API of system OpenSCADA

OpenSCADA 0.8.0 *(http://oscada.org)*

Roman Savochenko (*rom_as@oscada.org*)

Maxim Lysenko (*mlisenko@oscada.org*)

June 7, 2012

Contents table

API of system OpenSCADA....1
Introduction....4
1. The internal structure, API system OpenSCADA.....5
2. The overall structure of the system. Modularity (TSubSYS, TModule)....6
2.1. The root object system (TSYS)....6
2.2. Object of the messages system (TMess)....10
2.3. Object subsystem (TSubSYS)....11
2.4. Object Module (TModule)....12
3. Subsystem "Database" (TBDS)....13
3.1. Object of subsystem «Database» (TBDS)....13
3.2. Modular object of types of databases (TTipBD)....14
3.3. The object of the database (TBD)....14
3.4. The object of the table (TTable)....15
4. The subsystem “Data acquisition” (TDAQS)....16
4.1. Object of subsystem «Data acquisition» (TDAQS)....17
4.2. Modular object of the controller's type (TTipDAQ)....17
4.3. Controller's object (TController)....18
4.4. Parameters' type object (TTipParam)....19
4.5. Object of the physical level parameter (TParamContr)....20
4.6. Object of the value (Tvalue)....20
4.7. Attribute's object (TVal).....21
4.8. Object of the templates library of parameters of the "DAQ" subsystem (TPrmTmplLib) 22
4.9. The object of the parameter's template of the "DAQ" subsystem (TPrmTempl)....22
5. Subsystem “Archives” (TArchiveS)....23
5.1. The object of the subsystem "Archives" (TArchiveS)....24
5.2. The object of the values' archive (TVArchive)....25
5.3. Object of the values' buffer (TValBuf)....26
5.4. The modular object of the archiver's type (TTipArchivator)....27
5.5. The object of the messages' archiver (TMArchivator)....27
5.6. The object of the values' archiver (TVArchivator)....28
5.7. The object of the archive's element in the archiver (TVArchEl)....29
6. Subsystem "Transports" (TTransportS)....30
6.1. The object of the «Transports» subsystem (TTransportS)....30
6.2. The modular object of the transports' type (TTipTransport)....31
6.3. The object of the incoming transports (TTransportIn)....31
6.4. The object of the outgoing transports (TTransportOut)....32
7. Subsystem “Communication interfaces' protocols” (TProtocolS)....33
7.1. The object of the “Communication interfaces' protocols” subsystem (TProtocolS)....33
7.2. The modular object of the protocol (TProtocol)....33
7.3. The object of the incoming protocol's session (TProtocolIn)....33
8. Subsystem “User interfaces” (TUIS)....34
8.1. The object of the “User interfaces” subsystem (TUIS)....34
8.2. The modular object of the user interface (TUI)....34
9. Subsystem “Specials” (TSpecialS)....35
9.1. The object of the “Specials” subsystem (TSpecialS)....35
9.2. The modular object of the specials (TSpecial)....35
10. Subsystem “Security” (TSecurity)....36
10.1. The object of the "Security" subsystem. (TSecurity)....36
10.2. The user's object (TUser)....36
10.3. The users' group object (TGroup)....37
11. Subsystem “Modules' sheduling” (TModSchedul)....38
11.1. The object of the subsystem “Modules' sheduling” (TModSchedul)....38
12. Components of the object model of the OpenSCADA system....39

12.1. The function object (TFunction)....39
12.2. The object of the function's parameter (IO)....40
12.3. The object of the function's value (TValFunc)....41
13. Data in the OpenSCADA system and their storage in the DB (TConfig)....42
13.1. Data object (TConfig)....42
13.2. Data cell (TCfg)....43
13.3. Data structure object (TElem)....44
13.4. Data structure cell (TFld)....44
13.5. The object which preacts about changing of the structure (TValElem)....45
13.6. Data cell (TVariant)....46
13.7. User object (TVarObj)....47
14. The control interface of the system and the dynamic tree of the system objects (TCntrNode)....48
14.1. The syntax of the request and response of the control interface....50
14.2. Tag of the information structure for describing the groups of child branches of the page....50
14.3. Tags of the description of the information structure of the control interface....50
14.3.1. Area tag <area>....51
14.3.2. Data tags....51
14.4 Hierarchical dependences of the information elements of the control language....55
14.5. Object of the dynamic tree node (TCntrNode)....57
15. XML in the OpenSCADA system (XMLNode)....60
15.1. XML-tag (XMLNode)....60
16. Resources in the OpenSCADA system (Res, ResAlloc, AutoHD)....62
16.1. Resource object (Res)....62
16.2. Resource object (ResAlloc)....62
16.3. Template (AutoHD)....62
16.4. Object of the string with the access shared by the resource (ResString)....63
17. Organization and structure of the database of the system components....64
17.1. System tables....64
17.2. Tables of the "Data acquisition" subsystem....64
17.3. Tables of the "Transports" subsystem....65
17.4. Tables of the "Archives" subsystem....66
17.5. Tables of the "Security" subsystem....66
17.6. The structure of the databases of the modules....66
18. Service functions of the OpenSCADA control interface....67
18.1. Group access to the values of the parameter's attributes of the subsystem "Data acquisition", as well as to the detailed information....67
18.2. Access to archived data of the archives of messages....67
18.3. Access to archived data of the values' archive....68
19. API of modules of modular subsystems....70
20. Debugging and Testing the OpenSCADA project....75
21. Rules for design and commenting of the sources of OpenSCADA and its modules....76
22. Conventional signs in the text and source code....77

Introduction

This paper describes an application programming interface (API) of the OpenSCADA system.

OpenSCADA is the project of an open SCADA-system based on the modular principle. The document contains an exhaustive description of the internal system architecture of OpenSCADA. In addition to information on the architecture the background information on methods and attributes of the system is provided. The document is intended for programmers wishing to understand the architecture of OpenSCADA and develop extensions for it. The document is not intended for users and integrators of OpenSCADA!

For understanding of this document it is necessary for you to know the concept of Object Oriented Programming (OOP) and Universal Modeling Language (UML), and for the possibility of studying the source code of the project it is required the knowledge of programming language C++. In addition, the document contains the mentioning of the following technologies: relational databases, XML.

1. The internal structure, API system OpenSCADA.

For the purpose of visual and affordable perception of the OpenSCADA system architecture in general the Figure 1 shows the static class diagram of OpenSCADA on the universal modeling language (UML). Based on the chart it is clear that the OpenSCADA system contains modular subsystems: «Archives», «Databases», «Transports», «Transport protocols», «User interfaces», «Data acquisition» and «Special», as well as subsystems: «Safety» and «Module's management». The chart graphically presents the relationship between modular subsystems and modules.

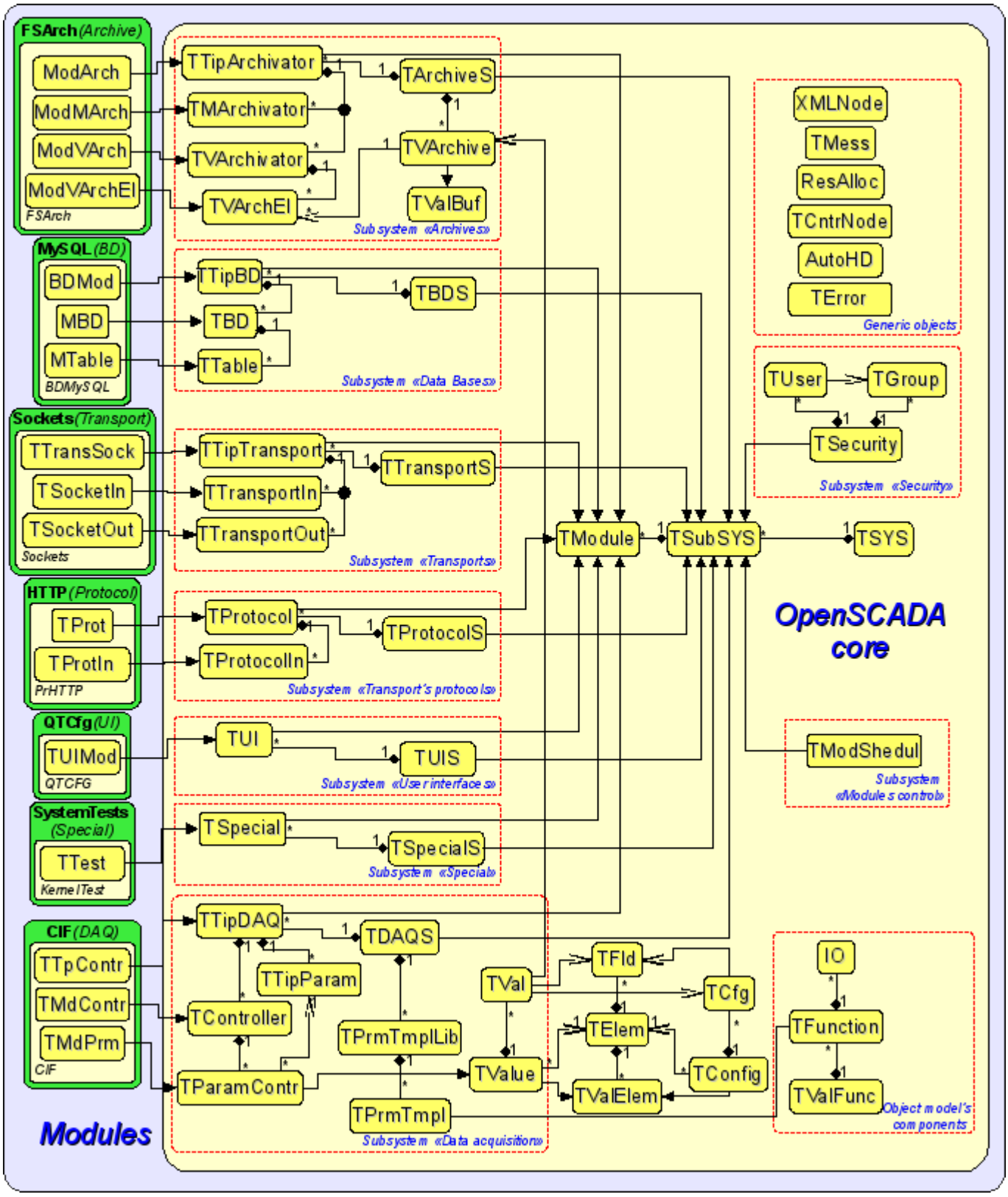

Fig. 1. The static class diagram

2. The overall structure of the system. Modularity (TSubSYS, TModule)

The root, from which we construct the whole system is the object of TSYS. The root contains a subsystem (TSubSYS). Subsystems can be: the ordinary and modular. The difference between the modular subsystems is clearly shown in Fig. 1. Thus, the modular subsystems necessarily contain a list of modular structures (TModule), such as subsystem TArchiveS contain modular objects TTipArchivator. At the same time, the normal subsystem does not contain such objects. For example TSesurity subsystem (Figure 2).

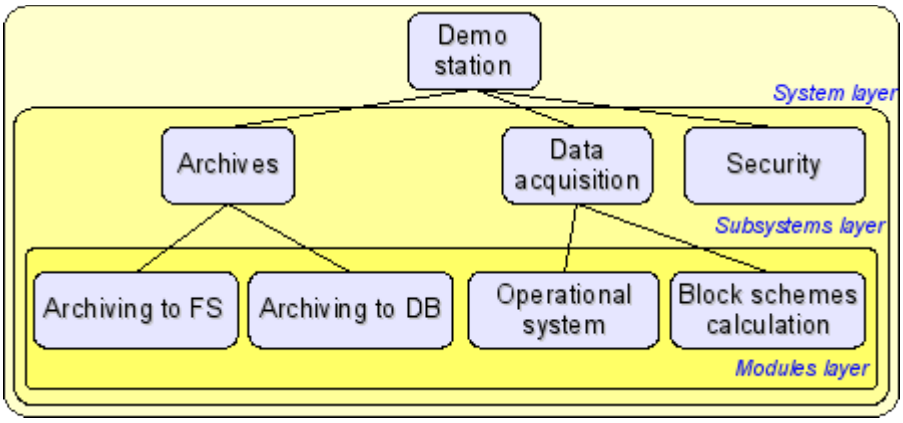

Fig. 2. Hierarchical structure of OpenSCADA.

In the process of initializing the root (TSYS) the global variable SYS is defined. The variable SYS can be used for direct access to the root of the system from any of its node. Initialization of the root is performed only one time from the main calling function. After starting the management is captured by the system object till stop. The root object concentrate all system functions of the OpenSCADA system.

Extension of the root object (TSYS) is the object TMess, which performs the functions of service of the flow of system messages. The object is accessible through the global variable Mess, which is initialized by the root of the system. The object contains the functions of encoding, decoding, and localization of messages.

In the subsystems (TSubSYS) the functions typical for each subsystem individually with the general access for all subsystems are carried out through the object TSubSYS. Modular subsystem is able to extend its functionality by means of modules. For this purpose, modular subsystem provides access to the modules of its type in the form of modular objects.

Module is the component of the modular subsystem. In general, for all modules and subsystems, the module provides information about itself, its origin and export functions. Individual module implements the functionality according to its own needs.

2.1. The root object system (TSYS)

Inherits:	TCntrNode.

Data:

Information variables of the program:

- *PACKAGE_LICENSE* — Distribution license of the program
- *PACKAGE_DESCR* — Brief description of the program
- *PACKAGE_AUTHOR* — Author of the program
- *PACKAGE_SITE* — Web site of the program support

Methods for coding of symbol sequences (enum — TSYS::Code):

- *PathEl* — element of the path (the characters: '/' and '%' to the form '%2f');
- *HttpURL* — address of the browser (http url);
- *Html* — special characters for use in html;
- *JavaSc* — end of line character for JavaScript;
- *SQL* — SQL-query value;

- *Custom* — selective coding of the specified characters;
- *base64* — Mime encoding in the standard Base64;
- *FormatPrint* — Coding/masking of the formating elements for functions like "printf";
- *oscdID* — coding of the nodes' identifiers.
- *Bin* — Encoding of the binary data in text and vice versa.
- *Reverse* — Invert the order of characters in the line.
- *ShieldSimb* — Shielded symbols like "\n" encoding to real code.

Types of representations of the integer in the function TSYS::int2str(), and TSYS::ll2str() (enum — TSYS::IntView):

- *Dec* — decimal;
- *Oct* — octal;
- *Hex* — hexadecimal.

Standard error codes in the OpenSCADA system (enum — TSYS::Errors):

- *DBInit* (1) — Error initializing the database;
- *DBConn* (2) — Error connecting to the database;
- *DBInernal* (3) — Internal error of the DB;
- *DBRequest* (4) — Error in the query to the database;
- *DBOpen* (5) — Error opening the database;
- *DBOpenTable* (6) — Error opening the table;
- *DBClose* (7) — Error closing the database;
- *DBTableEmpty* (8) — The database table is empty;
- *DBRowNoPresent* (9) — An entry in the table is missing.

Templates:

- *TO_FREE* — The value of the free object (NULL).
- *STR_BUF_LEN* — Standard length of string buffers in the OpenSCADA (3000).
- *STD_WAIT_DELAY* — Standard quantum of the waiting time cycles (100ms).
- *STD_WAIT_TM* — Standard interval of the event waiting.
- *__func__* — The full name of the calling function.
- *vmin (a,b)* — Determination of the minimum value.
- *vmax (a,b)* — Definition of the maximum value.

Public methods:

- *TSYS(int argi, char **argb, char **env);* — The initializing constructor.
- *int start( );* — Start of the system. The function is finished only with the finishing of the system. The return code is returned.
- *void stop( );* — The command of the system stop.
- *int stopSignal( );* — The return code in the case of system shutdown. Can be used as a sign of «System shutdown» of the various subsystems.
- *string id( );* — station ID.
- *string name( );* — Localized name of the station.
- *string user( );* — The system user on behalf of which the system is running .
- *string host( );* — Host name for the station run.
- *void list(vector<string> &list);* — A list of registered subsystems in the system.
- *bool present(const string &name);* — Check the availability of the subsystem.
- *void add(TSubSYS *sub);* — Add/registration of the subsystem.
- *void del(const string &name);* — Removing the subsystem.
- *AutoHD<TSubSYS> at(const string &name);* — Connection to the specified subsystem.
- *AutoHD<TUIS> ui( );* — Direct access to the subsystem «User interfaces».
- *AutoHD<TArchiveS> archive( );* — Direct access to the subsystem «Archives».
- *AutoHD<TBDS> db( );* — Direct access to the subsystem «Databases».
- *AutoHD<TControllerS> daq( );* — Direct access to the subsystem «Data acquisition».
- *AutoHD<TProtocolS> protocol( );* — Direct access to the subsystem «Protocols».
- *AutoHD<TTransportS> transport( );* — Direct access to the subsystem «Transports».
- *AutoHD<TSpecialS> special( );* — Direct access to the subsystem «special».
- *AutoHD<TModSchedul> modSchedul( );* — Direct access to the subsystem «Module».

- *AutoHD <TSesurity> sesurity( );* — Direct access to the subsystem «Security».
- *string workDir( );* — Working directory of the station.
- *string icoDir( );* — Directory of the icons of OpenSCADA.
- *string modDir( );* — Directory of the modules of OpenSCADA.
- *void setWorkDir(const string &wdir);* — Setting the working directory of the station.
- *void setIcoDir(const string &idir);* — Specifying of the directory of icons of OpenSCADA.
- *void setModDir(const string &mdir);* — Specifying the directory of modules of OpenSCADA.
- *string cfgFile( );* — Name of configuration file of the system.
- *XMLNode &cfgRoot( );* — Parsed structure of the configuration file.
- *XMLNode *cfgNode(const string &path, bool create = false);* — Node of configuration getting by it path <*path*>. Elements of the path creation by it miss <*create*>.
- *void modifCfg( );* — Configuration mark to modification, for next saving to file.
- *string workDB( );* — The full name of the working database.
- *string selDB( );* — Chosen DB. Used for selective loading from the specified database in the subsystem «DB».
- *bool chkSelDB(const string &wDB);* — The function of checking for compliance of the specified database <wDB> with the selected selDB( ).
- *void setWorkDB(const string &wdb);* — Setting of the full name of the working database.
- *void setSelDB(const string &vl);* — Installation of the selected database for the specified boot.
- *bool saveAtExit( );* — Sign - «Save configuration on exit».
- *void setSaveAtExit(bool vl);* — Setting of the sign - «Save configuration on exit».
- *int savePeriod( );* — Frequency of the automatically saving the station in the database (seconds).
- *void setSavePeriod(int vl);* — Set the frequency of the automatically saving of the station to the database (seconds).
- *string optDescr( );* — Localized help on the options of the command line and parameters of the configuration file.
- *static void sighandler(int signal);* — Function of the default handler of the signals of the system as a whole.
- *unsigned long long sysClk( );* — The estimated frequency of the processor on which the system works (Hz).
- *void clkCalc( );* — Calculation of frequency of the processor on which the system works. It is called periodically for the systems with variable CPU frequency.
- *unsigned long long shrtCnt( );* — The function of the measurement of small intervals of time by the counter of the CPU cycles. Returns the value of the counter of CPU cycles.
- *static long HZ( );* — Time of the system teak of the CPU.
- *bool cntrEmpty( );* — Debug counters checking to empty.
- *double cntrGet(const string &id);* — Debug counter <*id*> getting.
- *void cntrSet(const string &id, double vl);* — Debug counter <*id*> setting to value <*vl*>.
- *void cntrIter(const string &id, double vl);* — Debug counter <*id*> iteration to value <*vl*>.
- *void taskCreate(const string &path, int priority, void *(*start_routine)(void *), void *arg, int wtm = 5, pthread_attr_t *pAttr = NULL, bool *startSt = NULL);* — Create task (thread) with id <*path*>, priority <*priority*> (-1...99), task's function <*start_routine*> and it argument <*arg*>, and also wait for user's code start by flag <*startSt*>.
- *void taskDestroy(const string &path, bool *endrunCntr = NULL, int wtm = 5, bool noSignal = false);* — Destroy task with id <*path*>, start control flag <*startCntr*>. Use <*noSignal*> for exclude signal *SIGALRM* send to the task.
- *static int sysSleep(float tm);* — System sleep in seconds up to nanoseconds (1e-9).
- *static long long curTime( );* — The actual time in microsecond from the beginning of the era (01.01.1970).
- *static void taskSleep(long long per, time_t cron=0);* — The function of the flow sleep by the grid of absolute time with the period <per> in the nanosecond or for the scheduled time <cron>.
- *static time_t cron(const string &vl, time_t base=0);* — Planning the execution time on the format of the standard Cron <*vl*> beginning at the basic time <*base*> or at the current time if the base is not specified.
- *static bool eventWait(bool &m_mess_r_stat, bool exempl, const string &loc, time_t time=0);* — Function of the event waiting <*exempl*> for the variable <*m_mess_r_stat*> within a specified time interval <*time*> for the source <*loc*>.

• *static string int2str(int val, IntView view=Dec);* — Transformation of a signed integer to the string of the type of view <*view*>.
• *static string uint2str(unsigned val, IntView view=Dec);* — Transformation of unsigned integer to the string type of view <*view*>.
• *static string ll2str(long long val, IntView view=Dec);* — Transformation of a long integer (64bit) to the string type of view <*view*>.
• *static string real2str(double val, int prec=15, char tp='g');* — Real transformation with an accuracy <*prec*> signs and type <*tp*> to the string.
• *static double realRound(double val, int dig=0, bool toint=false);* — Rounding the real number to the specified digit <*dig*> after the decimal point with the possibility of transformation to integer after rounding <*toint*>.
• *static string time2str(time_t tm, const string &format);* — UNIX time <*tm*> conversion to string, by format <*format*> of POSIX-function strftime().
• *static string time2str(double utm);* — Convert time interval to string like "1hour 23min 10sec".
• *static string cpct2str(double cnt);* — Traffic counter <*cnt*> (bytes) convert to string like "12.5KiB".
• *static string addr2str(void *addr);* — Convert addresses into string.
• *static void *str2addr(const string &str);* — Convert string into the address.
• *static string strNoSpace(const string &val);* — Deletes from the original string <*val*> empty characters at the beginning and end of it.
• *static string strSepParse(const string &path, int level, char sep, int *off=NULL);* — Parsing of the string <*path*> into the components separated by the dividing symbol <*sep*>, beginning with the offset <*off*> and controlling the offset of the element end in itself.
• *static string strParse(const string &str, int level, const string &sep, int *off = NULL, bool mergeSepSymb = false);* — Parse function's strSepParse() expanded version which allow using multi-symbols separators and one symbols merging.
• *static string strLine(const string &str, int level, int *off = NULL);* — Parsing text lines for different ways to the end of a line (CR, LF and CR/LF).
• *static string pathLev(const string &path, int level, bool encode=true, int *off=NULL);* — The allocation of the elements of the path <*path*> with the ability of their decode, starting with the offset <*off*> and controlling the offset of the element end in itself.
• *static string path2sepstr(const string &path, char sep='.');* — Transformation the path into string with the separator <*sep*> of the elements.
• *static string sepstr2path(const string &str, char sep='.');* — Transformation of the string with the separator <*sep*> of the elements into the path.
• *static string strEncode(const string &in, Code tp, const string &symb="\t\n"); — Encoding of the string by the specified rule* <tp>//.
• *static string strDecode(const string &in, Code tp=Custom);* — Decoding of the string by the specified rule <*tp*>.
• *static string strMess(const char *fmt, ...);* — Formation of the string by the template <*fmt*> and arguments. It is implemented on the basis of printf.
• *string strCompr(const string &in, int lev=-1);* — Compression of the string <*in*> with the compression level <*lev*>.
• *string strUncompr(const string &in);* — Decompression of the string <*in*>.
• *static inline uint16_t getUnalign16(const void *p);* — Unaligned read for unsigned integer in 16-bit from buffer by offset.
• *static inline uint32_t getUnalign32(const void *p);* — Unaligned read for unsigned integer in 32-bit from buffer by offset.
• *static inline uint64_t getUnalign64(const void *p);* — Unaligned read for unsigned integer in 64-bit from buffer by offset.
• *static inline int getUnalignInt(const void *p);* — Unaligned read for integer from buffer by offset.
• *static inline float getUnalignFloat(const void *p);* — Unaligned read by real "float" from buffer by offset.
• *static inline double getUnalignDbl(const void *p);* — Unaligned read by real "double" from buffer by offset.
• *static float floatLE(float in);* — Real number "float" conversion from internal to format IEEE754 Little-Endian (LE).
• *static float floatLErev(float in);* — Real number "float" conversion from format IEEE754 Little-

Endian (LE) to internal.
• *static double doubleLE(double in);* — Real number "double" conversion from internal to format IEEE754 Little-Endian (LE).
• *static double doubleLErev(double in);* — Real number "double" conversion from format IEEE754 Little-Endian (LE) to internal.

