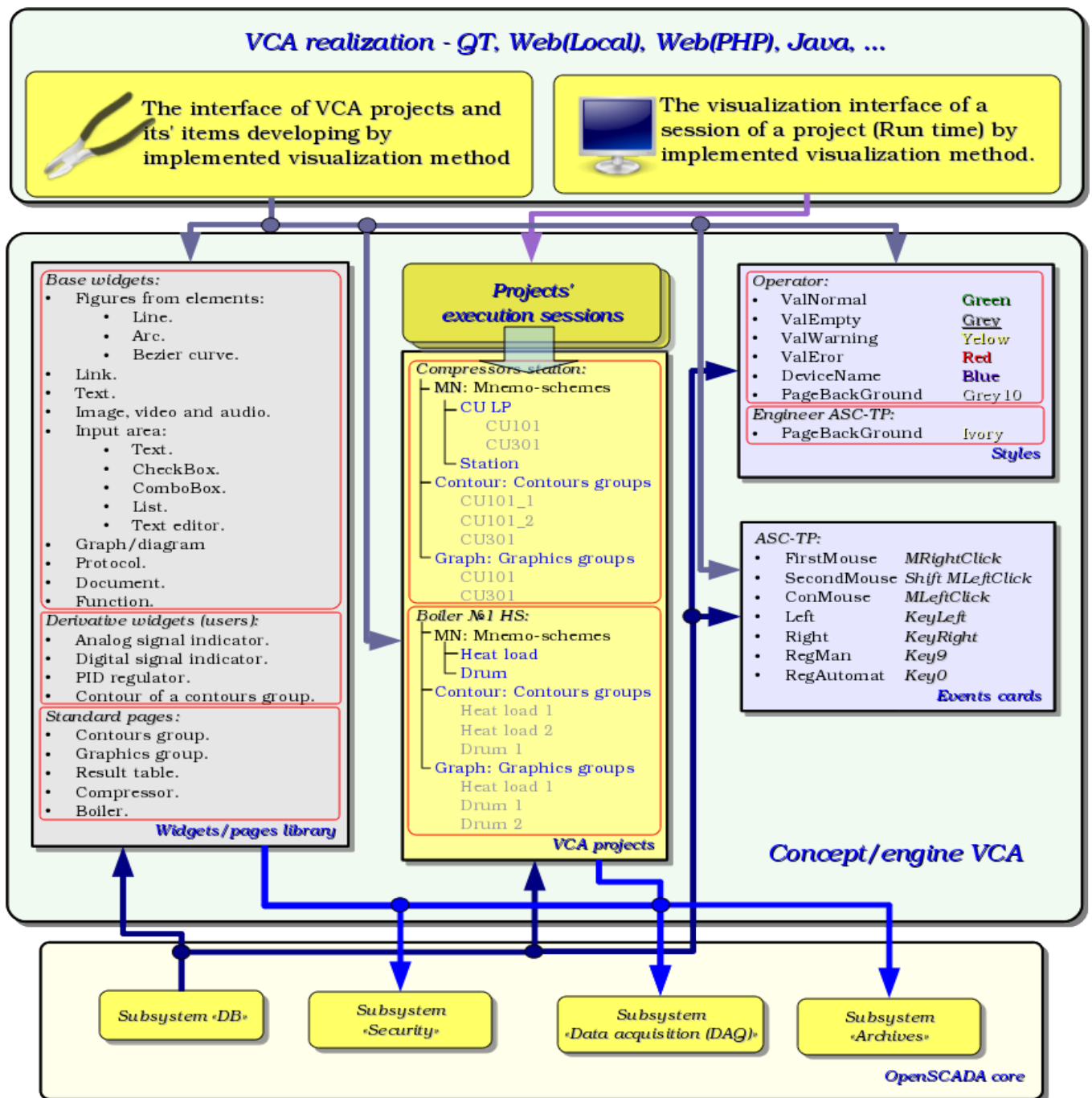


Fig.3 Generalized structure of the VCA.

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).

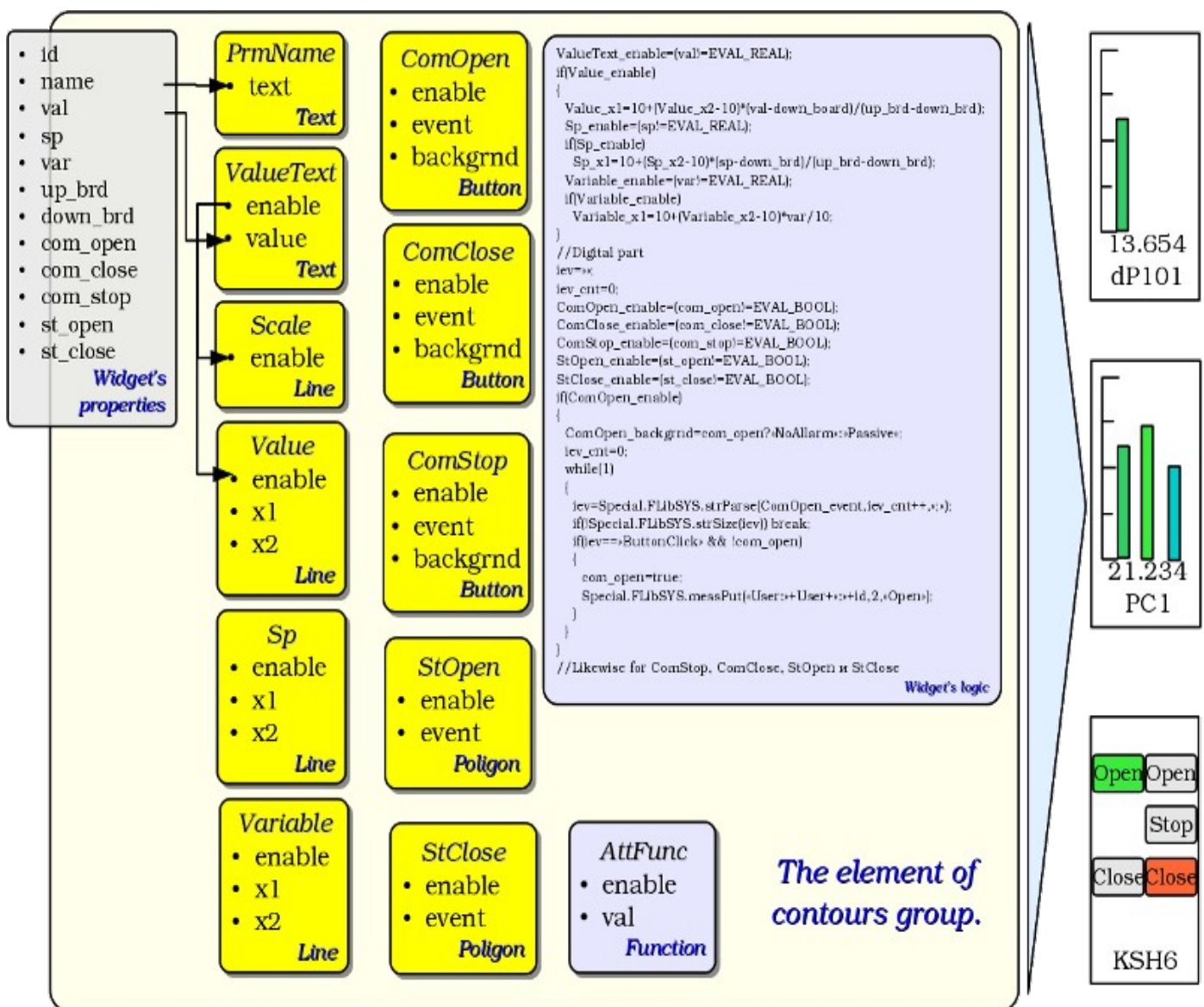


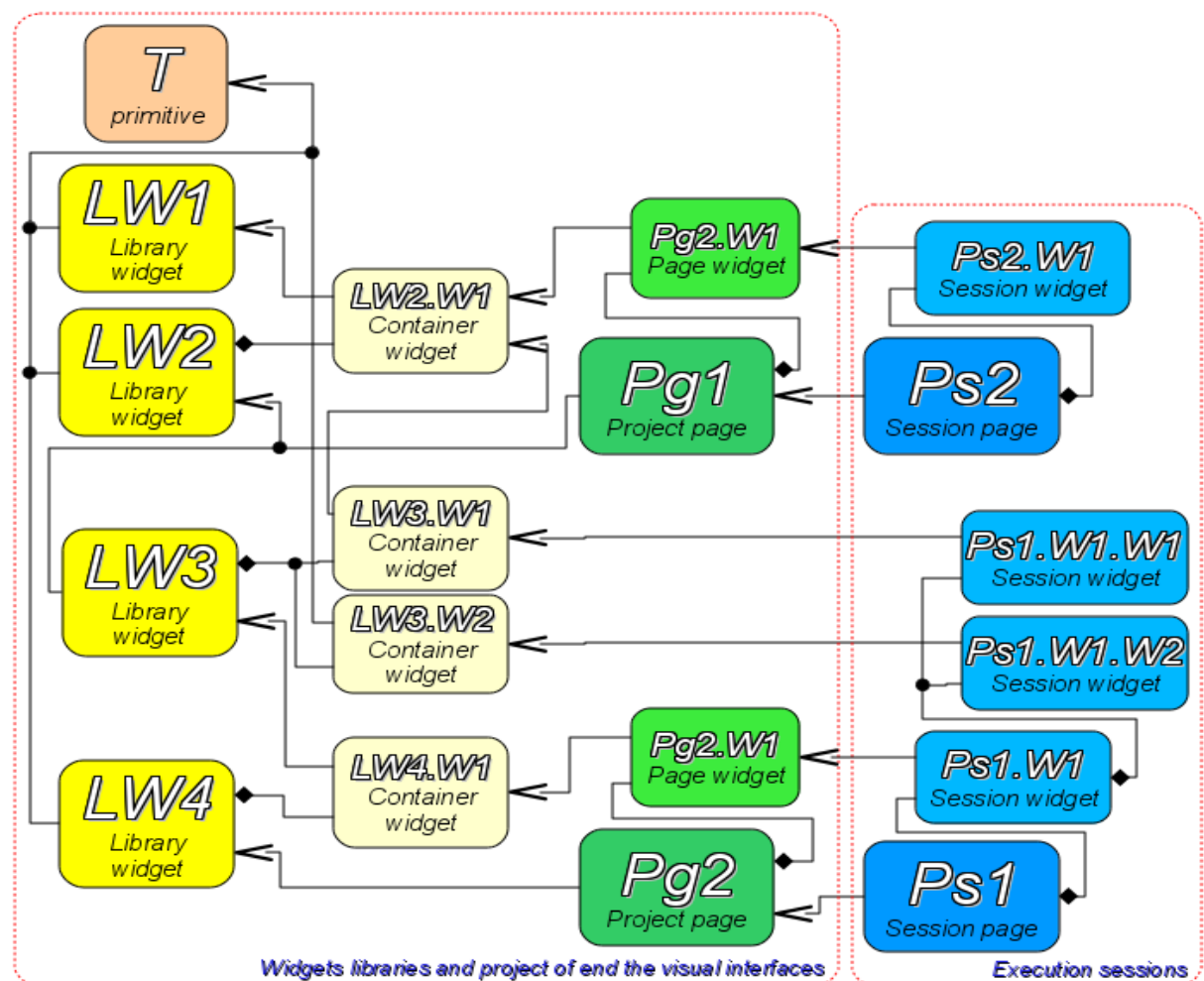
Fig.3.1.1 Example of the structure of the derived widget.

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.



Terminal widget – The final element of the visualization, or primitive. On the side of visualization becomes a visible image.

Library widget – Stored library widget. Be sure to inherit the visual image of the terminal widget and override its data. Inheritance terminal widget can be both direct and through a series of intermediate elements.

Container library widget – In fact, a link to another widget library (LW2.W1 -> LW1) or a reference library container (LW3.W1 -> LW2.W1).

Project page – Element of interface visualization and control (VC) - The page is used to construct a hierarchical interface VC for the end user.

Page widget – Page element for define data of library widget to the needs of the project page.

Session page – Session page for the execution page of the project in the context of the whole interface clause.

Session widget – End element of visualization. Arranged in a hierarchical relationship, the corresponding inheritance in container terminal widget widgets and widget libraries project.

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 the 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.

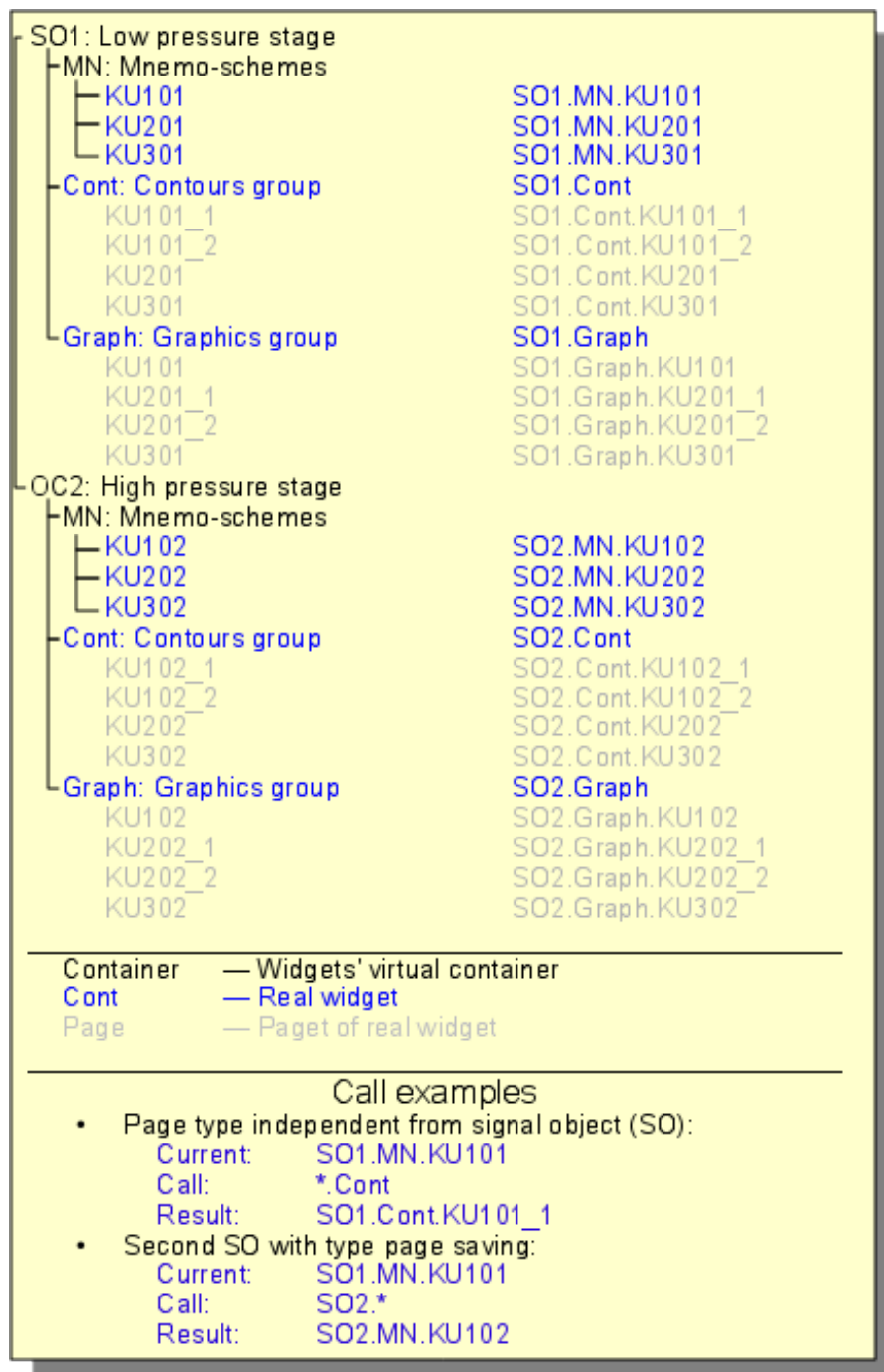


Fig.3.2 Hierarchical view of components of the project of classical interface of VC of the technological process.

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;
- *I* - 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.

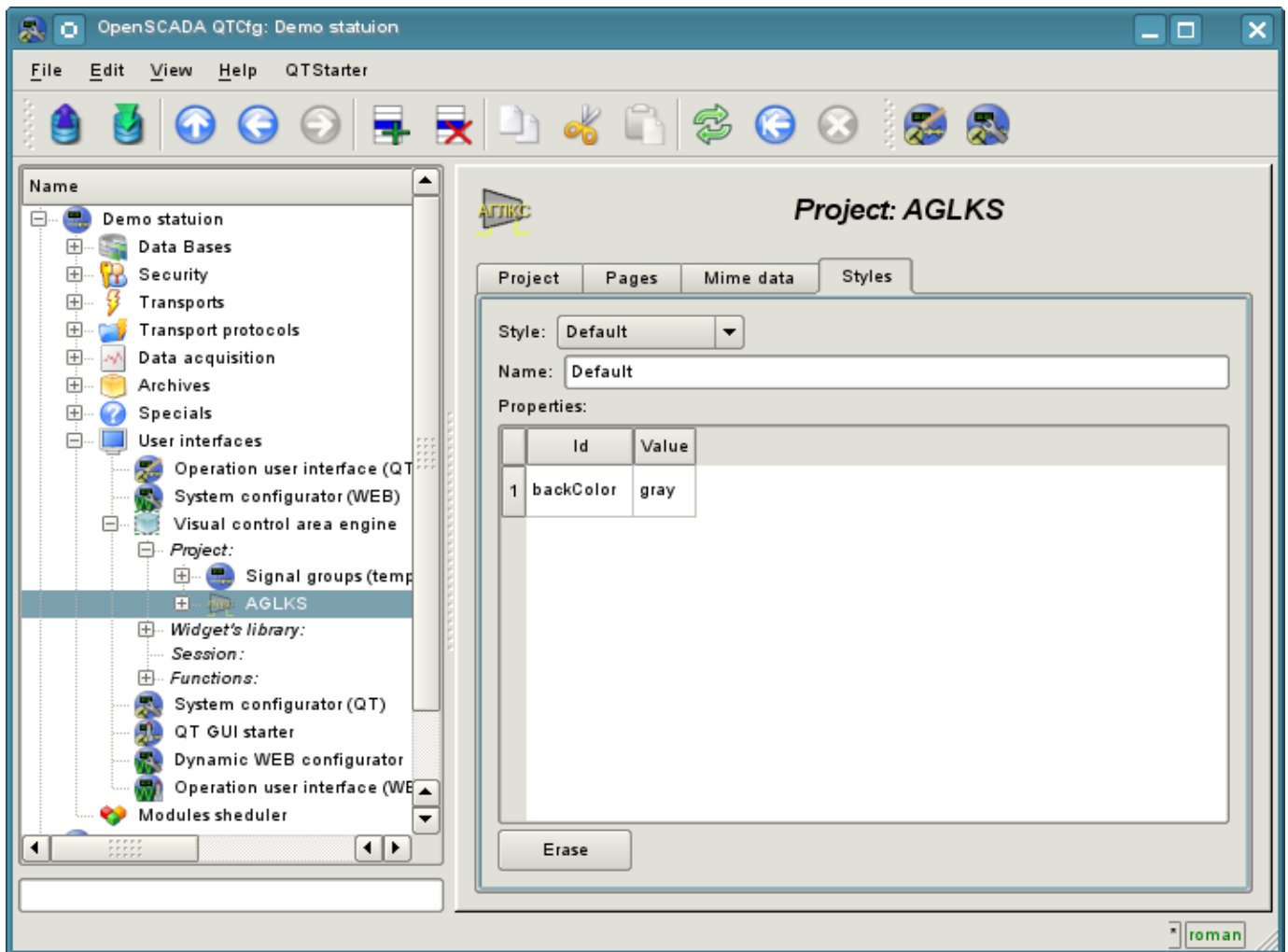


Fig. 3.3 "Styles" tab of the configuration page of the project.

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: <i>key_[pres rels][Ctrl Alt Shift]{Key}</i>	
<i>*SC#3b</i>	Scan code of the kye.
<i>*#2cd5</i>	Code of the unnamed key.
<i>*Esc</i>	"Esc".
<i>*BackSpace</i>	Removing of the previous character - "<--".
<i>*Return, *Enter</i>	Enter - "Enter".
<i>*Insert</i>	Insertion - "Insert".
<i>*Delete</i>	Deleting - "Delete".
<i>*Pause</i>	Pause - "Pause".
<i>*Print</i>	Print of the screen - "Print Screen".
<i>*Home</i>	Home - "Home".
<i>*End</i>	End - "End".
<i>*Left</i>	Left - "<-".
<i>*Up</i>	Up - '^'.
<i>*Right</i>	Right - "->".
<i>*Down</i>	Down - 'V'.
<i>*PageUp</i>	Page up - "PageUp".
<i>*PageDown</i>	Page down - "PageDown".
<i>*F1 - *F35</i>	Function key from "F1" to "F35".
<i>*Space</i>	Space - ' '.

Id	Description
*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 - '"
<i>Keyboard focus events.</i>	
ws_FocusIn	Focus is obtained by a widget.
ws_FocusOut	Focus is lost by a widget.
<i>Mouse events:</i>	
key_mouse[Pres Rels][Left Right Midle]	Pressed/released the mouse button.
key_mouseDbClick	Double-click the left mouse button.
<i>Events handshake on the side of the visualizer.</i>	
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.
<i>Events of the primitive of elemental figure ElFigure:</i>	
ws_Fig{n}[Left Right Midle]	Activating of the figure (fill) {n} by the mouse button.
<i>Events of the primitive of form elements FormEl:</i>	
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.
<i>Events of the primitive of media content Media:</i>	
ws_MapAct{n}[Left Right Midle]	Media area with the number {n} is activated by the mouse button.

Events are the main mechanism of notification and is actively used for user interaction. For the event processing there are two mechanisms: the scripts 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

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, for placement in a status line, displaying in the protocol, and placement in the archive of messages;
- *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 quitted 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.

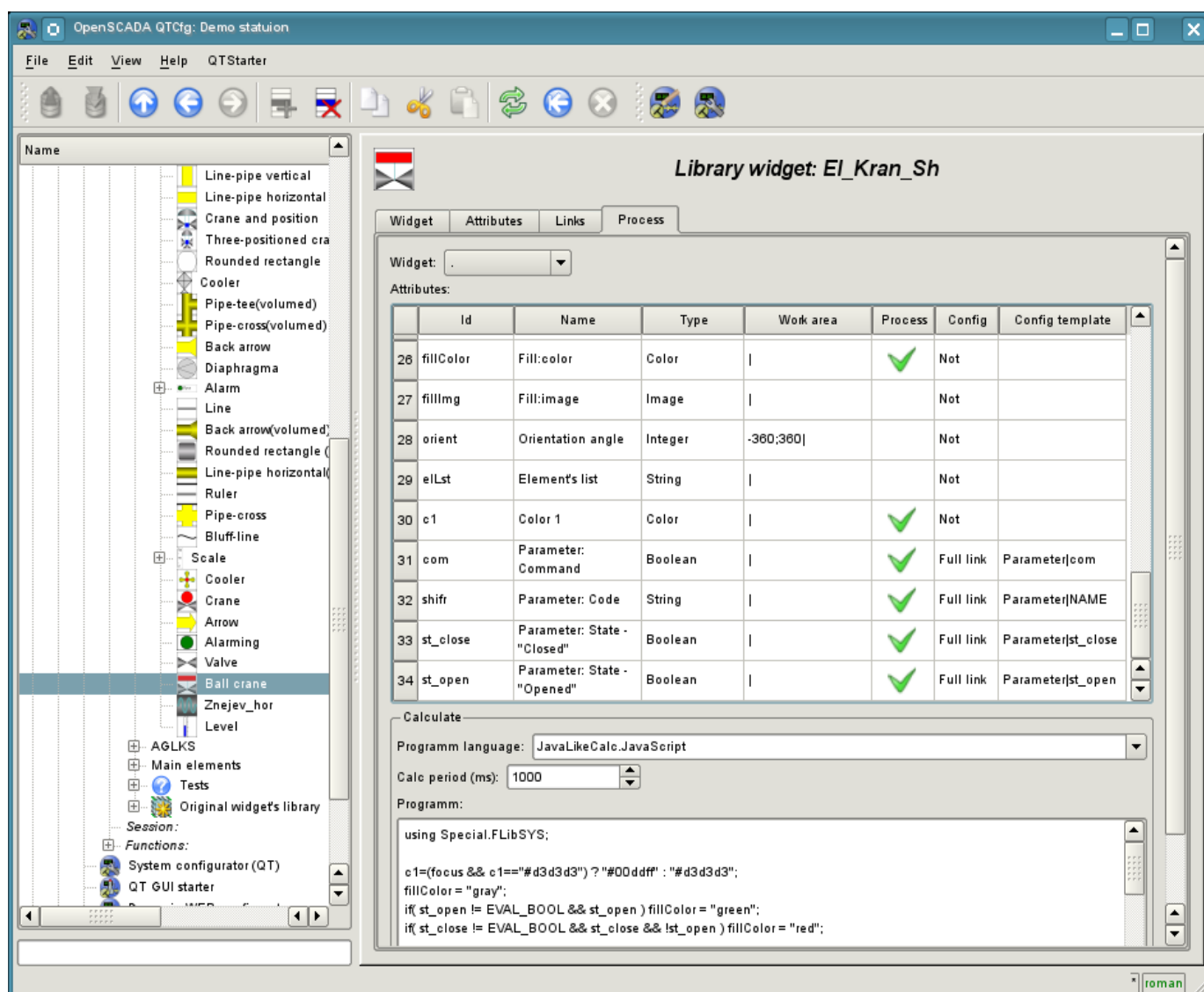


Fig. 3.7.a The tab "Processing" of the configuration page of the widget.

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".
- *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. At that moment this type of link is not intended for installation by the user manually, and is installed automatically in the mode of dynamic linkage!

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.

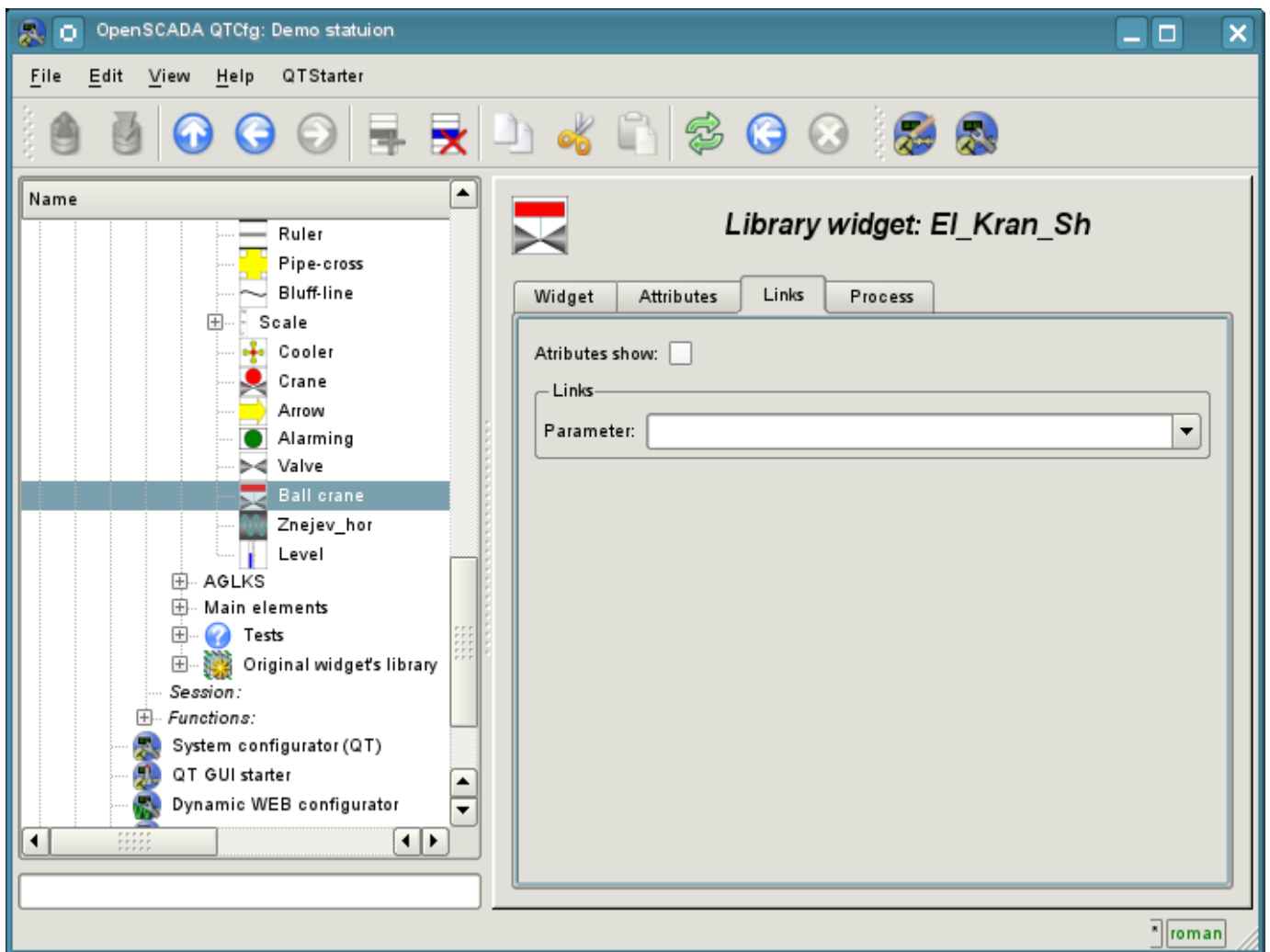


Fig. 3.7.b Tab "Links" of the page of configuration of the widget with the group assignment of the attributes by the only specifying of the parameter.

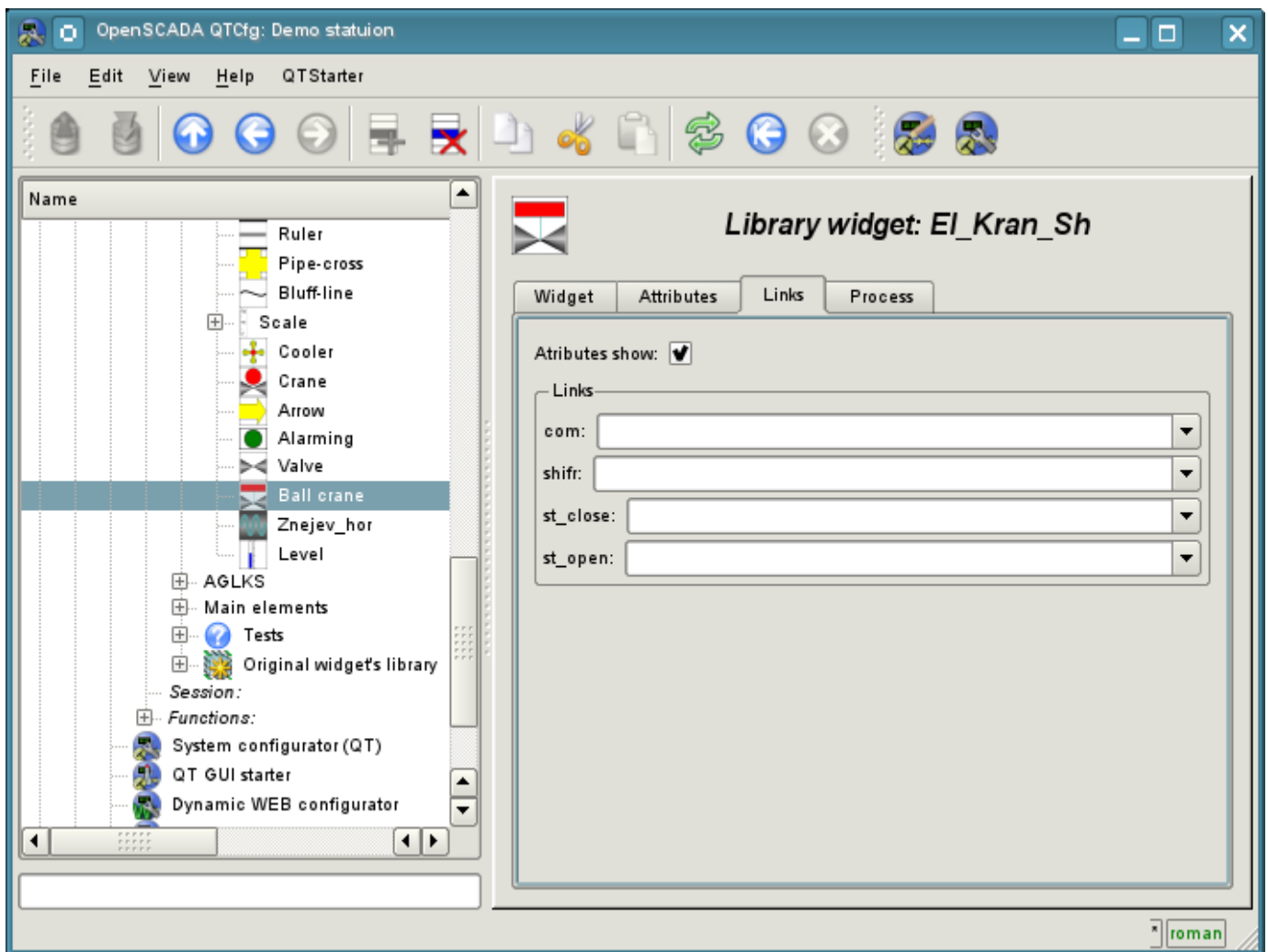


Fig. 3.7.c Tab "Links" of the page of configuration of the widget with the individual appointment of the attributes.

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

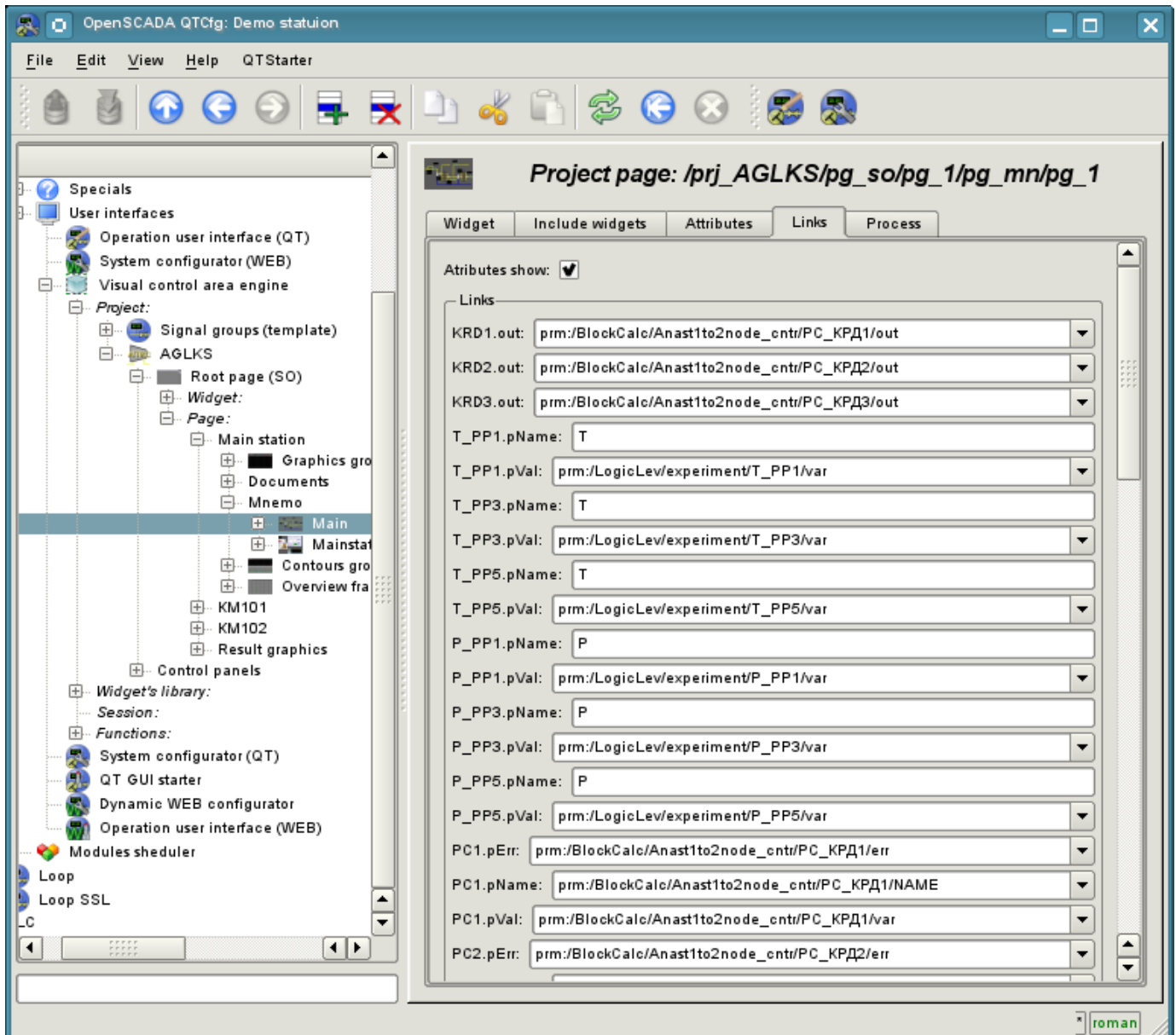


Fig. 3.7.d Tab "Links" of the page of configuration of the container of widgets, including widgets with links.

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	<p>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:</p> <ul style="list-style-type: none"> • Line. • Arc. • Bézier curve. • Fill of the enclosed space. <p>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.</p>

Id	Name	Function
FormEl	Elements of the form.	Includes support for standard form components: <ul style="list-style-type: none"> • 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 for other widgets placement with the purpose of creation of new, more complex widgets and pages of final visualization.
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	No	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 designed to provide information about the location of the element.
parent	Parent	-	Ancestor or parent of the widget. The attribute is read-only and designed to provide information about the location of ancestor from which the widget is inherited from.
root	Root	1	Id of the widget-primitive (basic element) which underlies the image of visualization of the widget.
owner	Owner	-	Widget's owner and group, separated by ":". By default the "root:UI".
perm	Access	-	Access to widget into by select octal numbers of view "RWR_R_". By default the 0664(RWRWR_).
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.

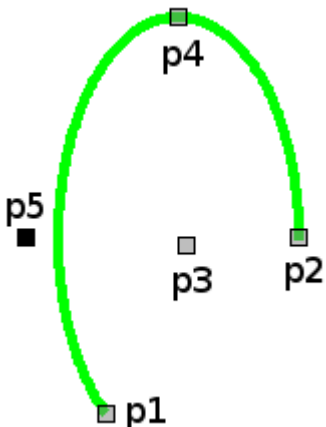
Id	Name	No	Value
active	Active	6	The state of the element - "Active". Active element may receive focus in the execution mode, and thus receive keyboard and other events with their subsequent processing.
geomX	Geometry:x	7	Geometry, coordinate 'x' of the element position.
geomY	Geometry:y	8	Geometry, 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.
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.
contextMenu	Context menu	17	Configuration of the own context menu of the element. The configuration is stored as the lines of entries of context menu in the format:<Name of the entry>:<EventId>. Where: <ul style="list-style-type: none"> • <Name of the entry> - Name of the entry of menu. • <EventId> - Event Id which is generated to the widget (usr_<EventId>) while the selection the menu entry.
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).
<i>Additional attributes for items placed into the project in the role of a page.</i>			
pgOpen	Page:open state	-	Sign "The page is open".
pgNoOpenProc	Page:no open process	-	Sign "Execute the page, even if it is closed".
pgOpenSrc	Page:open source	3	Full address of the page which has opened this one.
pgGrp	Page:group	4	The group of the page.
<i>Additional attributes of the execution mode.</i>			
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	Permission	-3	Virtual attribute of the rights verification of active user on the viewing and control over the widget.

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	No	Value
lineWdth	Line:width	20	Line width.
lineClr	Line:color	21	Line color.(Transparency of the color is defined as follows: red-127 #ff0000-127, where "127" is the value of transparency.)
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.
fillColor	Fill:color	25	Fill color.
fillImg	Fill:image	26	Fill image.
orient	Orientation angle	28	The rotation angle of the content of widget.
eLst	Element's list	27	<p>List of graphic primitives in the following format:</p> <ul style="list-style-type: none"> Line. Record form in the list: <code><line:p1 (x y):p2 (x y):[width w{n}]:[color c{n}]:[border_width w{n}]:[border_color c{n}]:[line_style s{n}]></code> Arc. Record form in the list: <code><arc:p1 (x y):p2 (x y):p3 (x y):p4 (x y):p5 (x y):[width w{n}]:[color c{n}]:[border_width w{n}]:[border_color c{n}]:[line_style s{n}]></code>  <ul style="list-style-type: none"> Bézier curve. Record form in the list: <code><bezier:p1 (x y):p2 (x y):p3 (x y):p4 (x y):[width w{n}]:[color c{n}]:[border_width w{n}]:[border_color c{n}]:[line_style s{n}]></code> Fill. Record form in the list: <code><fill:p1 (x y),p2 (x y),...,pn (x y):[fillClr c{n}]:[fillImg i{n}]></code>
<i>The attributes for each point from the list of graphic figures eLst</i>			
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}.
i{n}	Image {n}	30+n*6+4	Image {n}.
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.
<i>Line edit:</i>			
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	<p>Configuration of the line. The format of the value of the field for different types of lines:</p> <p><i>Text</i> - the formatted 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.</p> <p><i>Combobox</i> - contains a list of the values of the editable combobox.</p> <p><i>Integer</i> - contains the configuration of input field of integer in the format: <Minimum>:<Maximum>:<Step of change>:<Prefix>:<Suffix>.</p> <p><i>Real</i> - 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>.</p> <p><i>Time, Date, Date and time</i> - to form the date following the the template with parameters: d - number of the day (1-31); dd - number of the day (01-31);</p>

Id	Name	№	Value
			ddd - acronym of the day ('Mon' ... 'Sun'); dddd - the full name of the day ('Monday' ... 'Sunday'); M - number of the month (1-12); 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.
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
<i>Text edit:</i>			
value	Value	21	The contents of the editor.
wordWrap	Word wrap	22	Automatic division of text by the words.
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
<i>Check box:</i>			
name	Name	26	Name/label of the checkbox.
value	Value	21	Value of the checkbox.
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
<i>Button:</i>			
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.
color	Color	23	Color of the button.
colorText	Color:text	27	The color of the text.
checkable	Checkable	24	Sign of functioning as a settled button.
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
<i>Combo box:</i>			
value	Value	21	Current value of the list.
items	Items	22	The entries of the list.
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
<i>List:</i>			
value	Value	21	The selected list value.
items	Items	22	The entries of the list.
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.

Id	Name	№	Value
<i>Slider and the scroll bar:</i>			
value	Value	21	Slider position.
cfg	Config	22	Configuration of the slider in the format: <VertOrient>:<Min>:<Max>:<SinglStep>:<PageStep>. Where: <ul style="list-style-type: none"> • 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.
backImg	Background:image	21	Background image.
bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color.
bordStyle	Border:style	24	Border style (None;Dotted;Dashed;Solid;Double;Groove;Ridge;Inset;Outset).
font	Font	25	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
color	Color	26	Text color.
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).
text	Text	30	The value of the text field.
numbArg	Arguments number	40	Arguments number.
<i>Attributes of the arguments</i>			
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: <ul style="list-style-type: none"> • <i>string</i> : [len] - string width; • <i>real</i>: [wdth];[form];[prec] - value width, the form of значения ('g','f'); • <i>integer</i>: [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.
backImg	Background:image	21	Background image.
bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color.
bordStyle	Border:style	24	Border style (None;Dotted;Dashed;Solid;Double;Groove;Ridge;Inset;Outset).
src	Source	25	Source of media data.
fit	Fit to widget size	26	Sign "Co-ordinate the contents with the size of the widget".
type	Type	27	Type of media (Image;Movie).
areas	Map areas	28	Number of active areas.
<i>The attributes of the video (Movie)</i>			
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.
<i>Active areas</i>			
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	No	Value
backColor	Background:color	20	Background color.
backImg	Background:image	21	Background image.
bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color.
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".
<i>Attributes of the trend/graph (Trend)</i>			
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.
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.
sclMarkFont	Scale:Markers:font	37	Font of markers of scale/grid in the form of {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}. .
valArch	Value archiver	38	Archives of parameters archiver.
parNum	Parameters number	39	The number of parameters that can be displayed on the one trend.
<i>Individual attributes of the parameters of trend/graph</i>			
prm{X}addr	Parametr {X} :address	50+10*{X}	Full address to the parameter {X} or to the archive of values.
prm{X}bordL	Parametr {X} :view border:lower	50+10*{X}+1	The lower limit of the parameter {X}.
prm{X}bordU	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 fro display of trend of the parameter {X}.
prm{X}val	Parametr {X} :value	50+10*{X}+4	Value of the parameter {X} under the cursor.

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	Имя	№	Значение
backColor	Background:color	20	Background color.
backImg	Background:image	21	Background image.
font	Font	22	Text font in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
headVis	Header visible	23	Show header for table or not.
time	Time, sek	24	Current time, seconds.
tSize	Size, sek	25	Query size, seconds.
trcPer	Tracing period (s)	26	Mode and frequency of tracing.
arch	Archival	27	Archiver of the message archive.
tmpl	Template	28	Template of the query in the archive.
lev	Level	29	The level of messages.
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	The displayed columns.
itProp	Item's properties	32	Item's properties number.
<i>Individual attributes of item's properties</i>			
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}. Include special symbols '*' and '?'.
it{X}fnt	Item {X}:font	42+5*{X}	Element {X} font.
it{X}color	Item {X}:color	43+5*{X}	Element {X} color.

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	Document style (CSS).
tmpl	Template	21	XHTML basic template of the document.
doc	Document	22	Pseudo-virtual attribute of the the current (selected) document.
font	Font	26	Basic font of the text in the full record {<Family> <Size> <Bold> <Italic> <Underline> <Strikeout>}.
bTime	Time:begin	24	Start time of the document, seconds.
time	Time:current	23	Time of the document generation, seconds.
n	Archive size	25	Number of documents or the depth of the archive.
<i>Attributes of the enabled archival mode</i>			
aCur	Cursor:archive	-	The position of the current document in the archive. Record of the value <0 produces the archiving of this document.
vCur	Cursor:view	-	Current visual document of the archive. Writing a value of -1 - to select the next document, -2 - to select the previous instrument.

Id	Name	No	Value
<i>Attributes of the archive</i>			
doc{X}	Document {X}	-	Archive document X (0...(n-1))

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.

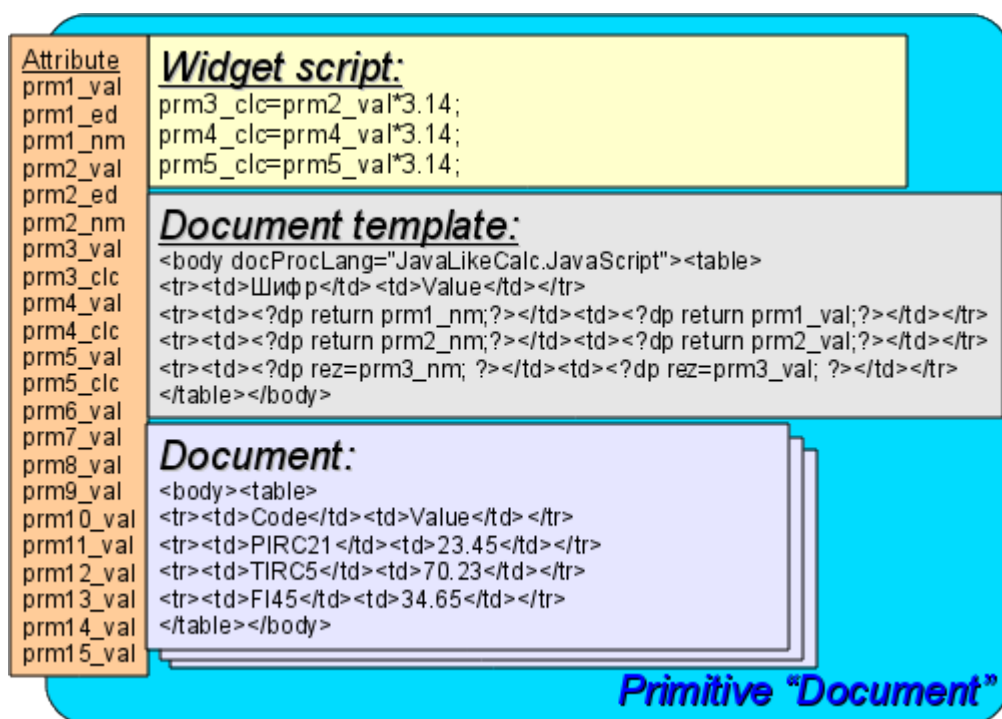


Fig. 3.8.7.a The block diagram of the primitive "Document".

It is 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
<i>Attributes</i>	
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.
<i>Special tags</i>	
<i>Special attributes of the standard tags</i>	
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*). 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!

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	No	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.
backImg	Background:image	21	Background image.
bordWidth	Border:width	22	Border width.
bordColor	Border:color	23	Border color.
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. I.e. 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 "LibWidgets").
- {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.
- LibWidgets(ID, ICO, PARENT, PROC, PROC_PER, USER, GRP, PERMIT, ATTRS, DBV) - 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;
 - DBV* - DB version.
- LibWidgetIO(IDW, IDC, ID, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) - Work attributes <ID> of the widget <IDW>.
 - IDW* - identifier of the widget;
 - IDC* - child widget identifier;
 - ID* - identifier of the IO;
 - 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, IDC, ID, NAME, IO_TP, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) - User attributes <ID> of the widget <IDW>.
 - IDW* - identifier of the widget;
 - IDC* - child widget identifier;
 - ID* - identifier of the IO;
 - 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(IDW, ID, PARENT, ATTRS, USER, GRP, PERMIT, DBV) - 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;

DBV - DB version.

- LibWidgetMime(*ID*, 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(*ID*, NAME, DSCR, DB_TBL, ICO, USER, GRP, PERMIT, PER, FLGS, DBV) - 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;

DBV - DB version.

- ProjPage(*OWNER*, *ID*, 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(*IDW*, *ID*, 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(*IDW*, *ID*, IO_VAL, SELF_FLG, CFG_TMPL, CFG_VAL) - Working attributes of the pages. The structure actually corresponds to the table LibWidgetIO.