Public attributes:
• *static bool finalKill* — Sign «The final killing of». It is used for forced disconnection of blocked objects on the the final stage of shutdown.
• *const int argc* — Counter of the arguments of the the command line.
• *const char **argv* — Buffer of the command line arguments.
• *const char **envp* — Pointer to the list of parameters of the environment.

2.2. Object of the messages system (TMess)

Data:
Types (levels) of messages (enum — TMess:: Type):
- *Debug* — debug;
- *Info* — information;
- *Notice* — the notification;
- *Warning* — warning;
- *Error* — error;
- *Crit* — critical situation;
- *Alert* — alert;
- *Emerg* — emergency.

The structure of the message (class — TMess:: SRec):
- *time_t time;* — time of the message;
- *int utime;* — microsecond of the message time;
- *string categ;* — category of messages (usually the way inside the system);
- *Type level;* — the level of the message;
- *string mess;* — message.

Templates:
• *_(mess)* — A wrapper over the translation of the messages function for the provision of the accepted by the most of the programs translation of the messages.
• *FTM(rec)* — Get a full time of the message, in microseconds, using the two fields of time of the message structure.
• *message(cat, lev, fmt, args ...)* — Formation of the the full message.
• *mess_debug(cat, fmt, args ...)* — Formation of the debug message.
• *mess_info(cat, fmt, args ...)* — Formation of the information message.
• *mess_note(cat, fmt, args ...)* — Formations of the message - notification.
• *mess_warning(cat, fmt, args ...)* — Formation of the warning message.
• *mess_err(cat, fmt, args ...)* — Formation of the error message.
• *mess_crit(cat, fmt, args ...)* — Formation of the critical condition message.
• *mess_alert(cat, fmt, args ...)* — Formation of the alarm message.
• *mess_emerg(cat, fmt, args ...)* — Formation of the emergency message.

Public methods:
• *void load( );* — Download.
• *void save( );* — Saving.
• *string codeConv(const string &fromCH, const string &toCH, const string &mess);* — Conversion of the codepage of the messages.
• *string codeConvIn(const string &fromCH, const string &mess);* — Conversion the codepage of the messages into the internal system codepage.
• *string codeConvOut (const string &toCH, const string &mess);* — Conversion of the message from the internal system codepage.

• *static const char *I18N(const char *mess, const char *d_name = NULL);* — Getting the message in system the language.
• *static string I18Ns(const string &mess, const char *d_name = NULL);* — Getting the message in the system language.
• *string lang( );* — Language of the system, as en_US.UTF-8.
• *string lang2Code( );* — Language of the system in bi-symbolic codepage (en).
• *string lang2CodeBase( );* — The language of basic variables of the text messages in bi-symbolic codepage(en).
• *string &charset( );* — The system codepage.
• *int logDirect( );* — Receivers to whom the system messages (stdout, stderr, syslog, archive) are sent;
• *int messLevel( );* — The level below which the messages are ignored.
• *bool isUTF8( );* — Internal codepage is UTF-8.
• *void setLang(const string &lang);* — Setting of the system language (localization).
• *void setLang2CodeBase(const string &vl);* — Setting the language of the basic message of text variables in the bi-symbolic codepage (en).
• *void setLogDirect(int dir);* — Setting receivers to which the system messages are sent. For <dir> the bit mask is used. Where:
1 — to syslog;
2 — to stdout;
4 — to stderr;
8 — to the archive.
• *void setMessLevel(int level);* — Setting a minimum level of processed messages.
• *void put(const char *categ, int8_t level, const char *fmt, ...);* — Create a message for the current time.
• *void get(time_t b_tm, time_t e_tm, vector<TMess::SRec> &recs, const string &category = "", int8_t level = Debug);* — Get the message from the archive for the period of time <*b_tm*> — <*e_tm*> under the category template <*category*> and minimum level <*level*>.

2.3. Object subsystem (TSubSYS)

Inherits:	*TCntrNode.*
Inherited:	*TArchiveS, TProtocolS, TBDS, TFunctionS, TSesurity, TModShedul, TTransportS, TUIS, TSpecialS, TControllerS.*

Public methods:
• *TSubSYS(const char *id, const char *name, bool mod = false);* — Initialize the constructor. Sign <mod> indicates that subsystem is module one.
• *string subId( );* — ID of the subsystem.
• *string subName( );* — Localized name of the subsystem.
• *bool subStartStat( );* — Sign of the subsystem execution.
• *bool subModule( );* — Sign of the modularity of the subsystem.
• *virtual int subVer( );* — Version of the subsystems.
• *virtual void subStart( );* — Start of the the subsystem.
• *virtual void subStop( );* — Stop of the subsystem.
• *void modList(vector <string> &list);* — List <*list*> of the modules of the modular subsystem.
• *bool modPresent(const string &name);* — Check for the availability of the module <*name*>.
• *void modAdd(TModule *modul);* — Add/registration of the module <*modul*>.
• *void modDel(const string &name);* — Deleting of the module <*name*>.
• *AutoHD <TModule> modAt(const string &name);* — Connection to the module <*name*>.
• *virtual void perSYSCall(unsigned int cnt);* — Periodic call from system thread, with period 10 seconds and seconds counter <*cnt*>.
• *TSYS &owner( );* — The system - the owner of the subsystem.

2.4. Object Module (TModule)

Inherits:	*TCntrNode.*
Inherited:	*TProtocol, TTipBD, TTipArchive, TTipTransport, TUI, Tspecial, TTipController.*

Data:

The data structure which identifies the module (class — TModule::SAt):

- *SAt(const string &iid, const string &itype = "", int itver = 0);* — initialize the constructor;
- *bool operator ==(const TModule::SAt &amst) const;* — comparison function identifiers modules;
- *string id;* — the identifier of the module;
- *string type;* — the type of module (subsystem);
- *int t_ver;* — version of the type of module (subsystem) to which the module is designed.

The structure of exported functions (class — TModule::ExpFunc):

- *string prot;* — a prototype of the function;
- *string dscr;* — localized description of the function;
- *void (TModule::*ptr) ();* — relative address of the function (with respect to the object module).

Public methods:

- *string modId( );* — ID of the module.
- *string modName( );* — Localized name of the module.
- *virtual void modStart( );* — Start of the module.
- *virtual void modStop( );* — Stop of the module.
- *virtual void modInfo(vector<string> &list);* — A list of information items <*list*> of the module.
- *virtual string modInfo (const string & name);* — Getting the contents of the specified information item <*name*>.
- *virtual void perSYSCall(unsigned int cnt);* — Periodic call from system thread, with period 10 seconds and seconds counter <*cnt*>.
- *void modFuncList(vector<string> &list);* — The list of exported functions <*list*> of the module.
- *bool modFuncPresent(const string &prot);* — Check the availability of the specified function by its prototype <*prot*>.
- *ExpFunc &modFunc (const string & prot);* — Get the information about exported function of the module <*prot*>.
- *void modFunc(const string &prot, void(TModule::**offptr)());* — Getting the relative addresse <*offptr*> of exported function <*prot*>.
- *const char *I18N(const char *mess);* — Localization of modular message <*mess*> in accordance with the current locale.
- *TSubSYS &owner( );* — Subsystem - the owner of the module.

Protected Attributes:

- *string mName;* — Name of the module.
- *string mDescr;* — Description of the module.
- *string mType;* — Type of module.
- *string mVers;* — Version of the module.
- *string mAutor;* — Author of the module.
- *string mLicense;* — License of the module.
- *string mSource;* — Source/origin of the module.

Protected methods are:

- *void modFuncReg (ExpFunc *func);* — Registration of the exported by module functions.

3. Subsystem "Database" (TBDS)

Subsystem «Databases» is represented by the object *TBDS*, which contains a modular objects of the following types of DB *TTipBD*. Each type of database contains objects of individual databases of that type *TBD*. Each database in its turn, contains the objects of their tables TTable (Fig. 3).

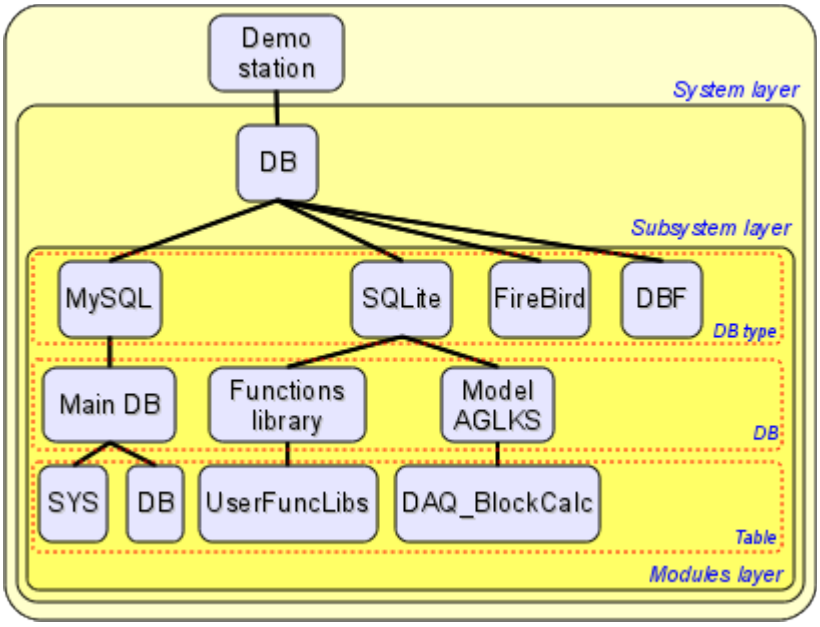

Fig. 3. Hierarchical structure of the database subsystem.

The subsystem provides the basic functions to access the type of database, as well as generalized functions for the manipulation of the databases and tables. For example, to hide the source of data, which may be a configuration file, the functions of an abstract access to the data source are provided. For the storage system-wide data the system table and the function of the abstract to access it are provided. Consequently, system-wide data can be stored in the configuration file and in the database table. Priority source, in this case, is the database table.

Being a modular object, the type of database (TTipBD) provides access to the implementation of the mechanism of one or another database. Access is made through a public databases of the module of a given type of database. Open/registered database is described in the table of databases to be opened or in the configuration file. There is, the so-called, the working database, which is always opens and is shown in the configuration file. DB which support the SQL-queries can grant access based on direct SQL-queries.

While working, the components of the OpenSCADA system open tables (TTable) available in the database and work with them.

3.1. Object of subsystem «Database» (TBDS)

Inherits:	*TSubSYS, TElem.*

Data:

Flags of the queries to the system table (enum – TDBS::ReqGen):

- *OnlyCfg* — request only to the configuration file.
- *UseTranslate* — use the translation of text variable.

Public methods:

- *int subVer( );* — Subsystem version.
- *static string realDBName(const string &bdn);* — Conversion of the full template name of the database or table (of view *.*.myTbl) in the real name. Actually the replacement of the special elements '*' by the elements of the working DB.
- *void dbList(vector<string> &ls, bool checkSel = false);* — A list of available databases.. <*checkSel*> points to the necessity to verify the fact of boot from the selected database and to insert to the db list the only selected one.

• *AutoHD<TTable> open(const string &bdn, bool create = false);* — Opening the table <*bdn*> of the DB by its full path with the creation <*create*> in the case of its absence.
• *void close(const string &bdn, bool del = false);* — Closing the table <*bdn*> of the DB by its full path with the possibility of deleting of it after closing <*del*>.
• *bool dataSeek(const string &bdn, const string &path, int lev, TConfig &cfg);* — Total scanning of the records of the data source. The configuration file of the DB is the source if data. In the case of DB absence the configuration file is used. If the DB name <bdn> or path <path> to the configuration file are not indicated, their processing is skipped.
• *bool dataGet(const string &bdn, const string &path, TConfig &cfg);* — Getting the record from the data source (database or configuration file). If the DB name <bdn> or path <path> to the configuration file are not indicated, their processing is skipped.
• *void dataSet(const string &bdn, const string &path, TConfig &cfg);* — Set/Save the record in the data source (database or configuration file). If the DB name <bdn> or path <path> to the configuration file are not indicated, their processing is skipped.
• *bool dataDel(const string &bdn, const string &path, TConfig &cfg, bool useKeyAll = false);* — Deleting the record in the data source (database or configuration file). If the DB name <*bdn*> or path <*path*> to the configuration file are not indicated, their processing is skipped. <*useKeyAll*> is used to indicate the necessity of the all keys set fro their using while deleting with the restoration of the original state of the keys selection when the function is finished. If this flag is not set, the early selected keys are used to perform the operation.
• *static string genDBGet(const string &path, const string &oval = "", const string &user = "root", char rFlg = 0);* — Get system-wide data from the configuration file or system tables on behalf of user name <*user*>. If data are not available the value <*oval*> is returned.
• *static void genDBSet(const string &path, const string &val, const string &user = "root", char rFlg = 0);* — Set/Save the system-wide data in the configuration file or system tables on behalf of user name <*user*>.
• *string fullDBSYS( );* — The full name of the system table.
• *string fullDB( );* — The full name of the table with a description of the registered databases.
• *TElem &openDB_E( )* — The structure of the table of registered databases.
• *AutoHD<TTipBD> at(const string &iid)* — Referencing to the DB module(DB type).
• *string optDescr( );* — Localized help on the command line options and parameters in the configuration file.

3.2. Modular object of types of databases (TTipBD)

Inherits:	*TModule.*
Inherited:	By root objects of the modules of subsystem «DB».

Public methods:
• *bool fullDeleteDB( );* — Sign of the complete removal of the database.
• *void list(vector<string> &list);* — The list of registered (opened) databases.
• *bool openStat(const string &idb);* — Check the availability of the specified opened database.
• *void open(const string &iid);* — Opening of the database.
• *void close(const string &iid, bool erase = false);* — Closing the database. If the sign <*erase*> is set, the database will be completely removed.
• *AutoHD<TBD> at(const string &name);* — Connection to the opened database.
• *TBDS &owner( );* — Subsystem - the owner of the module.

3.3. The object of the database (TBD)

Inherits:	*TCntrNode, TConfig.*
Inherited:	By the database objects of the modules of subsystem «DB».

Public methods:
• *TBD(const string &iid, TElem *cf_el);* — Initializing constructor.
• *string id( );* — DB identifier.
• *string name( );* — DB name.

- *string dscr( );* — DB description.
- *string addr( );* — Address of the database. The form of recording is different for each type of database.
- *string codePage( );* — Codepage in which the data of the database are stored.
- *bool enableStat( );* — The state of the database: «Enabled».
- *bool toEnable( );* — Sign of the DB: "To Enable".
- *void setName(const string &inm);* — Setting the DB name.
- *void setDscr(const string &idscr);* — Setting the DB description.
- *void setAddr(const string &iaddr);* — Setting the DB address.
- *void setCodePage(const string &icp);* — Setting of the codepage for the storage of the DB data.
- *void setToEnable(bool ivl);* — Setting of the sign: "To Enable".
- *virtual void enable( );* — Enabling of the DB.
- *virtual void disable( );* — Disabling of the DB.
- *virtual void allowList(vector<string> &list);* — The list of the tables in this DB.
- *void list(vector<string> &list);* — The list of opened tables.
- *bool openStat(const string &table);* — The sign that indicates of the fact that the requested table is opened.
- *void open(const string &table, bool create);* — Opening of the table. If the sign <create> is set, in the case of absence the table will be created.
- *void close(const string &table, bool del = false);* — Closing of the table. If the sign <del> is set, the table will be completely removed.
- *AutoHD<TTable> at(const string &name);* — Connection to the table.
- *virtual void sqlReq(const string &req, vector< vector<string> > *tbl = NULL, char intoTrans = EVAL_BOOL);* — Sending the SQL-request <*req*> to the DB and receiving the result in tabular form <*tbl*>. If set <*intoTrans*> to true then will open transaction for the request, else if set to false then transaction will close.
- *virtual void transCloseCheck( )* — The function call periodic for opened transaction check to close for old transaction or overloaded.
- *TTipBD &owner( );* — DB type – the owner of the DB.

Protected methods:

- *virtual TTable *openTable(const string &table, bool create);* — The modular method for opening the table.

3.4. The object of the table (TTable)

Inherits:	*TCntrNode.*
Inherited:	By tables objects of the modules of subsystem "DB".

Public methods:

- *TTable(const string &name);* — Initializing constructor.
- *string name( );* — Table name.
- *virtual void fieldStruct(TConfig &cfg);* — Getting the structure of the table.
- *virtual bool fieldSeek(int row, TConfig &cfg);* — Scanning the records of the table.
- *virtual void fieldGet(TConfig &cfg);* — Request the specified record. The requested record is determined by the values of key cells of the original record <*cfg*>.
- *virtual void fieldSet(TConfig &cfg);* — Setting the values of the specified record. In te case of absence the record will be removed.
- *virtual void fieldDel(TConfig &cfg);* — Deleting of the specified record.
- *TBD &owner( );* — DB – the owner of the table.

4. The subsystem “Data acquisition” (TDAQS)

The subsystem "Data acquisition" is represented by the *TDAQS* object which contains modular objects of the data sources' types *TTipDAQ* and the objects of the libraries of parameters' templates of subsystem "Data acquisition" *TPrmTmplLib*. Object of the data sources types contains objects of the controllers TController and objects of the parameters' types *TTipParam*. Parameters' types objects are provided by the controller module and contain the DB structure of the separate parameters' types (analog, digital ...). Controllers' objects contain parameters' objects *TParamContr*. Each parameter is associated with only one type of the parameter. For the attribute storage parameter is inherited from the values object *TValue*, which contains the attributes' values *TVal*. The library of the parameters' templates of this subsystem contains templates' objects *TPrmTmpl*. An example of the described hierarchical structure is shown in Fig. 4.

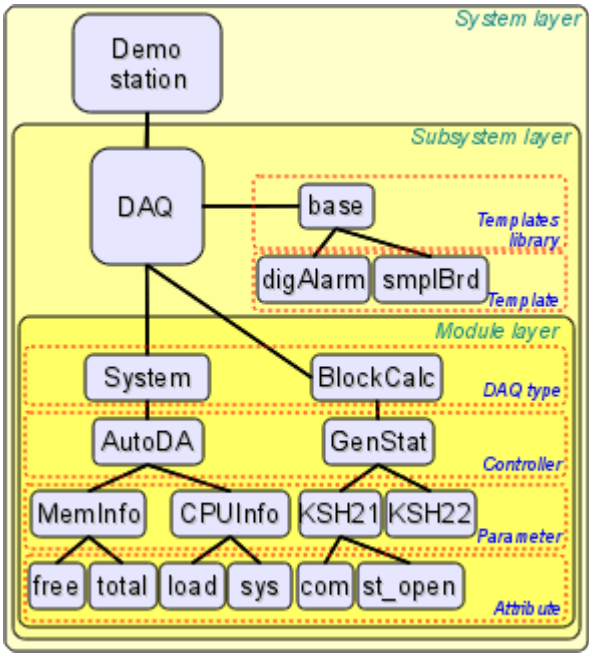

Fig. 4. Hierarchical structure of the data acquisition subsystem.

Subsystem contains the types of data sources. The source may be virtually any substance providing any data. Type of source can be divided into individual sources (controllers) within the limits of the particular type. For example, if we take the data from the operating system (OS), then the single source can be the separate operating system of the separate PC.

Data source (controller) is further divided (contains) into the parameters. The parameter is the part of the data source. In the case of the OS it will be, for example,: used RAM, the processor's frequency and many other parts.

Parameter, in its turn, contains the attributes, which provide the data. In addition to the basic data attributes can provide the related or detailing data. In the case of the same operating system and the memory usage, the attributes may not only provide the used memory, and also how much it all, how much in the swap, etc.

Some of the implementation of the data sources may provide the possibility of setting the structure of the parameter based on previously developed parameters' templates. For this purpose subsystem contains templates' libraries, which, in their turn, provide the parameters' templates . The example shows a library of templates "base" with the templates "digAlrm" and "smplBrd".

At the level of the subsystem the redundancy mechanism for data sources is provided. Redundancy means the possibility of coordinated work of several OpenSCADA stations to perform common task of data acquisition from the same data sources.

4.1. Object of subsystem «Data acquisition» (TDAQS)

Inherits:	*TSubSYS.*

Public methods:

- *int subVer( );* — Version of the subsystem.
- *void subStart( );* — Start the subsystem.
- *void subStop( );* — Stop the subsystem.
- *AutoHD<TTipDAQ> at(const string &name);* — Connection to the type of data source.
- *string tmplLibTable( );* — Name of table to store the parameters' templates of the subsystem «Data acquisition».
- *void tmplLibList(vector<string> &list);* — The list of available parameters' templates.
- *bool tmplLibPresent(const string &id);* — Check the availability of the parameter's template <id>.
- *void tmplLibReg(TPrmTmplLib *lib);* — Registration of the parameter's template <lib>.
- *void tmplLibUnreg(const string &id, int flg = 0);* — Deleting/removing of the registration of the parameter's template <id>.
- *AutoHD<TPrmTmplLib> tmplLibAt(const string &id);* — Connection to the parameter's template <id>.
- *bool rdActive( );* — Sign of the activity of the redundancy scheme. Points to the fact that there are at least one active backup station.
- *int rdStLevel( );* — Redundancy level of the current station.
- *void setRdStLevel(int vl);* — Set the redundancy level of the current station.
- *int rdTaskPer( );* — Frequency of the redundancy service task in seconds.
- *void setRdTaskPer(int vl);* — Set the frequency of the redundancy service task.
- *int rdRestConnTm( );* — Time of the retry attempts to reconnect to the backup station after its loss in seconds.
- *void setRdRestConnTm(int vl);* — Set of the time of the retry attempts to reconnect to the backup stations.
- *float rdRestDtTm( );* — Maximum depth of the archive data recovery when starting in hours.
- *void setRdRestDtTm(float vl);* — Set of the maximum depth of the archive data recovery when starting in hours.
- *void rdStList(vector<string> &ls);* — The list of stations in the reserve.
- *void rdActCntrList(vector<string> &ls, bool isRun = false);* — The list of active controllers working in the redundancy scheme. When we indicate <*isRun*> there will be in the list only running controllers in this station controllers.
- *string rdStRequest(const string &cntr, XMLNode &req, const string &prevSt = "", bool toRun = true);* — Request <*req*> to the backup station on behalf of the controller <*cntr*>. The station for the request is selected after specified in <*prevSt*> for the running remote controller when indicating <*toRun*>.
- *TElem &elLib();* — The structure of the table of the libraries if parameters' templates.
- *TElem &tplE();* — The structure of the table of parameters' templates.
- *TElem &tplIOE();* — The structure of the attributes of parameters' templates.
- *TElem &errE();* — The structure of the attribute(s) of parameters' errors.

4.2. Modular object of the controller's type (TTipDAQ)

Inherits:	*TModule, TElem.*
Inherited:	Root object of the modules of subsystem «Data acquisition».

Public methods:

- *string DAQPath( );* — Getting for DAQ-address of the element.
- *void modStart( );* — Start of the module.
- *void modStop( );* — Stop of the module.
- *void list(vector<string> &list);* — The list of the controllers.
- *bool present(const string &name);* — Check for the availability of the specified controller.
- *void add(const string &name, const string &daq_db = "*.*");* — Add the controller.
- *void del(const string &name);* — Delete the controller.

• *AutoHD<TController> at(const string &name, const string &who = "");* — Connect to the controller.
• *bool tpPrmPresent(const string &name_t);* — Check for the availability of the specified controller type.
• *unsigned tpPrmToId(const string &name_t);* — Getting of the index of the parameters' types by their names.
• *int tpParmAdd(const char *id, const char *n_db, const char *name);* — Addition/registration of the parameters' types.
• *unsigned tpPrmSize( );* — Quantity of the parameters' types.
• *TTipParam &tpPrmAt(unsigned id);* — Get the object of the parameters' type.
• *virtual bool compileFuncLangs(vector<string> *ls = NULL);* — Request of the list of languages for which it is realized the possibility of formation of user procedures in this module, and check for it support fact.
• *virtual void compileFuncSynthHighl(const string &lang, XMLNode &shgl);* — The rules of syntax highlight request for specifying language.
• *virtual string compileFunc(const string &lang, TFunction &fnc_cfg, const string &prog_text, const string &usings = "", int maxCalcTm = 10);* — Compiling/setup of the user function on the supported programming language <lang> and on the source code of procedure <prog_text>, based on the parameters of procedure <fnc_cfg>. The result is the address to the prepared function object.
• *virtual bool redntAllow( );* — The indication of support of redundancy mechanisms by the module. It must be simply overridden and return **true**.

Protected methods:
• *virtual TController *ContrAttach(const string &name, const string &daq_db);* — Connecting of the controller. It must be obligatory overridden in the descendant of the module.

4.3. Controller's object (TController)

Inherits:	*TCntrNode, TConfig.*
Inherited:	Objects of the modules of subsystem «Data acquisition».

Data:
Redundancy modes (enum TController::Redundant):
- *Off* — Turned off;
- *Asymmetric* — asymmetric;
- *Symmetric* — symmetric.

Public methods:
• *TController(const string &name_c, const string &daq_db, TElem *cfgelem);* — Initializing constructor of the controller.
• *string DAQPath( );* — Getting for DAQ-address of the element.
• *string id( );* — Controller ID.
• *string workId( );* — Work ID of the controller, including the ID of the module.
• *string name( );* — Controller's name.
• *string descr( );* — Controller's description.
• *virtual string getStatus( );* — Request function of the controller's status.
• *string DB( );* — DB name of the controller's instance.
• *string tbl( );* — The name of the table of DB of the controller's instance.
• *string fullDB( );* — The full name of the table of DB of the controller's instance.
• *void setName(const string &nm);* — Set the controller's name.
• *void setDescr(const string &dscr);* — Set the controller's description.
• *void setDB(const string &idb);* — Setting of the DB name of the controller's instance.
• *bool toEnable( );* — Sign «To enable the controller».
• *bool toStart( );* — Sign «To start the controller».
• *bool enableStat( );* — Status «Enable».
• *bool startStat( );* — Status «Run».
• *void start( );* — Controller's start.

- *void stop( );* — Controller's stop.
- *void enable( );* — Controller's enabling.
- *void disable( );* — Controller's stopping.
- *void list(vector<string> &list);* — Parameter's list in the controller.
- *bool present(const string &name);* — Check for the availability of the parameter <*name*>.
- *void add(const string &name, unsigned type);* — Addition of the parameter <*name*> of the type <*type*>.
- *void del(const string &name, bool full = false);* — Deleting of the parameter <*name*>. If the field <*full*> is specified the controller will be completely removed.
- *AutoHD<TParamContr> at(const string &name, const string &who = "th_contr");* — Connection to the controller's parameter <*name*>.
- *bool redntUse( );* — Getting the data from the backup station mode.
- *void setRedntUse(bool vl);* — Changing of the getting the data from the backup station mode.
- *Redundant redntMode( );* — Redundancy mode.
- *void setRedntMode(Redundant vl);* — Setting of the redundancy mode.
- *string redntRun( );* — Configuration of the preferred execution.
- *void setRedntRun(const string &vl);* — Setting the configuration of the preferred execution.
- *virtual void redntDataUpdate(bool firstArchiveSync = false);* — An operation of the data receiving from the backup station execution. It is called automatically by the service task of the redundancy scheme and before start to synchronize archives with the defined parameter <*firstArchiveSync*>.
- *void alarmSet(const string &mess, int lev = -TMess::Crit, const string &prm = "");* — Formation of the alarm (violation) for the object controller <*prm*>, or the controller as a whole if the object is not specified, the message <*mess*> and level <*lev*>. The negative value of the level <*lev*> is used to set and positive for removal of the violation. This function generates the violation and message with a category: **al{ModId}:{CntrId}[.{PrmId}]**.
- *TTipDAQ &owner( );* — The type of the data source (module) - the owner of the controller.

Protected attributes:

- *bool en_st;* — Sign «Enable».
- *bool run_st;* — Sign «Running».

Protected methods:

- *virtual void enable_( );* — Enabling of the controller. Intercepted by the child.
- *virtual void disable_( );* — Disabling of the controller. Intercepted by the child.
- *virtual void start_( );* — Starting of the controller. Intercepted by the child.
- *virtual void stop_( );* — Stopping of the controller. Intercepted by the child.
- *virtual TParamContr *ParamAttach(const string &name, int type);* — Modular method of the creation/opening of the new parameter.

4.4. Parameters' type object (TTipParam)

Inherits:	*TElem.*

Public methods:

- *TTipParam(const char *id, const char *name, const char *db);* — Initializing constructor.

Public attributes:

- *string name;* — Parameter's type name.
- *string descr;* — Parameter's type description.
- *string db;* — DB of the parameter's type.

4.5. Object of the physical level parameter (TParamContr)

Inherits:	*TConfig, TValue.*
Inherited:	Objects of the module's parameters of subsystem «Data acquisition».

Public methods:

- *TParamContr(const string &name, TTipParam *tpprm);* — Initializing constructor.
- *string DAQPath( );* — Getting for DAQ-address of the element.
- *string id( );* — Parameter's ID (key).
- *string name( );* — Parameter's name.
- *string descr( );* — Parameter's description.
- *bool toEnable( );* — Sign «To enable parameter».
- *bool enableStat( )* — Status «Enable».
- *void setName(const string &inm);* — Setting of the parameter's name.
- *void setDescr(const string &idsc);* — Setting of the parameter's description.
- *void setToEnable(bool vl);* — Setting of the sign «Ro enable the parameter».
- *TTipParam &type( );* — Parameter's type.
- *virtual void enable( );* — To enable parameter.
- *virtual void disable( );* — To disable parameter.
- *bool operator==(TParamContr & PrmCntr);* — Parameter's comparison.
- *TParamContr &operator=(TParamContr & PrmCntr);* — Copying of the parameter.
- *TController &owner( );* — Controller – the parameter's owner.

Public attributes:

- *long long mRedntTmLast* — the time of the last data receiving through the redundancy mechanism.

Protected methods:

- *virtual void setType(const string &tpId);* — is called to change the parameter type <*tpId*> and can be processed in the object module for self data change.

4.6. Object of the value (Tvalue)

Inherits:	*TCntrNode, TValElem.*
Inherited:	*TParamContr.*

Public methods:

- *virtual string DAQPath( );* — Getting for DAQ-address of the element.
- *void vlList(vector<string> &list);* — Getting the attributes' list.
- *bool vlPresent(const string &name);* — Check for the availability of the specified attribute.
- *AutoHD<TVal> vlAt(const string &name);* — Connection to the attribute.

Protected methods:

- *TConfig *vlCfg( )* — Getting of the associated configuration object. If NULL is returned then there is no associated configuration object.
- *void setVlCfg(TConfig *cfg);* — Setting of the associated configuration object <*cfg*>.
- *bool vlElemPresent(TElem *ValEl);* — Check for the availability if the attributes' elements <*ValEl*>.
- *void vlElemAtt(TElem *ValEl);* — Attach of the data structure <*ValEl*>.
- *void vlElemDet(TElem *ValEl);* — Detach of the data structure <*ValEl*>.
- *TElem &vlElem(const string &name);* — Get the data structure by its name <*name*>.
- *virtual TVal* vlNew( );* — Creation of the TVal instance. It can be overridden in the module for the creation of derived objects of the parameters' attributes of the subsystem "Data acquisition".
- *virtual void vlSet(TVal &val, const TVariant &pvl)* — Predicting function of the value setting. It is used for direct (synchronous) writing with the previous value into <*pvl*>.
- *virtual void vlGet(TVal &val);* — Predicting function of the value getting. It is used for direct (synchronous) reading.
- *virtual void vlArchMake(TVal &val);* — Notifying function of the creation of the archive for the

<val> attribute. It is used to configure the created archive in accordance with the peculiarities of the data source.

4.7. Attribute's object (TVal).

Inherits:	*TCntrNode.*

Data:

Additional flags to the object TFld (enum TVal::AttrFlag):

- *DirRead* — Flag for the direct reading of the value;
- *DirWrite* — Flag for the direct writing of the value.

Public methods:

- *TVal( );* — Default constructor.
- *TVal(TFld &fld);* — Initialization as the repository of dynamic data.
- *TVal(TCfg &cfg);* — Initialization as the reflection of static data (DB).
- *string DAQPath( );* — Getting for DAQ-address of the element.
- *void setFld(TFld &fld);* — Initialization as the repository of dynamic data.
- *void setCfg(TCfg &cfg);* — Initialization as the reflection of static data (DB).
- *string name( );* — Attribute's name.
- *int64_t time( );* — Time marker of the the last/current value.
- *string getSEL(long long *tm = NULL, bool sys = false);* — Request of the selected type value for the specified time <*tm*>. If NULL then it will be returned the last value.
- *TVariant get(long long *tm = NULL, bool sys = false);* — Request of the value for the specified time <*tm*>. If NULL then it will be returned the last value.
- *string getS(long long *tm = NULL, bool sys = false);* — Request of the string type value for the specified time <*tm*>. If NULL then it will be returned the last value.
- *double getR(long long *tm = NULL, bool sys = false);* — Request of the real type value for the specified time <*tm*>. If NULL then it will be returned the last value.
- *int getI(long long *tm = NULL, bool sys = false);* — Request of the integer type value for the specified time <*tm*>. If NULL then it will be returned the last value.
- *char getB(long long *tm = NULL, bool sys = false);* — Request of the boolean type value for the specified time <*tm*>. If NULL then it will be returned the last value.
- *AutoHD<TVarObj> getO(int64_t *tm = NULL, bool sys = false);* — Request of the object type value.
- *void setSEL(const string &value, long long tm = 0, bool sys = false);* — Setting of the selected type value <*value*>.
- *void set(const TVariant &value, long long tm = 0, bool sys = false); — Setting of the value <value>.*
- *void setS(const string &value, long long tm = 0, bool sys = false);* — Setting of the string type value <*value*>.
- *void setR(double value, long long tm = 0, bool sys = false);* — Setting of the real type value <*value*>.
- *void setI(int value, long long tm = 0, bool sys = false);* — Setting of the integer type value <*value*>.
- *void setB(char value, long long tm = 0, bool sys = false);* — Setting of the boolean type value <*value*>.
- *void setO(AutoHD<TVarObj> value, int64_t tm = 0, bool sys = false);* — Setting of the object type value <*value*>.
- *AutoHD<TVArchive> arch( );* — Getting the associated with the value archive.
- *void setArch(const AutoHD<TVArchive> &vl);* — Setting the associated with the value archive.
- *bool reqFlg( ); bool resB1( ); bool resB2( )* — Get some realisation-specific flags.
- *void setReqFlg(bool vl); void setResB1(bool vl); void setResB2(bool vl)* — Set some realisation-specific flags.
- *TFld &fld( );* — Descriptor of the attribute's structure.

4.8. Object of the templates library of parameters of the "DAQ" subsystem (TPrmTmplLib)

Inherits:	*TCntrNode, TConfig.*

Public methods:

- *TPrmTmplLib(const char *id, const char *name, const string &lib_db);* — Initializing constructor.
- *string id( );* — Library ID.
- *string name( );* — Library name.
- *string descr( );* — Library description.
- *string DB( );* — DB of the library instance.
- *string tbl( );* — DB table of the library instance.
- *string fullDB( );* — Full address of the DB table of the library instance.
- *bool startStat( );* — Sign "Library started".
- *void start(bool val);* — Satart/stop of the library.
- *void setName(const string &vl);* — Setting of the library name.
- *void setDescr(const string &vl);* — Setting of the library description.
- *void setFullDB(const string &vl);* — Setting of the full address of the DB table of the library instance.
- *void list(vector<string> &ls);* — Templates list in the library.
- *bool present(const string &id);* — Check for the presence of the template <id> in the library.
- *AutoHD<TPrmTempl> at(const string &id);* — Connection to the template <id>.
- *void add(const char *id, const char *name = "");* — Addition of the template <id.name> into the library.
- *void del(const char *id, bool full_del = false);* — Deleting of the template <id> from the library.
- *TDAQS &owner( );* — Object- subsystem "DAQ", owner of the library.

4.9. The object of the parameter's template of the "DAQ" subsystem (TPrmTempl)

Inherits:	*TFunction, TConfig.*

Data:

Additional flags to the attribute's object of the function IO (enum TPrmTempl::IOTmplFlgs):

- *AttrRead* — Read only attribute;
- *AttrFull* — Full access attribute;
- *CfgPublConst* — Public constant;
- *CfgLink* — External link;
- *LockAttr* — Blocked attribute.

Public methods:

- *TPrmTempl(const char *id, const char *name = "");* — Initializing constructor of the template.
- *string id( );* — Parameter's template ID.
- *string name( );* — Parameter's template name.
- *string descr( );* — Parameter's template description.
- *int maxCalcTm( );* — The limit on the maximum time calculation of procedure template.
- *string progLang( );* — Parameter's template programming language.
- *string prog( );* — Parameter's template program.
- *void setName(const string &inm);* — Setting the name of the parameter's template.
- *void setDescr(const string &idsc);* — Setting the description of the parameter's template.
- *void setMaxCalcTm(int vl);* — Setting the limit on the maximum time calculation of procedure template.
- *void setProgLang(const string &ilng);* — Setting the programming language of the parameter's template.
- *void setProg(const string &iprg);* — Setting the program of the parameter's template.
- *void setStart(bool val);* — Start/stop of the parameter's template.
- *AutoHD<TFunction> func( );* — Connection to the function formed by the template.
- *TPrmTmplLib &owner( );* — Object of the templates' library - the owner of the template.

5. Subsystem “Archives” (TArchiveS)

Subsystem "Archives" is represented by an object TArchiveS, which contains at the subsystem level the modular objects of the archivers types TTipArchivator. Each object of the archiver type contains objects of the messages' archivers TMArchivator and values' archivers TVArchivator. In addition, the subsystem object contains the methods of the messages archive and objects of the values' archives TVArchive. The object of the values' archive TVArchive contains the buffer of values through the inheritance of the buffer object TValBuf. To connect the archive of values with the archivers the object of the value element TVArchEl is provided. This object is contained in the archiver and it is referenced by the archive. Structure of the subsystem "Archives" is presented in Fig. 5.

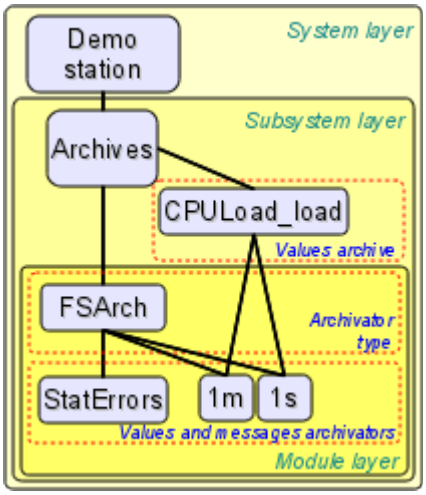

Fig. 5. The hierarchical structure of subsystem "Archives".

Subsystem "Archives" contains the mechanisms for archiving of messages and values. It directly contains the messages' archive together with its buffer. Contains methods for accessing the archives of the values and for the archivers of values and messages. Besides it performs the actively data acquisition from sources of values for the archives of values, as well as archiving the archive of messages by the archivers.

Archive of values (TVArchive) contains the buffer (TValBuf) for intermediate values' accumulation before archiving. It is connected with the source of values in the person of OpenSCADA system parameters in active or passive mode, as well as with other sources in the passive mode. To archive to the physical storage it is connected with the values' archivers of various types.

Object of the buffer TValBuf contains an array of values of the main types of OpenSCADA system: string, integer, real and boolean. It is supported the storage of values in the modes of hard, soft grid and in the free access mode. It is also provided the mode of high-resolution time (microseconds). It is used for direct storage of large arrays of values, and for the exchange of large arrays by the frame-accurate method of access.

Root object of the module of subsystem "Archives" (TTipArchivator) contains information about the specific type of module. Within the individual modules it can implement their own module-wide functions. In general, for modules of this type it contains methods to access the repositories of values and messages.

Object of the messages' archiver (TMArchivator) contains the specific implementation of the message storage. In general, for messages' archivers the interface of access to the implementation of an archiving mechanism in modules is provided.

Object of the values' archiver (TVArchivator) contains the specific implementation of the repository of values. In general, for the values' archivers the access interface to implementation of the archiving mechanism and the appointment of archives of values for service by archiver are provided.

Object of the archive element TVArchEl links the archive objects with the archivers. It is used to access the archiver from the archive, as well as to archives from the archiver, ie for cross-calls.

5.1. The object of the subsystem "Archives" (TArchiveS)

Inherits:	*SubSYS.*

Public methods:

- *int subVer( );* — Subsystem's version.
- *int messPeriod( );* — Period of archiving of messages from the buffer (seconds).
- *int valPeriod( );* — Acquisition of values period for the active archivers (milliseconds).
- *int valPrior( );* — Priority of the tasks of the values acquisition for the active archivers.
- *void setMessPeriod(int ivl);* — Setting the period of the archiving of messages from the buffer (seconds).
- *void setValPeriod(int ivl);* — Setting the values' acquisition period for active archivers (milliseconds).
- *void setValPrior(int ivl);* — Setting the priority of the values' acquisition task for the active archivers.
- *void subStart( );* — Start of the subsystem.
- *void subStop( );* — Stop of the system.
- *void valList(vector<string> &list);* — The list of the values'archives in the subsystem.
- *bool valPresent(const string &iid);* — Check for the availability of the values' archive <*iid*>.
- *void valAdd(const string &iid, const string &idb = "*.*");* — Addition of the new values' archive <*iid*>.
- *void valDel(const string &iid, bool db = false);* — Deleting of the values' archive <*iid*>.
- *AutoHD<TVArchive> valAt(const string &iid);* — Connection/addressing to the values' archive <*iid*>.
- *void setActValArch(const string &id, bool val);* — Setting of the archive <*id*> into the active status <*val*>. Active archive will be provided with the periodic flow of values (is determined by the periodicity of the archive) of the subsystem.
- *AutoHD<TTipArchivator> at(const string &name);* — Connection/addressing to the archiver's type (module) <*name*>.
- *void messPut(time_t tm, int utm, const string &categ, int8_t level, const string &mess);* — Location of the value <*mess*> with the level <*level*> of the category <*categ*> and time <*tm*>+<*utm*> into the buffer, and then into the messages' archive.
- *void messPut(const vector<TMess::SRec> &recs);* — Location of the group of values <*recs*> into the buffer, and then into the messages' archive.
- *void messGet(time_t b_tm, time_t e_tm, vector<TMess::SRec> & recs, const string &category = "", int8_t level = TMess::Debug, const string &arch = "", time_t upTo = 0);* — Values' request <*reqs*> for the specified period of time <*b_tm*>, <*e_tm*> for the specified category (by the template) <*category*> and level <*level*> from the archivator <*arch*>.
- *time_t messBeg(const string &arch = "");* — Beginning of the messages' archive as a whole or for the specified archiver <*arch*>.
- *time_t messEnd(const string &arch = "");* — End of the messages' archive as a whole or for the specified archiver <*arch*>.
- *TElem &messE( );* — DB structure of the messages' archivers.
- *TElem &valE( );* — DB structure of the values' archivers.
- *TElem &aValE( );* — DB structure of the values' archives.

Public methods:

- *static int max_req_mess;* — Maximum size of the messages request (by defaults 3000).
- *bool SubStarting;* — The subsystem starting flag.