- ProjPageUserIO(*IDW*, *ID*, 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(*IDW*, *ID*, 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 almSndPlay()* - The widget's path for that on this time played the alarm message.
- *int almQuittance(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>.
- *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, ELP 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);
```

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
<i>Request command of the visual attributes of the widget: <code><get path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattr"/></code></i>		
tm	Time/counter of changes	Time/counter of changes set up for the query of the only changed attributes.
<code><el id="{attr}" p="{a_id}">{val}</el></code>	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}.
<i>The set command of the visual attributes of the widget: <code><set path="/UI/VCAEngine/{wdg_addr}/%2fserv%2fattr"/></code></i>		
<code><el id="{attr}">{val}</el></code>	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
<i>Command of request of the list of open sessions for the project: <code><list path="/UI/VCAEngine/%2fserv%2fsess"/></code></i>		
prj	Indication of the project	Specifies the project for which to return the list of open sessions.
<code><el>{Session}</el></code>	Control of the sessions' list	In the child element the open for the requested project sessions are specified.
<i>The command of the connection/opening of the session: <code><connect path="/UI/VCAEngine/%2fserv%2fsess"/></code></i>		
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 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.
<i>The command of disconnection/closing of the session: <code><disconnect path="/UI/VCAEngine/%2fserv%2fsess"/></code></i>		
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).

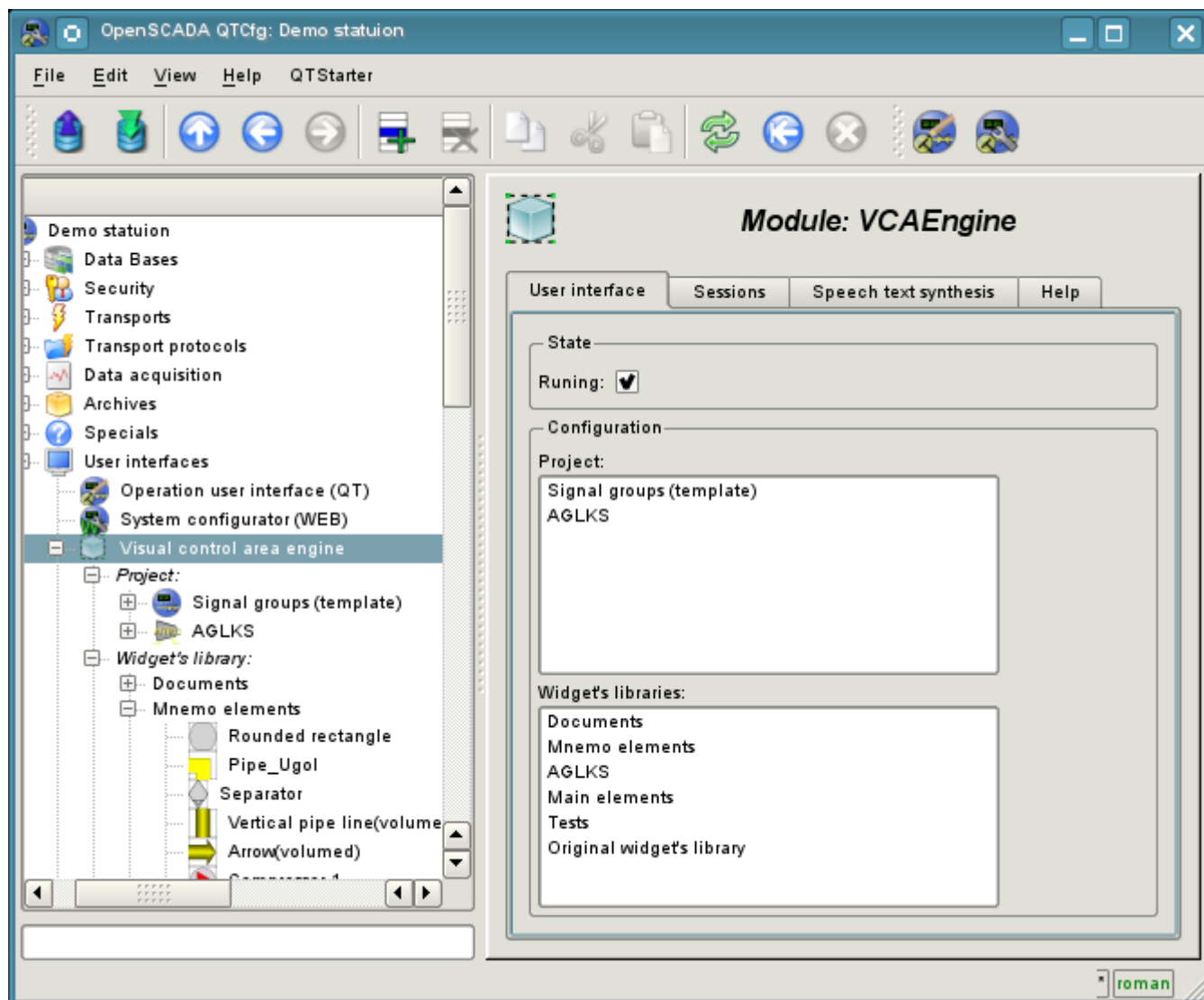


Fig.4.1 Main configuration page of the module.

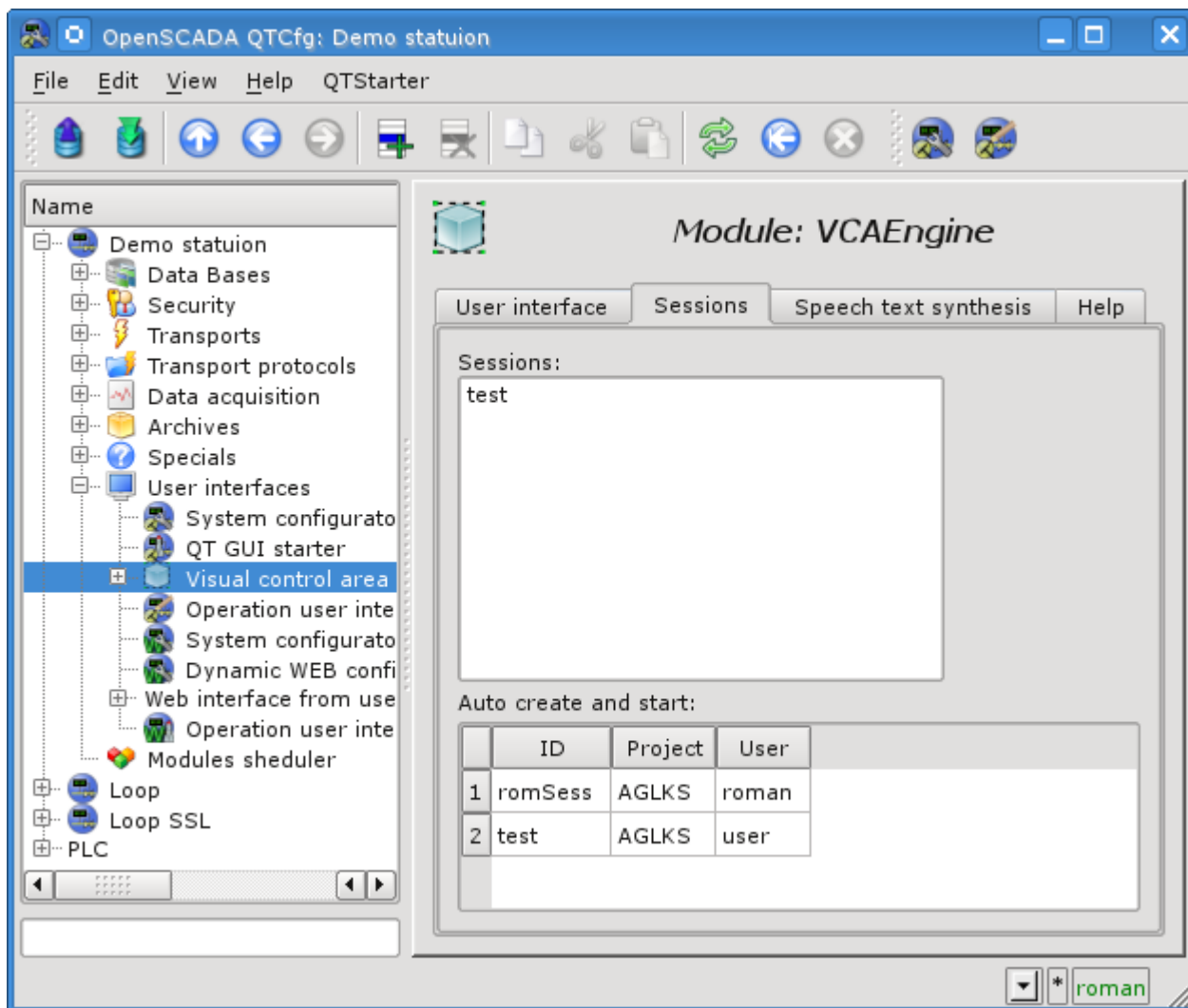


Fig.4.2 The inset "Sessions" of configuration page of the module.

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.

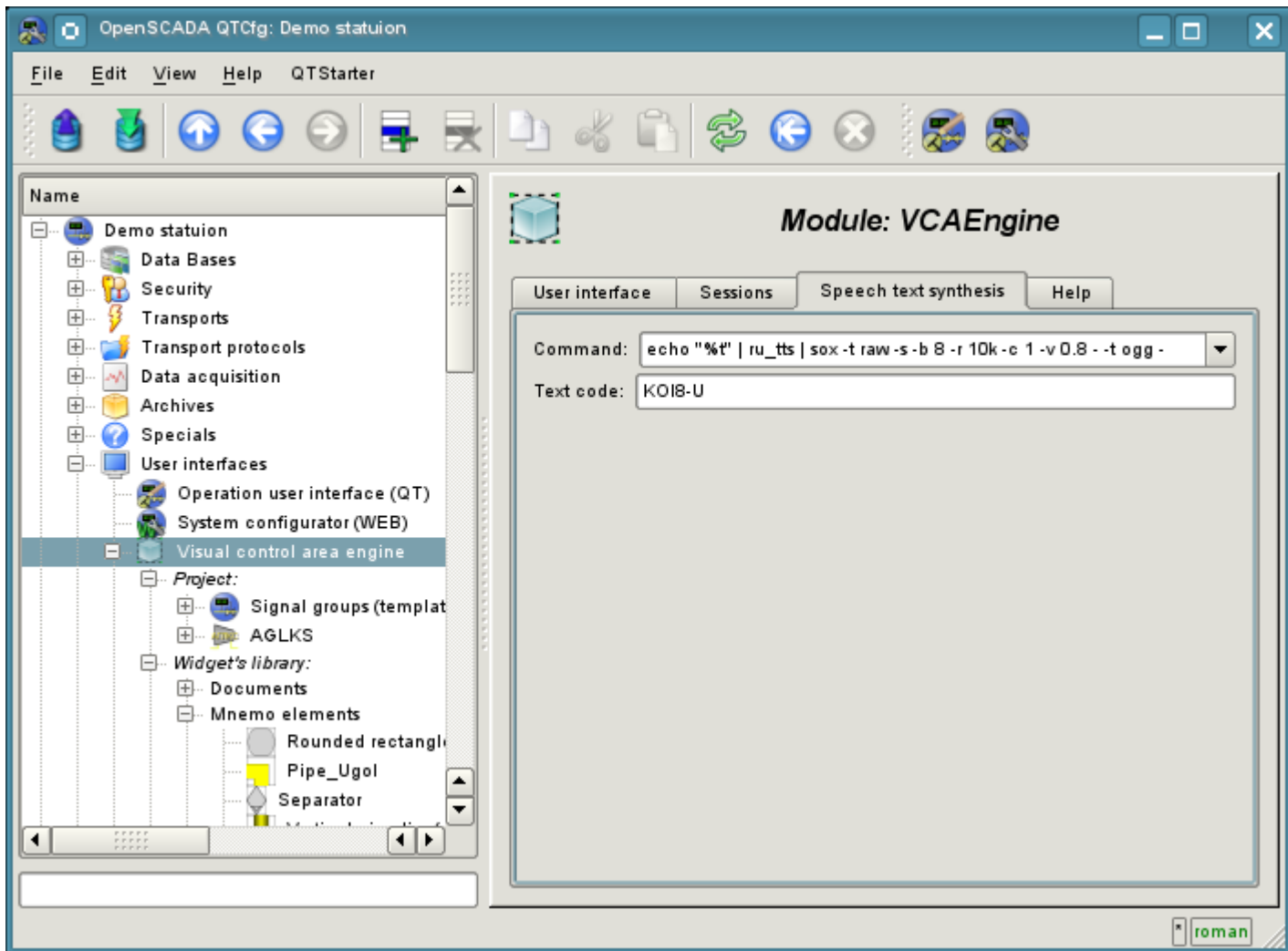


Fig.4.3 The inset for speech synthesis engine configuration.

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).

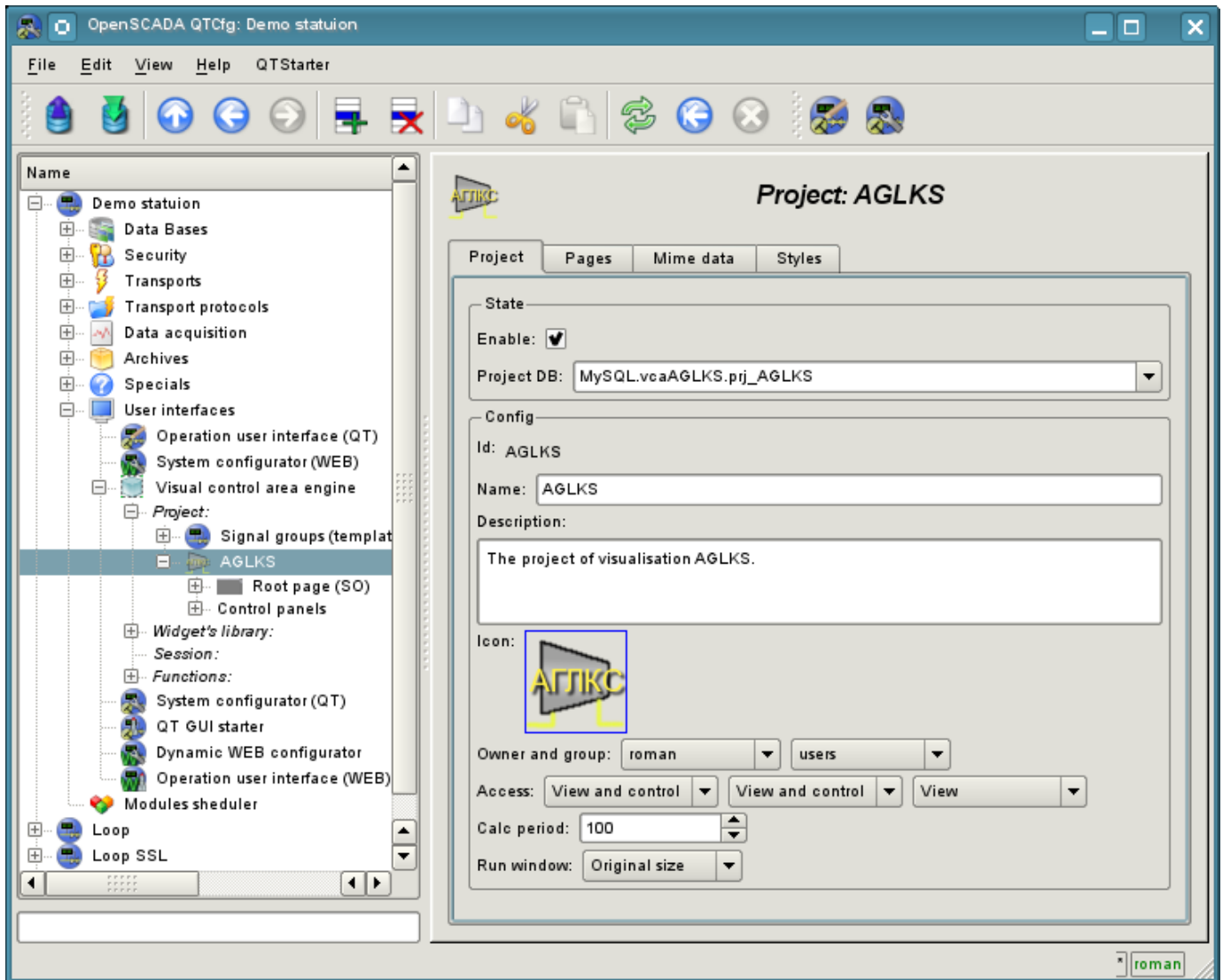


Fig.4.4 The configuration page of the projects.

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).

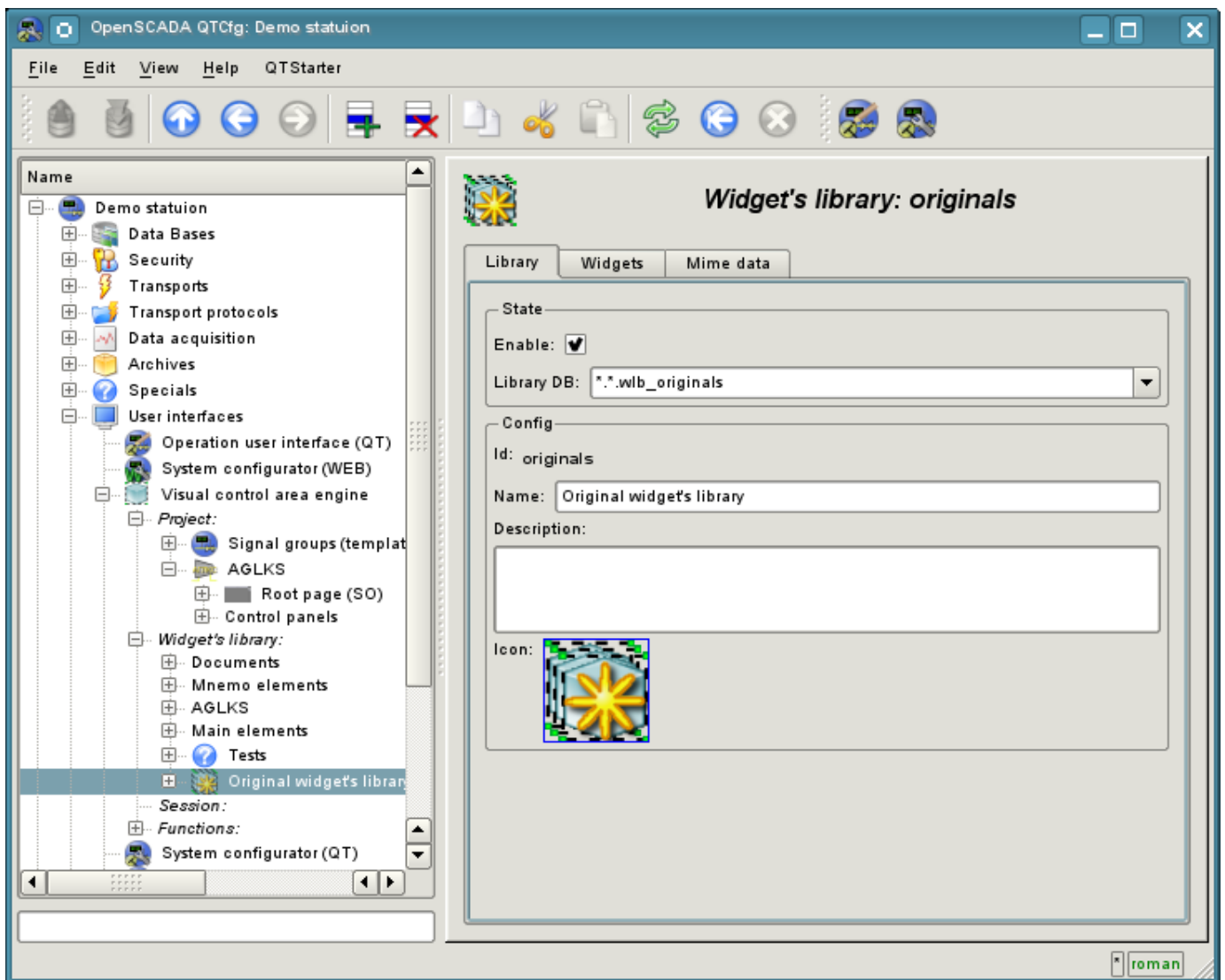


Fig.4.5 The configuration page of the widgets libraries.

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.

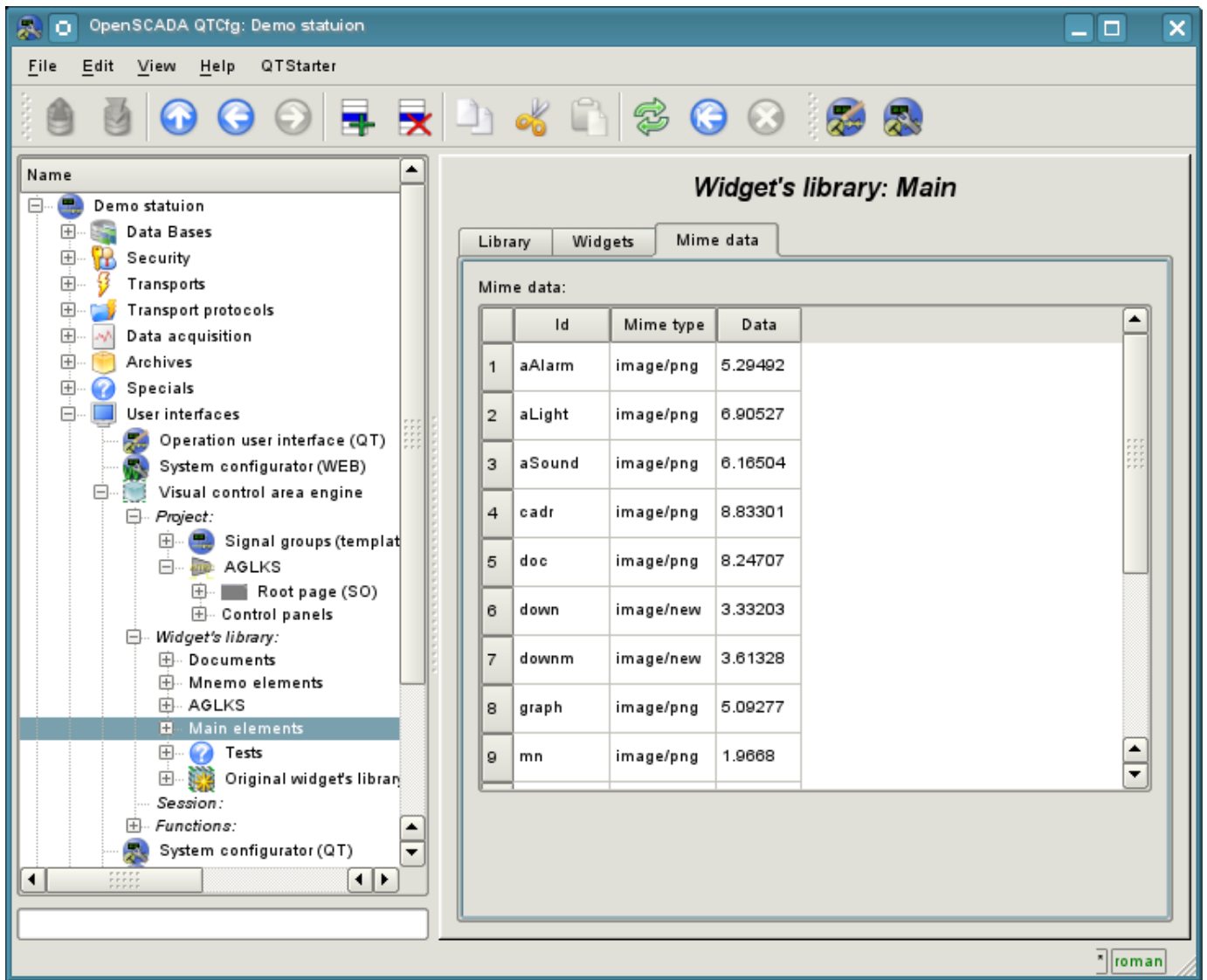


Fig.4.6 The configuration tab of the Mime-data of the container.

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

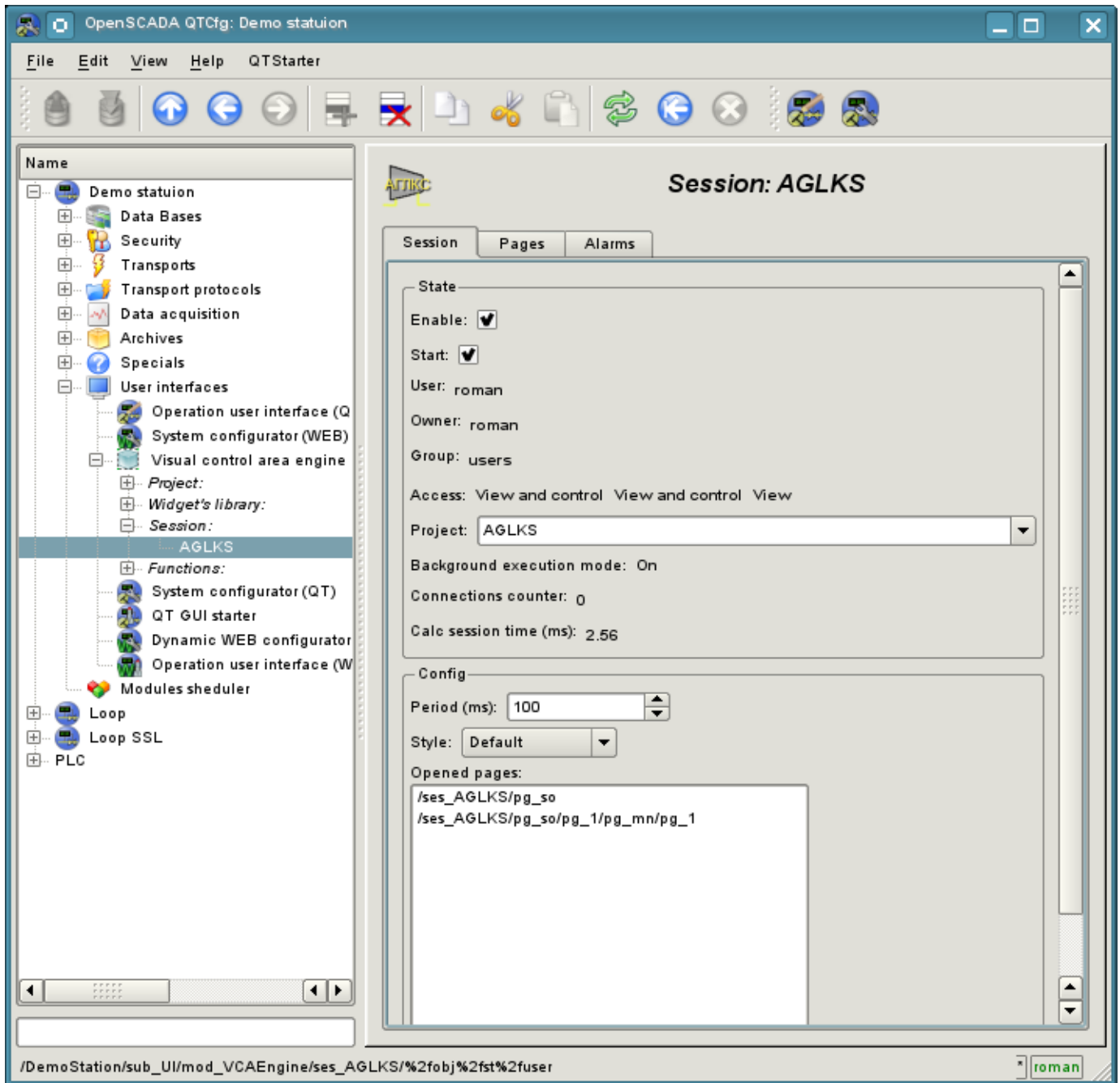


Fig.4.7 The configuration page of the sessions of the projects.

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).

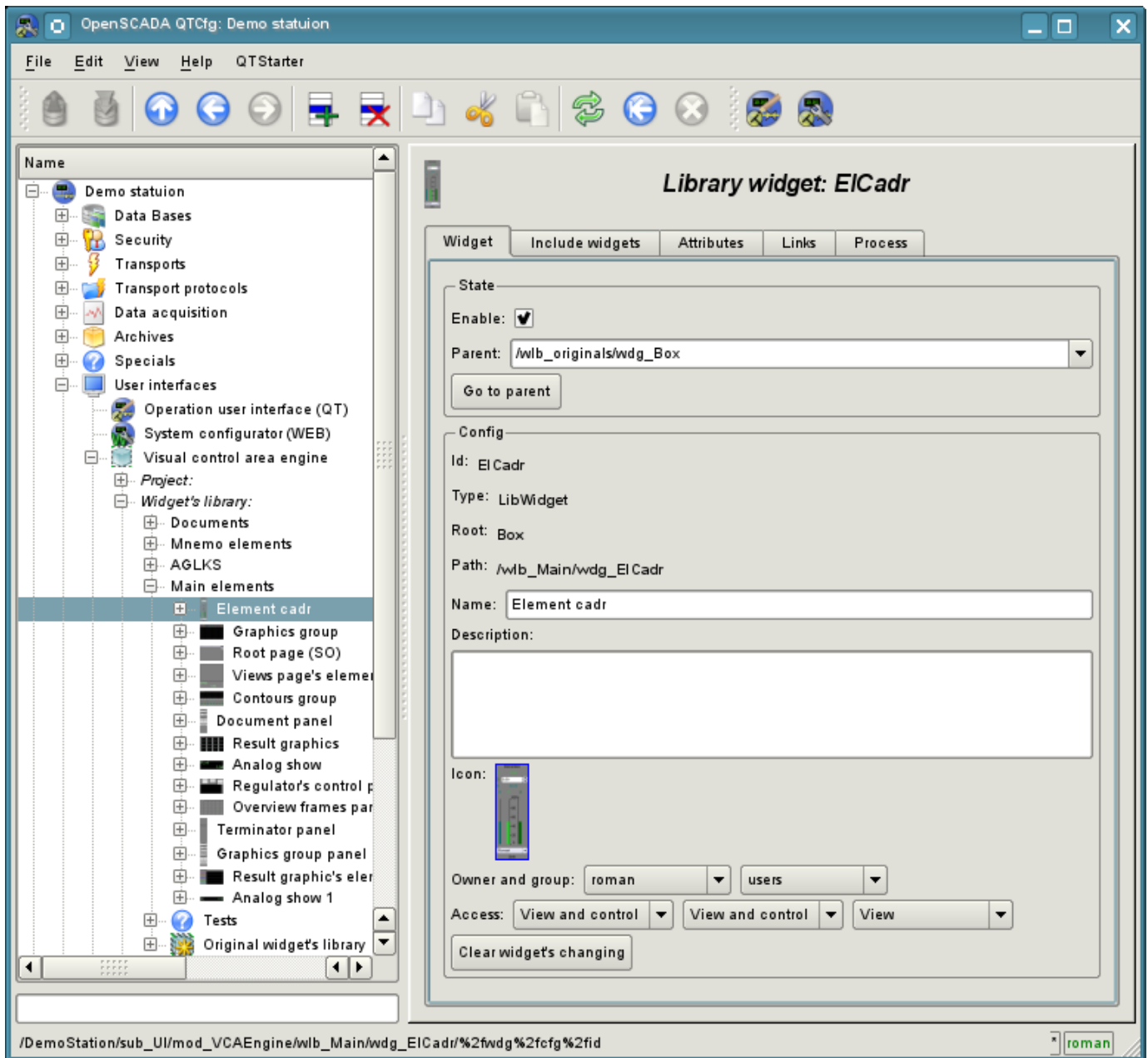


Fig.4.8 Main tab of the configuration of visual elements.

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.

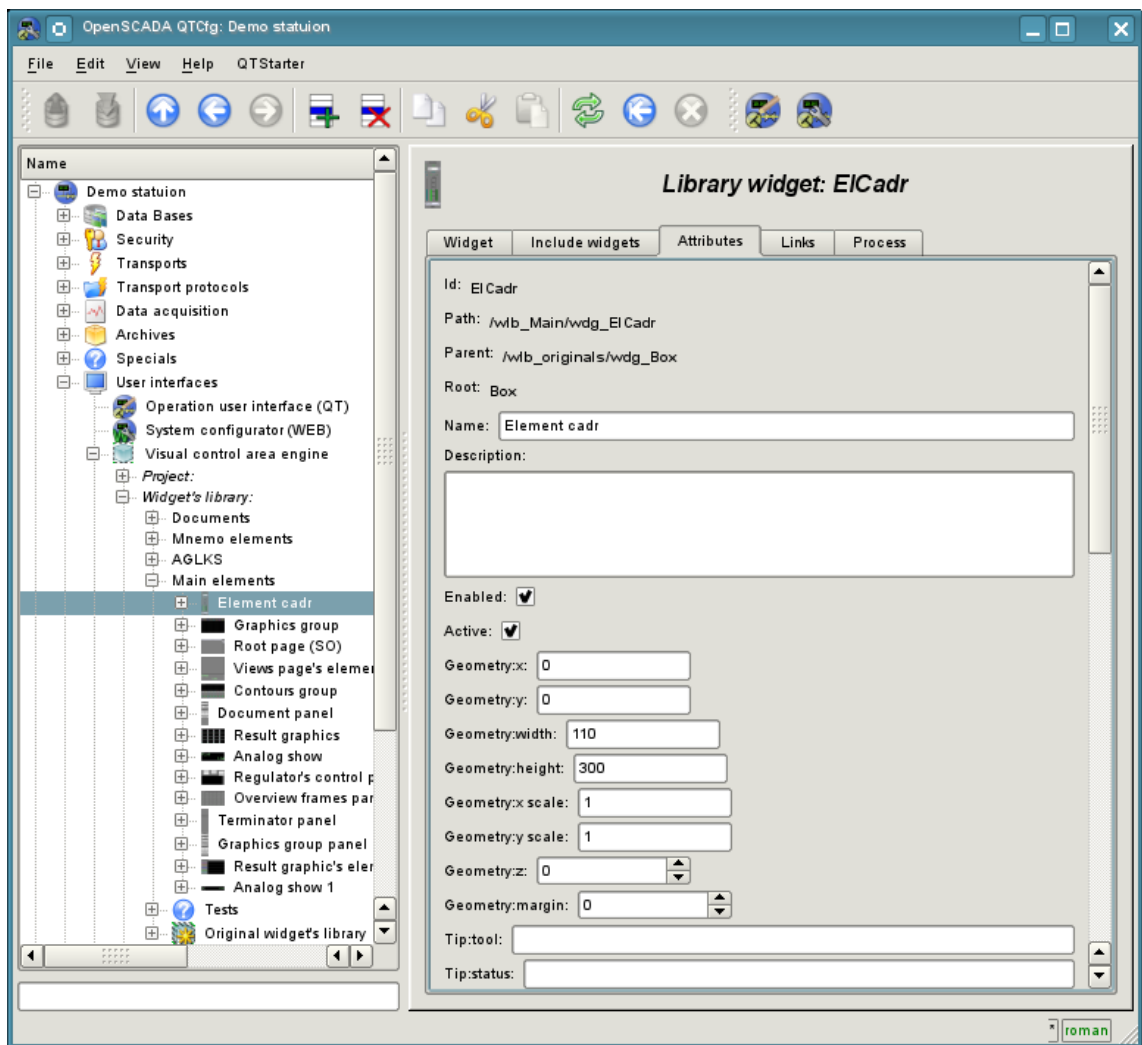


Fig.4.9 Tab of the attributes of visual elements.

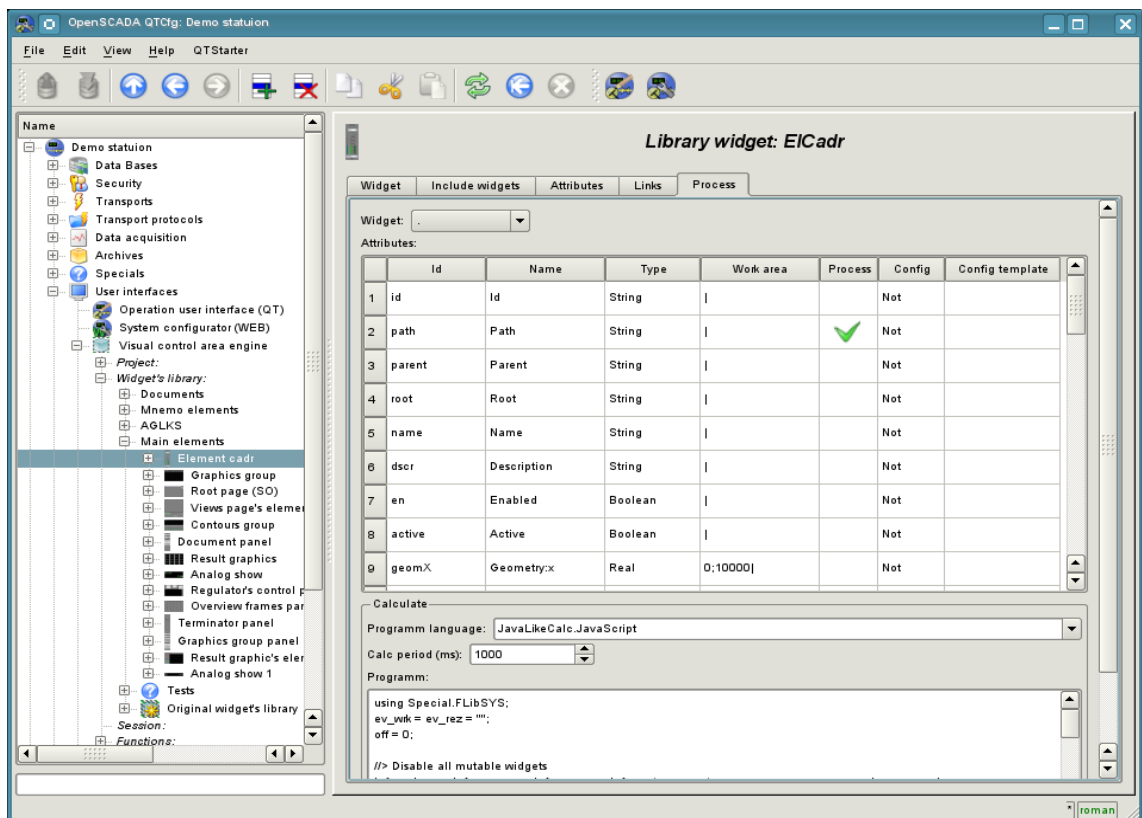


Fig.4.10 Tab of the processing of visual elements.

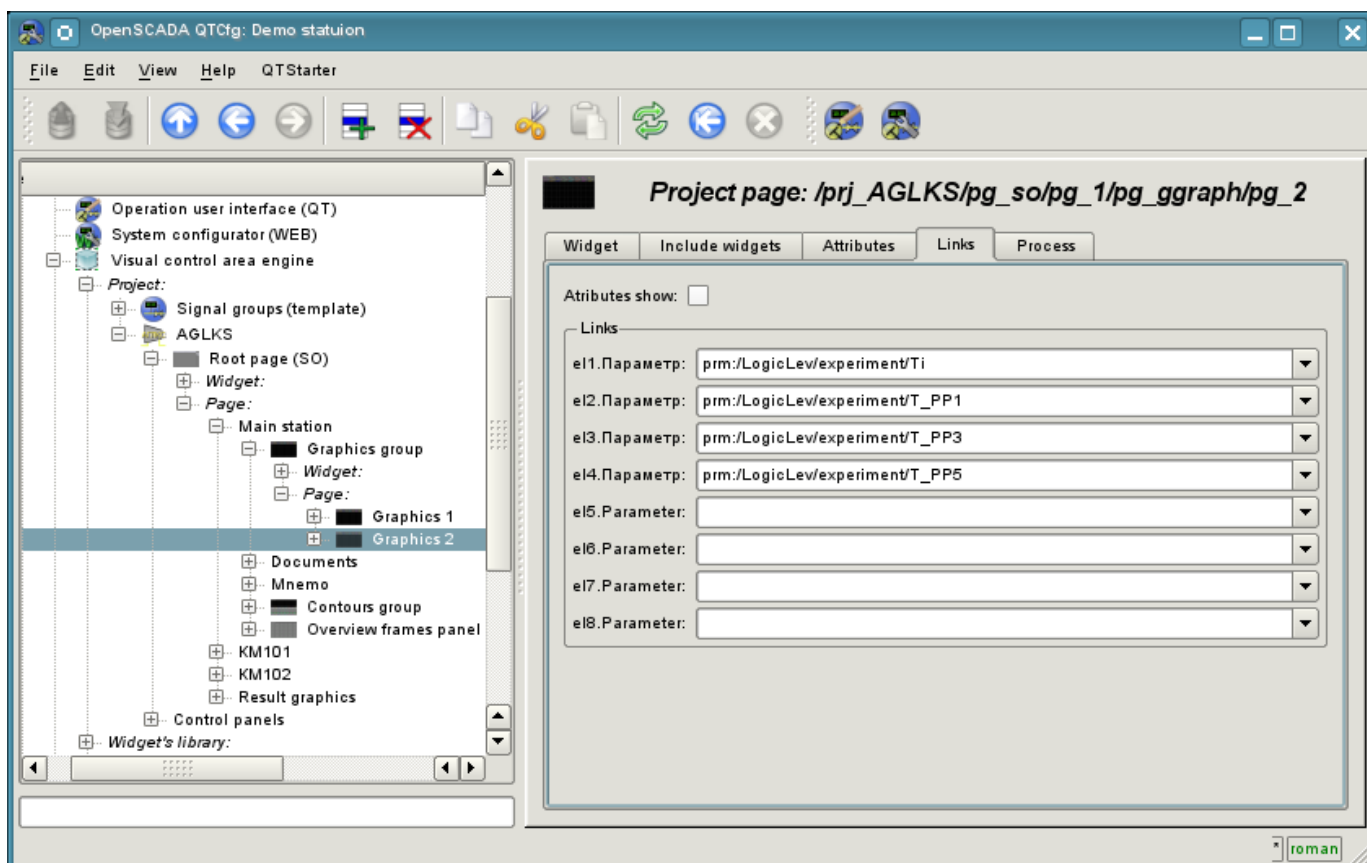


Fig.4.11 Tab of the links of the visual elements.

The module <Vision> of subsystems "User Interfaces"

<i>Module:</i>	Vision
<i>Name:</i>	Operation user interface (QT)
<i>Type:</i>	User interfaces
<i>Source:</i>	ui_Vision.so
<i>Version:</i>	1.0.0
<i>Author:</i>	Roman Savochenko
<i>Developers:</i>	Roman Savochenko, Maxim Lysenko, Ksenia Yashina
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Visual operation user interface.
<i>License:</i>	GPL

Vision module provides a mechanism of the final visualization control area (VCA) into the OpenSCADA. The module is based on the multi-platform library of graphical user interface (GUI) of firm TrollTech - QT (<http://www.trolltech.com/qt/>). In its work, the module uses the data of the VCA engine (module [VCAEngine](#)).

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 be 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 [проект создания концепции CBY](#) is established. The result of this project - the direct visualization module (based on the library QT), direct visualization module [WebVision](#) and VCA engine [VCAEngine](#).

1. Purpose

This module of the direct visualization of the VCA is designed for the formation and execution of VCA interfaces among the graphic library QT.

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. Tool of the graphical formation of the VCA interface

Development of the VCA interface is performed in a single window, realizing many documents interface (MDI) interface (Fig. 1). This approach allows you to simultaneously edit multiple frames of various sizes. The following mechanisms for managing the development are used: toolbars, menus and context menus. Most actions are duplicated by different mechanisms, that allows you to quickly find the tool by the preferred method. Navigational interfaces are implemented by the attached windows. Configuration of the toolbars and attached windows is saved on exit and restored at startup that lets you to customize the interface for yourself.

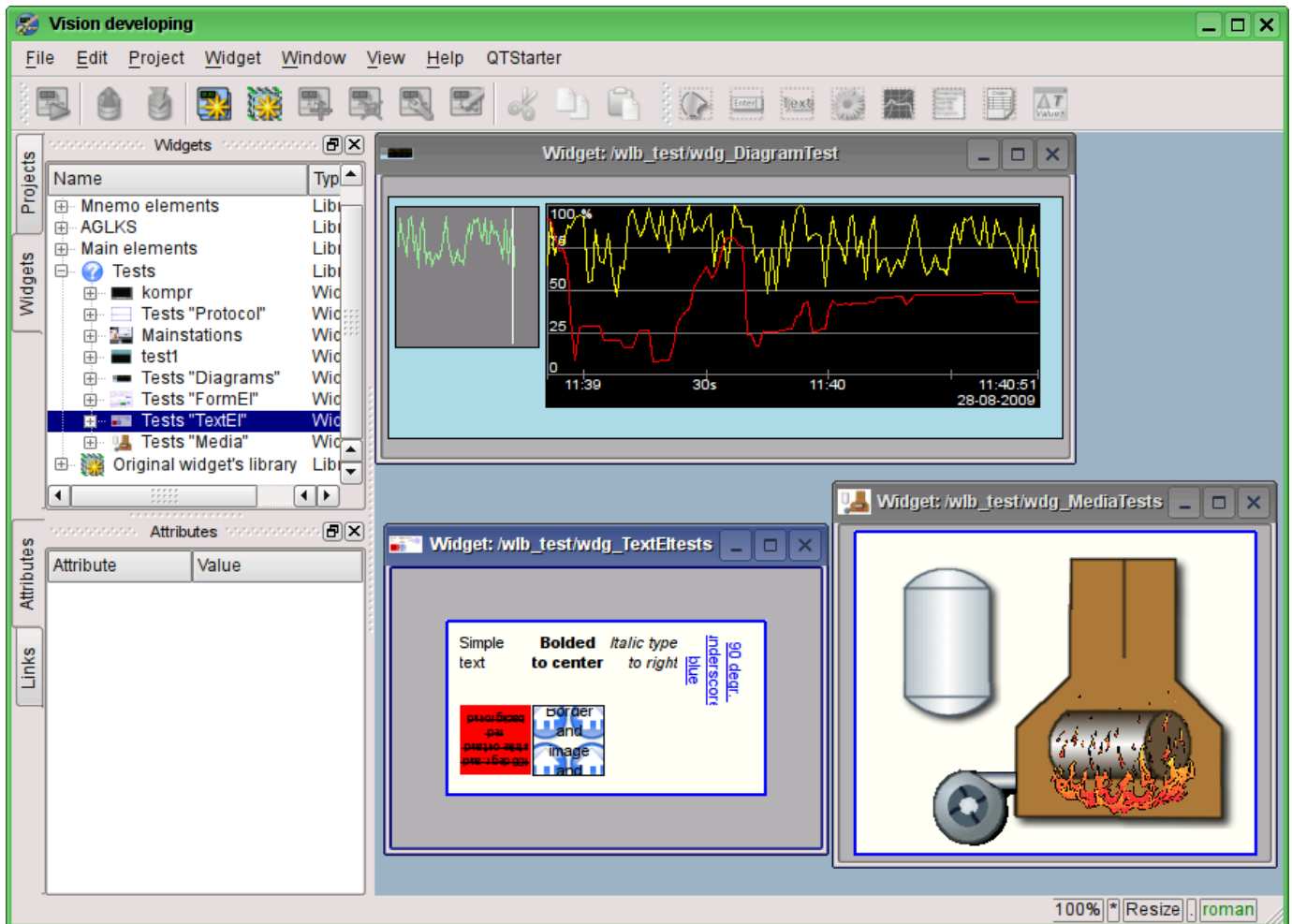


Fig.1. The window of the VCA interface development.

Access to major components of the VCA is made by attached windows, in the Figure 1 these windows are shown on the left side. These windows contain:

- Tree of the widget libraries. Using the navigator you can quickly find the needed widget or library and to do with them necessary operations. The following operations are implemented: add, delete, copy, settings of the widgets and libraries, as well as cleaning and visual editing of the widget. For adaptive management the context menu is supported with the following items:
 - "New library" - creation of the new library.
 - "Add visual item" - adding of the visual element to the library.
 - "Delete visual item" - deleting of the visual element from the library.
 - "Visual item changes clear" - cleaning of the visual element with inheritance of modified properties or setting them by default.
 - "Visual item properties" - configuration of the visual element.
 - "Visual item edit" - visual editing of the element.
 - "Visual item cut" - cut/move of the visual element at the time of paste.
 - "Visual item copy" - copy of the visual element at the time of paste.
 - "Visual item paste" - paste of the visual element.

- "Load from DB" - uploading the data of the visual element from the database.
- "Save to DB" - saving data of visual element to the database.
- "Refresh libraries" - performs rereading of the configuration and composition of the libraries of the data model.
- The tree of pages of the project. Provides the mechanism for "Drag and drop" for creation of the user frames based on the elements of libraries. In order to provide the adaptive management the context menu is supported with the following items:
 - "Run project execution" - starting of the execution of the chosen project.
 - "New project" - creation of the new project.
 - "Add visual item" - adding of the visual element to the project/page.
 - "Delete visual item" - deleting of the visual element from the project/page.
 - "Visual item changes clear" - cleaning of the visual element with inheritance of modified properties or setting them by default.
 - "Visual item properties" - configuration of the visual element.
 - "Visual item edit" - visual editing of the element.
 - "Visual item cut" - cut/move of the visual element at the time of paste.
 - "Visual item copy" - copy of the visual element at the time of paste.
 - "Visual item paste" - paste of the visual element.
 - "Load from DB" - uploading the data of the visual element from the database.
 - "Save to DB" - saving data of visual element to the database.
 - "Refresh libraries" - performs rereading of the configuration and composition of the libraries of the data model.
- attributes of widgets;
- external links of widgets.

In the main space of the working window the pages of projects, frames of the widgets' libraries, user elements and elements of primitives at the time of their visual editing are placed.

At the top of the working window there is the menu. All the tools needed for development the VCA interfaces are placed in the menu. Menu has the following structure:

- "File" - General operations.
 - "Load from DB" - uploading the data of the visual element from the database.
 - "Save to DB" - saving data of visual element to the database.
 - "Close" - close the editor's window
 - "Quit" - quit from the OpenSCADA system.
- "Edit" - Editing operations of the visual elements.
 - "Visual item cut" - cut/move of the visual element at the time of paste.
 - "Visual item copy" - copy of the visual element at the time of paste.
 - "Visual item paste" - paste of the visual element.
- "Project" - Operations over the projects.
 - "Run project execution" - starting of the execution of the chosen project.
 - "New project" - creation of the new project.
 - "Add visual item" - adding of the visual element to the project.
 - "Delete visual item" - deleting of the visual element from the project.
 - "Visual item changes clear" - cleaning of the visual element with inheritance of modified properties or setting them by default.
 - "Visual item properties" - configuration of the visual element.
 - "Visual item edit" - visual editing of the element.
- "Widget" - Operations over the widgets and the libraries of widgets.
 - "New library" - creation of the new library.
 - "Add visual item" - adding of the visual element to the library.
 - "Delete visual item" - deleting of the visual element from the library.
 - "Visual item changes clear" - cleaning of the visual element with inheritance of modified properties or setting them by default.
 - "Visual item properties" - configuration of the visual element.