5.2. The object of the values' archive (TVArchive)

Inherits:	*TCntrNode, TValBuf, TConfig*

Data:

The data acquisition mode/source (struct — TVArchive::SrcMode):

- *Passive* — passive mode of the data acquisition, the source puts the data into the archive itself;
- *PassiveAttr* — passive mode of the data acquisition from the parameter's attribute, the parameter's attribute puts the data into the archive itself;
- *ActiveAttr* — active mode of the data acquisition from the parameter's attribute, the parameter's attribute is periodically polled by the subsystem "Archives";

Public methods:

- *TVArchive(const string &id, const string &db, TElem *cf_el);* — Initializing constructor of the archive, where <*id*> — archive ID, <*db*> — DB for storage and <*cf_el*> — DB structure of the values' archives.
- *string id( );* — Archive's ID.
- *string name( );* — Archive's name.
- *string dscr( );* — Archive's description.
- *SrcMode srcMode( );* — Linkage mode with the data source.
- *AutoHD<TVal> srcPAttr(bool force = false, const string &ipath = "");* — Connect to associated attribute of parameter of data source.
- *string srcData( );* — Parameters of the data source, in the case of access mode to the parameter this is the address of the parameter.
- *bool toStart( );* — Sign: "Start the archive when starting the system".
- *bool startStat( );* — Status: "Archive started".
- *string DB( );* — DB address of the values' archive.
- *string tbl( );* — DB table of the values' archive.
- *string fullDB( );* — Full name of the DB table of the values' archive.
- *long long end(const string &arch = BUF_ARCH_NM);* — End of the archive time at whole (arch="") or the specified archiver, buffer (arch="<bufer>").
- *long long begin(const string &arch = BUF_ARCH_NM);* — Start of the archive time at whole (arch="") or the specified archiver, buffer (arch="<bufer>").
- *long long period(const string &arch = BUF_ARCH_NM);* — Periodicity of the archive buffer or the specified archiver (microseconds).
- *TFld::Type valType( );* — Type of the archived value.
- *bool hardGrid( );* — Using of the hard grid in the archive buffer.
- *bool highResTm( );* — Using of the high-resolution time in the archive buffer (microseconds).
- *int size( );* — Archive buffer size (units).
- *void setName(const string &inm);* — Setting the name of the archive.
- *void setDscr(const string &idscr);* — Setting the description of the archive.
- *void setSrcMode(SrcMode vl = SaveCur, const string &isrc = "<*>", bool noex = false);* — Setting the linkage mode with the data source.
- *void setToStart(bool vl);* — Setting the sign: "Start the archive when starting the system".
- *void setDB(const string &idb);* — Setting the DB address of the values' archive.
- *void setValType(TFld::Type vl);* — Setting the type of the archived value.
- *void setHardGrid(bool vl);* — Setting the using of the hard grid in the archive buffer.
- *void setHighResTm(bool vl);* — Setting of using of the high-resolution time in the archive buffer (microseconds).
- *void setSize(int vl);* — Setting of the archive buffer size (units).
- *void setPeriod(long long vl);* — Setting the periodicity of the archive buffer.
- *void start( );* — Start of the archive.
- *void stop(bool full_del = false);* — Stop of the archive with it's completely deleting <*full_del*>.
- *TVariant getVal(long long *tm = NULL, bool up_ord = false, const string &arch = "", bool onlyLocal = false);* — Request of the single value for the time <*tm*> and with the sign of the moving to the top <*up_ord*> from the specified archiver <*arch*>, buffer (arch = "<bufer">) or from all archivers in the course of falling quality (arch =""). To process the request only by the local station <*onlyLocal*> is

set.

- *void getVals(TValBuf &buf, long long beg = 0, long long end = 0, const string &arch = "", int limit = 100000, bool onlyLocal = false);* — Request of the values' frame <*buf*> for the time from <*beg*> to <*end*> from the specified archiver <*arch*>, buffer (arch = "<bufer>") or from the all archivers in the course of falling quality (arch =""), limiting the size of the request <*limit*> of the record. To request only the local archives, without compensation gaps in the archives from the backup stations, set <*onlyLocal*>.
- *void setVals(TValBuf &buf, long long beg, long long end, const string &arch);* — Set of the values' frame <*buf*> for the time from <*beg*> to <*end*> to the specified archiver <*arch*>, buffer (arch = "<bufer">) or to all archivers (arch ="").
- *void getActiveData( );* — To interrogate the data source. Is used for periodic data acquisition by the active archives in the subsystem.
- *void archivatorList(vector<string> &ls);* — List of archivers which serves the archive.
- *bool archivatorPresent(const string &arch);* — Check the archiver for it's servicing of the given archive.
- *void archivatorAttach(const string &arch);* — Connecting the archive to be served by the specified archiver.
- *void archivatorDetach(const string &arch, bool full = false);* — Disabling this archive from the servicing of the specified archiver.
- *void archivatorSort( );* — Sort of the serving archivers in the order of decreasing quality.
- *string makeTrendImg(long long beg, long long end, const string &arch, int hsz = 650, int vsz = 230);* — Building of the (pdf) image of the trend for the specified amount of time <*beg*>, <*end*> and for this archiver <*arch*>.
- *TArchiveS &owner( );* — Subsystem "Archives" - the owner of the archive of values.

5.3. Object of the values' buffer (TValBuf)

Inherited:	*TVArchive*

Public methods:

- *TValBuf( );* — Buffer initializer with default settings.
- *TValBuf(TFld::Type vtp, int isz, long long ipr, bool ihgrd = false, bool ihres = false);* — Buffer initializer with the specified parameters.
- *void clear( );* — Cleaning the buffer.
- *TFld::Type valType( );* — Type of the value stored in the buffer.
- *bool hardGrid( );* — Hard grid working mode.
- *bool highResTm( );* — High resolution (microseconds) working mode of the buffer.
- *int size( );* — Maximum buffer size (units).
- *int realSize( );* — Real buffer size (units).
- *long long period( );* — Frequency of values in the buffer (microseconds). If the frequency is zero the buffer operates in the mode of free access.
- *long long begin( );* — Start time of the buffer (microseconds).
- *long long end( );* — End time of the buffer (microseconds).
- *bool vOK(long long ibeg, long long iend);* — Checking for values in the buffer for the specified period of time from <*ibeg*> to <*iend*>.
- *void setValType(TFld::Type vl);* — Setting of the value type stored by the buffer.
- *void setHardGrid(bool vl);* — Setting the hard grid mode.
- *void setHighResTm(bool vl);* — High resolution (microseconds) working mode setting.
- *void setSize(int vl);* — Buffer size (units) setting.
- *void setPeriod(long long vl);* — Frequency of values in the buffer (microseconds) setting.
- *virtual void getVals(TValBuf &buf, long long beg = 0, long long end = 0);* — Request of the values' frame <*buf*> for the time from <*beg*> to <*end*>.
- *virtual string getS(long long *tm = NULL, bool up_ord = false);* — Request of the string type value fro the time <*tm*> and with the sign of moving to the top <*up_ord*>.
- *virtual double getR(long long *tm = NULL, bool up_ord = false);* — Request of the real type value fro the time <*tm*> and with the sign of moving to the top <*up_ord*>.
- *virtual int getI(long long *tm = NULL, bool up_ord = false);* — Request of the integer type value fro the time <*tm*> and with the sign of moving to the top <*up_ord*>.

• *virtual char getB(long long *tm = NULL, bool up_ord = false);* — Request of the boolean type value fro the time <*tm*> and with the sign of moving to the top <*up_ord*>.
• *virtual void setVals(TValBuf &buf, long long beg = 0, long long end = 0);* — Setting the values' frame from /<buf> *fro the time from* <beg> *to* <end>//.
• *virtual void setS(const string &value, long long tm = 0);* — Setting the string type value with the time <*tm*>.
• *virtual void setR(double value, long long tm = 0);* — Setting the real type value with the time <*tm*>.
• *virtual void setI(int value, long long tm = 0);* — Setting the integer type value with the time <*tm*>.
• *virtual void setB(char value, long long tm = 0);* — Setting the boolean type value with the time <*tm*>.

Protected methods:
• *void makeBuf(TFld::Type v_tp, int isz, long long ipr, bool hd_grd, bool hg_res);* — Rebuilding the buffer for the specified parameters.

5.4. The modular object of the archiver's type (TTipArchivator)

Inherits:	*TModule.*
Inherited:	By the root objects of the modules of subsystem "Archives".

Public methods:
• *void messList(vector<string> &list);* — The list of the messages' archivers.
• *bool messPresent(const string &iid);* — Check for the presence of the specified messages' archiver.
• *void messAdd(const string &iid, const string &idb = "*.*");* — Adding of the messages' archiver.
• *void messDel(const string &iid, bool full = false);* — Deleting of the messages' archiver.
• *AutoHD<TMArchivator> messAt(const string &iid);* — Connection to the messages' archiver.
• *void valList(vector<string> &list);* — The list of the values' archivers.
• *bool valPresent(const string &iid);* — Check for the presence of the specified values' archiver.
• *void valAdd(const string &iid, const string &idb = "*.*");* — Adding of the values' archiver.
• *void valDel(const string &iid, bool full = false);* — Deleting of the values' archiver.
• *AutoHD<TVArchivator> valAt(const string &iid);* — Connection to the values' archiver.
• *TArchiveS &owner( );* — Subsystem "Archives" - the owner of the archiver's type.

Protected methods:
• *virtual TMArchivator *AMess(const string &iid, const string &idb);* — The modular method of creating of the messages' archiver.
• *virtual TVArchivator *AVal(const string &iid, const string &idb);* — The modular method of creating of the values' archiver.

5.5. The object of the messages' archiver (TMArchivator)

Inherits:	*TCntrNode, TConfig*
Inherited:	By the objects of the archivers of messages of modules of the subsystem "Archives".

Public methods:
• *TMArchivator(const string &id, const string &db, TElem *cf_el);* — Initializing constructor of the messages' archiver with the ID <*id*>, for the storage in the database <*db*> with the structure <*cf_el*>.
• *string id( );* — Archiver's ID.
• *string workId( );* — Working identifier, it includes the name of the module.
• *string name( );* — Archiver's name.
• *string dscr( );* — Archiver's description.
• *bool toStart( );* — The sign "To start the archiver".
• *bool startStat( );* — Archivers' status "Run".
• *string &addr( );* — Address of the archiver's storage.
• *int &level( );* — The level of messages, serviced by the archiver.
• *void categ(vector<string> &list);* — Categories (templates) of messages, serviced by the archiver.
• *string DB( );* — DB address of the archiver.

- *string tbl( );* — Table address of the archiver's DB.
- *string fullDB( );* —The full table address of the archiver's DB.
- *void setName(const string &vl);* — Setting the archiver's name.
- *void setDscr(const string &vl);* — Setting the archiver's description.
- *void setToStart(bool vl);* — Setting the sign "To start the archiver".
- *void setAddr(const string &vl);* — Setting the address of the archiver's storage.
- *void setLevel(int lev);* — Setting the level of messages, serviced by the archiver.
- *void setDB(const string &idb);* — Setting the DB adress of the archiver.
- *virtual void start( );* — Start of the archiver.
- *virtual void stop( );* — Stop of the archiver.
- *virtual time_t begin( );* — The beginning of the archive of the specified archiver.
- *virtual time_t end( );* — The end of the archive of the specified archiver.
- *virtual void put(vector<TMess::SRec> &mess);* — Place the group of messages in the messages' archive of the specified archiver.
- *virtual void get(time_t b_tm, time_t e_tm, vector<TMess::SRec> &mess, const string &category = "", char level = 0, time_t upTo = 0);* — Get the messages from the archive of the specified archiver for the specified filter parameters.
- *TTipArchivator &owner( );* — Archiver's type – the owner of the messages' archiver.

Protected attributes:
- *bool run_st;* — Sign "Run".

Protected methods:
- *bool chkMessOK(const string &icateg, TMess::Type ilvl);* — Check the message to the filter conditions.

5.6. The object of the values' archiver (TVArchivator)

Inherits:	*TCntrNode, TConfig*
Inherited:	By the objects of the archivers of values of modules of the subsystem "Archives".

Public methods:
- *TVArchivator(const string &id, const string &db, TElem *cf_el);* — Initializing constructor of the values' archiver with the ID <*id*>, for the storage in the database <*db*> with the structure <*cf_el*>.
- *string id( );* — Archiver's ID.
- *string workId( );* — Working identifier, it includes the name of the module.
- *string name( );* — Archiver's name.
- *string dscr( );* — Archiver's description.
- *string addr( );* — Address of the archiver's storage.
- *double valPeriod( );* — Frequency of values of the archiver (microseconds).
- *int archPeriod( );* — Frequency of the values' archiving by the archiver. Time after which the archiver makes archiving of the frame of values from the archive's buffer.
- *bool toStart( );* — The sign "To start the archiver".
- *bool startStat( );* — Archivers' status "Run".
- *string DB( );* — DB address of the archiver.
- *string tbl( );* — Table address of the archiver's DB.
- *string fullDB( );* — The full table address of the archiver's DB.
- *void setName(const string &inm);* — Setting the archiver's name.
- *void setDscr(const string &idscr);* — Setting the archiver's description.
- *void setAddr(const string &vl);* — Setting the address of the archiver's storage.
- *virtual void setValPeriod(double iper);* — Setting the frequency of values of the archiver (microseconds).
- *virtual void setArchPeriod(int iper);* — Setting the frequency of the values' archiving by the archiver. Time after which the archiver makes archiving of the frame of values from the archive's buffer.
- *void setToStart(bool vl);* — Setting the sign "To start the archiver".
- *void setDB(const string &idb);* — Setting the DB adress of the archiver.
- *virtual void start( );* — Start of the archiver.

• *virtual void stop(bool full_del = false);* — Stop of the archiver with the possibility of it's full deleting <*full_del*>.
• *void archiveList(vector<string> &ls);* — The archive's list, serviced by the archiver.
• *bool archivePresent(const string &iid);* — Check for the servicing of the specified archive by the specified archiver.
• *TTipArchivator &owner( );* — Archiver's type – the owner of the values' archiver.

Protected methods:
• *TVArchEl *archivePlace(TVArchive &item);* — Add the archive <*item*> to the processing by the archiver.
• *void archiveRemove(const string &id, bool full = false);* — Remove the archive <*id*> from the processing by the archiver, with the possibility of its full deleting <*full*>.
• *virtual TVArchEl *getArchEl(TVArchive &arch);* — Getting the object of the archive's element for the specified archive.

Protected attributes:
• *Res a_res* — Resource of the archiving process.
• *bool run_st* — Sign "Archive is run".
• *vector<TVArchEl *> arch_el;* — Array of elements of archives.

5.7. The object of the archive's element in the archiver (TVArchEl)

Inherited:	By the objects of the archivers of values of modules of the subsystem "Archives".

Public methods:
• *TVArchEl(TVArchive &iachive, TVArchivator &iarchivator);* — Initializing constructor fro the connection the archive <*iachive*> with the archiver <*iarchivator*>.
• *virtual void fullErase( );* — Full deleting of the element.
• *virtual long long end( );* — End time of data (microseconds).
• *virtual long long begin( );* — Start time of data (microseconds).
• *long long lastGet( );* — Time of last transfer of the data from the buffer to the storage.
• *TVariant getVal(long long *tm, bool up_ord, bool onlyLocal = false);* — Request of the value for the time <*tm*> and with the sign of moving to the top <*up_ord*>, with the specifying of the request of only local archive in <*onlyLocal*>.
• *void getVals(TValBuf &buf, long long beg = 0, long long end = 0, bool onlyLocal = false);* — Request of the values' frame <*buf*> for the time from <*beg*> to <*end*>, with the specifying of the request of only local archive in <*onlyLocal*>.
• *void setVals(TValBuf &buf, long long beg = 0, long long end = 0);* — Setting the values' frame from the <*buf*> for the time from <*beg*> to <*end*>.
• *TVArchive &archive( );* — Elements' archive.
• *TVArchivator &archivator( );* — Elements' archiver.

Public methods:
• *long long prev_tm;* — The previous value time. It is used for averaging.
• *string prev_val;* — The previous value time. It is used for averaging.

Protected methods:
• *virtual TVariant getValProc(long long *tm, bool up_ord);* — The processing function of the single value from the archive request.
• *virtual void getValsProc(TValBuf &buf, long long beg, long long end);* — Function of the processing the request of the module to get the data.
• *virtual void setValsProc(TValBuf &buf, long long beg, long long end);* — Function of the processing the request of the module to set the data.

6. Subsystem "Transports" (TTransportS)

"Transports" subsystem is represented by the TTransportS object, which contains modular objects of the transports' types TTipTransport on the subsystem-level. Each type of transport contains objects TTransportIn of the incoming and TTransportOut of the outgoing transports. The overall structure of the subsystem is shown in Fig. 6.

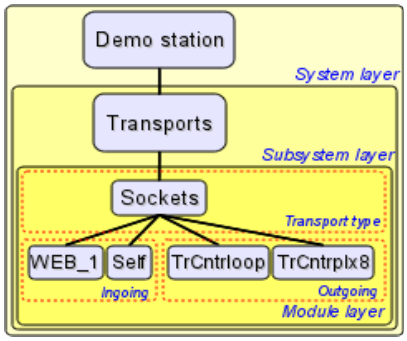

Fig. 6. The layered structure of the transports subsystem.

The root object of the "Transports" subsystem's module provides information about the specific type of module and about the external OpenSCADA hosts/stations. As part of the single module it can be implemented the own general-module functionality. In general, for all modules, the access methods for both: inbound and outbound transports of the specific module are contained.

The object of the incoming transport TTransportIn provides an interface to the implementation of the modular method of incoming transport.

The object of the outgoing transport TTransportOut provides an interface to the implementation of the modular method of outgoing transport.

6.1. The object of the «Transports» subsystem (TTransportS)

Inherits:	*TSubSYS.*

Data:

The structure of the external OpenSCADA hosts/stations (class TTransportS::ExtHost):

- *ExtHost(const string &iuser_open, const string &iid, const string &iname, const string &itransp, const string &iaddr, const string &iuser, const string &ipass);* — Constructor of the structure initialization.
- *string user_open;* — The user who created the record about the external host/station.
- *string id;* — Id of the external host/station.
- *string name;* — The name of an external host/station.
- *string transp;* — Transport which is used to access an external host/station.
- *string addr;* — Address for the transport, which is used to access an external host/station.
- *string user;* — The users of the external host/station.
- *string pass;* — The password of the external host/station user.
- *bool link_ok;* — The sign "Connection with the external host/station is done".

Public methods:

- *int subVer( );* — Subsystem's version.
- *void inTrList(vector<string> &ls);* — Full list of incoming transports.
- *void outTrList(vector<string> &ls);* — Full list of outgoing transports.
- *bool sysHost( );* — Sign - "Show the system hosts".
- *void setSysHost(bool vl);* — Setting of the sign - "Show the system hosts".
- *string extHostsDB( );* — Database for the storing the list of external hosts.
- *void extHostList(const string &user, vector<string> &list);* — The list of external hosts.
- *bool extHostPresent(const string &user, const string &id);* — Check for an external host presence *id* on behalf of the user *user* ("*" — for the system hosts).
- *AutoHD<TTransportOut> extHost(TTransportS::ExtHost host, const string &pref = "");* — Creation

- request of the outgoing transport to service the external host *host* with the prefix of the identification of the system node *pref*.

• *ExtHost extHostGet(const string &user, const string &id);* — Getting the information object from the external host <*id*> on behalf of the user <*user*> ("*" — for the system hosts).

• *void extHostSet(const ExtHost &host);* — Setting of the external host/station <*host*>.

• *void extHostDel(const string &user, const string &id);* — Deleting of the external host/station <*id*> on behalf of the user *user* ("*" — for the system hosts).

• *int cntrIfCmd(XMLNode &node, const string &senderPref, const string &user = "");* — Transfer of the control area request of the OpenSCADA <*node*> to the remote station.

• *void subStart( );* — Start of the subsystem.

• *void subStop( );* — Stop of the subsystem.

• *TElem &inEl( );* — DB structure of the incoming transports

• *TElem &outEl( );* — DB structure of the outgoing transports

• *AutoHD<TTipTransport> at(const string &id);* — Addressing/connection to the type of transport <*id*>.

6.2. The modular object of the transports' type (TTipTransport)

Inherits:	*TModule.*
Inherited:	By the root object of the subsystem's "Transports" modules..

Public methods:

• *void inList(vector<string> &list);* — The list of the incoming transports.

• *bool inPresent(const string &name);* — Check for an incoming transport presence.

• *void inAdd(const string &name, const string &db = "*.*");* — Addition of the incoming transport.

• *void inDel(const string &name, bool complete = false);* — Deleting of the incoming transport. It is possible to completely delete with the database included, by setting the <*complete*> sign.

• *AutoHD<TTransportIn> inAt(const string &name);* — Connection to the incoming transport.

• *void outList(vector<string> &list);* — The list of the outgoing transports.

• *bool outPresent(const string &name);* — Check for an outgoing transport presence.

• *void outAdd(const string &name, const string &db = "*.*");* — Addition of the outgoing transport.

• *void outDel(const string &name, bool complete = false);* — Deleting of the outgoing transport. It is possible to completely delete with the database included, by setting the <*complete*> sign.

• *AutoHD<TTransportOut> outAt(const string &name)* — Connection to the outgoing transport.

• *TTransportS &owner( );* — "Transports" subsystem — the owner of the transport's type.

Protected methods:

• *virtual TTransportIn *In(const string &name, const string &db);* — The modular method of the creating/opening of the new "incoming" transport.

• *virtual TTransportOut *Out(const string &name, const string &db);* — The modular method of the creating/opening of the new "outgoing" transport.

6.3. The object of the incoming transports (TTransportIn)

Inherits:	*TCntrNode, TConfig.*
Inherited:	By the objects of incoming transports of the subsystem's "Transports" modules.

Public methods:

• *TTransportIn(const string &id, const string &db, TElem *el);* — Initializing constructor.

• *string id( );* — Transport's Id.

• *string workId( );* — Full ID including the ID of the module.

• *string name( );* — Transport's name.

• *string dscr( );* — Transport's description.

• *string addr( );* — Address.

• *string protocol( );* — Linked transport protocol.

• *virtual string getStatus( );* — Getting the status of the incoming transport.

• *bool toStart( );* — The sign "To start".

- *bool startStat( );* — The status "Running".
- *string DB( );* — Transport's DB address.
- *string tbl( );* — Transport's DB table.
- *string fullDB( );* — The full name of the transport's DB table.
- *void setName(const string &inm);* — Setting the name of transport in <*inm*>.
- *void setDscr(const string &idscr);* — Setting the description of transport in <*idscr*>.
- *virtual void setAddr(const string &addr);* — Setting the address of transport in <*addr*>.
- *void setProtocol(const string &prt);* — Setting of the linked transport protocol.
- *void setToStart(bool val);* — Setting of the sign "To start".
- *void setDB(const string &vl);* — Setting of the transport's DB address.
- *virtual void start( );* — Start of the transport.
- *virtual void stop( );* — Stop of the transport.
- *TTipTransport &owner( );* — Transport's type – the owner of the incoming transport.

Protected attributes:
- *bool run_st;* — The sign «Run».

6.4. The object of the outgoing transports (TTransportOut)

Inherits:	*TCntrNode, TConfig.*
Inherited:	By the objects of outgoing transports of the subsystem's "Transports" modules.

Public methods:
- *TTransportOut(const string &id, const string &db, TElem *el);* — Initializing constructor.
- *string id( );* — Transport's Id.
- *string workId( );* — Full ID including the ID of the module.
- *string name( );* — Transport's name.
- *string dscr( );* — Transport's description.
- *string addr( );* — Transport's address.
- *int prm1( );* — The first backup parameter.
- *int prm2( );* — The second backup parameter.
- *bool toStart( );* — The sign "To start".
- *bool startStat( );* — The status «Running».
- *virtual string getStatus( );* — Getting the status of the transport.
- *string DB( );* — Transport's DB address.
- *string tbl( );* — Transport's DB table.
- *string fullDB( );* — The full name of the transport's DB table.
- *void setName(const string &inm);* — Setting the name of transport.
- *void setDscr(const string &idscr);* — Setting the description of transport.
- *virtual void setAddr(const string &addr);* — Setting the address of transport.
- *void setPrm1(int vl);* — Setting of the first backup parameter.
- *void setPrm2(int vl);* — Setting of the second backup parameter.
- *void setToStart(bool val);* — Setting of the sign "To start".
- *void setDB(const string &vl);* — Setting of the transport's DB address.
- *virtual void start( );* — Start of the transport.
- *virtual void stop( );* — Stop of the transport.
- *virtual int messIO(const char *obuf, int len_ob, char *ibuf = NULL, int len_ib = 0, int time = 0, bool noRes = false);* — Sending of the data over the transport. The waiting time <*time*> of the connection is specified in milliseconds. <*noRes*> is used by the protocols for exclusive blocking of the transport for the time of working with him and to avoid its own blocking by the function.
- *void messProtIO(XMLNode &io, const string &prot);* — Sending of the data in the XML tree <*in*> over the transport using the transport protocol <*prot*>.
- *TTipTransport &owner( );* — Type of transport - the owner of outgoing transport.