 - "Visual item edit" - visual editing of the element.
 - "View" - Management of the arrangement of visual elements on the frame.

- "Rise widget" - rising the widget to the top.
- "Lower widget" - lowering the widget to the very bottom.
- "Up widget" - to rise the widget above.
- "Down widget" - to lower the widget below.
- "Align to left" - alignment of the widget to the left.
- "Align to vertical center" - alignment of the widget vertically to the center.
- "Align to right" - alignment of the widget to the right.
- "Align to top" - alignment of the widget to the top.
- "Align to horizontal center" - horizontal alignment of the widget in the center.
- "Align to bottom" - alignment of the widget to the bottom.
- "Library: {Name of the library}" - menu items to access the frames/widgets in the library.
- "Window" - Management of the windows of MDI-interface.
 - "Close" - to close the active window.
 - "Close all" - to close all the windows.
 - "Tile" - to tile all the windows for visibility at the same time.
 - "Cascade" - to cascade all the windows.
 - "Next" - to activate the next window.
 - "Previous" - to activate the previous window.
 - "Widget: {Name of the widget}" - items of activation of the specific window.
- "View" - Management of the visibility of the working window and the toolbars on it.
 - "Visual items toolbar" - visual element toolbar.
 - "Widgets view functions" - the toolbar for management of the visibility and arrangement of widgets on the panels.
 - "Elementary figures tools" - Additional toolbar for the editing the primitive of elementary figures ("ElFigure").
 - "Projects" - attached window of management of projects' tree.
 - "Widgets" - attached window of management of widgets' libraries tree.
 - "Attributes" - attached window of the attributes' manager.
 - "Links" - attached window of the links' manager.
 - "Library: {Name of the library}" - management of the visibility of widgets' libraries toolbars.
- "Help" - Help for OpenSCADA and fro Vision module.
 - "About" - information about this module.
 - "About QT" - Information about the QT library, used by this module.
 - "What's this" - query of the description of the elements of the window's interface.

Above, under menu, or on the sides, there are the toolbars. Toolbars can be hidden or located, which is controlled in the menu item "View". The following toolbars are present:

- "Visual items toolbar" - Management toolbar of the visual items:
 - "Run project execution for selected item" - runs the project for execution and activates the selected page of the project.
 - "Load item data from DB" - uploading the data of the chosen elements from the database.
 - "Save item data to DB" - saving data of chosen elements to the database.
 - "New project" - creation of the new project.
 - "New library" - creation of the new library.
 - "Add visual item" - adding of the visual element to the project.
 - "Delete visual item" - deleting of the visual element from the project.
 - "Visual item's properties" - configuration of the visual element.
 - "Visual item edit" - visual editing of the element.
 - "Visual item cut" - cut/move of the visual element at the time of paste.
 - "Visual item copy" - copy of the visual element at the time of paste.
 - "Visual item paste" - paste of the visual element.
- "Widgets view functions" - The toolbar of visibility and arrangement management of widgets on the panels:
 - "Rise widget" - rising the widget to the top.
 - "Lower widget" - lowering the widget to the very bottom.

- "Up widget" - to rise the widget above.
- "Down widget" - to lower the widget below.
- "Align to left" - alignment of the widget to the left.
- "Align to vertical center" - alignment of the widget vertically to the center.
- "Align to right" - alignment of the widget to the right.
- "Align to top" - alignment of the widget to the top.
- "Align to horizontal center" - horizontal alignment of the widget in the center.
- "Align to bottom" - alignment of the widget to the bottom.
- "Elementary figure tools" - Additional toolbar of the editing of the elementary figures primitive ("ElFig").
 - "Cursor" - return to the cursor for the action over the figures on the widget.
 - "Add line" - adding the line to the elementary figure.
 - "Add arc" - adding the arc to the elementary figure.
 - "Add besier curve" - adding the Bézier curve to the elementary figure.
 - "Connections" - the enabling of the of connections at the elementary figure.
- "Library: {Name of the library}" - Management of the visibility of toolbars of the widget libraries. The contents of the panel depends on the contents of the library and includes call buttons of the library items.
- "QTStarter toolbar" - The toolbar, created by the module of the module of starting the QT library modules. It contains buttons to start the UI modules of OpenSCADA, based on the QT Library. With this toolbar you can open multiple copies of the windows of the module or other modules.

At the bottom of the development window of the VCA there is the status line. On the right side of the status line there are indicators of the visual scale of the edited frame, of the mode of changing of the size of the elements, of the mode of the current page of the of the VCA engine station and the user on whose behalf the development of the VCA interface is done. By double-clicking on the indicator of the user it can be changed the current user, enter the new username and password. In the main field of the status line it is displayed various information and assistance messages.

To edit the properties of the visual elements there are two dialogues. The first dialogue allows you to edit the properties of containers of visual elements (widget libraries and projects), figure 2. The second dialogue serves to edit the properties of the visual elements, Fig. 3. Changes, made in the dialogues, at once, get to the VCA engine. To save these changes to the database or restore from the database it is necessary to use the appropriate tools of the main development window.

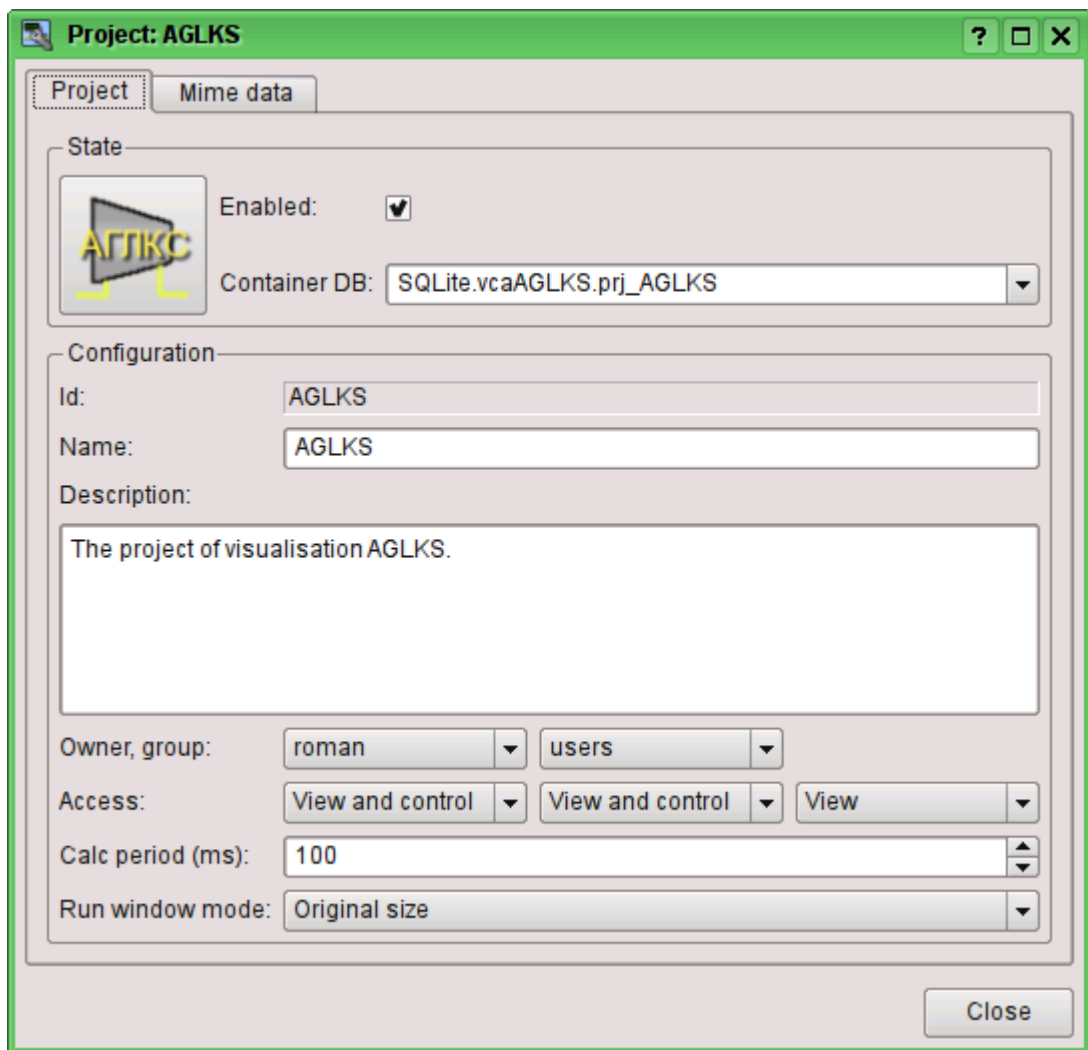


Fig.2. Dialogue of the editing the properties of the containers of visual elements.

With the help of the main tab of that dialog you can set:

- The state of the elements' container, namely: "Enabled", the database container.
- Id, name and description of the container.
- For project: user, group of users and user access, users' group and all the rest.
- For the project: the period for calculating of the project and the mode of opening the windows in the execution.

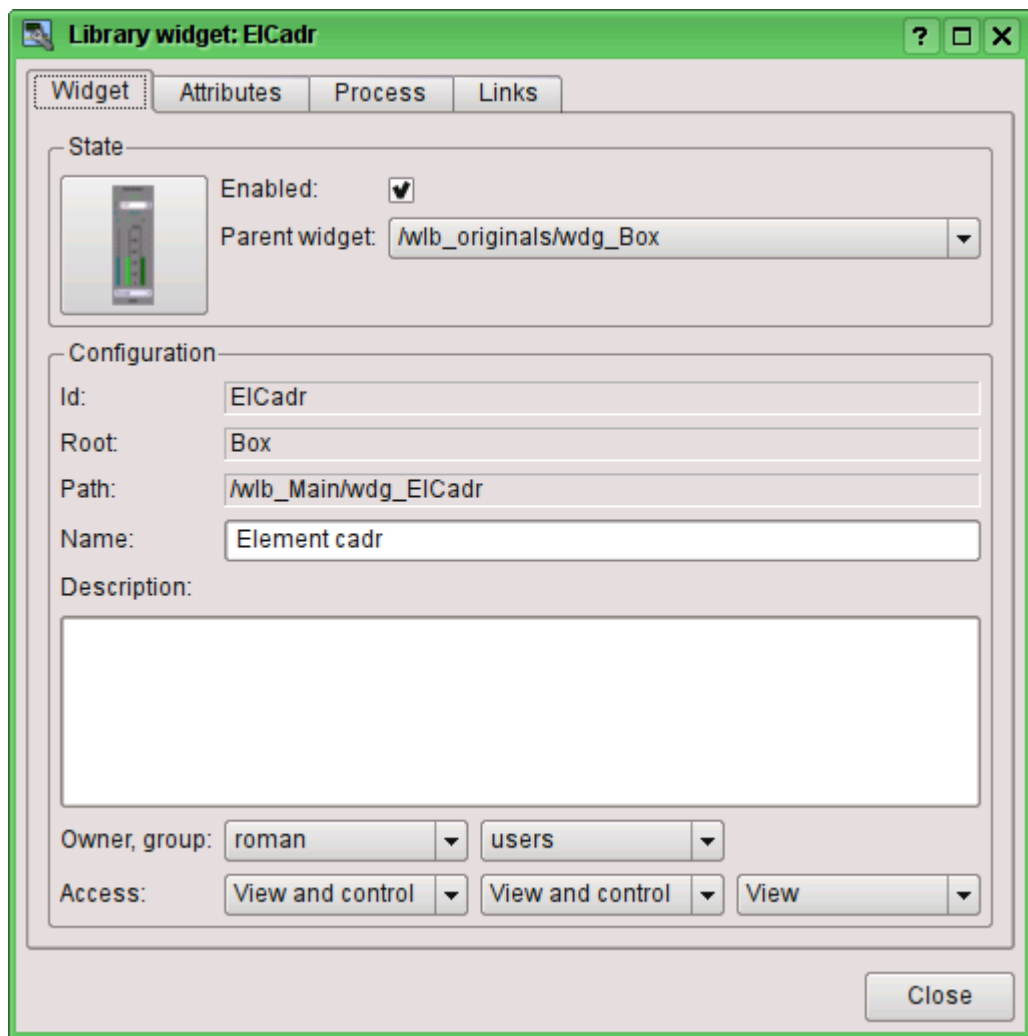


Fig.3. Dialogue of editing the properties of visual elements.

With the help of the main tab of that dialog you can set:

- The state of element, namely: "Enabled", the parent widget.
- Id, root, path, name and description of the element.
- User, group of users of the element and user access, user groups and all the rest.

Dialogue of editing the properties of the containers of visual elements contains two tabs: configuration tab of the the main parameters (Fig.2) and the configuration tab of the mime-data of containers (Fig. 4).

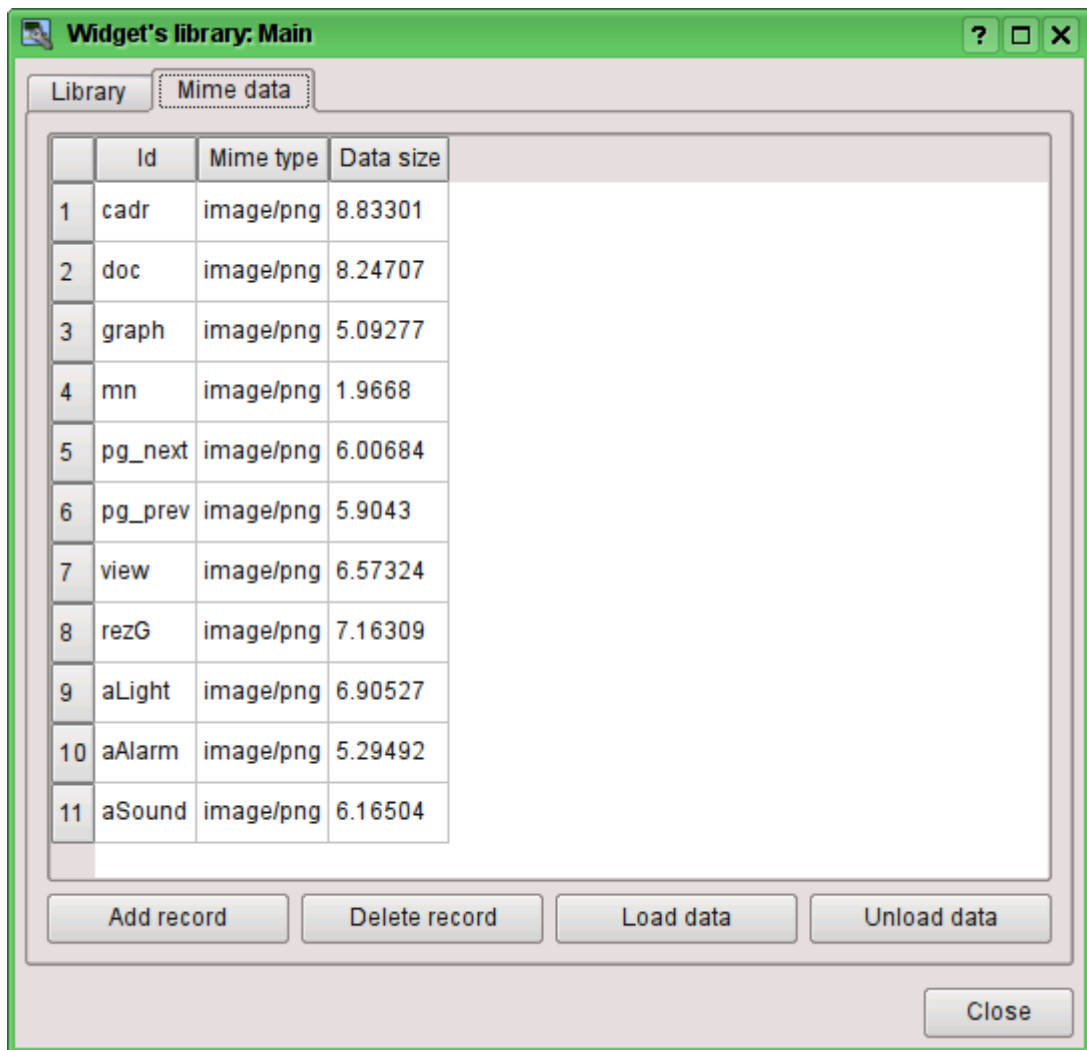


Fig.4. Editing tab of the mime-data of the container of visual elements.

Dialogue of the editing the properties of the visual elements contains four tabs: configuration tab of the main parameters (Fig.2), the tab of attributes of the element (Fig. 5), the tab of the processing of the element (Fig. 6) and the tab of links of the elements (Fig.7). At different levels of the hierarchy of visual elements any tabs can be available, but some are not.

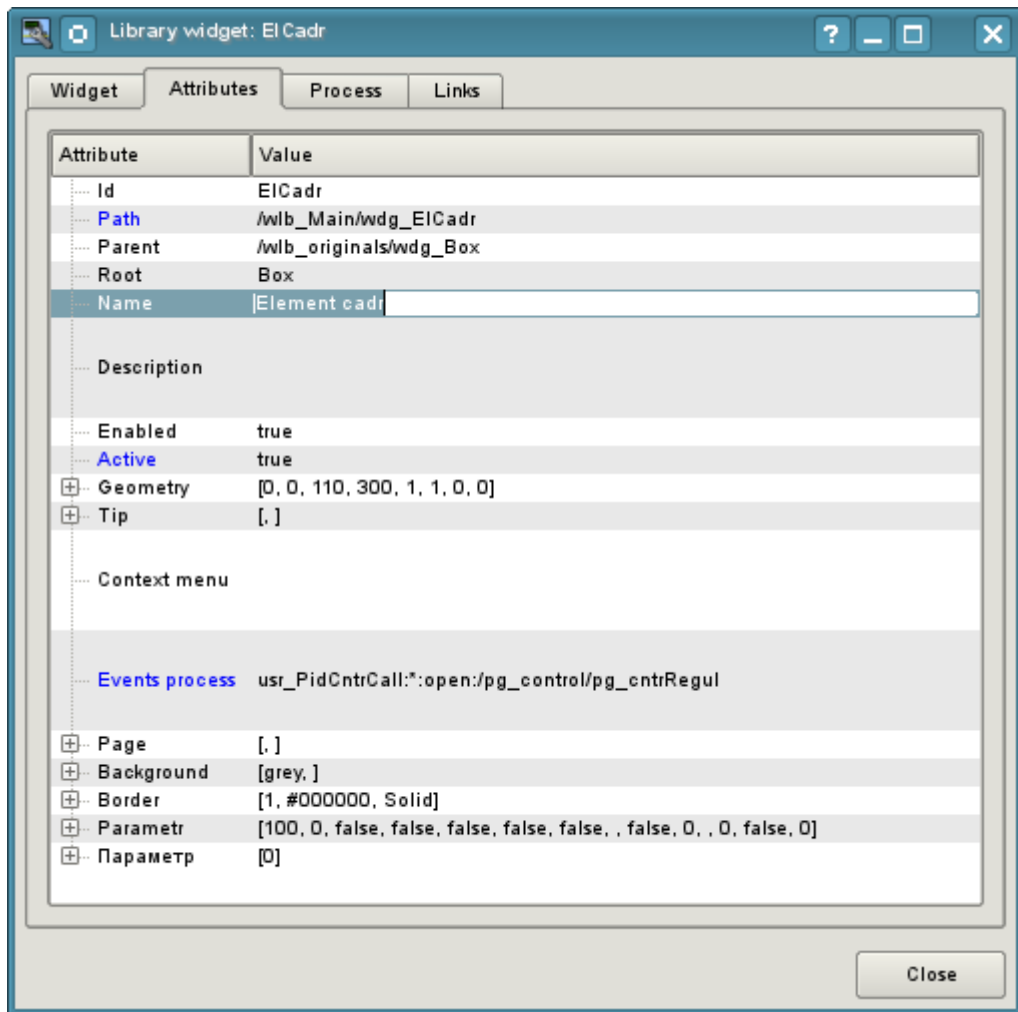


Fig.5. Attributes of the editing dialogue of the properties of the visual element tab.

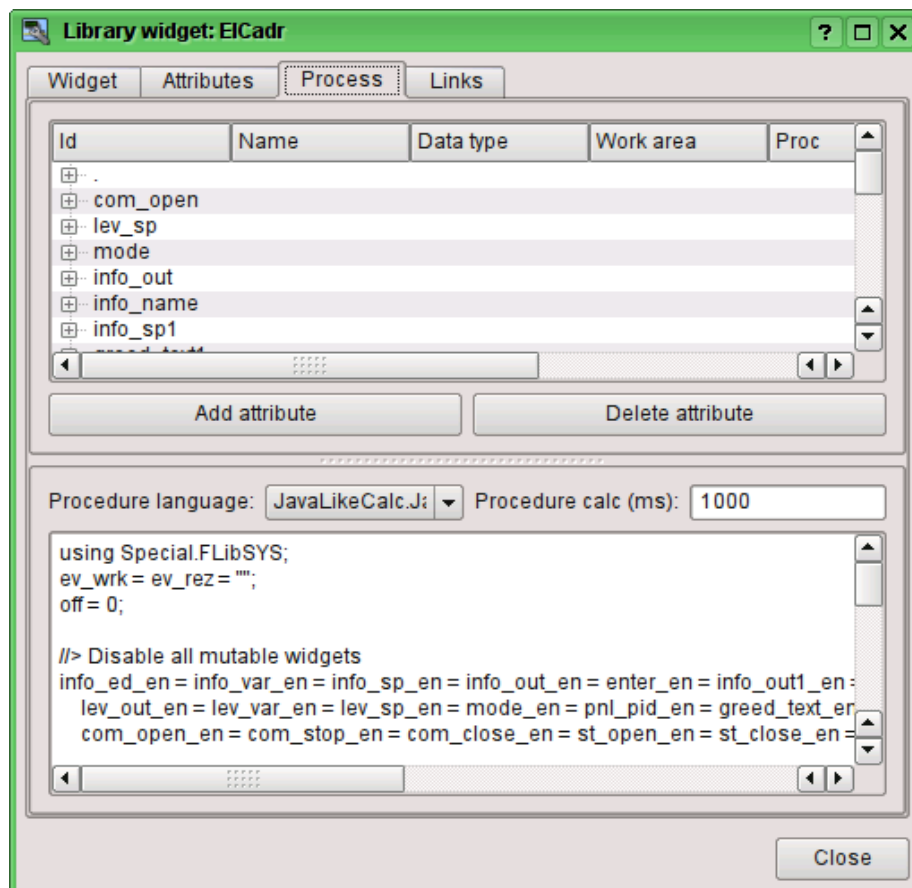


Fig.6. Processing tab of the dialogue of the editing the properties of the visual element.

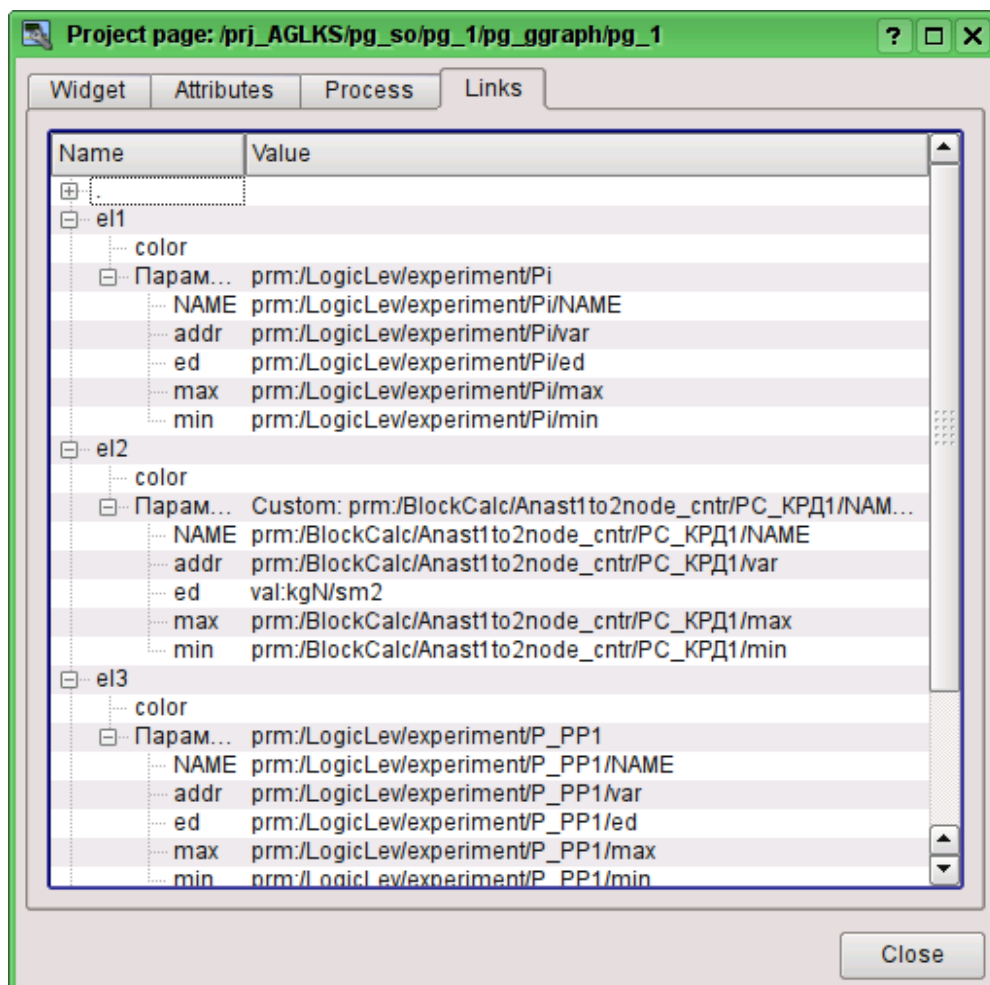


Fig.7. Tab of links of the editing dialog of the properties of visual element.

2.1. Styles

It is known that people can have individual characteristics in the perception of graphical information. If these features are not taken into account it is possible to get the rejection and exclusion of the user to the VC interface. Such rejection and exclusion can lead to fatal errors in the management of TP, as well as traumatize the human by the permanent working with the such interface. In SCADA systems it is accepted the agreement, which regulate the requirements for establishing a unified VC interface which is normally perceived by most of people. The people with some deviations are not taken into account.

To take this into account, and provide the ability to centrally and easily change the visual properties of the interface, the project provides the implementation of visualization interface styles manager.

User can create many styles, each of which will hold the color, font and other properties of the elements of the frame. A simple change of style will quickly transform the VC interface, and the possibility of appointing an individual style to the user will 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. 6). 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. 8). On this tab you can create new styles, delete old ones, change the field of the style and delete unnecessary.

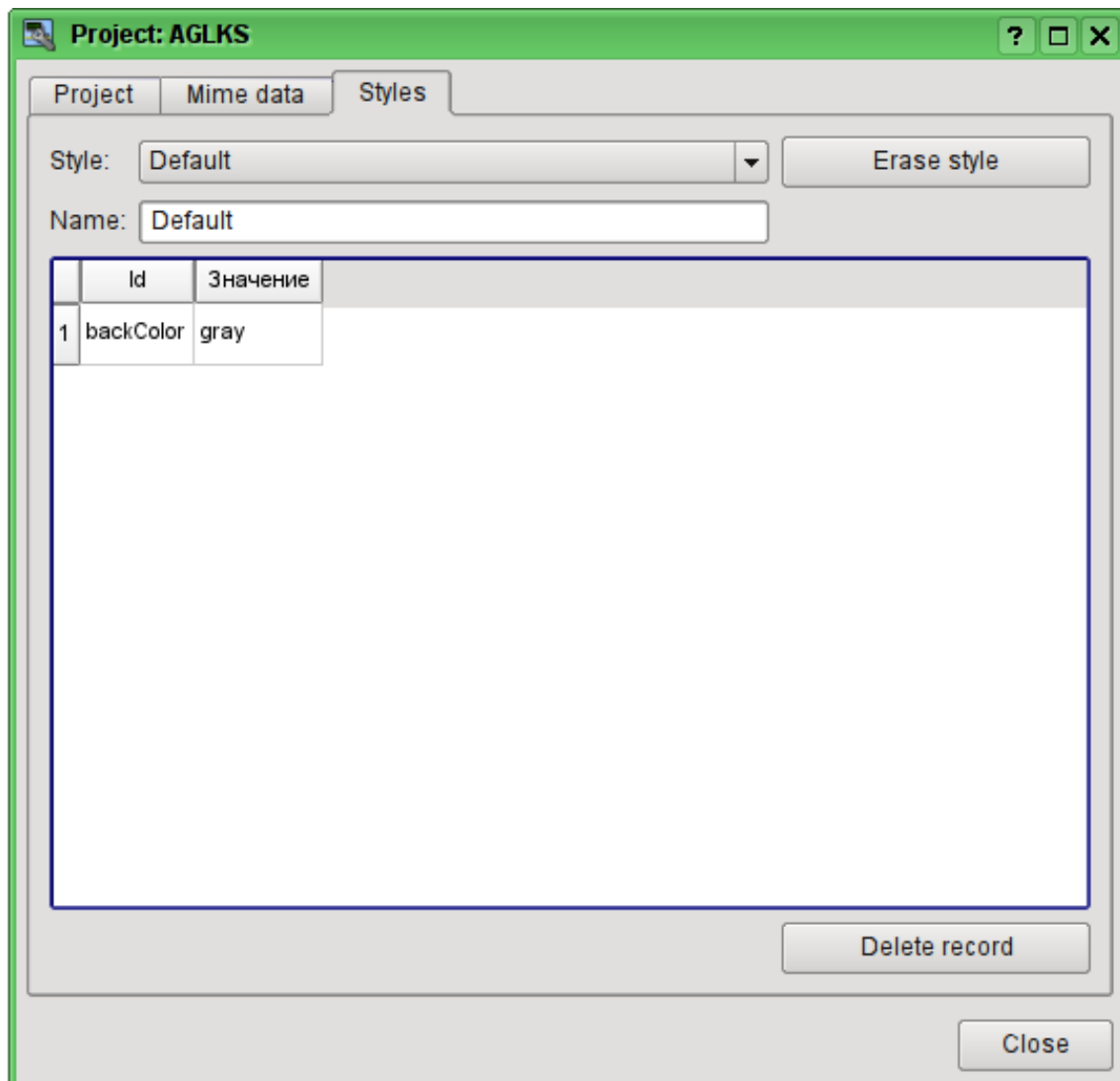


Fig. 8 Styles tab of the configuration page of the project.

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.

2.2. 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.6). 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.

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".

- *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. At that moment this type of link is not intended for installation by the user manually, and is installed automatically in the mode of dynamic linkage!

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.