Protected attributes:
- *bool run_st;* — The sign «Running».

7. Subsystem “Communication interfaces' protocols” (TProtocolS)

Subsystem "Communication interfaces' protocols" is presented by the TProtocolS object, which contains modular objects of the separate protocols TProtocol on the a subsystem's level. Each protocol contains objects of the opened sessions of the incoming protocols TProtocolIn.

TProtocolS object provides an access to the both: the incoming and the outgoing protocols of the individual types of transport protocols. The inner side of outgoing protocol based on the steaming principle with the individual structure of the stream for each implementation of the protocol.

7.1. The object of the “Communication interfaces' protocols” subsystem (TProtocolS)

Inherits:	*TSubSYS.*

Public methods:

- *int subVer( );* — Subsystem's version.
- *AutoHD<TProtocol> at(const string &id);* — Connection to the protocol's <*id*> module.
- *string optDescr( );* — Localized help for command-line options and parameters of the configuration file.

7.2. The modular object of the protocol (TProtocol)

Inherits:	*TModule.*
Inherited:	By the root object of the subsystem's "Protocols" modules.

Public methods:

- *virtual void itemListIn(vector<string> &ls, const string &curIt = "");* — Items list of protocol for individual addressing into transport.
- *void list(vector<string> &list);* — The list of open incoming sessions.
- *bool openStat(const string &name);* — Check for the openness of the incoming session with the specified name.
- *void open(const string &name, const string &tr);* — The opening of the session on behalf of the incoming transport <*tr*>.
- *void close(const string &name);* — Closing of the incoming session.
- *AutoHD<TProtocolIn> at(const string &name);* — Connecting to the open incoming sessions.
- *virtual void outMess(XMLNode &io, TTransportOut &tro);* — Sending the data in the XML tree <*in*> over the protocol and transport <*tro*>.

7.3. The object of the incoming protocol's session (TProtocolIn)

Inherits:	*TCntrNode.*
Inherited:	By the session objects of the incoming modules' protocol of the subsystem "Protocols".

Public methods:

- *TProtocolIn(const string &name);* — Initializing constructor.
- *string name( );* — The name of the incoming session.
- *const string &srcTr( );* — The address of the transport-source of the opening of the session of the incoming protocol.
- *void setSrcTr(const string &vl);* — Setting of the address of the transport-source of the opening of the session of the incoming protocol.
- *virtual bool mess(const string &request, string &answer, const string &sender);* — The transfer of unstructured data to their processing by the protocol.
- *TProtocol &owner( );* — The protocol - the owner of incoming sessions.

8. Subsystem “User interfaces” (TUIS)

The subsystem "User Interfaces" is presented by the TUIS object, which contains modular objects of the TUI user interfaces' on the subsystem's level.

8.1. The object of the “User interfaces” subsystem (TUIS)

Inherits:	*TSubSYS.*

Public methods:

- *int subVer( );* — Subsystem's version.
- *void subStart( );* — Start of the subsystem.
- *void subStop( );* — Stop of the subsystem.
- *AutoHD<TUI> at(const string &name);* — Connecting to the user's interface module.
- *static bool icoPresent(const string &inm, string *tp = NULL);* — Check for the availability of icon *inm*> in the standard directory. The name of the icon is indicate without extension. The extension/type of the loaded image is placed in <tp>.
- *static string icoGet(const string &inm, string *tp = NULL);* — Upload the image icon <*inm*> from the standard directory.
- *static string icoPath(const string &ico, const string &tp = "png");* — The full path to the icon, including the working directory.

8.2. The modular object of the user interface (TUI)

Inherits:	*TModule.*
Inherited:	By the root objects of the “User interfaces” subsystem.

Protected methods:

- *void cntrCmdProc(XMLNode *opt);* — Service of the commands of the control interface by the system.

Protected attributes:

- *bool run_st;* — The sign "Running".

9. Subsystem “Specials” (TSpecialS)

The subsystem "Specials" is presented by TSpecialS object, which contains modular objects of TSpecial special on the subsystem's level.

9.1. The object of the “Specials” subsystem (TSpecialS)

Inherits:	*TSubSYS.*

Public methods:

- *int subVer( );* — Subsystem's version.

9.2. The modular object of the specials (TSpecial)

Inherits:	*TModule.*
Inherited:	By the root objects of the subsystem's “Specials” modules.

Protected attributes:

- *bool run_st;* — The sign "Running".

10. Subsystem “Security” (TSecurity)

Security subsystem is presented by an object TSesurity, which contains group objects of TGroup and users TUser.

User object TUser contains user information and checks the authenticity of the user in accordance with the specified password.

TGroup user object contains information about the group of users and checks the user's belonging to the group.

10.1. The object of the "Security" subsystem. (TSecurity)

Inherits:	*TSubSYS.*

Public methods:

- *bool access(const string &user, char mode, int owner, int group, int access);* — Check the access of the user <*user*> with the mode rights <*mode*> to the resource with the owner <*owner*> and group <*access*>.
- *void usrList(vector<string> &list);* — The list of the users <*list*>.
- *void usrGrpList(const string &name, vector<string> &list);* — The list of the users' groups <*list*>, in which the user <*name*> is included.
- *bool usrPresent(const string &name);* — Check fro the presence of the specified user <*name*>.
- *int usrAdd(const string &name, const string &db = "*.*");* — Addition of the user <*name*> with the saving to the DB <*db*>.
- *void usrDel(const string &name, bool complete = false);* — Deleting of the user <*name*> with the possibility of the completely deleting <*complete*>.
- *AutoHD<TUser> usrAt(const string &name);* — Attaching of the user <*name*>.
- *void grpList(vector<string> &list);* — The list of the users' groups <*list*>.
- *bool grpPresent(const string &name);* — Check fro the presence of the specified users' group <*name*>.
- *int grpAdd(const string &name, const string &db = "*.*");* — Addition of the users' group <*name*> with the saving to the DB <*db*>.
- *void grpDel(const string &name, bool complete = false);* — Deleting of the users' group <*name*> with the possibility of the completely deleting <*complete*>.
- *AutoHD<TGroup> grpAt(const string &name);* — Connection to the users' group <*name*>.

10.2. The user's object (TUser)

Inherits:	*TCntrNode, TConfig.*

Public methods:

- *TUser(const string &name, const string &db, TElem *el);* — Initializing constructor.
- *string name( );* — The user's name.
- *string lName( );* — The full user's name.
- *string descr( );* — The user's description.
- *string picture( );* — The user's image.
- *bool sysItem( );* — The sign of the system user.
- *bool auth(const string &pass);* — Check the authenticity of the user by the password <*pass*>.
- *string DB( );* — User's DB address.
- *string tbl( );* — Address of the table of the user's DB.
- *string fullDB( );* — The full name of the table of the user's DB.
- *void setLName(const string &nm);* — Setting of the full user's name to <*nm*>.
- *void setPicture(const string &pct);* — Setting the user's image to <*pct*>.
- *void setPass(const string &pass);* — Setting the user's password to <*pass*>.
- *void setSysItem(bool vl);* — Setting the sign of the system user to <*vl*>.
- *void setDB(const string &vl);* — Setting the user's DB address.
- *TSecurity &owner( );* — "Security" subsystem - the owner of the user.

10.3. The users' group object (TGroup)

Inherits:	*TCntrNode, TConfig.*

Public methods:

- *TGroup(const string &name, const string &db, TElem *el);* — Initializing constructor.
- *string name( );* — The users' group name.
- *string lName( );* — The full users' group name.
- *string descr( );* — The group's description.
- *bool sysItem( );* — The sign of the system user.
- *string DB( );* — Users' group DB address.
- *string tbl( );* — Address of the table of the users' group DB.
- *string fullDB( );* — The full name of the table of the users' group DB.
- *void setLName(const string &nm);* — Setting of the full users' group name to <*nm*>.
- *void setSysItem(bool vl);* — Setting the sign of the system user to <*vl*>.
- *void setDB(const string &vl);* — Setting the users' group DB address.
- *bool user(const string &name);* — Check for the user's belonging to the group <*name*>.
- *void userAdd(const string &name);* — Addition of the user <*name*> to the group.
- *void userDel(const string &name);* — Deleting of the user <*name*> from the group.
- *TSecurity &owner( );* — "Security" subsystem - the owner of the users' group.

11. Subsystem “Modules' sheduling” (TModSchedul)

Subsystem “Modules' sheduling” is presented by the object TModSchedul.

The subsystem contains the control mechanism for modules contained in shared libraries.

11.1. The object of the subsystem “Modules' sheduling” (TModSchedul)

Inherits:	*TSubSYS.*

Data:

The structure of information about the shared library (struct – TModSchedul::SHD):

- *void *hd;* — the title of the shared library (if NULL, the library is present, but not connected);
- *vector<string> use;* — list of connected modules;
- *time_t tm;* — modification time of the library;
- *string name;* — full name/path of the shared library.

Public methods:

- *string allowList( );* — The list of permitted shared libraries (modules).
- *string denyList( );* — The list of prohibited shared libraries (modules).
- *int chkPer( );* — The verification period of the directory with the modules (s).
- *void setAllowList(const string &vl);* — Setting of the list of permitted shared libraries (modules).
- *void setDenyList(const string &vl);* — Setting of the list of prohibited shared libraries (modules).
- *void setChkPer(int per);* — Setting the verification period of the directory with the modules (s). If the frequency is equal to zero, the check will be disabled.
- *void subStart( );* — Subsystem's start.
- *void subStop( );* — Subsystem's stop.
- *int loadLibS( );* — Loading the shared libraries and initialization of the modules. Return numbers loaded modules.
- *SHD &lib(const string &name);* — Getting the shared library object <*name*>.
- *void libList(vector<string> &list);* — The list of shared libraries <*list*>.
- *int libLoad(const string &path, bool full);* — Loading the shared libraries from the specified path <*path*>. Return numbers loaded modules.
- *void libAtt(const string &name, bool full = false);* — Attaching of the specified shared library <*name*>.
- *void libDet(const string &name);* — Detaching of the specified shared library <*name*>.

12. Components of the object model of the OpenSCADA system

Object model of the OpenSCADA system is based on the function object *TFunction*, on the parameters of the function *IO* and on the values' frame of the function *TValFunc*. Later the function's objects are included in the object tree, forming an object model of the system. Using the functions of the object model is made by linking the frame of values *TValFunc* with function.

The idea in general is shown in Fig. 7.

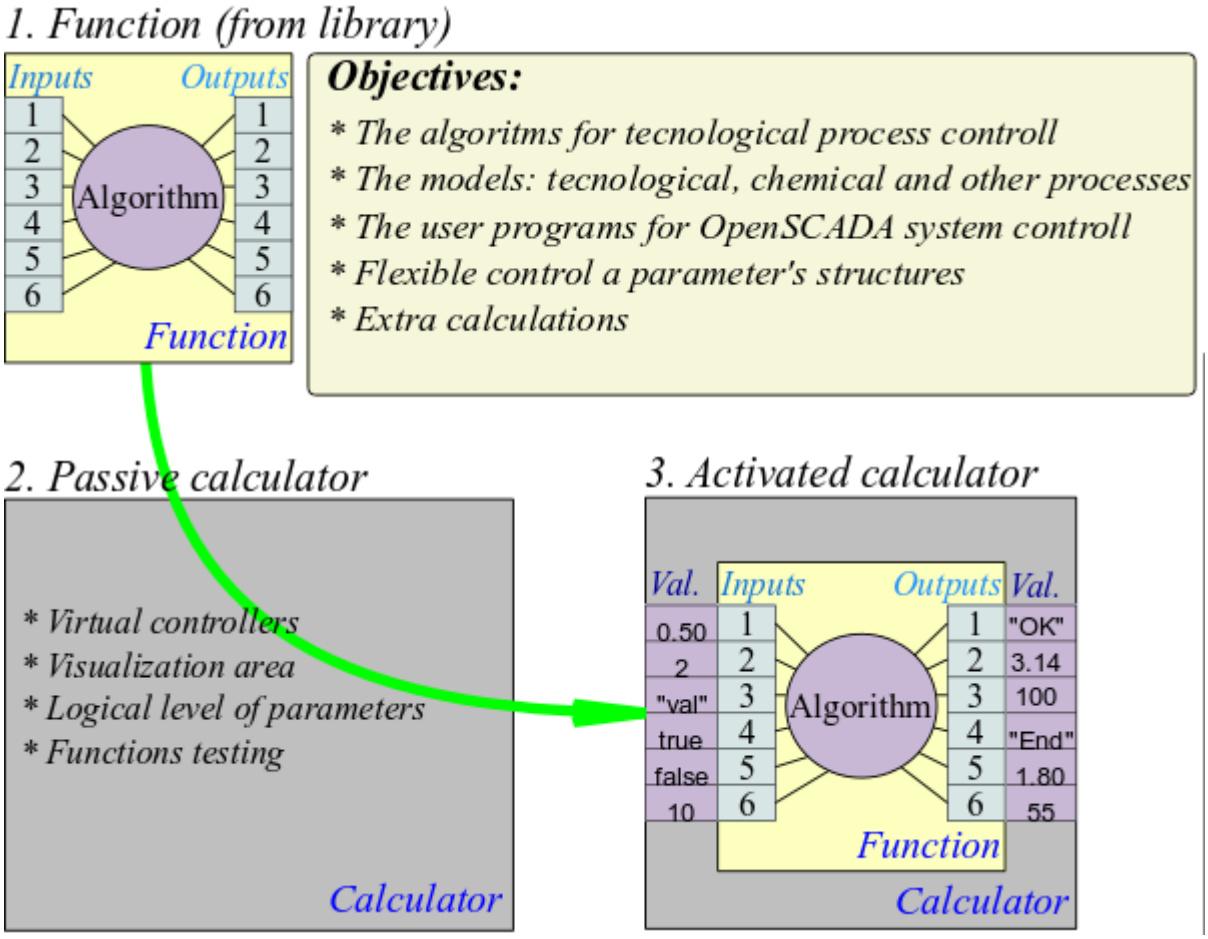

Fig. 7. The basis of the programming area of the OpenSCADA system.

Object of the function (TFunction) provides an interface for the formation of function parameters and algorithm of the computing in the object which inherits it.

Object of the function's parameter (IO) contains the configuration of the single parameter.

Object of the values' frame (TValFunc) contains values in accordance with the structure of the linked function. While the execution of the algorithm by the associated function the values of this object are used.

12.1. The function object (TFunction)

Inherits:	*TCntrNode*
Inherited:	By the modules and nodes of the systems, which contains the functions for publication in the object model of the system.

Public methods:

- *TFunction(const string &iid);* — Initializing constructor of the function with the identifier <*id*>.
- *TFunction &operator=(TFunction &func);* — Copying of functions.
- *string id( );* — Function ID.
- *virtual string name( );* — Localized name of the function.
- *virtual string descr( );* — Description of the function.
- *bool startStat( );* — Status - "Running".
- *int use( );* — Counter of the function's using.

- *virtual void setStart(bool val);* — Start/stop <*val*> of the function.
- *void ioList(vector<string> &list);* — The list of the function's parameters <*list*>.
- *int ioId(const string &id);* — Getting the index of the function's parameter <*id*>.
- *int ioSize( );* — The quantity of the function's parameters.
- *IO *io(int id);* — Getting the function's parameter by the index <*id*>.
- *void ioAdd(IO *io);* — Addition of the function's parameter <*io*>.
- *int ioIns(IO *io, int pos);* — Insert of the function's parameter <*io*> into the position <*pos*>. Returns the actual position of the new parameter.
- *void ioDel(int pos);* — Deleting the function's parameter in the position <*pos*>.
- *void ioMove(int pos, int to);* — Moving the function's parameter from the position <*pos*> to the position *to*>.
- *virtual void calc(TValFunc *val);* — The calculation of the function's algorithm of the specified values <*val*>.
- *void valAtt(TValFunc *vfnc);* — Is called by the object of the values' frame <*vfnc*> in the case of attaching with the function.
- *void valDet(TValFunc *vfnc);* — Is called by the object of the values' frame <*vfnc*> in the case of detaching from the function.
- *virtual void preIOCfgChange( );* — Is called before the changing of the configuration of the function.
- *virtual void postIOCfgChange( );* — Is called after the changing of the configuration of the function.

Protected attributes:
- *string mId;* — Function ID.
- *bool run_st;* — Sign - "Running".
- *TValFunc *mTVal;* — Reference to the values' object, used for testing the function. May be absent.

12.2. The object of the function's parameter (IO)

Data:

Parameter's type (enum — IO::Type):
- *IO::String* — string/text;
- *IO::Integer* — integer;
- *IO::Real* — real;
- *IO::Boolean* — boolean;
- *IO::Object* — object.

Parameter's flags (enum — IO::IOFlgs):
- *IO::Default* — default mode (input);
- *IO::Output* — output;
- *IO::Return* — return.

Public methods:
- *IO(const char *id, const char *name, IO::Type type, unsigned flgs, const char *def = "", bool hide = false, const char *rez = "");* — Initializing constructor.
- *IO &operator=(IO &iio);* — Copying of the function parameters.
- *const string &id( );* — ID of the function's parameter.
- *const string &name( );* — The localized name of the function's parameter.
- *const Type &type( );* — The type of the function's parameter.
- *unsigned flg( );* — The flags of the function's parameter.
- *const string &def( );* — Default value.
- *bool hide( );* — Sign - "Hide".
- *const string &rez( );* — The reserve property of the function's parameter.
- *void setId(const string &val);* — Set the identifier to <*val*>.
- *void setName(const string &val);* — Set the name to <*val*>.
- *void setType(Type val);* — Set the type to <*val*>.
- *void setFlg(unsigned val);* — Set the flags to <*val*>.
- *void setDef(const string &val);* — Set the default value to <*val*>.
- *void setHide(bool val);* — Set/unset the sign - "Hide".

- *void setRez(const string &val);* — Set the reserve property to <*val*>.

12.3. The object of the function's value (TValFunc).

Public methods:

- *TValFunc(const string &iname = "", TFunction *ifunc = NULL, bool iblk = true);* — Initializing constructor.
- *string user( );* — User on whose behalf the function is executed.
- *const string &vfName( );* — Name of the values' object.
- *bool blk( );* — The sign - "Blocking of the parameters' changes of the function".
- *bool dimens( );* — The sign - "To measure the computation time".
- *bool mdfChk( );* — The sign - "To check attributes to modify".
- *void setUser(const string &iuser);* — Set the user on whose behalf the function will be executed.
- *void setVfName(const string &nm);* — Set the name of the values' object to <*nm*>.
- *void setDimens(bool set)* — Set/unset <*set*> of the sign - "To measure the computation time".
- *void setMdfChk(bool set);* — Set/unset <*set*> of the sign - "To check attributes to modify".
- *void ioList(vector<string> &list);* — The list of the function's parameters <*list*>.
- *int ioId(const string &id);* — Getting the parameter's index <*id*>.
- *int ioSize( );* — The whole quantity of the parameters.
- *IO::Type ioType(unsigned id);* — Parameter's type <*id*>.
- *unsigned ioFlg(unsigned id);* — Parameter's flags <*id*>.
- *bool ioHide(unsigned id);* — Parameter's sign <*id*> - "Hide".
- *bool ioMdf(unsigned id);* — Parameter's sign <*id*> - "Modified".
- *TVariant get(unsigned id);* — Get the value of the parameter <*id*>.
- *string getS(unsigned id);* — Get the value (string) of the parameter <*id*>.
- *int getI(unsigned id);* — Get the value (integer) of the parameter <*id*>.
- *double getR(unsigned id);* — Get the value (real) of the parameter <*id*>.
- *bool getB(unsigned id);* — Get the value (boolean) of the parameter <*id*>.
- *AutoHD<TVarObj> getO(unsigned id);* — Get the value of the parameter's object <*id*>.
- *void set(unsigned id, const TVariant &val); — Set the value <val> of the parameter <id>.*
- *void setS(unsigned id, const string &val);* — Set the value <*val*> (string) of the parameter <*id*>.
- *void setI(unsigned id, int val);* — Set the value <*val*> (integer) of the parameter <*id*>.
- *void setR(unsigned id, double val);* — Set the value <*val*> (real) of the parameter <*id*>.
- *void setB(unsigned id, bool val);* — Set the value <*val*> (boolean) of the parameter <*id*>.
- *void setO(unsigned id, AutoHD<TVarObj> val);* — Set the value <*val*> of the parameter's object <*id*>.
- *virtual void calc(const string &user = "");* — To calculate on behalf of the user <*user*> or user, specified earlier.
- *double calcTm( );* — Computation time.
- *void setCalcTm(double vl);* — Initializing of the time of value's calculation <*vl*>.
- *TFunction *func( );* — Attached function.
- *void setFunc(TFunction *func, bool att_det = true);* — To attach with the specified function <*func*>.
- *virtual void preIOCfgChange( );* — Is called before the configuration changes.
- *virtual void postIOCfgChange( );* — Is called after the configuration changes.
- *TValFunc *ctxGet(int key);* — Get context for key <*key*>.
- *void ctxSet(int key, TValFunc *val);* — Set context for key <*key*>.
- *void ctxClear( );* — External functions call context clear.

13. Data in the OpenSCADA system and their storage in the DB (TConfig)

Storing data in the system based on the objects *TConfig* and *TElem*. These objects store the structure and fields' values of the database, allowing for direct loading and saving the configuration via the "DB" subsystem. For the specialized different types data storage the TVariant object is provided.

TElem object contains the structure of database record. Structure of the record contains extensive information about the elements, their types, sizes and other parameters. Information in this structure is enough to create, control and manage the real structure of the database. Elementary unit of the record is the cell *Tfld*.

TSonfig object is the heir of *TElem* and contains the actual values of elements. *TConfig* is used as the parameter in the functions of the manipulating with the table's records in the "DB" subsystem. Elementary unit of the record is the cell *TCfg*.

To provide an opportunity to inform the data storehouse about the changes in the structure it is provides an object *TValElem*, from which it is inherited the storehouse *TConfig* and the list of which is contained in the *TElem* structure.

13.1. Data object (TConfig)

Inherits:	*TValElem*
Inherited:	*TParamContr, TController, TMArchivator, TPrmTempl, TPrmTmplLib, TUser, TGroup, TTransportIn, TTransportOut, TBD, TVArchive, TVArchivator, а также модульные объекты хранящие свои данные в БД.*

Public methods:

- *TConfig(TElem *Elements = NULL);* — Initializing constructor.
- *TConfig &operator=(TConfig &cfg);* — Copying from <*cfg*>.
- *void cfgList(vector<string> &list);* — Elements' list <*list*>.
- *bool cfgPresent(const string &n_val);* — Check for the element's presence <*n_val*>.
- *TCfg &cfg(const string &n_val);* — Getting of th element <*n_val*>.
- *TCfg *at(const string &n_val, bool noExpt = false);* — Getting of the element's pointer <*n_val*>. If the element is absent an exception is generated or the null pointer is returned when setting <*noExpt*>.
- *void cfgViewAll(bool val = true);* — Set/unset the sign of the visibility for all the elements.
- *void cfgKeyUseAll(bool val);* — Set/unset the sign of the using the key for all the elements.
- *TElem &elem( );* — The using structure.
- *void setElem(TElem *Elements, bool first = false);* — Setting the structure in <*Elements*>.
- *void cntrCmdMake(XMLNode *fld, const string &path, int pos, const string &user = "root", const string &grp = "root", int perm = 0664);* — Formation of the information description of elements of the configuration for the management interface of OpenSCADA.
- *void cntrCmdProc(XMLNode *fld, const string &elem, const string &user = "root", const string &grp = "root", int perm = 0664);* — Requests' of the OpenSCADA management interface to the elements of the configuration processing.
- *bool noTransl( );* — The sign: "Do not broadcast messages when working with the database".
- *void setNoTransl(bool vl);* — Setting the sign of the broadcasting the messages.
- *TVariant objFunc(const string &id, vector<TVariant> &prms, const string &user);* — The object's functions for access to configuration.