Fig. 7 presents the tab with the possibility of group and individual assignment of 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.

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 2.1 is cited.

Table 2.1. 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. Execution of the VCA interfaces

Execution of the VCA interface is to run a new project session or connect to the existing one on the level of VCA engine. Then the module of direct visualization represents and manages the data of the session. The main window mode of execution mode of this module has the form presented at Fig.8.

Update of the contents of the open pages of the visualization interface with the frequency of the project session execution. In the updating process it is performed:

- request a list of opened pages, with a sign of page modification, at the model and consistency checking of the actually opened pages to that list;
- request of the branch of the modified pages;
- update of the contents of the modified pages and their widgets, in accordance with the received modified data.

At the closure of "RunTime" window closing of the session of the project is done in the VCA engine.

The mechanism of the request of the only modified data is based on an absolute counter of the session execution. If you want to make real changes in the attributes of widgets the memorizing of the value of this counter is done, which allows the identification of modified attributes. This approach can increase productivity and reduce the load on network sharing in the case of access to the VCA engine via network.

Hierarchically the module provides an opportunity to accommodate the project pages in the main execution window (Fig.8), as well as putting them inside of the container widgets, as well as by the opening of additional windows over the main.

When you expand the main execution window, or when moving to the full-screen mode the scaling of the page content of the VCA interface is done, filling the entire space of the window and allowing to execute the projects, developed on one screen resolution, at different resolutions.

The main window consists of menu (top) status line (bottom), and the executable contents of the session between them. Menu in the execution mode is positioned as the OpenSCADA administrator tool, containing the self-system functions and it is available only to privileged users, occupying the group "root". Menu has the following structure:

- "File" - General operations.
 - "Print" - Print:
 - "Page" - page of the user interface;
 - "Diagram" - diagram on the user interface;
 - "Document" - document on the user interface.
 - "Export" - Export:
 - "Page" - page of the user interface;
 - "Diagram" - diagram on the user interface;
 - "Document" - document on the user interface.
 - "Close" - Close the editor window.
 - "Quit" - Quit from the OpenSCADA system.
- "Alarm" - Alarm quittance:
 - "Alarm level" - all alarms;
 - "Light alarm" - lighting notification;
 - "Speaker alarm" - notification with the whistle;
 - "Sound/speech alarm" - sound/speech notification.
- "View" - Display options of the project session.
 - "Full screen" - Switcher of the full screen execution mode.
- "Help" - Help through the OpenSCADA and Vision module.
 - "About" - Information about this module.
 - "About QT" - Information about the QT library, used by the module.

On the right side of the status line the indicators of the time, the current VCA engine station and users on whose behalf the VCA interface is executed, as well as the panel with the alarm quittance buttons, print and export. By double-clicking on the indicator of the user it can be changed by the typing of the new username and password, and by clicking on the quittance button - to quit alarms completely or only the desired

notification. In the main field of the status line various messages and assistance messages are displayed.

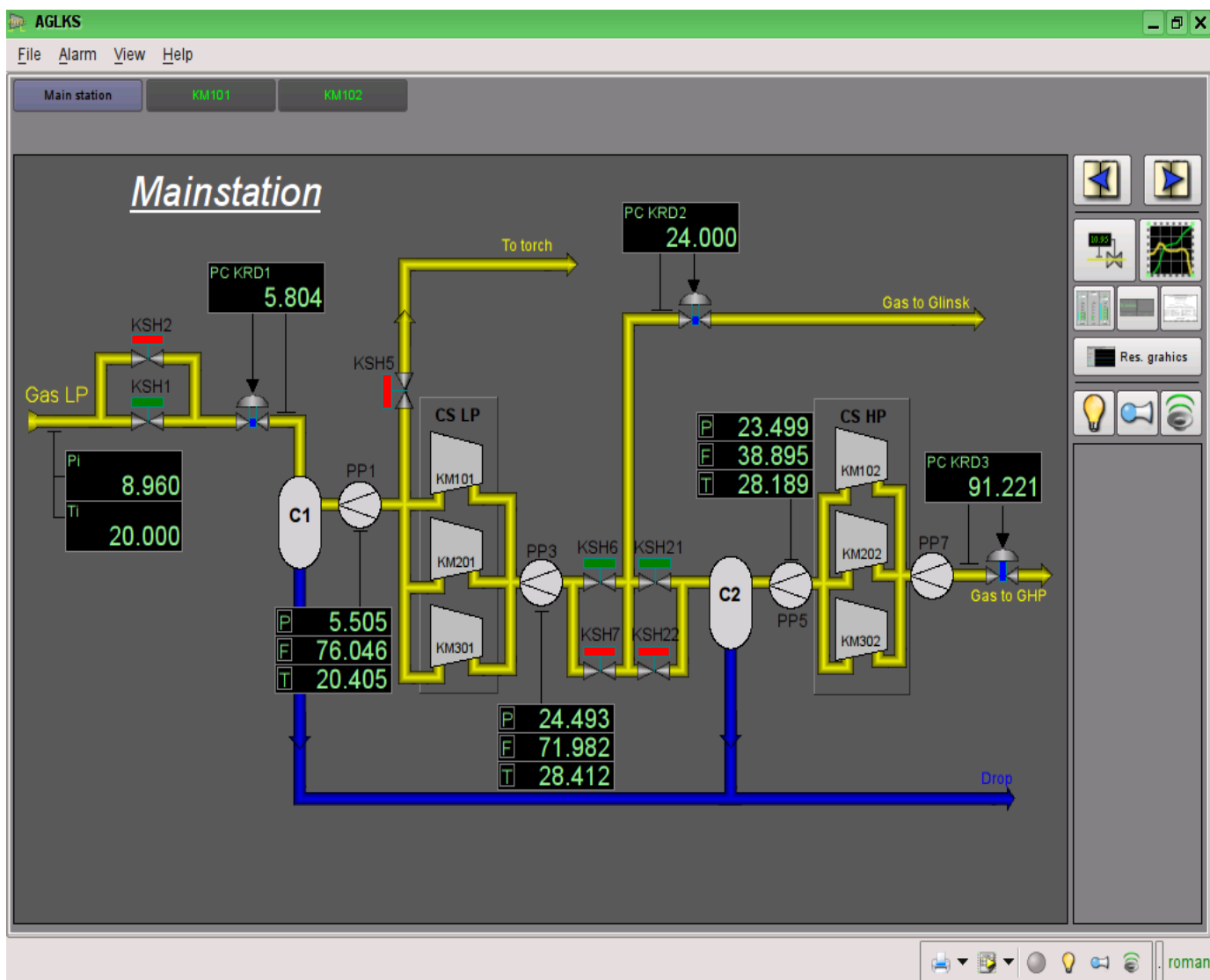


Fig.8. The main window if the execution mode.

4. Conception of basic elements (primitives)

In this version of that module not all the primitives' images of this project are implemented. In general the project provides the following primitives:

Id	Name	Purpose
ElFigure	Elementary graphic figures	<p>Primitive is the basis for drawing basic graphic shapes with their possible combinations in the single object. The support of the following elementary figures is provided:</p> <ul style="list-style-type: none"> • Line. • Arc. • Bézier curve. • Fill of the enclosed space. <p>For all figures contained in the widget common properties of thickness, color, etc. are set, but this does not exclude the possibility of indicating of aforementioned attributes specific to each figure separately.</p>
FormEl	Form elements.	<p>Includes support of standard form components:</p> <ul style="list-style-type: none"> • Line edit. • Text edit. • Check box. • Button. • Combo box. • List. • Slider. • Scroll bar.
Text	Text	Text element(labels). It is characterized by the type of font, color, orientation and alignment.
Media	Media	Element of representation of raster and vector images of various formats, playback of the animated images, playback of audio segments and view of video segments. Perhaps it will be useful to include the OpenGL support for it!
Diagram	Diagram	Element of the diagram with the support of the possibility of displaying multiple streams of trends and different modes of display, from minimalist to full, two-, three-dimensional, circular, etc.
Protocol	Protocol	Element of the protocol, visualizer of the system messages, with support for multiple operating modes with the different sizes and settings
Document	Document	The element of generating the reports, journals and other documentation on the basis of specified data.
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.
Box	Box	Contains the mechanism for other widgets placement with the purpose of creation of new, more complex widgets and pages of final visualization.

Lets examine the implementation of each primitive.

4.1. Elementary figure primitive (ElFigure)

Support of the following elementary figures is provided: lines, elliptical arcs, Bézier curves and fill of the enclosed space with the color and/or image. For the elementary figures the following operations are provided:

- creation/deleting of the figures;
- copying of the figure(s);
- moving and resizing of the figures by mouse and keyboard;
- possibility to connect the elementary figures to each other, getting more complex figures, for which all the properties of the source elementary figures are available;
- possibility of simultaneous movement of several figures;
- fill of the enclosed space with the color and/or image;
- generation of mouse key events at the time of the mouse-click on the filled spaces;
- scaling;
- rotation.

Fig. 9 shows a part of the screen with a frame containing the elementary figures.

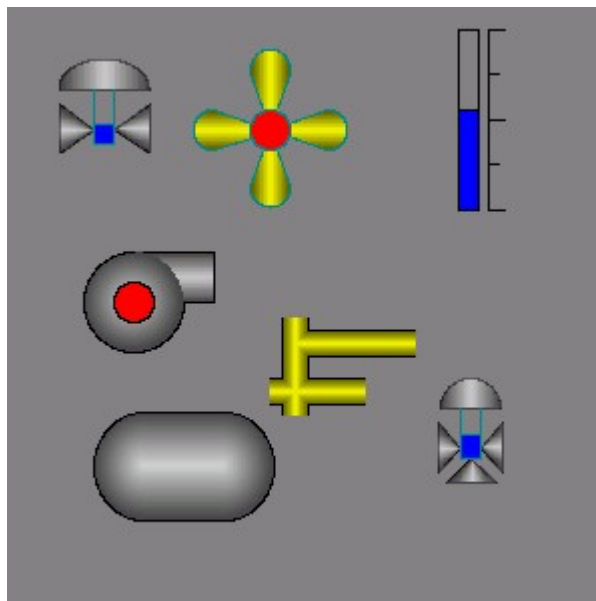


Fig.9 Realization of elementary figures in the Vision.

The figures underlying this widget, containing the points (the start and end ones) that can be connected with the according points of other figures; and the points with the help of which the geometry of the figure can be changed.

It is possible to add the figure using the mouse:

1. Select the desired figure from the context menu.
2. Set with the left mouse-button start and end points (for line with the SHIFT key hold its orthogonal drawing is made).

The deleting of the figure(s) it is possible by pressing "Del", having selected figure(s).

The copying of the figure(s) it is possible by pressing keys "Ctrl"+"C", having selected figure(s).

Moving/resizing of the figure it is possible by using the mouse or keyboard:

1. Select the figure, by clicking on it with the left mouse button.
2. Drag (with the help of mouse or control keys) the figure or one of its control points in the desired location and release the mouse button (key).

It is possible to move several figures, selected by means of holding "Ctrl" and clicking on the desired figures (this option works when the button Connections (Connections) is disabled) or by mouse selection.

The connection of the figures with each other it is possible by the following way:

1. Press the Connections button.
2. Select one of the figures and move its start or end point to the desired start or end point of the other figure so that it will get to the appeared circle, release the left mouse button. Connected figures are moving as well as the individual, the general point is moved for all connected figures, to which it refers(priority is given to the arc, two arcs can't be connected directly with each other).

To fill the enclosed space from the figures it is possible with the following way:

1. Press the Connections button.
2. Create the enclosed space.
3. Make the double-click of the left mouse button inside of it.

To delete the fill of the enclosed space it is possible from the context menu of the widget; by braking the enclosed space or by double-click of the left mouse button on the already existing filled space.

Rotation of the figure is made around the center of the widget.

4.2. Text primitive (Text)

Support of the text element with the following properties is provided:

- Font with the properties: type/class of the font, size, bold, italic, strikeout and underline.
- Text color.
- Text orientation.
- Automatic word wrap.
- Alignment of the text horizontally and vertically with all options..
- Displaying the background as the color and/or image.
- Display the border around the text, with the specified color, width and style.
- Formation of the text from the attributes of different types and properties.

Fig. 10 represents a part of the screen with the frame containing the text examples using various parameters.

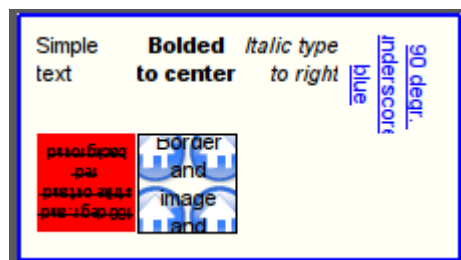


Fig.10. Realization of the basic text element in the Vision.

4.3. Primitive of the form element (FormEl)

Support of the form elements on the VCA frames is provided. The following form elements are included:

Line edit - It is represented by the following types: "Text", "Combo", "Integer", "Real", "Time", "Date", "Date and time". All kinds of line editor support the confirmation of entry.

Text edit - It is the flat-text editor with the confirmation or denial of entry.

Check box - Provides a field of binary flag.

Button - Provides the button with the support of: the color of the button, the image of the button, and mode of fixation.

Combo box - Provides the selection field of the element from the list of the items.

List - Provides the list box with the control of the current element.

Slider - Slider element.

Scroll bar - Strip of the scroll bar.

The following modes are realized: «Enable» and «Active», as well as transfer of changes and events to the data model of the VCA (engine).

Fig. 11 represents a part of the screen with the frame containing the above-listed elements of the form.

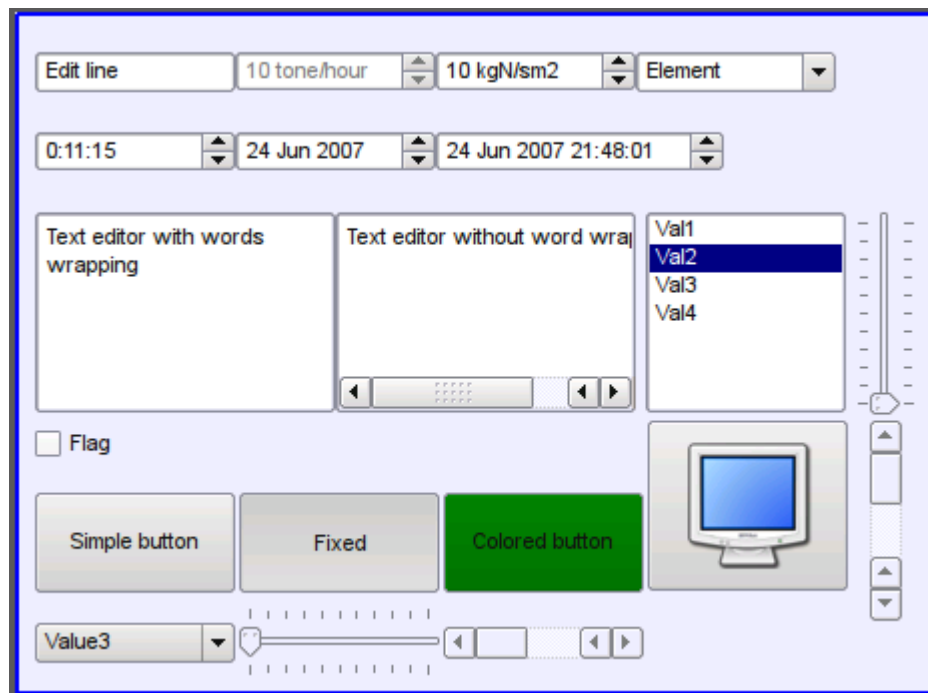


Fig.11. Realization of the form elements in the Vision.

4.4. Primitive of the displaying the media materials (Media)

Support of the element of the displaying of media materials with the following properties is provided:

- The indication of the source of media data (images or video material).
- View of the images of most well-known formats with the possibility of inscribing of it in the size of the widget.
- Playback of the simple animated images and video formats with the possibility to control the playback speed.
- Displaying of the the background as a color and/or image.
- Display the border around the text, with the specified color, width and style.
- Formation of the active areas and generating the events when they are activated.

Fig. 12 represents a part of the screen with the frame containing examples of viewing/playback of media data.

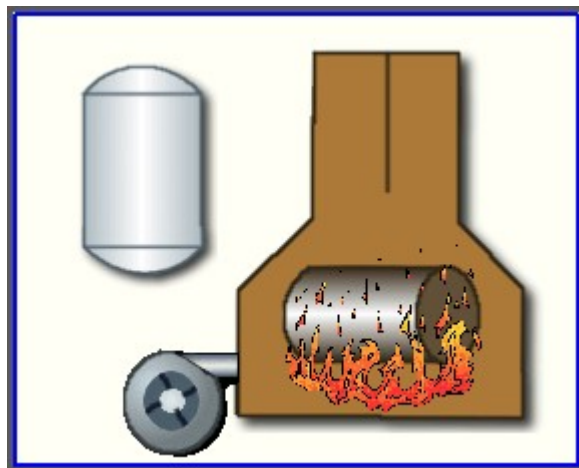


Fig.12. Realization of the basic element of the displaying of media materials in the Vision.

4.5. Primitive of the construction of diagrams/graphs (Diagram)

Support of the element of the construction of diagrams/graphs with the following properties is provided:

- Construction of graphs/trends:
 - Construction graph for: archive data, current data and the formation of an intermediate buffer for the display of the parameters without archive.
 - Construction of a single graphs with the value of the parameter on the ordinate axis, and the combined graphs of up to 10 parameters, with the percentage scale.
 - Ability to adapt the parameter's graph to the value, the regrowth of scale.
 - Wide range of scalability and adaptation of the horizontal scale, with automatic averaging at the server level and the primitive itself.
 - Ability to display the size grid and markers on the horizontal and vertical, with adaptation to the displaying range.
 - Support of the active mode, with the cursor and getting values under the cursor.

Fig. 13 represents a part of the screen with the frame containing examples of the trend-diagrams.

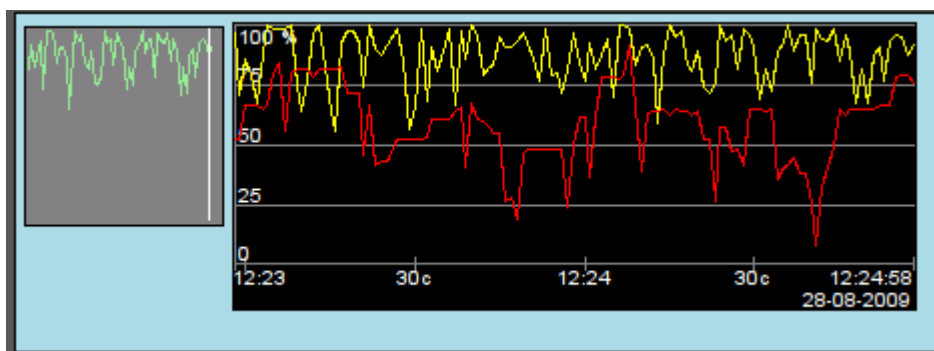


Fig.13. Realization of the basic element of a diagram-trend displaying in the Vision.

4.6. Primitive of the protocol formation (Protocol)

Support of the element of the formation of the protocol with the following properties is provided:

- Formation of the protocol from the archive of messages for the specified time and depth.
- Request of the data from the messages archivers.
- Selection of data from the archives by the level of importance and the category of messages template.
- Support the tracking mode for the appearance of messages in the archive of messages.

Fig. 14 represents a part of the screen with the frame containing an example of the protocol.

	Time	mcsec	Level	Category	Me
1	28.08.2009 13:14:23	8275	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM102/	Start controller!
2	28.08.2009 13:14:23	3510	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM101/	Start controller!
3	28.08.2009 13:14:23	25948	1	/DemoStation/sub_DAQ/mod_LogicLev/cntr_experiment/	Start controller!
4	28.08.2009 13:14:23	23775	1	/DemoStation/sub_DAQ/mod_System/cntr_AutoDA/	Start controller!
5	28.08.2009 13:14:23	21291	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM302/	Start controller!
6	28.08.2009 13:14:23	18554	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM301/	Start controller!
7	28.08.2009 13:14:23	1676	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_Anast1to2node_cntr/	Start controller!

Fig.14. Realization of the basic element of a protocol displaying in the Vision.

4.7. Primitive of the report formation (*Document*)

Support element of the report formation with the following properties is provided:

- Adaptive formation of a document structure based on Hypertext Markup Language. This provides support for the broad features of formatting of the documents.
- Formation of the documents on command or on schedule. It is necessary for creation of reports into the archive and then view the archive.
- Formation of a document in real time mode. It is necessary to form documents completely dynamically, and based on the archives for the specified time.
- Using of the the attributes of the widget for transmission of values and addresses to the archives in the report. It allows you to use the widget of the document as a template when 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. 15 shows the frame containing a sample of the document.

"Dneprovskij metkombinat" LTD
(enterprise name, where flowmeter located)

DAY REPORT

over "_____ " 2008
made _____
(date) (time)

"FLOWTEC-TM" _____ Pipeline _____
(calculator or corrector name) (thread name)

The flowmeter characterisation included standard narrowing:

Contract hour	12 25	Sensor's type	Угловой	Atm. pressure, kPa	95
Molar part of N ₂ , %	70	Ct. of harshness	0.2	Cuting, kPa	80
Molar part of CO ₂ , %	10	Ct of blunting	0.1	Upper border, kPa	150
Pipe diameter, mm	100	Coefficient L (alpha)	0.3	Trigger threshold, kPa	85
Diameter of narrowing, mm	60	Relative square of narrowing	0.6	Dynamic ductility, kgF/m ²	32

Hours data

Date	Time		Capacity, 1000x m3	Aver. pressure diff., pPa	Average pressure, MPa	Average temperatura, °C	Average density , kg.m ³
	begin	end					
28 08 2009	09:00	10:00	Empty	Empty	Empty	Empty	Empty
28 08 2009	10:00	11:00	Empty	Empty	Empty	Empty	Empty
28 08 2009	11:00	12:00	2341.62	6.02	9.25	15.44	1.01
28 08 2009	12:00	12:25	2331.34	6.02	9.62	15.48	0.97
All over day			4672.96	12.04	18.86	30.92	1.98

Fig.15 Implementation of the basic visualization element of the report documentation in the Vision.

4.8. Primitive of the box container (Box)

Support of the primitive of the container concurrently serves as the project pages is provided. This primitive is the only element-container, which may include links to frames from the library, thereby creating the user elements of desired configuration. Primitive implements the provided by the project properties. The properties of this primitive are:

Container -- Allows you to form the desired objects by grouping in the limits of the primitive.

Page -- Elements constructed on the basis of the primitive may serve as a page of user interface.

Container of pages -- Property of substitution of its own contents by another page in the execution process. Used to create frames on the pages of user interface. For example, the main page of traditional SCADA system with alarm objects is constructed in this way.

Background -- Supports ability to specify the background as color or image.

Border -- Supports the displaying of the border, with the specified color, width and style.

Example of editing of the frame, based on the primitive, is shown in Fig. 1, and Fig. 8 shows a page containing the container of the pages, built on the basis of the primitive.

5. The overall configuration of the module

To adjust your own behavior in not obvious situations the module provides the ability to customize individual settings through the management interface of the OpenSCADA (Fig. 16). These settings are:

- Initial user of the configurator - points on behalf of what user to open configurator without requiring the password.
- The list of projects for their automatic execution with the launch of the module. To provide the possibility to indicate the opening of the window of the project execution on the desired display of many display systems the recording format of the project "PrjName-1" is provided, where 1 - the number of the target display.
- The name of the remote OpenSCADA station with visualization engine VCA.
- The link to the configuration page of the external OpenSCADA stations.

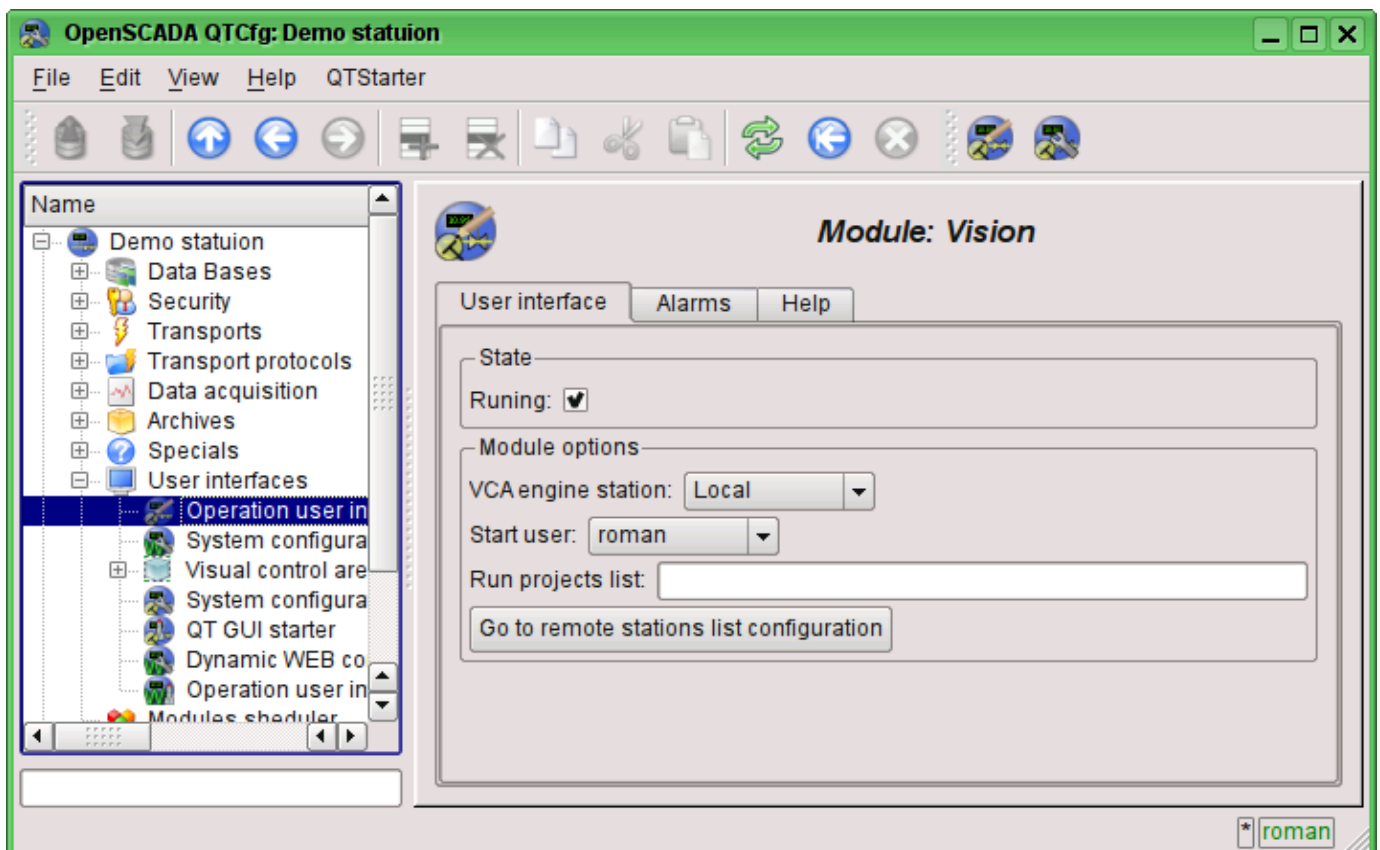


Fig.16. The configuration page of the module.

The module <WebVision> of subsystems “User Interfaces”

<i>Module:</i>	WebVision
<i>Name:</i>	Operation user interface (WEB)
<i>Type:</i>	User interfaces
<i>Source:</i>	ui_WebVision.so
<i>Version:</i>	0.9.0
<i>Author:</i>	Roman Savochenko
<i>Developers:</i>	Roman Savochenko, Maxim Lysenko, Ksenia Yashina
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Web visual user interface for the project execution of visual control area (VCA).
<i>License:</i>	GPL

WebVision module provides a mechanism of the final visualization of the visual control area (VCA) in the OpenSCADA system. The module is based on WEB technologies (XHTML, JavaScript, CSS, AJAX). In its work, the module uses the data from the VCA engine (module [VCAEngine](#)).

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 be 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 be 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 [проект создания концепции CBY](#) is established. The result of this project - the direct visualization module based on the WEB technologies, the direct visualization module [Vision](#) and VCA engine [VCAEngine](#).

1. Purpose

This module of the direct visualization of the VCA serves only for the execution of interfaces of the VCA in the area of WEB-technologies.