Protected methods:

- *virtual bool cfgChange(TCfg &cfg);* — Is called in the case of changing the contents of the configuration element.

13.2. Data cell (TCfg)

Inherits:	*TVariant*

Data:

Additional flags to *TFld* (enum — TCfg::AttrFlg):

- *TCfg::TransltText* — Translate the text variables of the record.
- *TCfg::NoVal* — Do not reflect an element on the value of the TValue object.
- *TCfg::Key* — key field.
- *TCfg::Hide* — attribute is hidden.

Requests flags (enum — ReqFlg):

- *TFld::ForceUse* — Forcing the setting of the flag of the element's using during the setting of its value.

Public methods:

- *TCfg(TFld &fld, TConfig &owner);* — Initializing constructor.
- *const string &name( );* — Cell name.
- *bool operator==(TCfg &cfg);* — Comparison of the cells.
- *TCfg &operator=(TCfg &cfg);* — Copying of the cells.
- *bool view( );* — The sign - "Cell is visible".
- *bool keyUse( );* — The sign - "Use the key" for the dataSeek() and dataDel() requests.
- *bool noTransl( );* — The sign "No translation" is provided to disable the translation of text variables for the record at the time of a request.
- *void setView(bool vw);* — Setting of the sign "The cell is visible" в <*vw*>.
- *void setKeyUse(bool vl);* — Setting of the sign "Use the key" in <*vl*>.
- *void setNoTransl(bool vl);* — Setting of the sign "No translation".
- *TFld &fld( );* — Configuration of the cell.
- *string getSEL( );* — Get the value of the selective type.
- *string getS( );* — Get the value of the string type.
- *const char *getSd( );* — Get the direct access to the string type value.
- *double &getRd( );* — Get the direct access to the real type value.
- *int &getId( );* — Get the direct access to the integer type value.
- *char &getBd( );* — Get the direct access to the boolean type value.
- *void setSEL(const string &val, char RqFlg = 0);* — Set the value of the selective type to the <*val*> with the request flags <*RqFlg*>.
- *void setS(const string &val); TCfg &operator=(const string &vl); TCfg &operator=(const char *vl); void setS(const string &val, char RqFlg);* — Set the value of the string type to the <*val*> with the request flags <*RqFlg*>.
- *void setR(double val); TCfg &operator=(double vl); void setR(double val, char RqFlg);* — Set the value of the real type to the <*val*> with the request flags <*RqFlg*>.
- *void setI(int val); TCfg &operator=(int vl); void setI(int val, char RqFlg);* — Set the value of the integer type to the <*val*> with the request flags <*RqFlg*>.
- *void setB(char val); TCfg &operator=(bool vl); void setB(char val, char RqFlg);* — Set the value of the boolean type to the <*val*> with the request flags <*RqFlg*>.

13.3. Data structure object (TElem)

Inherited:	By the *TTipParam, TControllerS, TTipController*, as well as by the modular objects, combining the functions of the structure storage .

Public methods:

- *TElem(const string &name = "");* — Initialization of the structure with the specified name <*name*>.
- *string &elName( );* — The name if the structure.
- *void fldList(vector<string> &list);* — The cell's list in the structure <*list*>.
- *unsigned fldSize( );* — The number of cells in the structure.
- *unsigned fldId(const string &name, bool noex = false);* — Getting the index of the cell from its identifier <*name*>. Set <*noex*> on no the cell present caused return overall cells number, without exception generation.
- *bool fldPresent(const string &name);* — Check for the presence of the specified cell <*name*>.
- *int fldAdd(TFld *fld, int id = -1);* — Addition/insert of the cell <*fld*> to the position <*id*> (-1 — insert to the end).
- *void fldDel(unsigned int id);* — Deleting of the cell <*id*>.
- *TFld &fldAt(unsigned int id);* — Getting of the cell <*id*>.
- *void valAtt(TValElem *cnt);* — Is called automatically when the structure is attached to the data storage <*cnt*>.
- *void valDet(TValElem *cnt);* — Is called automatically when the structure is detached from the data storage <*cnt*>.

13.4. Data structure cell (TFld)

Data:

Cell's type (enum – TFld::Type):

- *TFld::Boolean*(0) — boolean type;
- *TFld::Integer*(1) — integer type;
- *TFld::Real*(4) — real type;
- *TFld::String*(5) — string type.

Cell's flags (enum — TFld::AttrFlg):

- *TFld::NoFlag*(0x00) — No flags;
- *TFld::Selected*(0x01) — selection mode from the available values, the selective type;
- *TFld::SelfFld*(0x02) — To create the own copy of this cell;
- *TFld::NoWrite*(0x04) — unwritable;
- *TFld::HexDec*(0x08) — integer type: hexadecimal notation;
- *TFld::OctDec*(0x10) — integer type: octal notation;
- *TFld::DateTimeDec*(0x20) — integer type: содержит дату в UTC;
- *TFld::FullText*(0x08) — Full-text, multi-string mode of the text type;
- *TFld::NoStrTransl*(0x10) — no string variable translation set.

Public methods:

- *TFld( );* — Initialization by default.
- *TFld(TFld &ifld);* — Copying constructor.
- *TFld(const char *name, const char *descr, Type type, unsigned char flg, const char *valLen = "", const char *valDef = "", const char *vals = "", const char *nSel = "", const char *res = "");* — Initialization with the specified configuration.
- *TFld &operator=(TFld &fld);* — Copy of the cell from <*fld*>.
- *const string &name( );* — Cell's name.
- *const string &descr( );* — Cell's description.
- *int len( );* — The size of the cell's value (symbols in symbolic representation).
- *int dec( );* — The size of the fractional part of a real (symbols in symbolic representation).
- *Type type( );* — Cell's type.
- *static Type type(IO::Type tp);* — Cell's type from IO::Type type.
- *IO::Type typeIO( );* — Cell's type in IO::Type.

• *unsigned flg( );* — Cell's flag.
• *const string &def( );* — Default value.
• *string values( );* — Working range of the value or a list of possible values for the selective type (as — "vl1;vl2;vl3").
• *string selNames( );* — List of names of values for the selective type (as — "Value 1;Value 2;Value 3").
• *const string &reserve( );* — Reserve parameter.
• *void setDescr(const string &dscr);* — Set the descrition to the <*dscr*>.
• *void setLen(int vl);* — Set the cell's size to the <*vl*>.
• *void setDec(int vl);* — Set the fraction part of the real to the <*vl*>.
• *void setDef(const string &def);* — Set the default value to the <*def*>.
• *void setFlg(unsigned flg);* — Set the flags to <*flg*>.
• *void setValues(const string &vls);* — Set the working range of value or the list of possible values for the selective type (as — "vl1;vl2;vl3") to the <*vls*>.
• *void setSelNames(const string &slnms);* — Set the list of values' names for the selective type (as — "Value 1;Value 2;Value 3") to the <*slnms*>.
• *void setReserve(const string &ires);* — Set of the reserve parameter to <*res*>.
• *const vector<string> &selValS();* — List of values' variants for the string type.
• *const vector<int> &selValI();* — List of values' variants for the integer type.
• *const vector<double> &selValR();* — List of values' variants for the real type
• *const vector<bool> &selValB();* — List of values' variants for the boolean type
• *const vector<string> &selNm();* — List of the names of values' variants.
• *string selVl2Nm(const string &val);* — Get the selected name from the value <*val*> of the string type.
• *string selVl2Nm(int val);* — Get the selected name from the value <*val*> of the integer type.
• *string selVl2Nm(double val);* — Get the selected name from the value <*val*> of the real type.
• *string selVl2Nm(bool val);* — Get the selected name from the value <*val*>of the boolean type.
• *string selNm2VlS(const string &name);* — Get the value of string type from the selected name <*name*>.
• *int selNm2VlI(const string &name);* — Get the value of integer type from the selected name <*name*>.
• *double selNm2VlR(const string &name);* — Get the value of real type from the selected name <*name*>.
• *bool selNm2VlB(const string &name);* — Get the value of boolean type from the selected name <*name*>.
• *XMLNode *cntrCmdMake(XMLNode *opt, const string &path, int pos, const string &user = "root", const string &grp = "root", int perm = 0664);* — Create a form element in accordance with the parameters of the cell.

13.5. The object which preacts about changing of the structure (TValElem)

Inherited:	*TValue, TConfig.*

Protected methods:
• *virtual void detElem(TElem *el);* — Notification by the element <*el*> of the container about it's wish to detach.
• *virtual void addFld(TElem *el, unsigned id) = 0;* — Notification about the addition of the cell <*id*> of the element <*el*>.
• *virtual void delFld(TElem *el, unsigned id) = 0;* — Notification about the deleting of the cell <*id*> of the element <*el*>.

13.6. Data cell (TVariant)

Data:

Error values for the different data types (define):

- *EVAL_BOOL* — Error value of the boolean type (2);
- *EVAL_INT* — Error value of the integer type (-2147483647);
- *EVAL_REAL* — Error value of the real type (-3.3E308);
- *EVAL_STR* — Error value of the string type ("<EVAL>").

Типы данных (enum — TVariant::Type):

- *TVariant::Null* — data type and data is not set.
- *TVariant::Boolean* — boolean type (boolean, 8бит).
- *TVariant::Integer* — integer type (integer, 32бит).
- *TVariant::Real* — real type (double).
- *TVariant::String* — string.
- *TVariant::Object* — object.

Public methods:

- *TVariant( );* — Default constructor.
- *TVariant(char ivl);* — Boolean type constructor.
- *TVariant(int ivl);* — Integer type constructor.
- *TVariant(double ivl);* — Real type constructor.
- *TVariant(string ivl);* — String type constructor.
- *TVariant(const char *var);* — String type constructor.
- *TVariant(AutoHD<TVarObj> ivl); TVariant(TVarObj *ivl);* — Object constructor.
- *TVariant(const TVariant &var);* — Copying constructor.
- *bool operator==(const TVariant &vr);* — Check the object for equality.
- *bool operator!=(const TVariant &vr);* — Check the object for not equality.
- *TVariant &operator=(const TVariant &vr);* — Copy of the object.
- *bool isNull( ) const;* — Signs that the object is not initiated.
- *bool isEVal( ) const;* — Value into the object is error.
- *Type type( ) const;* — Value's type.
- *void setType(Type tp);* — Type set.
- *bool isModify( );* — Modification flag. It serves in the object functions to indicate the modification of variables.
- *void setModify(bool vl = true);* — Set of the modification flag.
- *virtual char getB( ) const; operator char( );* — Getting the value as the boolean one.
- *virtual int getI( ) const; operator int( );* — Getting the value as the integer one.
- *virtual double getR( ) const; operator double( );* — Getting the value as the real one.
- *virtual string getS( ) const; operator string( );* — Getting the value as the string one.
- *virtual AutoHD<TVarObj> getO(bool noex = false) const; operator AutoHD<TVarObj>( );* — Getting the object. Set <*noex*> for disable exception generation on NULL object get.
- *virtual void setB(char val);* — Set to the value the boolean one.
- *virtual void setI(int val);* — Set to the value the integer one.
- *virtual void setR(double val);* — Set to the value the real one.
- *virtual void setS(const string &val);* — Set to the value the string one.
- *virtual void setO(AutoHD<TVarObj> val);* — Set the object.

13.7. User object (TVarObj)

Inherited:	*TArrayObj*

Public methods:

- *TVarObj( );* — Constructor.
- *int connect( );* — Connection to the object.
- *int disconnect( );* — Disconnection from the object.
- *virtual void propList(vector<string> &ls);* — Object's properties list <*ls*>.
- *virtual TVariant propGet(const string &id);* — Request of the object property with an ID <*id*>.
- *virtual void propSet(const string &id, TVariant val);* — Set the object property with an ID <*id*> to the value <*val*>.
- *virtual string getStrXML(const string &oid = "");* — Conversion of the contents of the object in the XML flow.
- *static AutoHD<TVarObj> parseStrXML(const string &str, XMLNode *nd = NULL, AutoHD<TVarObj> prev = NULL);* — Backward serialization for XML-stream to object.
- *virtual TVariant funcCall(const string &id, vector<TVariant> &prms);* — Call of the object function with the ID <*id*> and with the parameters <*prms*>.

14. The control interface of the system and the dynamic tree of the system objects (TCntrNode)

For complete coverage of the key components of the system by the network of the same structure objects the object of the dynamic tree node TCntrNode is provided. This objects provides the following functions:

- providing of the uniform access to system components, including the blocks of the dynamic access;
- building the distributed control interface.

Any object that has the need to provide dynamic access to itself or its components must be inherited from the object of the dynamic tree node TCntrNode. This relationship is automatically includes the node into the dynamic tree of objects covered by both: the direct and the reverse links, and also provides an opportunity to create containers for its own child nodes. In addition to this node is able to preact the inclusion and exclusion/remove of the node from the tree with the possibility of reject from the exclusion/removal.

The control interface of the system is included into the TCntrNode object and, accordingly, covers all the nodes of the dynamic tree of the system, allowing you to consistently manage the system regardless of the client tool used. The control interface of the system is based on the markup language XML. You can think of many ways to use control interface of system, as an example, we'll note the following most meaningful solutions:

- Web interface of the configuration ;
- GUI configuration interface (QT, GTK+, ...);
- reflection of the configuration in the network for distributed management of multiple OpenSCADA-stations from the single management area;
- using as the protocol for objects' data access from the network;
- provision of service functions to access of the exterior application and the single OpenSCADA components to the internal data.

The control interface of the system is implemented by the following components:

- information structure of the user configuration page;
- dynamic part in the form of requests for getting, modification of data and service requests;
- container or group of the above elements.

Information hierarchical structure contains information about public control elements and can be used to build custom dialogs for the node's control of the system.

The dynamic part contains the scripts of requests' to the control elements service which are described in the information structure; as well as hidden control elements in the form of service functions used for unified access to the node.

The container can also gather into the single request several information structures and dynamic parts, thereby optimizing the time of the request especially on the high latency network interfaces.

Overall control interface is built from individual components of the dynamic tree. Hierarchical inheritance from the TCntrNode object allows to realize multi-level addition of the configuration of the control interface. General view of the dynamic node tree is shown in Fig. 8.

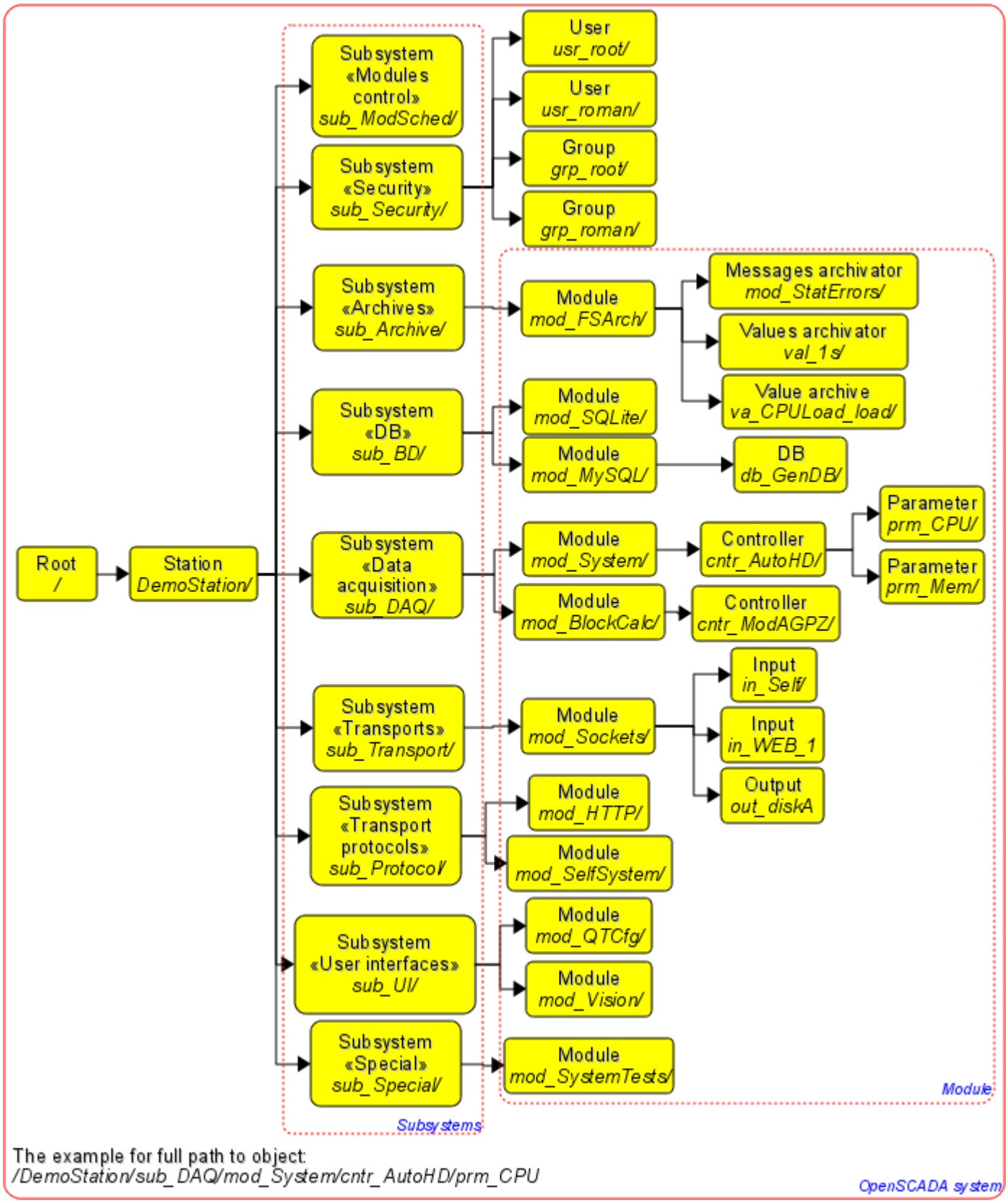

Fig. 8. Example of dynamic tree of the node in the OpenSCADA system.

System's nodes, containing data for the control interface of the system must also be connected to the dynamic tree of objects.

Connection of the node to the dynamic tree is done as follows:

- inheritance of the *TCntrNode* object or its child;
- information structure formation;
- service of the requests to dynamic data.

14.1. The syntax of the request and response of the control interface

Whole exchange with the control interface is done through the XML language. The internal exchange is done through the parsed structure of XML language (DOM), and external — through the transformation in the stream the characters of continuous XML-file and vice versa.

Request is executed by sending a container with some parameters in the attributes. The result is placed in the resulting container with the changes in certain attributes of the root container. In general, the request container can be written as follows: *<cmd path="/TreePath" user="user" force="1"/>* Where:

cmd — request command;
path — path to the node or to the branch of the tree;
user — user of the system on behalf of which the request is sent;
force — the sign: to execute the request without warning.

As the confirmation of the request's result it is set the result's attribute <rez> in values: 0-request succeeds, 1-warning (with the possibility of execution), 2-error. In the case of error and warning the messages are written in the text of the container and in the attribute *mcat* (message category): *<cmd path = "/TreePath" user="user" force="1" rez="2" mcat="sub_DAQ/mod_BlockCalc>; Unable to delete the node </cmd>*

Grouping request <CntrReqs> is processed at the API level of the node and does not require separate processing in the user code. In fact, into the tag <CntrReqs> may be placed any other requests with the possibility of hierarchical grouping by including of the internal tags <CntrReqs>. The only attribute of this tag is an attribute *path*, which indicates the path to the node and is the basis for internal requests.

```
<CntrReqs path="/sub_DAQ/cntr_gate">
      <get path="/%2fprm%2fcfg%2fNAME"/>
      <get path="/%2fprm%2fcfg%2fDESCR"/>
      <list path="/%2fserv%2fattr"/>
</CntrReqs>
```

14.2. Tag of the information structure for describing the groups of child branches of the page

Each page can contain groups of child branches. For descriptions of groups of branches there is the tag *<branches>*. Tag contains the description of groups of branches through embedded tags *<grp>*. The group tag can be accessed as for the "read" (visibility) and for modification (the execution of commands for adding and removing the elements of the group), hence the element of the triad of access may be:

00 — there is no any access;
04 — there is an access only for reading;
02 — there is only access fro the writing, usually this value has no meaning because the write access means access for reading;
06 — there is an access to read, and to write.

```
<branches id='br'>
      <grp id='/br/in_' descr='Incoming transport' acs='04'/>
      <grp id='/br/out_' descr='Outgoing transport' acs='04'/>
</branches>
```

Actions over the group of elements coincide with the actions over the list of visual elements "list", which is described below.

14.3. Tags of the description of the information structure of the control interface

Information tags for the XML form the alphabet of the formation of the description the configuration dialogs. Request command of the information part is: *<info path="/TreePath" user="user"/>*

As the result of the request it will be received the information structure of the page in accordance with the privileges of the specified user.

14.3.1. Area tag <area>

Areas are described with <area> tag and are intended to group the elements on different features. Area may include other elements and areas. The root ones form the bookmarks in the presentation of the user interface. To tag, you can access only to <read> or visibility, hence the element of the triad of access can be set to 00, if access is absent, or 04, if it is present.

```
<area id="base" dscr="Base information">
      <fld id="host" dscr="Host name" tp="str"/>
      <fld id="user" dscr="Operated user" tp="str"/>
      <fld id="sys" dscr="Station system" tp="str"/>
      <area id="other" dscr="Other options">
            <fld id="val" dscr="Value" tp="real"/>
      </area>
</area>
```

14.3.2. Data tags

Tags, which describe data, are shown in the table 1.

Table 1. Tags, which describe data

Tag	Description
<fld>	The simplest data of the string, integer, real or boolean types.
<list>	Lists with data of string, real and boolean types.
<table>	Tables with the data in cells of string, integer, real, boolean types.
<img>	Images.

a) Tag <fld>

```
<fld id="host" dscr="Host name" tp="str"/>
<fld id="user" dscr="Operated user" tp="str"/>
<fld id="sys" dscr="Station system" tp="str"/>
```

The tag can be accessed as to "read" and also to "write", therefore an element of the triad of access may be:

00 — there is no any access;
04 — there is an access only for reading;
02 — there is only access fro the writing, usually this value has no meaning because the write access means access for reading;
06 — there is an access to read, and to write.

Type of the element, discribed by the <fld> tag, is specified by the <tp> attribute (table 2).

Table 2. Attribute's <tp> values of the <fld> tag.

Tag <tp>	Description
str	String type. *<fld id="host" dscr="Host name" tp="str"/>*
dec	Integer in decimal representation. *<fld id="debug" dscr="Debug level" tp="dec"/>*
oct	Integer in octal representation. *<fld id="cr_file_perm" dscr="Make files permissions(default 0644)" tp="oct" len="3"/>*
hex	Integer in hexadecimal representation.
real	Real value.
bool	Boolean sign ("false"|"true"). *<fld id="log_sysl" dscr="Direct messages to syslog" tp="bool"/>*
time	Time in seconds (from 01/01/1970). *<fld id="v_beg" dscr="Start time" tp="time"/>*

Table 3. Actions over the element, described by the <fld> tag.

Operation	**Action**
Interrogation	*Request:* command «get»: *<get path="/fld_teg" user="user"/>*. *Result:* confirmation with the value in the text of the tag or an error message.
Modification	*Request:* command "set": *<set path="/fld_teg" user="user">value</set>* *Result:* confirmation or an error message.

Request for the rules of syntax highlight, for text fields (the attribute "rows" set).	*Request:* command "SnthHgl": *<SnthHgl path="/fld_teg" user="user"/>*
	Result: confirmation whith the rules of syntax highlight list:
	`<rule expr="\b(if|else|for|while|in|using|new|var|break|continue|return|Array|Object)\b" color="darkblue" font_weight="1"/>`
	`<rule expr="(\?|\:)" color="darkblue" font_weight="1"/>`
	`<rule expr="(\b0[xX][0-9a-fA-F]*\b|\b[+-]?[0-9]*\.?[0-9]+[eE]?[-+]?[0-9]*\b|\btrue\b|\bfalse\b)" color="blue"/>`
	`<rule expr="&quot;[^&quot;]*&quot;" color="darkgreen"/>`
	`<rule expr="//[^\n]*" color="gray" font_italic="1"/>`
	`<blk beg="/\\*" end="\\*/" color="gray" font_italic="1"/>`

b) Tag <list>

```
<list id="mod_auto" dscr="List of shared libs(modules)" tp="str" dest="file"/>
```

The tag can be accessed as to "read" and also to "write"(modification), therefore an element of the triad of access may be:

00 — there is no any access;
04 — there is an access only for reading;
02 — there is only access fro the writing, usually this value has no meaning because the write access means access for reading;
06 — there is an access to read, and to write.

Type of the elements in the list is specified by the attribute *<tp>*. *<tp>* attribute's values are given in Table 1.

Table 4. Actions over the list.

Operation	**Action**

Interrogation	*Request:* command «get»: *<get path="/fld_teg" user="user"/>*
	Result: confirmation with the result in the tag's text or an error message. The result is formed in the form:
	`<get path="/fld_teg" user="user" rez="0">`
	`<el id="0">./MODULES/arh_base.o</el>`
	`<el id="1">./MODULES/cntr_sys.o</el>`
	`</get>`

String's addition	*Request:* command "add": *<add path="/fld_teg" user="user" id="tst">Test</add>* To add the string with ID "tst" and value "Test". If the list is not indexed, then the id attribute is absent. *Result:* confirmation or an error message.
Insert the string	*Request:* command "ins": *<ins path="/fld_teg" user="user" pos="3" p_id="tst1" id="tst">Test</ins>* To insert the string with the ID "tst" and value "Test" in position 3 with the string "tst1". In the case of the index list p_id attribute contains the identifier, else — the text of the string. If the list is not indexed, then the ID attribute is absent. *Result:* confirmation or an error message.
Deleting of the string	*Request:* command "del": *<del path="/fld_teg" user="user" pos="3" id="tst">Test</del>* To delete the string with ID "tst" and value "Test" from the position 3. If the list is not indexed, then the ID attribute is absent. *Result:* confirmation or an error message.

Operation	Action
Edit of the string	*Request::* command "edit": *<edit path="/fld_teg" user="user" pos="3" p_id="tst1" id="tst" >Test</edit>* To replace the string at position 3 with the identifier "tst1" with the another string with the identifier "tst" and value "Test". In the case of the index list p_id attribute contains the identifier, else — the text of the string. If the list is not indexed, then the ID attribute is absent. *Result:* confirmation or an error message.
Mooving of the string	*Request:* command "move": *<move path="/fld_teg" user="user" pos="3" to="5"/>* To move the string from position 3 to position 5. *Result:* confirmation or an error message.

c) Tag <table>

```
<table id="a_mess" key="0" col_lst="0;1;2">
      <list id="0" dscr="Id" acs="4" tp="str"/>
      <list id="1" dscr="Name" acs="4" tp="str"/>
      <list id="2" dscr="Type" acs="4" tp="str"/>
      <list id="3" dscr="Hide" acs="4" tp="bool"/>
</table>
```

The tag of the table or single column can be accessed as to "read" and also to "write"(modification), therefore an element of the triad of access may be:

00 — there is no any access;

04 — there is an access only for reading;

02 — there is only access fro the writing, usually this value has no meaning because the write access means access for reading;

06 — there is an access to read, and to write.

If the attribute *<key>* is specified and it lists the key columns, the work with the table moves to the addressing mode by the identifiers for columns and keys.

Table 5. Actions over the table.

Operation	Action

Interrogation	*Request:* command "get": *<get path="/fld_teg" user="user" cols="0;2" rows="100;1000"/>*
	To get columns 0-2 and rows of them from 100 to 1000 table.
	Result: Confirmation with the data of the table or an error message. The result is formed in the form:
	`<get path="/fld_teg" user="user" cols="0;2" rows="100;1000" rez="0">`
	`<list id="0" tp="str">`
	`<el id="100">Sat Feb 21 18:04:16 2004</el>`
	`</list>`
	`<list id="1" tp="str">`
	`<el id="100">SYS</el>`
	`</list>`
	`<list id="2" tp="str">`
	`<el id="100">*:(TSYS)Broken PIPE signal allow!</el>`
	`</list>`
	`</get>`

Adding of the string	*Request:* command "add": *<add path="/fld_teg" user="user"/>* *Result:* confirmation or an error message.
Insert of the string	*Request:* command "ins": *<ins path="/fld_teg" user="user" row="3"/>* To insert the string in position 3. Command does not work when attribute <key> is set! *Result:* confirmation or an error message.

Operation	Action
Deleting of the string	*Request:* command "del": *<del path="/fld_teg" user="user" row="3"/>* or *<del path="/fld_teg" user="user" key_id="Test"/>* for the key mode To remove the string in position 3 or string in the position where the <id> column value is "Test". *Result:* confirmation or an error message.
Moving of the string	*Request:* command "move": *<move path="/fld_teg" user="user" row="3" to="5"/>* To move the string from position 3 to position 5. Command does not work when attribute <key> is set! *Result:* confirmation or an error message.
Change the cell	*Request:* command "set": *<set path="/fld_teg" user="user" row="3" col="id">Test</set>* or *<set path="/fld_teg" user="user" key_id="Test" col="id">Test1</set>* for the key mode To set the cell's value in row 3 and column "id" to "Test" or set of the column named "id" of the row to the position where the <id> column value is "Test" to value "Test1". Practically, this command renames a key element of the specified row. *Result:* confirmation or an error message.

d) Tag <img>

```
<img id="ico" descr="Page icon"/>
```

The tag can be accessed as to "read" and also to "write"(modification), therefore an element of the triad of access may be:

00 — there is no any access;

04 — there is an access only for reading;

02 — there is only access fro the writing, usually this value has no meaning because the write access means access for reading;

06 — there is an access to read, and to write.

Tag is provided to transfer images to the clients of control interface. Under the image may be: pages' icons, graphics of the values' arrays and other data that can be presented in graphical form.

The following requests' commands are supported:

- *<get path="/fld_teg" user="user"/>* — image request;
  The result is the confirmation with image data or an error message.
- *<set path="/fld_teg" user="user">img</set>* — загрузка изображения.
  The result is the confirmation or error message.

e) Commands with the parameters. Tag <comm>

```
<comm id="add">
      <fld id="tm" tp="time"/>
      <fld id="cat" tp="str"/>
      <fld id="lvl" tp="dec" min="0" max="7"/>
      <fld id="mess" tp="str"/>
</comm>
```

The tag can be accessed as to "read" or visibility + service of the requests, and for modification or execution of the command, hence an element of the triad of access can be set to 00, if access is absent at all; 04, if the command can be seen, and 06, if the command can be initiated.

It is provided for transmission of commands and actions to the node, and also can be used to create links to other pages. Commands may include options. Parameters are described with <fld>.

The following request commands are supported:

- Command execution:

```
<set path="/fld_teg" user="user"/>
      <fld id="tm">1023456244</fld>
      <fld id="cat">*</fld>
      <fld id="lvl">2</fld>
```

```
            <fld id="mess">Test mess</fld>
        </set>
```

- Load the link to another page:

```
        <get path="/fld_teg" user="user" tp="lnk"/>"
```

The result is the confirmation or error message.

f) Branches (child nodes)

```
<list id="k_br" dscr="Kernel branches" tp="br"/>
```

The branches are described by the usual list <list> with special attributes tp = "br". Methodology of the request and modification of branches coincides with the one of work with the list <list>.

14.4 Hierarchical dependences of the information elements of the control language

Example of the node page of the control language:

```
<oscada_cntr>
      <area id="a_gen" dscr="Generic control">
            <fld id="config" dscr="Config file" tp="str" dest="file"/>
            <fld id="cr_file_perm" dscr="Files" tp="oct" len="3"/>
            <fld id="cr_dir_perm" dscr="Directories" tp="oct" len="3"/>
            <comm id="upd_opt" dscr="Update options(from config)"/>
            <comm id="quit" dscr="Quit"/>
      </area>
      <area id="a_kern" dscr="Kernels">
            <list id="k_br" dscr="Kernels" tp="br"/>
      </area>
</oscada_cntr>
```

Table 6. Hierarchical dependencies of the information elements of the language:

Tag	Description	Attributes	Contents
oscada_cntr	The root element of the page. It is the only and serves to identify the origin of language of control interface.	*id* — ID; *dscr* — description.	area, img, branches
branches	Container of the groups of child branches of the node.	*id* — Container's ID. It is equal: *br*.	grp
grp	The group of the child nodes.	*id* — prefix of the group of child nodes in the system; *dscr* — description of the branches' group; *acs* — access options.	
area	Grouping of the standard tags.	*id* — ID; *dscr* — description; *acs* — access options.	area, fld, list, table, comm, img
comm	Commands to the node.	*id* — ID; *dscr* — description; *help* — command's help; *tp* — command's type (*lnk* — ссылка); *acs* — access options.	fld
fld	Description of the standard data types.	*id* — ID; *dscr* — description; *help* — help; *acs* — access options. *tp* — element's type: *str(len, dest, cols, rows(SnthHgl))* — string element; *dec(len, max, min, dest)* — integer in decimal representation;	

Tag	Description	Attributes	Contents
		oct(len, max, min, dest) — integer in octal; *hex(len, max, min, dest)* — integer in hexadecimal; *real(len, max, min, dest)* — real number; *bool* — boolean sign; *time* — time/date in seconds (from 01/01/1970). **Connected:** *len* — value length (char.); *min* — minimum of value; *max* — maximum of value; *cols* — number of columns; *rows* — number of rows; *dest* — input method: *data* — source of binary data (base64). *select(select)* — selective type; *sel_ed(select)* — selective type with the possibility of editing. *select* — path to the hidden list; *sel_list* — static list (separator ';'); *sel_id* — static list of the identifiers (separator ';').	
list	List of standard data types.	*id* — ID; *dscr* — description; *help* — help list; *acs* — access options; *tp* — as in <fld> except: *br(br_pref)* — child nodes. *idm* — indexed list (0|1); *s_com* — ways of the list's modification [add][,ins][,edit][,del]: *add* — add rows; *ins* — insert rows; *edit* — modify rows; *del* — delete rows. **Connected:** *br_pref* — child node's prefix; *dest* — as in <*fld*>.	
table	Table of standard data types.	*id* — ID; *dscr* — description; *help* — table help; *acs* — access options; *key* — key columns (key=«id,name,per»); *cols* — list of columns in the request attribute; *rows* — range of rows in the attribute of the request; *s_com* — types of the table's modification [add][,del][,ins][,move]: *add* — add rows; *ins* — insert rows; *del* — delete rows; *move* — move rows.	list
img	Image.	*id* — ID; *dscr* — description; *help* — image help;	

Tag	Description	Attributes	Contents
		acs — access options; *h_sz* — horizontal limitation; *v_sz* — vertical limitation.	

14.5. Object of the dynamic tree node (TCntrNode)

Inherited:	By all the dynamic and controlled objects directly or through the childs.

Data:

Named rights of access to control elements (define):

- *R_R_R_* (0444) — access to all read-only;
- *R_R___* (0440) — read access only to the owner and group;
- *R_____* (0400) — read access only to the owner;
- *RWRWRW* (0666) — full access to all;
- *RWRWR_* (0664) — full access to the owner and group, and for the rest — the read-only;
- *RWR_R_* (0644) — full access to the owner and for the group and the rest — the read-only;
- *RWR___* (0640) — full access to the owner, read-only — to the group and closed for the rest;
- *RW____* (0600) — full access to the owner, but for the group and everyone else is closed.

Dynamic node's flags (enum TCntrNode::Flag):

- *TCntrNode::MkDisable* — switching-off (0);
- *TCntrNode::Disable* — disabled (1);
- *TCntrNode::MkEnable* — switching-on (2);
- *TCntrNode::Enable* — enabled (3);
- *TCntrNode::SelfModify* — sign of the node's modification (0x04);
- *TCntrNode::SelfModifyS* — sign for node's modification save into *load_()* (0x08).

Flags of the enabling/disabling modes of the node (enum TCntrNode::Flag):

- *TCntrNode::NodeConnect* — connection of the node;
- *TCntrNode::NodeRestore* — restoration of the connection of the node;
- *TCntrNode::NodeShiftDel* — the sign -deleting of the node is suspended.

Modification of the node flags (enum TCntrNode::ModifFlag):

- *TCntrNode::Self* — the given node is modified;
- *TCntrNode::Child* — the child nodes are modified;
- *TCntrNode::All* — the given node and the child ones are modified.

Public methods:

• *TCntrNode(TCntrNode *prev = NULL);* — Initialization with an indication of the parent node <*prev*>.

• *virtual string objName( );* — Object name. Object-heir should extend the definition of aggregate object name.

• *virtual TCntrNode &operator=(TCntrNode &node);* — Virtual copy function of the dynamic tree nodes.

• *void cntrCmd(XMLNode *opt, int lev = 0, const string &path = "", int off = 0);* — Command of work with the control interface of the system. The transport transitions to the full path of the following form are supported: </sub_Security/usr_root/%2fgen> где %2fgen — encoded sub-path to a particular field of the page (/gen).

• *static XMLNode *ctrId(XMLNode *inf, const string &n_id, bool noex = false);* — Getting the XML node by the attribute's value "id" <*n_id*>. XML requests of the node by the full path to it are supported (node1/node2/node3).

• *static XMLNode *ctrMkNode(const char *n_nd, XMLNode *nd, int pos, const char *req, const string &dscr int perm=0777, const char *user="root", const char *grp="root", int n_attr=0, ...);* — Adding the control element to the page. It is possible to specify the set of additional attributes in the number of <*n_attr*> as follows: <attribute1>,<values1>,<attribute2>,<values2>,... .

• *bool ctrRemoveNode(XMLNode *nd, const char *path);* — Removing the control element <*path*> from the page <*nd*>.

• *static bool ctrChkNode(XMLNode *nd, const char *cmd="get", int perm=0444, const char*

**user="root", const char *grp="root", char mode=04, const char *warn = NULL);* — Checking for the dynamic command getting <*cmd*> and for the existence of rights to its execution.
- *virtual Res &nodeRes( );* — Resource of the node's using.
- *virtual const char *nodeName( );* — Node's name.
- *string nodePath(char sep = 0, bool from_root = false);* — Getting the full path to the node, beginning from the root <*from_root*>, and using the separator <*sep*> or standard path recording.
- *void nodeList(vector<string> &list, const string& gid = "");* — Child nodes list <*list*> in the specified group <*gid*>.
- *AutoHD<TCntrNode> nodeAt(const string &path, int lev = 0, char sep = 0, int off = 0, bool noex = false);* — Attachment to the child node.
- *void nodeDel(const string &path, char sep = 0, int flag = 0);* — Deleting the node using its path.
- *static void nodeCopy(const string &src, const string &dst, const string &user = "root");* — Copying the nodes of the dynamic tree.
- *TCntrNode *nodePrev(bool noex = false);* — Address of the parent node.
- *char nodeFlg( );* — Node's flag.
- *char nodeMode( );* — Node's status.
- *unsigned nodeUse(bool selfOnly);* — Number of the connections to the node. <*selfOnly*> — only self connections, without childs.
- *unsigned nodePos( );* — The position of the node in the parent-node's container. It is valid only for ordered containers.
- *int isModify(int mflg = TCntrNode::All);* — Checking for the modification of the node or nodes' branch.
- *void modif(bool save = false);* — Setting the sign of the node's modification, with save sign <*save*>.
- *void modifG( );* — Setting the sign of the nodes' branch modification.
- *void modifClr(bool save = false);* — Clear the sign of the node's modification or save sign <*save*>.
- *void modifGClr( ); — Clear the sign of the nodes' branch modification.*
- *void load(bool force = false);* — Load the dynamic tree node.
- *void save( );* — Saving the dynamic tree node.
- *virtual void AHDConnect( );* — Connection to the node (capture of the resource).
- *virtual bool AHDDisConnect( );* — Disconnection from the node (release of the resource). Resource will remove on zero result.
- *virtual TVariant objPropGet(const string &id);* — Request of the node's property as the user's object.
- *virtual void objPropSet(const string &id, TVariant val);* — Setting the node's property as the user's object.
- *virtual TVariant objFuncCall(const string &id, vector<TVariant> &prms);* — Node's function calling as the user's object.
- *virtual AutoHD<TCntrNode> chldAt(int8_t igr, const string &name, const string &user = "");* — Attachment to the child node <*name*> of the container <*gr*> of the user <*user*>.
- *void chldList(int8_t igr, vector<string> &list);* — The list of the child nodes <*list*> in the specified container <*gr*>.
- *bool chldPresent(int8_t igr, const string &name);* — Check for the presence of the specified child node <*name*> in the container <*gr*>.

Protected methods:
- *virtual void cntrCmdProc(XMLNode *req);* — Function of requests' processing of the control interface. It must be redefined by the child.
- *void nodeEn(int flag = 0);* — Node's enabling.
- *void nodeDis(long tm = 0, int flag = 0);* — Disabling of the node with the flag transfer.
- *void nodeDelAll( );* — Clear all containers with child nodes.
- *void setNodePrev(TCntrNode *node);* — Setting the parent node to <*node*>.
- *void setNodeMode(char mode);* — Setting the node's status.
- *virtual void chldAdd(int8_t igr, TCntrNode *node, int pos = -1);* — Adding the child node <*node*> to the container <*gr*> and position <*pos*>.
- *void chldDel(int8_t igr, const string &name, long tm = -1, int flag = 0, bool shDel = false);* — Deleting the child node <*name*> from the container <*gr*> with the flag <*flag*>.
- *int8_t grpSize( );* — Number of the containers with the child nodes.
- *int8_t grpId(const string &sid);* — Getting the index of the group by its ID.

- *GrpEl &grpAt(int8_t id);* — Access to the structure of the group.
- *unsigned grpAdd(const string &id, bool ordered = false);* — Adding the container of the child nodes with the prefix <*id*> and the possibility of an orderly storage <*ordered*>. Returns the ID of the new container.
- *virtual void preEnable(int flag);* — The connection preact. It is called before the actual connection.
- *virtual void postEnable(int flag);* — The connection preact. It is called after the actual connection.
- *virtual void preDisable(int flag);* — The disconnection preact. It is called before the actual disconnection.
- *virtual void postDisable(int flag);* — The disconnection preact. It is called before the actual disconnection (before deleting).
- *virtual void load_( );* — The function of the loading child node calling.
- *virtual void save_( );* — The function of the saving child node calling.

15. XML in the OpenSCADA system (XMLNode)

XML in the OpenSACDA system is represented by the object of the XML-tag — XMLNode.

15.1. XML-tag (XMLNode)

Data:

Options of the function of XML-file generation (enum - XMLNode::SaveView):

- *XMLNode::BrOpenPrev* — insert the end of the line before the opening tag;
- *XMLNode::BrOpenPast* — insert the end of the line after the opening tag;
- *XMLNode::BrClosePast* — insert the end of the line after the closing tag;
- *XMLNode::BrTextPast* — insert the end of the line after the tag text;
- *XMLNode::BrSpecBlkPast* — insert the end of the line after the computed instruction;
- *XMLNode::BrAllPast* — insert the end of the line after the all elements;
- *XMLNode::XMLHeader* — insert the standard xml-header;
- *XMLNode::Clean* — result clean from comments and procedure insertions;
- *XMLNode::XHTMLHeader* — insert the standard XHTML-header;
- *XMLNode::MissTagEnc* — to miss for tags name encoding.
- *XMLNode::MissAttrEnc* — to miss for attributes name encoding.

Public methods:

- *XMLNode(const string &name = "");* — Initialization of the tag with the name <*name*>.
- *XMLNode(const XMLNode &nd);* — Copy constructor.
- *XMLNode &operator=(const XMLNode &prm);* — Copying a branch of the XML-tree from <*prm*>.
- *string name( ) const;* — Tag's name.
- *XMLNode* setName(const string &s);* — Setting the tag's name to <*s*>.
- *string text(bool childs = false, bool recursive = false) const;* — Tag's text. <*childs*> — for text getting from special text nodes.
- *XMLNode* setText(const string &s, bool childs = false);* — Setting the tag's text to <*s*>. <*childs*> — for set text to special text node.
- *void attrList(vector<string> & list) const;* — The list of attributes <*list*> in the tag.
- *XMLNode* attrDel(const string &name);* — Deleting of the attribute <*name*>.
- *void attrClear( );* — Clear the tag's attributes.
- *string attr(const string &name, bool caseSens = true) const;* — Getting the attribute <*name*>.
- *XMLNode* setAttr(const string &name, const string &val);* — Setting/creation of the attribute <*name*> with the value <*val*>.
- *void load(const string &vl, bool full = false, const string &cp = "UTF-8");* — Load/parsing of the XML-file, with source encoding <*cp*>. <*full*> — for full XML-loading with text and comments blocks to special nodes.
- *string save(unsigned flgs = 0, const string &cp = "UTF-8");* — Saving/creation of the XML-file with the formatting parameter <*flgs*> and encoding to <*cp*>.
- *XMLNode* clear( );* — Clear the tag (recursively, including all attachments).
- *bool childEmpty( ) const;* — Check for empty attached tags.
- *unsigned childSize( ) const;* — The number of attached tags.
- *void childAdd(XMLNode *nd);* — Adding the attached tag.
- *XMLNode* childAdd(const string &name = "");* — Adding the attached tag.
- *int childIns(unsigned id, XMLNode *nd);* — Insert of the attached tag to the position <*id*>.
- *XMLNode* childIns(unsigned id, const string &name = "");* — Insert of the attached tag with the name <*name*> to the position <*id*>.
- *void childDel(const unsigned id);* — Deleting of the attached tag <*id*>.
- *void childDel(XMLNode *nd);* — Deleting of the attached tag using its address <*nd*>.
- *void childClear(const string &name = "");* — Clear of the attached tag <*name*>.
- *XMLNode* childGet(const int, bool noex = false) const;* — Getting the attached tag by its index number.
- *XMLNode* childGet(const string &name, const int numb = 0, bool noex = false) const;* — Getting of the attached <*numb*> indexed tag by the tag's name <*name*>. <*noex*> indicates the prohibition of the

generation of exception in the case of absence of the tag.