The user interface is formed in the WEB-browser, by reference to the WEB-server and receiving from it XHTML-document over HTTP. In this case, the WEB-server - OpenSCADA system, which supports standard communication mechanisms of the TCP-networks (module Transport.Sockets), hypertext transfer protocol (module Protocol.HTTP), as well as encryption of traffic between the browser and the server (Transport.SSL). On this basis, to gain access to the user interface provided by this module, you need to configure the transport in the OpenSCADA (Transport.Sockets or Transport.SSL) in conjunction with the protocol HTTP (Protocol.HTTP). In the delivery of the OpenSCADA system there are configuration files containing settings of the Transport.Sockets for ports 10002 and 10004. Consequently, the interface of the module in the default configuration of the OpenSCADA will be available at URL: <http://localhost:10002> or <http://localhost:10004>.

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. Execution of the VCA interfaces

Execution of the VCA interface is to run a new project session or connect to the existing one on the level of VCA engine (Fig.2). Before the connection request to the session the authentication of the user request is done . Then the module of direct visualization represents and manages the data of the session. The main window of the execution mode of this module has the form presented at Fig.3.

The interface of the execution window is fully dynamically built by the JavaScript script on the basis of the contents of the session of the project through direct XML requests to the server.

Update of the contents of the open pages of the visualization interface with the frequency of 1 second. In the updating process it is performed:

- request a list of opened pages, with a sign of page modification, at the model and consistency checking of the actually opened pages to that list;
- request of the branch of the modified pages;
- update of the contents of the modified pages and their widgets, in accordance with the received modified data.

The mechanism of the request of the only modified data is based on an absolute counter of the session execution. If you want to make real changes in the attributes of widgets the memorizing of the value of this counter is done, which allows the identification of modified attributes. This approach can increase productivity and reduce the load on network sharing in the case of access to the VCA engine via network.

Hierarchically the module provides an opportunity to accommodate the project pages in the main execution window of the WEB-browser (Fig.3), as well as putting them inside of the container widgets.



Fig.1.Authentication page.

OpenSCADA. Operation user interface (WEB)

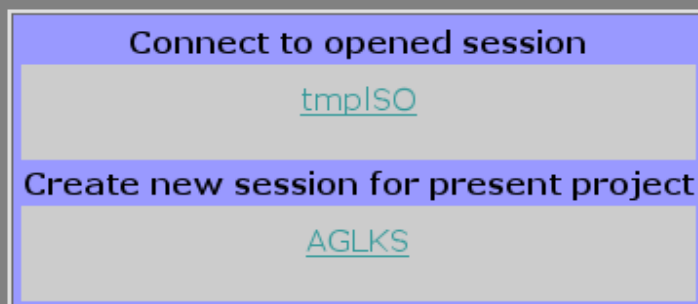


Fig.2. Connection or the creation of a new session of the project's execution of the VCA.

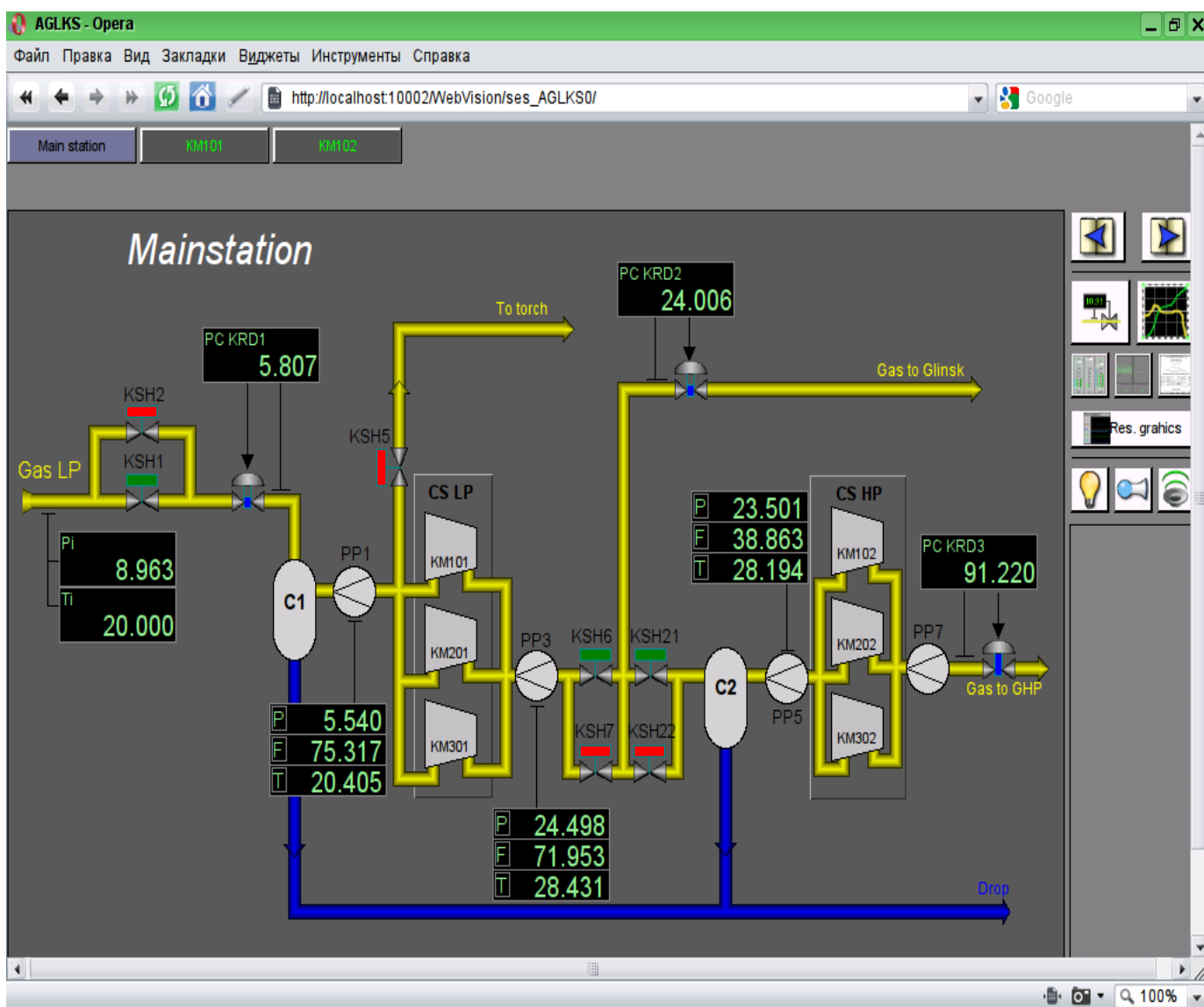


Fig.3. The main execution window.

3. Conception of basic elements (primitives)

In this version of that module not all the primitives' images of this project are implemented. In general the project provides the following primitives:

Id	Name	Purpose
ElFigure	Elementary graphic figures	<p>Primitive is the basis for drawing basic graphic shapes with their possible combinations in the single object. The support of the following elementary figures is provided:</p> <ul style="list-style-type: none"> • Line. • Arc. • Bézier curve. • Fill of the enclosed space. <p>For all figures contained in the widget common properties of thickness, color, etc. are set, but this does not exclude the possibility of indicating of aforementioned attributes specific to each figure separately.</p>
FormEl	Form elements.	<p>Includes support of standard form components:</p> <ul style="list-style-type: none"> • Line edit. • Text edit. • Check box. • Button. • Combo box. • List. • Slider. • Scroll bar.
Text	Text	Text element(labels). It is characterized by the type of font, color, orientation and alignment.
Media	Media	Element of representation of raster and vector images of various formats, playback of the animated images, playback of audio segments and view of video segments. Perhaps it will be useful to include the OpenGL support for it!
Diagram	Diagram	Element of the diagram with the support of the possibility of displaying multiple streams of trends and different modes of display, from minimalist to full, two-, three-dimensional, circular, etc.
Protocol	Protocol	Element of the protocol, visualizer of the system messages, with support for multiple operating modes with the different sizes and settings
Document	Document	The element of generating the reports, journals and other documentation on the basis of specified data.
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.
Box	Box	Contains the mechanism for other widgets placement with the purpose of creation of new, more complex widgets and pages of final visualization.

Lets examine the implementation of each primitive.

3.1. Elementary figure primitive (ElFigure)

Support of the elementary figures is provided: lines, elliptical arcs, Bézier curves and fill of the enclosed space with the color and/or image. For the elementary figures the following operations are provided:

- creation/deleting of the figures;
- copying of the figure;
- moving and resizing of the figures by mouse and keyboard;
- possibility to connect the elementary figures to each other, getting more complex figures, for which all the properties of the source elementary figures are available;
- possibility of simultaneous movement of several figures;
- fill of the enclosed space with the color and/or image;
- generation of mouse key events at the time of the mouse-click on the filled spaces;
- scaling;
- rotation.

Fig. 4 shows a part of the screen with a frame containing the elementary figures.

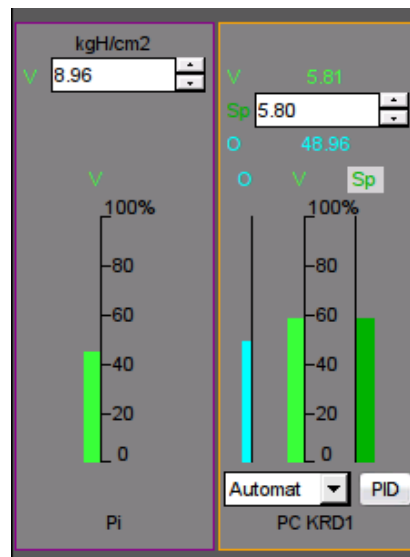


Fig.4 Realization of elementary figures in the WebVision.

3.2. Text primitive (Text)

Support of the text element with the following properties is provided:

- Font with the properties: type/class of the font, size, bold, italic, strikeout and underline.
- Text color.
- Text orientation.
- Automatic word wrap.
- Alignment of the text horizontally and vertically with all options..
- Displaying the background as the color and/or image.
- Display the border around the text, with the specified color, width and style.
- Formation of the text from the attributes of different types and properties.

Fig. 5 represents a part of the screen with the frame containing the text examples using various parameters.

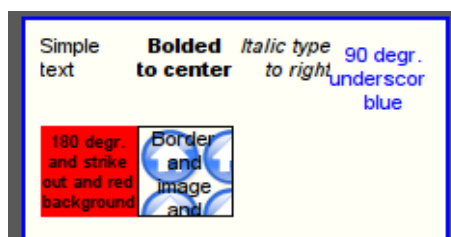


Fig.5. Realization of the basic text element in the WebVision.

3.3. Primitive of the form element (FormEl)

Support of the form elements on the VCA frames is provided. The following form elements are included:

Line edit - It is represented by the following types: "Text", "Combo", "Integer", "Real", "Time", "Date", "Date and time". All kinds of line editor support the confirmation of entry.

Text edit - It is the flat-text editor with the confirmation or denial of entry.

Check box - Provides a field of binary flag.

Button - Provides the button with the support of: the color of the button, the image of the button, and mode of fixation.

Combo box - Provides the selection field of the element from the list of the items.

List - Provides the list box with the control of the current element.

Slider - Slider element(Not done).

Scroll bar - Strip of the scroll bar(Not done).

The following modes are realized: «Enabled» and «Active», as well as transfer of changes and events to the data model of the VCA (engine). For all realized representations the active mode is supported, ie elements can be used to create the forms of user input.

Fig. 6 represents a part of the screen with the frame containing the above-listed elements of the form.

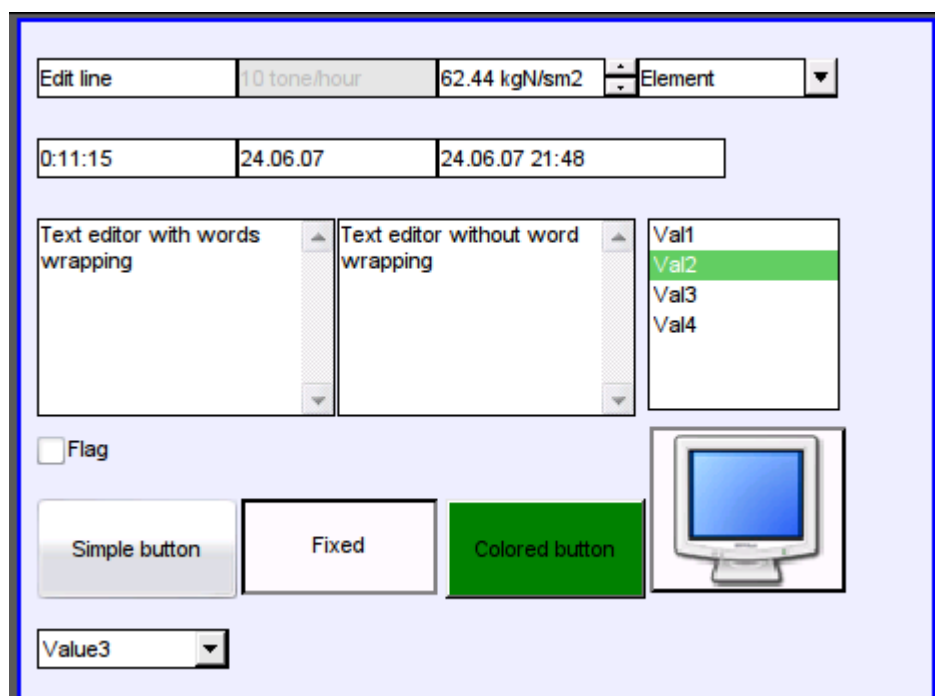


Fig.6. Realization of the form elements in the WebVision.

3.4. Primitive of the displaying the media materials (Media)

Support of the element of the displaying of media materials with the following properties is provided:

- The indication of the source of media data (images or video material).
- View of the images of most well-known formats with the possibility of inscribing of it in the size of the widget.
- Playback of the simple animated images and video formats with the possibility to control the playback speed.
- Displaying of the the background as a color and/or image.
- Display the border around the text, with the specified color, width and style.
- Formation of the active areas and generating the events when they are activated.

Fig. 7 represents a part of the screen with the frame containing examples of viewing/playback of media data.

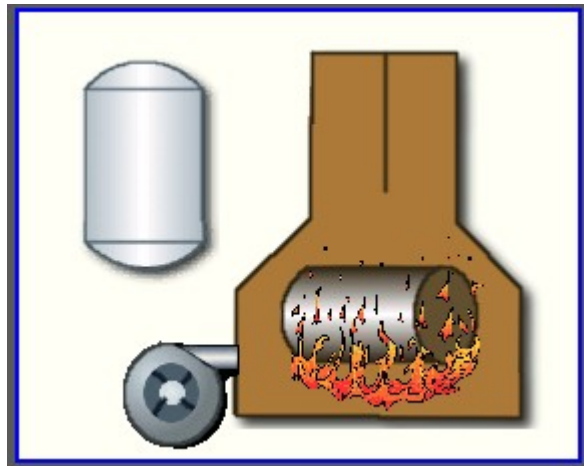


Fig.7. Realization of the basic element of the displaying of media materials in the WebVision.

3.5. Primitive of the construction of diagrams/graphs (Diagram)

Support of the element of the construction of diagrams/graphs with the following properties is provided:

- Construction of graphs/trends:
 - Construction graph for: archive data, current data and the formation of an intermediate buffer for the display of the parameters without archive.
 - Construction of a single graphs with the value of the parameter on the ordinate axis, and the combined graphs of up to 10 parameters, with the percentage scale.
 - Ability to adapt the parameter's graph to the value, the regrowth of scale.
 - Wide range of scalability and adaptation of the horizontal scale, with automatic averaging at the server level and the primitive itself.
 - Ability to display the size grid and markers on the horizontal and vertical, with adaptation to the displaying range.
 - Ability to set the cursor in the trend by mousea.

Fig. 8 represents a part of the screen with the frame containing examples of the trend-diagrams.

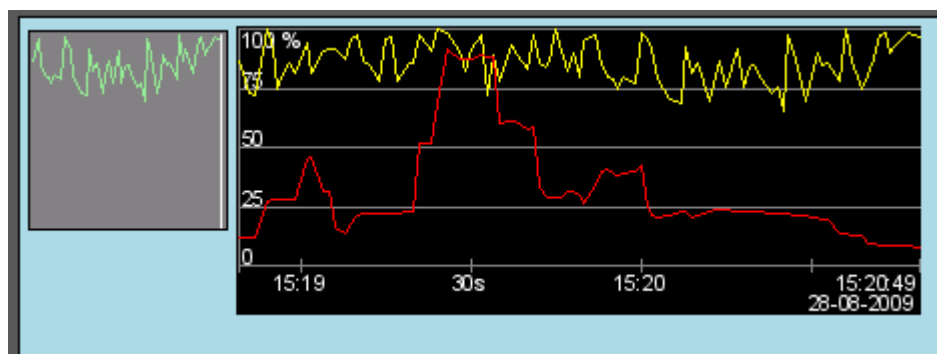


Fig.8. Realization of the basic element of a diagram-trend displaying in the WebVision.

3.6. Primitive of the protocol formation (*Protocol*)

Support of the element of the formation of the protocol with the following properties is provided:

- Formation of the protocol from the archive of messages for the specified time and depth.
- Request of the data from the messages archivers.
- Selection of data from the archives by the level of importance and the category of messages template.
- Support the tracking mode for the appearance of messages in the archive of messages.

Fig. 9 represents a part of the screen with the frame containing an example of the protocol.

#	Time	mcsec	Level	Category	
0	28.8.2009 15:25:46	71361	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM302/	Start controller!
1	28.8.2009 15:25:46	44349	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM301/	Start controller!
2	28.8.2009 15:25:46	354704	1	/DemoStation/sub_UI/mod_VCAEngine/	Start module.
3	28.8.2009 15:25:46	332330	4	/DemoStation/sub_Archive/	Value archivor <DBArch.1s> start error.
4	28.8.2009 15:25:46	322082	4	/DemoStation/sub_BD/mod_MySQL/db_arch/	Connect to DB error: Access denied for u
5	28.8.2009 15:25:46	318370	1	/DemoStation/sub_Archive/	Start subsystem.
6	28.8.2009 15:25:46	27115	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM202/	Start controller!
7	28.8.2009 15:25:46	18885	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM201/	Start controller!
8	28.8.2009 15:25:46	112010	1	/DemoStation/sub_DAQ/mod_LogicLev/cntr_experiment/	Start controller!
9	28.8.2009 15:25:46	103157	1	/DemoStation/sub_DAQ/mod_System/cntr_AutoDA/	Start controller!
10	28.8.2009 15:25:45	992582	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM102/	Start controller!
11	28.8.2009 15:25:45	953837	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_KM101/	Start controller!
12	28.8.2009 15:25:45	946934	1	/DemoStation/sub_DAQ/mod_BlockCalc/cntr_Anast1to2node_cntr/	Start controller!

Fig.9. Realization of the basic element of a protocol displaying in the WebVision.

3.7. Primitive of the report formation(*Document*)

Support element of the report formation with the following properties is provided:

- Adaptive formation of a document structure based on Hypertext Markup Language. This provides support for the broad features of formatting of the documents.
- Formation of the documents on command or on schedule. It is necessary for creation of reports into the archive and then view the archive.
- Formation of a document in real time mode. It is necessary to form documents completely dynamically, and based on the archives for the specified time.
- Using of the the attributes of the widget for transmission of values and addresses to the archives in the report. It allows you to use the widget of the document as a template when 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. 10 shows the frame containing a sample of the document.

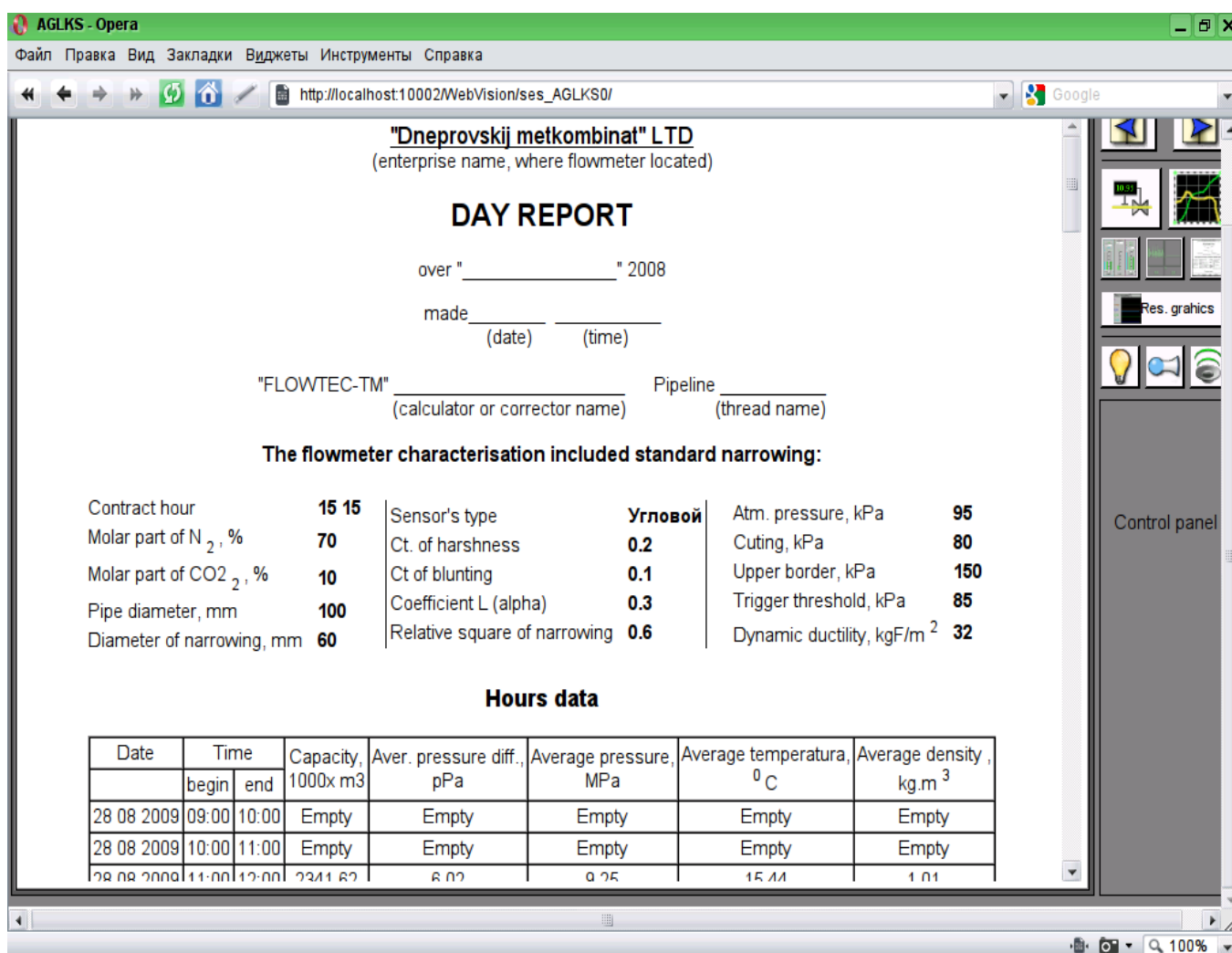


Fig.10 Implementation of the basic visualization element of the report documentation in the WebVision.

3.8. Primitive of the box container (Box)

Support of the primitive of the container concurrently serves as the project pages is provided. This primitive is the only element-container, which may include links to frames from the library, thereby creating the user elements of desired configuration. Primitive implements the provided by the project properties. The properties of this primitive are:

Container - Allows you to form the desired objects by grouping in the limits of the primitive.

Page - Elements constructed on the basis of the primitive may serve as a page of user interface.

Container of pages - Property of substitution of its own contents by another page in the execution process. Used to create frames on the pages of user interface. For example, the main page of traditional SCADA system with alarm objects is constructed in this way.

Background - Supports ability to specify the background as color or image.

Border - Supports the displaying of the border, with the specified color, width and style.

4. The overall configuration of the module

To adjust your own behavior in not obvious situations the module provides the ability to customize individual settings through the management interface of the OpenSCADA (Fig. 11). These settings are:

- The lifetime of the authentication session.

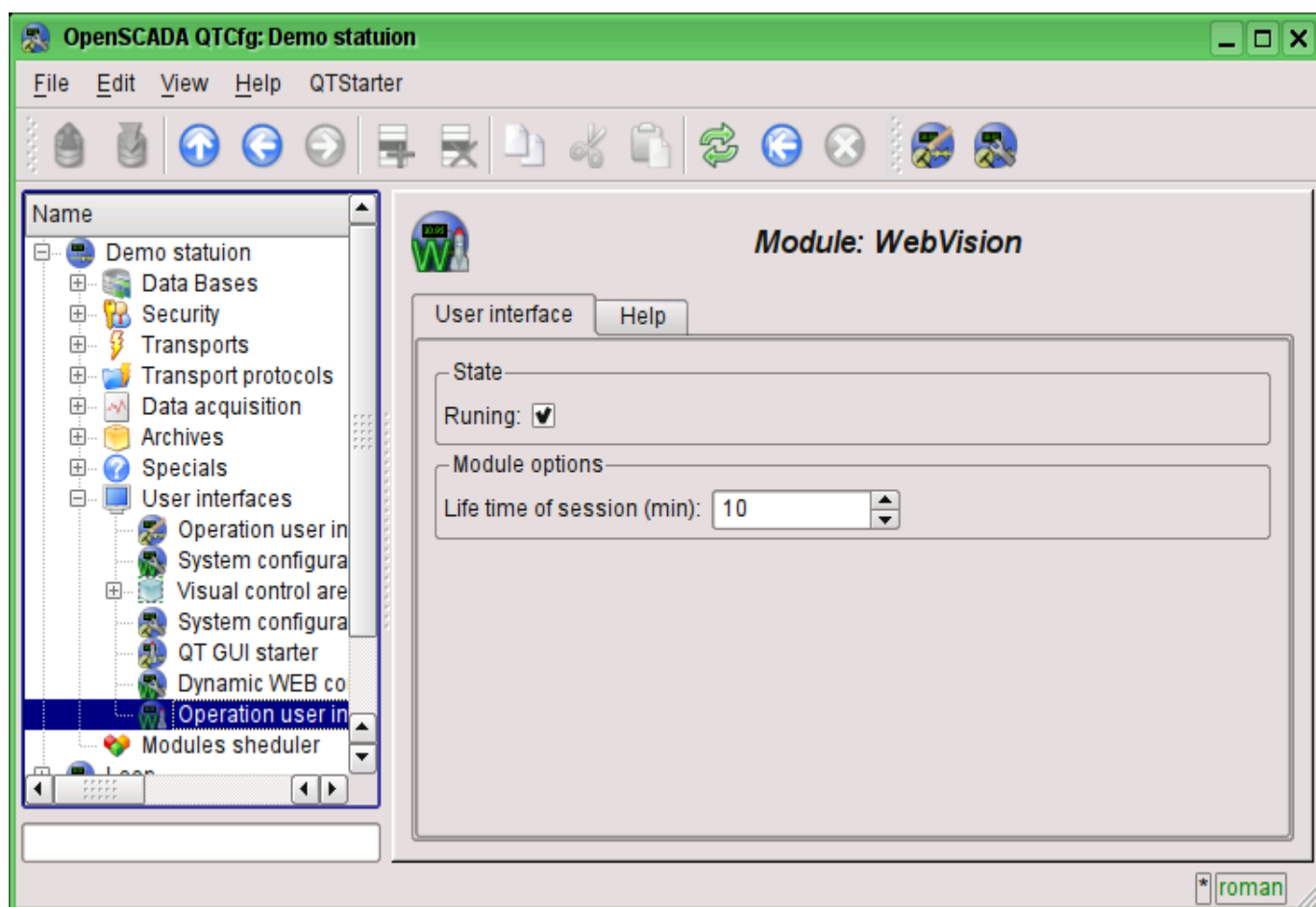


Fig.11. The configuration page of the module.

Conclusion

At this stage, the module may be used to build a real user interfaces that support core functions. However, some problems may arise due to the differences between browsers. For now it is guaranteed stable work on browsers: FireFox, Konqueror, Opera and Google Chromium.

The module <WebUser> of subsystems "User Interfaces"

<i>Module:</i>	WebUser
<i>Name:</i>	Web-interface from the user
<i>Type:</i>	User Interfaces
<i>Source:</i>	ui_WebUser.so
<i>Version:</i>	0.6.0
<i>Author:</i>	Roman Savochenko
<i>Translated:</i>	Maxim Lysenko
<i>Description:</i>	Allows you to create your own user web-interfaces in any language of OpenSCADA.
<i>License:</i>	GPL

WebUser module provides the user with a mechanism to create Web-pages, and can process other Web-requests with the help of the internal language of OpenSCADA, usually JavaLikeCalc, without necessity of low-level programming of OpenSCADA.

Except of the module's belonging to the system OpenSCADA it also belongs and is the module of the <HTTP> transport protocol module. Actually, the WebUser call is made from Protocol.HTTP. The call is made through enhanced communication mechanism through the exported to the WebUser module functions: HttpGet() and HttpSet().

Addressing of the pages begins with the second element of the URI. It is connected with the fact that the first element of the URI is used to identify the module of user Web-interface. For example URL: *http://localhost.localdomain:10002/WebUser/UserPage* can be deciphered as a call of the user page "UserPage" of the Web module WebUser on the host localhost.localdomain on port 10002. In the case of absence of the second element of URI and instruction to display an index of user pages in the configuration the index of the page is generated (Figure 1).

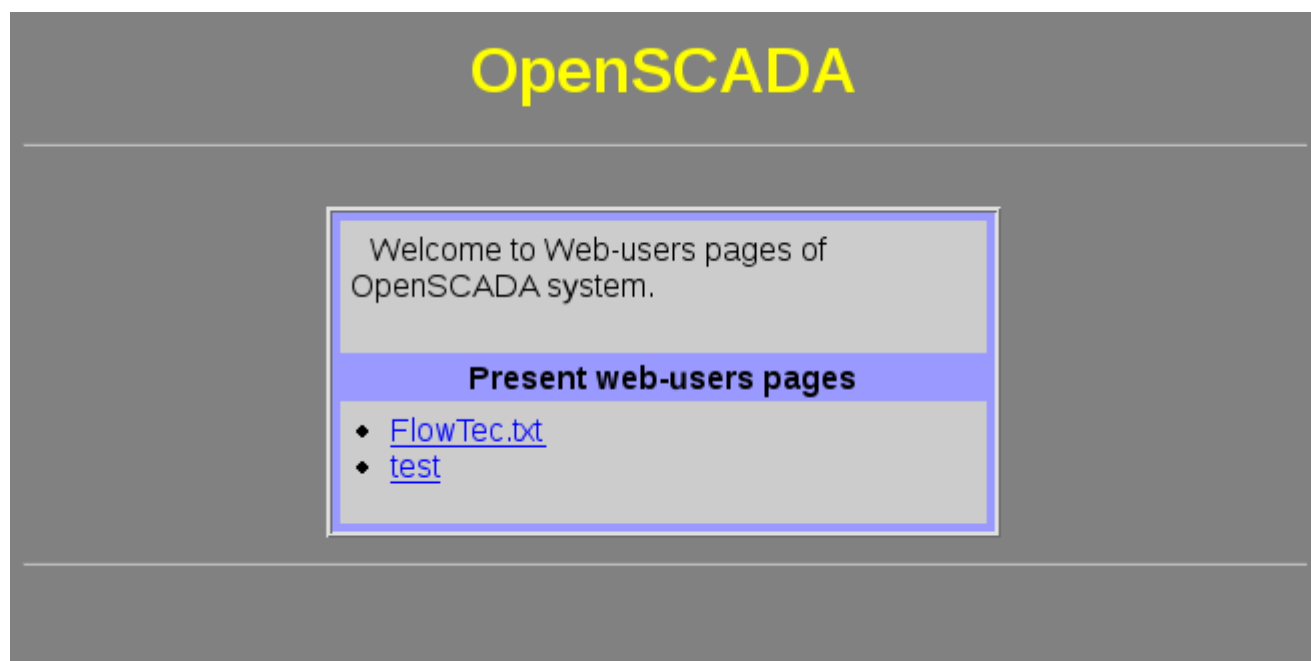


Fig.1. Index of user pages.

The main tab of the module configuration (Fig. 2) contains the state of the module, provides the ability to select the default page and allows you to make the list of user pages.

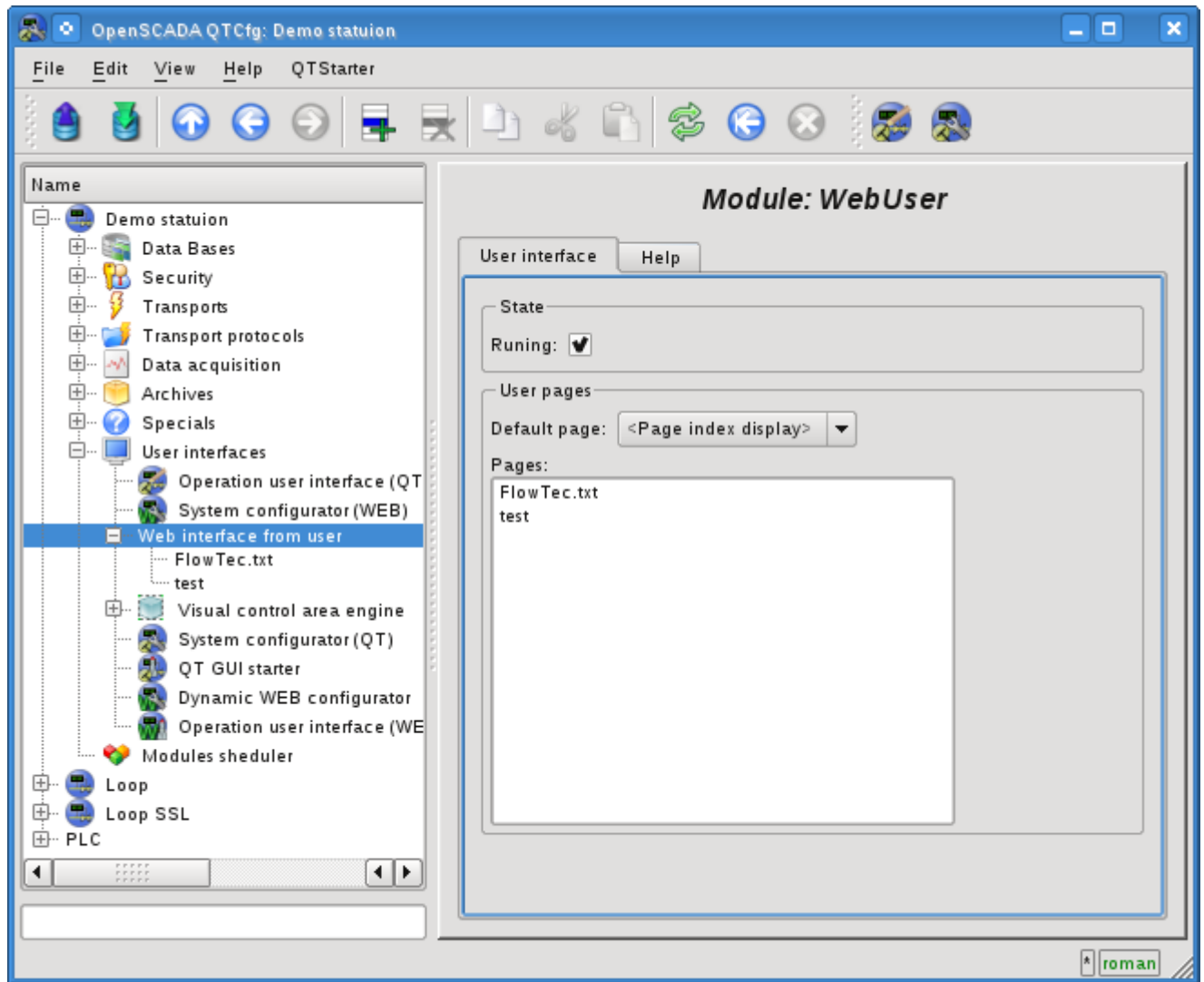


Fig.2. Main tab of the module's configuration.

1. WEB - pages

The module provides the ability to create multiple implementations of Web-pages in the object "User page" (Fig. 3).

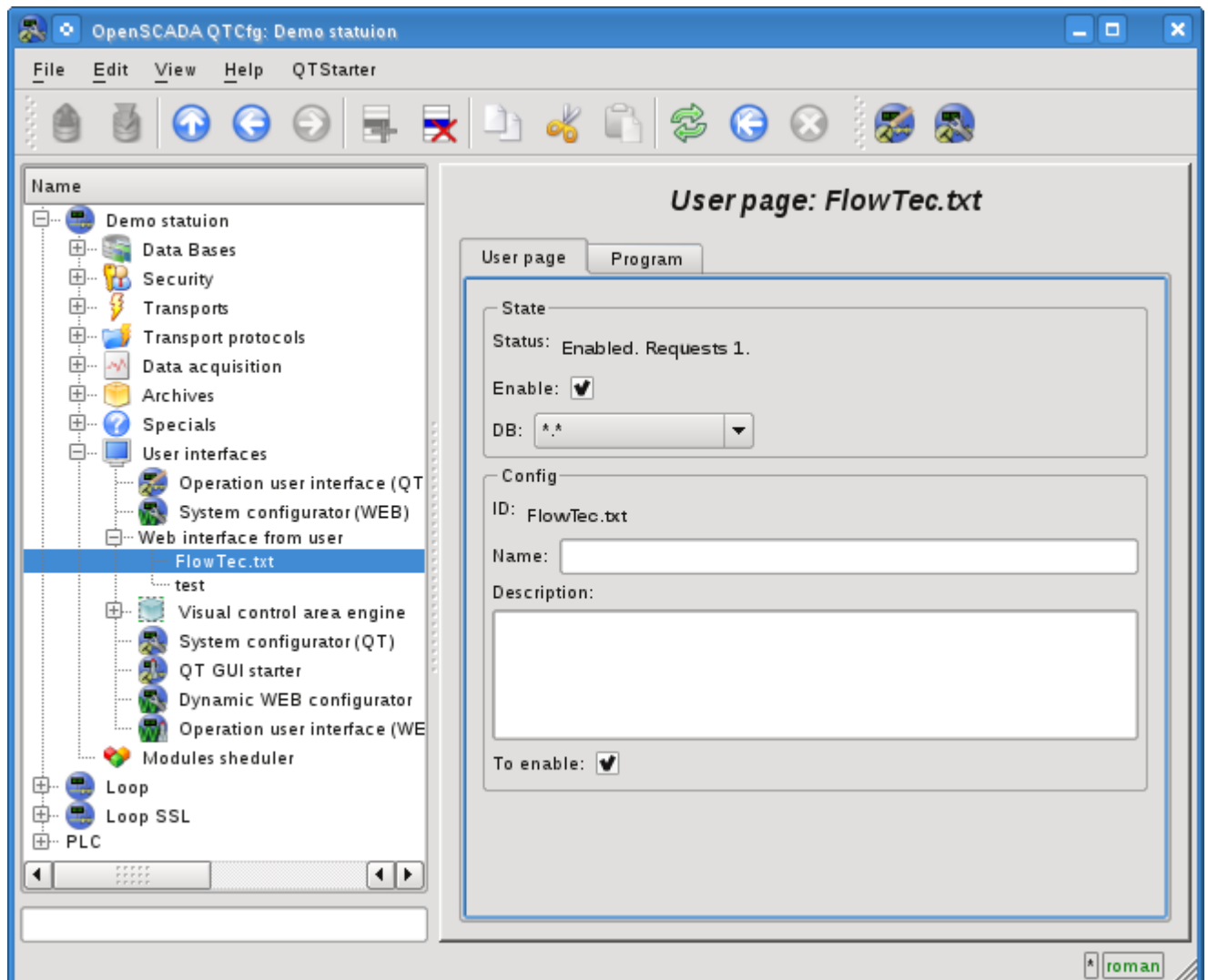


Fig.3. The main configuration page of the user page.

The main tab contains the basic settings of the user protocol:

- Section "Status" - contains properties that characterize the status of the user page:
 - *Enable* - the page status "Enabled".
 - *DB* - DB that stores configuration.
- Section "Config" - directly contains the configuration fields:
 - *ID* - information on the page's identifier.
 - *Name* - specifies the name of the page.
 - *Description* - brief description of the page and its purpose.
 - *To enable* - indicates the status "Enable", in which to transfer the page at startup.

All requests to the user pages are sent to the procedure of the processing of the requests of the user page, which is represented on the "Program" tab of the user page object (Figure 4).

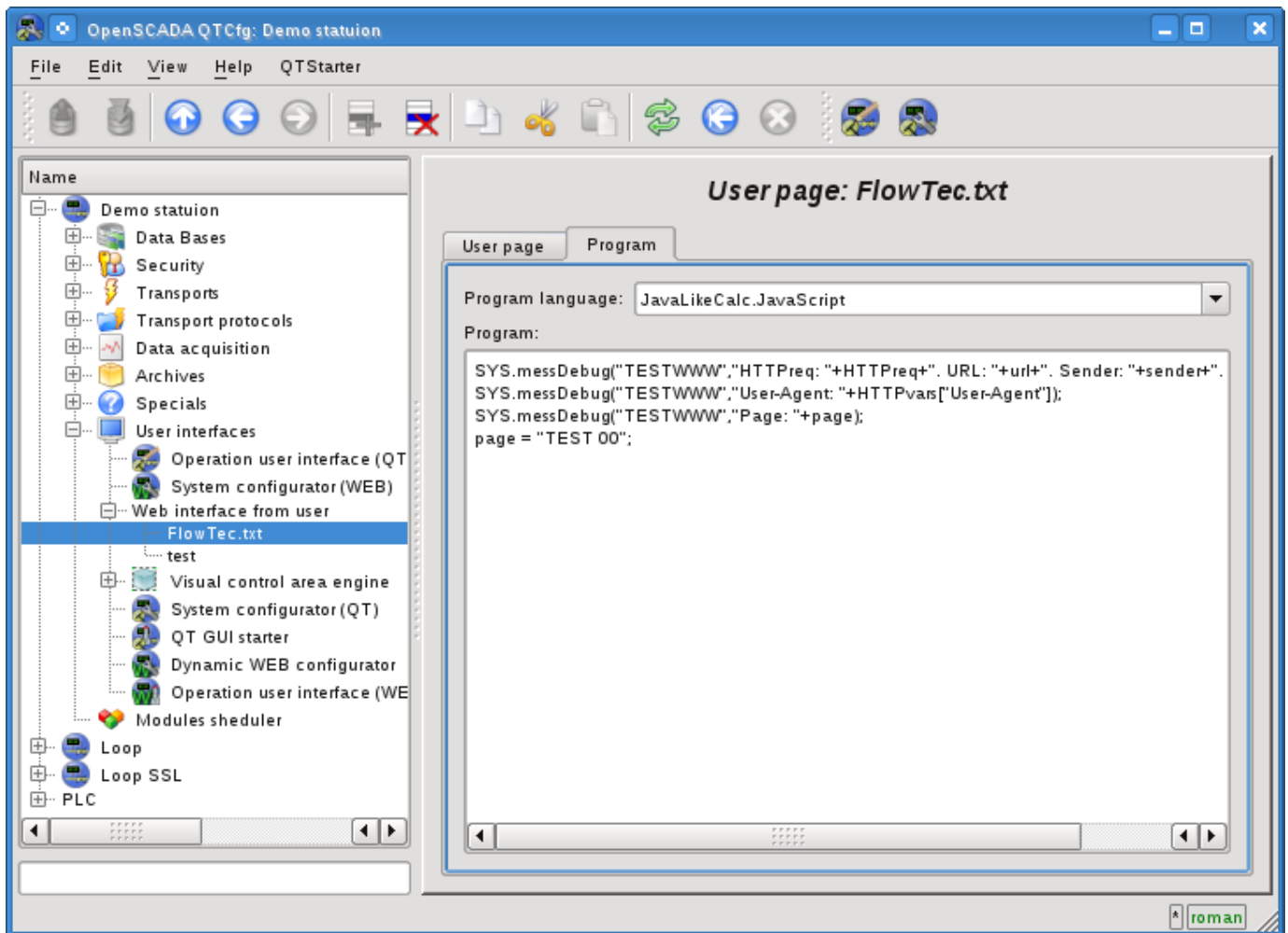


Fig.4. "Program" tab of the user page object.

Tab procedure's tab for processing the requests to the user's page contains the field for selecting the internal programming language of OpenSCADA and the text entry field for the processing procedure typing.

For the processing procedure the following exchange variables are predetermined:

- *rez* - processing result (by defaults - "200 OK");
- *HTTPreq* - the HTTP request method (GET,POST);
- *url* - URI of the request;
- *page* - contents of the Get/Post page for the request and respond as well;
- *sender* - request sender;
- *user* - authenticated user;
- *HTTPvars* - HTTP variables in the Object;
- *URLprms* - URL parameters in the Object;
- *cnts* - content items for POST in the Array<XMLNodeObj>.

The overall scenario of the user's page request:

- External network station generates an HTTP request with the following form of URI `"/WebUser/<UserPage>"` which falls on transport of OpenSCADA with the value of the configuration field "Protocol" equal to the "HTTP".
- Transport sends a request to the module of transport protocol Protocol.HTTP.
- Module of the transport protocol, in accordance with the first element of the URI, sends a request to this module.
- This module selects the object of the user's page which is specified in the second element of the URI.
- Initialization of the variables of HTTP-protocol for the procedure of the page is made:
 - *HTTPreq* - the value of the string "GET" or "POST", depending on the type of request, is set;

- *url* - address of the requested resource (URI);
 - *page* - the content of sending page for method "POST";
 - *sender* - address of the request sender;
 - *user* - address of an authenticated user, if the authentication has occurred;
 - *HTTPvars* - the parsed list of variables of the HTTP protocol in the form of object's properties;
 - *URLprms* - the parsed list of URL parameters in the form of object's properties;
 - *cnts* - parsed contents items for POST in Array <XMLNodeObj>, with the contents of elements in the text and properties in the attributes XMLNodeObj.
- Calling the procedure for execution, which, having processed the request, forms the contents of the page in the "page" and the result of the request in the "rez".
 - At the end the answer is formed with the return code of the HTTP from "rez" and with the contents from the "page".