- *XMLNode* childGet(const string &attr, const string &name, bool noex = false) const;* — Getting the attached <numb> indexed tag by its value <*name*> of the attribute <*attr*>. <*noex*> indicates the prohibition of the generation of exception in the case of absence of the tag.
- *XMLNode* parent( );* — Parent tag of the tag.

16. Resources in the OpenSCADA system (Res, ResAlloc, AutoHD)

Most of the units and subsystems of the OpenSCADA are dynamic, ie they allow the creation/deletion/configuration while the system is working. Taking into account the multi-threading of the system, this functionality imposes stringent requirements for synchronization of threads. For synchronization in the system the resources are used, functions of which are localized in the objects <Res> and <ResAlloc>. Object <Res> provides storage of the resource, providing the functions of capture/release for the read and write. In object <ResAlloc> the automatic release of the resource functions are implemented. Automatic resource involves the creation of a local resource object with its automatic release at fracture (in the destructor). Using of automatic resource makes the work with resources when using the exceptions much easier.

Any dynamic system object is inherited from TCntrNode object, which contains a mechanism to connect via the AutoHD template. The main function of the template is to store the link to the object and the capture of the resource, excluding the deleting of the object at the time of use. Template supports copying the resource and its release in case of destruction of the template object. For clarity of the access to the objects generated by TCntrNode the AutoHD template supports casting, based on the dynamic cast.

16.1. Resource object (Res)

Public methods:

- *Res( );* — Initialization of the resource.
- *static void resRequestW(long tm = 0);* — Request the resource for writing/modification with wait timeout <*tm*> (in milliseconds).
- *bool resTryW( );* — Request try the resource for writing/modification. On success return "true" else "false".
- *static void resRequestR(long tm = 0);* — Request the resource for reading with wait timeout <*tm*> (in milliseconds).
- *bool resTryR( );* — Request try the resource for reading. On success return "true" else "false".
- *static void resRelease( );* — Release of the resource.

16.2. Resource object (ResAlloc)

Public methods:

- *ResAlloc(Res &rid);* — Initialization of the automatically released resource for the previously selected identifier <*rid*>.
- *ResAlloc(Res &rid, bool write, long tm = 0);* — Initialization of the automatically released resource for the previously selected identifier <*rid*>. With the specification of the resource type <*write*> (read/write).
- *void request(bool write = false, long tm = 0);* — Request of the resource in the specified mode <*write*> (read/write).
- *void release( );* — Release of the resource.

16.3. Template (AutoHD)

Public methods:

- *AutoHD( );* — Initializing without linkage to the object.
- *AutoHD(ORes *node, const string &who = "");* — Initialization with the linkage to the object <*node*>. The object must contain the function AHDConnect() and AHDDisConnect().
- *AutoHD(const AutoHD &hd);* — Copyng constructor.
- *template <class ORes1> AutoHD(const AutoHD<ORes1> &hd_s, bool nosafe = false);* — Cast constructor in safe mode casting (by help dynamic_cast). Return free resource on impossible cast.
- *ORes &at( ) const;* — Getting the object by the resource.
- *AutoHD &operator=(const AutoHD &hd);* — Copying of the resources.
- *bool operator==(const AutoHD &hd);* — Compare objects into resource by pointer.

- *void free( );* — Release of the resource.
- *bool freeStat( ) const;* — The sign «The resource is free».

16.4. Object of the string with the access shared by the resource (ResString)

Public methods:

- *explicit ResString(const string &vl = "");* — Initialization of the string with the specified value <*vl*>.
- *ResString &operator=(const string &val);* — Function for implicit conversion from *std::string*.
- *operator string( );* — Function for implicit conversion to *std::string*.
- *size_t size( );* — The string size.
- *bool empty( );* — The string empty.
- *void setVal(const string &vl);* — Setting the string's value to <*vl*>.
- *const string &getVal( );* — Getting the string's value.
- *const string &getValRef( );* — Getting a direct reference to the string *std::string*. The resource is ignored when you use this function!

17. Organization and structure of the database of the system components

Nodes and subsystems of OpenSCADA may have their own tables in the database to store their own data. The structure of tables is individual and determined by the <TConfig> object. Nodes and subsystems must create and configure the <TConfig> object under their demands.

17.1. System tables

OpenSCADA system has two system tables: BD and SYS. Table BD contains records of registered databases and the table SYS contains data of system-wide parameters.

Table 7. Structure if the table of system-wide parameters (SYS).

User <user>	Parameter's ID <id>	Parameter's value <val>
root	/DemoStation/MessLev	0
user	/DemoStation/Workdir	/mnt/home/roman/work/OScadaD/share/OpenScada
user	/DemoStation/UI/QTStarter/StartMod	QTCfg

Table 8. Structure if the table of registered DB.

ID <ID>	DB Type <TYPE>	Name <NAME>	Description <DESCR>	Address <ADDR>	Codepage of the database contents <CODEPAGE>	To enable <EN>
LibBD	MySQL	Function's library		server.diya.org;roman;123456;oscadaUserLibs	KOI8-U	1
AnastModel	SQLite	AGLKS model		./DATA/AGLKSModel.db	UTF8	1
GenDB	MySQL	Main DB		server.diya.org;roman;123456;oscadaDemoSt	KOI8-U	1

17.2. Tables of the "Data acquisition" subsystem

Controllers (data sources) of the subsystem "Data acquisition" are stored in the tables of their subsystems named DAQ_<ModName>. The structures of these tables can differ significantly, but all of them have the obligatory fields. The overall structure of the controllers' tables is presented in table 9.

Table 9. The overall structure of the controllers' tables of the subsystem "Data acquisition" (DAQ_<ModName>).

ID <ID>	Controller's name <NAME>	Description <DESCR>	To enable <ENABLE>	To start <START>	Individual parameters
AutoDA	Automatic source	Data acquisition from active sources with automatic identification of them.	1	1	...

Like the controller's table, the parameter's table for different types of data sources can differ significantly, but also have the obligatory fields. In addition to the differences which is typical to the type of data source, parameter's tables can still be different for different types of parameters. The overall structure of the parameters' tables is given in Table 10.

Table 10. The overall structure of the parameters' tables of the subsystem "Data acquisition".

Parameter's shifr <SHIFR>	Parameter's name <NAME>	Parameter's description <DESCR>	To enable <EN>	Individual parameters
P3	P3	Pressure on the diaphragm	1	...

In addition to controllers and parameters the subsystem "Data acquisition" contains parameter's templates. Parameter's templates are grouped by templates' libraries and are stored in tables of three types: templates' library table (ParamTemplLibs) - table 11, parameter's templates table - table 12 and template's parameters table - table 13.

Table 11. Structure of the templates' library table.

ID <ID>	Name <NAME>	Description <DESCR>	DB table of the library <DB>
base	Basic templates	Basic templates' library	tmplib_base
S7		Templates' Library for Siemens S7 series controllers.	tmplib_S7

Table 12. Structure of the templates' table.

ID <ID>	Name <NAME>	Description <DESCR>	Text of the template procedure <PROGRAM>
digAlarm	Digital signal	Alarm over the discrete parameter	JavaLikeCalc.JavaScript
simpleBoard	Simple boards	Formation of the simple boards of the analog signal.	JavaLikeCalc.JavaScript

Table 13. Structure of the table of the template's parameters.

Template's ID <TMPL_ID>	Parameter's ID <ID>	Name <NAME>	Type <TYPE>	Flags <FLAGS>	Value <VALUE>	Position <POS>
digAlarm	in	Вход	3	144		2
digitBlock	cmdOpen	Open command	3	161		0

17.3. Tables of the "Transports" subsystem

Subsystem "Transports" is divided into incoming and outgoing transports. For each type of transport there is its own table with its own structure. Table names, respectively: Transport_In and Transport_Out. Tables can be supplemented by fields, typical to the type of transport.

Table 14. Structure of the incoming transport's table (Transport_in).

ID <ID>	Type <MODULE>	Name <NAME>	Description <DESCRIPT>	Address <ADDR>	Protocol <PROT>	To start <START>	Individual fields of the transports' types
web1	Sockets	Web 1	Work web transport for proced http requests.	TCP::10002:0	HTTP	1	...
Self	SelfSystem	TCP 1	Test TCP input socket!	Sockets	TCP::10001:1	1	...

Table 15. Structure of the outgoing transport's table (Transport_out).

ID <ID>	Type <MODULE>	Name <NAME>	Description <DESCRIPT>	Address <ADDR>	To start <START>	Individual fields of the transports' types
tcp_o1	Sockets	TCP Out 1	Output TCP transport 1	TCP::10001	1	...

For the centralized description of the list of external OpenSCADA stations it is used the table of external hosts (CfgExtHosts). The structure of this table is shown in Table 16.

Table 16. The structure of the table of external OpenSCADA hosts (CfgExtHosts).

User of the system <OP_USER>	ID <ID>	Name <NAME>	Transport <TRANSP>	Address of the remote host <ADDR>	User of the external host <USER>	Password of the user of the external host <PASS>
tcp_o1	Sockets	TCP Out 1	Output TCP transport 1	TCP::10001	1	...

17.4. Tables of the "Archives" subsystem

Subsystem "Archives" contains three tables with fixed names:

- Archives of values: Archive_val;
- Archivers of values: Archive_val_proc;
- Archivers of messages: Archive_mess_proc.

Tables of the archivers can be complemented by fields, typical for each type of archiver.

Table 17. Structure of the table of the values' archive (Archive_val).

ID <ID>	Name <NAME>	Description <DESCR>	To start <START>	The mode of the values' source <SrcMode>	Source of the values <Source>	Type of the values <VTYPE>	Buffer's periodicity <BPER>	Buffer's size <BSIZE>	Buffer's hard grid <BHGRD>	High resolution of the buffer's time <BHRES>	List of the serviced archivers <ArchS>
CPULoad_load			1	1	DAQ.System.AutoDA.CPULoad.load	4	1	100	0	0	FSArch.1s;DBArch.1m;FSArch.1m;
ai1_dP			0	0		4	0.0001	100	1	1	FSArch.POMP_20070301;FSArch.1s;

Table 18. Structure of the table of the values' archivers (Archive_val_proc).

ID <ID>	Archiver's type <MODUL>	Name <NAME>	Description <DESCR>	To start <START>	Address <ADDR>	Values' period <V_PER>	Archiving period <A_PER>	Individual fields of the archivers' types
1s	FSArch		One second	1	ARCHIVES/VAL/1s	1	60	...
POMP_20070301	FSArch			0	ARCHIVES/VAL/POMP_20070301	0.0001	60	...

Table 19. Structure of the table of the messages' archivers (Archive_mess_proc).

ID <ID>	Archiver's type <MODUL>	Name <NAME>	Description <DESCR>	To start <START>	Template of the messages' category <CATEG>	Messages' level <LEVEL>	Address <ADDR>	Individual fields of the archivers' types
StatErrors	FSArh	Station error		1	/DemoStation*	4	ARCHIVES/MESS/stError/	...
NetRequsts	FSArh	Network requests		1	/DemoStation/Transport/Sockets*	1	ARCHIVES/MESS/Net/	...

17.5. Tables of the "Security" subsystem

Subsystem "Security" contains two tables: table of the system's users (Security_user) and groups of the system (Security_grp).

Table 20. Structure of the table of the system's users (Security_user).

Name <NAME>	Description <DESCR>	Password <PASS>	Picture <PICTURE>
root	SuperUser	openscada	
user	User	user	

Table 21. Structure of the table of the system's users groups (Security_grp).

Name <NAME>	Description <DESCR>	Users in the group <USERS>
root	SuperUser's group	root;user
users	User's group	toot;user

17.6. The structure of the databases of the modules

Each module can have its own database tables to store individual data. Structure of database tables of the modules can be formed freely, based on internal needs.

18. Service functions of the OpenSCADA control interface

Service functions — is an interface to access the OpenSCADA system through the OpenSCADA control interface from external systems. This mechanism is the basis for all exchange mechanisms within OpenSCADA, implemented through weak links and standard exchange protocol of OpenSCADA. Its main advantage is the priority processing and the possibility of using non-standard packaging data. For access to normal data you can use the standard commands of the control interface.

18.1. Group access to the values of the parameter's attributes of the subsystem "Data acquisition", as well as to the detailed information

These requests let you to obtain detailed information about the parameters of the subsystem "Data acquisition", to acquire the values of all attributes of the parameters, and also to make the group setting. Detailed information about requests is given in Table 23.

Table 23. Attributes of the request commands of the attributes' parameters of the subsystem "Data acquisition"

Id	Name	Value
Request command of information about the attributes of the parameter: **<list path="/sub_DAQ/mod_{SRC}/cntr_{CNTR}/prm_{PRM}/%2fserv%2fattr"/>**		
<el id="iatr"/>	Information of the attributes	In separate tags the information about attribute is returned.
id	Attribute's ID	The character identifier of the single attribute.
nm	Attribute's name	The name of the single attribute.
flg	Attribute's flags	Flags of the single attribute.
tp	Attribute's type	The type of the single attribute.
vals	Values area of the attribute.	Values area of the single attribute.
names	The names of the attribute's values for the selective type.	The names of the single attribute's values for the selective type.
Team request command of the all values of attributes of the parameter: **<get path="/sub_DAQ/mod_{SRC}/cntr_{CNTR}/prm_{PRM}/%2fserv%2fattr"/>**		
<el id="iatr">val</el>	Attributes' values	In separate tags the values of the attributes are returned.
The set command of the the values of the specified attributes of the parameter: **<set path="/sub_DAQ/mod_{SRC}/cntr_{CNTR}/prm_{PRM}/%2fserv%2fattr"/>**		
<el id="iatr">val</el>	The values of attributes are specified	In separate tags the values of attributes are specified.

18.2. Access to archived data of the archives of messages

To request and set the archive data in the archiving subsystem of the messages archive's object the following command is provided <info path="/sub_Archive/%2fserv%2fmess"/>, <get path="/sub_Archive/%2fserv%2fmess"/> and <set path="/sub_Archive/%2fserv%2fmess"/> to request the information about the archive, archive's values and set values to archive, respectively. The detailed description of these commands is presented in Table 24.

Table 24. Attributes of request commands of information about the archive of values and archive data

Id	Name	Value
Request command of the information about the archive: **<info path="/sub_Archive/%2fserv%2fmess"/>**		
arch	Setting the name of the archive's archiver	Archive's archiver, for which to specify the parameters.
end	Control of the archive's top in the given archiver	As a result of the request points to the real top of the messages' archive in the given archiver <arch>.

Id	Name	Value
beg	Control of the archive's depth in the given archiver	As a result of the request points to the real depth of the messages' archive in the given archiver <arch>.
Request command of the archival and/or current data: **<get path="/sub_Archive/%2fserv%2fmess"/>**		
tm	Time set	Time of the block's top of the messages' archive .
tm_grnd	Time set of the foundation/beginning of the archive	Points to the foundation/beginning of the archive.
arch	Setting of the archive's archiver	Specifies at what archiver to request the values. If the archiver is not specified, the request will be made consistently in all archivers with the exception of duplicates.
cat	Setting the message's category	Specify the category/template of the requested messages.
lev	Setting the level of importance	Specifies the importance level of messages for which and above to get the messages.
<el>{mess}</el>	Messages	In separate tags the messages are returned.
time, utime	Message's time	Time of the separate message.
cat	Message's category	Category of the separate message.
lev	Message's level	Level of the separate message.
Set command of the archival data: **<set path="/sub_Archive/%2fserv%2fmess"/>**		
<el>{mess}</el>	Messages	In separate tags the messages for set.
time, utime	Message's time	Time of the separate message.
cat	Message's category	Category of the separate message.
lev	Message's level	Level of the separate message.

18.3. Access to archived data of the values' archive

To request the archive data in the archiving subsystem of the values archive's object and parameter's attribute object of "Data acquisition" subsystem the following command is provided <info path="{a_p_addr}/%2fserv%2fval"/> и <get path="{a_p_addr}/%2fserv%2fval"/> to request the information about the archive's and archive values respectively. Attributes of these commands, involving various mechanisms of the request are presented in Table 25.

Table 25. Attributes of the request commands of information about the archive of archive data

Id	Name	Value
Request command of the information about the archive: **<info path="{a_p_addr}/%2fserv%2fval"/>**		
arch	Setting the name of the archive's archiver	Archive's archiver, for which to specify the parameters.
end	Control of the archive's top in the given archiver	As a result of the request points to the real top of the values' archive in the given archiver <arch>. In the case of the archive's absence the attribute is set to "0".
beg	Control of the archive's depth in the given archiver	As a result of the request points to the real depth of the values' archive in the given archiver <arch>. In the case of the archive's absence the attribute is set to "0".
per	Control of the archive's periodicity in the given archiver	As a result of the request points to the real periodicity of the values' archive in the given archiver <arch>. In the case of the archive's absence the attribute is set to "0".
vtp	Control of the archive's/parameter's type of value	Returns the code of the type of the archive data values: 0 — Boolean, 1 — Integer, 4 — Real, 5 — String. But this property is only available in the case of request at the parameter without archive!

Id	Name	Value
Request command of archive or/and current data **<get path="{a_p_addr}/%2fserv%2fval"/>**		
tm	Setting and control of time	Time of the requested value or top of the archive values' block. If the attribute is not specified or equal to "0", the last value is returned. The value of this attribute is set to the value of time of obtained value or of the top of the archival data block.
tm_grnd	Setting and control of the foundation/beginning time of the archive	Points to the foundation/beginning of the archive. If the attribute is not specified or equal to "0", then the request of one value is made. The value of this attribute is set to the foundation of the block of archival data time.
per	Setting and control of the periodicity of the obtained values	Periodicity of requested values. If the attribute is not specified or equal to "0" or less than the real periodicity of the archive, then the values will be returned with the real periodicity of the archive and the value of this attribute will be set to the real periodicity value. If the attribute's value is more than the actual periodicity, the values of the archive will be averaged to the specified periodicity.
arch	Setting and control of the archive's archiver	Specifies at what archiver to request the values. If the archiver is not specified, the request will be made consistent with the priority from best to worst. The name of the archiver, from whom the values are obtained, will be indicated in this attribute when returning the result. If the archiver for this parameter is not set, or such archiver is not present, then the value of this attribute will be reset. In the case of the request, crossing several archivers, the processing occurs only for the first archiver with the specification of its name and dimension in the relevant attributes. To request the data of follow-up archivers the request is repeated starting from the dimension of the previous request.

mode	Setting and control of the mode of transfer of the archive data	The form in which it is desirable to obtain the archive data is specified. The following modes/forms of transfer of archives' data are provided:
		0 — Simple record: single value — one string without packaging of the related values. In the case of string archive the values are encoded to exclude the newlines characters. Example of the writing form:
		`34.5678`
		`23.6543`
		`65.8754`
		`34.6523`
		1 — Packed record on the principle of folding the related values: one value - one record. Before each value it is indicated its position number, separated by the space:
		`0 34.5678`
		`1 23.6543`
		`4 65.8754`
		`6 34.6523`
		2 — The array of unpackaged binary values encoded by Mime Base64. Can not be used for string archives.

real_prec	Setting the precision of the real values	Specifies the precision with which to transmit the data of real values in the "0" and "1" mode. By default, this precision is 6 digits.
round_perc	Setting the rounding percent	Indicates the rounding percentage of related numeric values for the mode of "1".

19. API of modules of modular subsystems

Modules in the OpenSCADA system are implemented as shared libraries. As it was previously mentioned, one such library can contain many modules of OpenSCADA subsystems, actually acting as a container.

The first step in connecting the shared (SO — shared object) libraries is the connection of the initialization functions. These functions should be defined as usual "C" functions to avoid distortion of function names. Usually this is done as follows:

```
//================== CUT =========================
extern "C"
{
#ifdef MOD_INCL
    //Need for define only at case builtin to OpenSCADA core support by the module
    TModule::SAt bd_DBF_module( int n_mod )
#else
    TModule::SAt module( int n_mod )
#endif
    {
    //Allowed into the shared object the modules descriptors forming
    }

#ifdef MOD_INCL
    //Need for define only at case builtin to OpenSCADA core support by the module
    TModule *bd_Tmpl_attach( const TModule::SAt &AtMod, const string &source )
#else
    TModule *attach( const TModule::SAt &AtMod, const string &source )
#endif
    {
    //Selected module connect
    }
}
//================== CUT =========================
```

Functions for the working with shared library:

TModule::SAt module(int n_mod);

The function is provided to sequentially poll the information about modules which are contained in the SO library. <n_mod> parameter indicates the serial number of the requested module and must be looked over, starting from zero. In the case of absence of the module with the given identifier the function must return the structure with the module name of zero length, that is the end of the scanning process.

*TModule *attach(const TModule::SAt &AtMod, const string &source);*

Connection to the specified module.

Actually all functions and data of the system are summarized in the API-systems described in this document. However, to simplify and accelerate the process of writing the modules the main function, redefined in modules, are listed in Table 22.

Table 22. The main functions used when creating the modules

Common API
Objects API (TCntrNode): • *virtual void load_( );* — Object loading from storage. • *virtual void save_( );* — Object saving into storage.
Common modules API (TModule): *Attributes:* • *string mId;* — Module's ID. • *string mName;* — Module's name. • *string mDescr;* — Module's description. • *string mType;* — Module's type. • *string mVers;* — Module's version. • *string mAutor;* — Module's author. • *string mLicense;* — Module's license. • *string mSource;* — Module's source/origin. *Methods:* • *virtual void modStart( );* — Module's start. • *virtual void modStop( );* — Module's stop. • *virtual void modInfo(vector<string> &list);* — The list of available elements of information about the module. There are the following information elements: Module — module's ID; Name — localized module's name; Type — module's type; Source — module's source (container); Version — module's version; Author — module's author; Description — module's description; License — module's license. • *virtual string modInfo(const string &name);* — Request of the specified information element. • *virtual void perSYSCall(unsigned int cnt);* — Call from system thread, with period 10 seconds and seconds counter <*cnt*>. • *void postEnable(int flag);* — Connecting the module to the dynamic tree of objects. • *void modFuncReg(ExpFunc *func);* — Registration of the exported function.
API of the modules of the "DB" subsystem.
DB type (child from TTipBD): • *virtual TBD *openBD(const string &id);* — Open/create DB.
DB (child from TBD): • *virtual void enable( );* — DB enable. • *virtual void disable( );* — DB disable. • *virtual void allowList(vector<string> &list);* — Tables' list in the DB. • *virtual void sqlReq(const string &req, vector< vector<string> > *tbl = NULL, char intoTrans = EVAL_BOOL);* — Sending the SQL-request <*req*> to the DB and receiving the result in tabular form <*tbl*>. If set <*intoTrans*> to true then will open transaction for the request, else if set to false then transaction will close. • *virtual void transCloseCheck( );* — The function call periodic for opened transaction check to close for old transaction or overloaded. • *virtual TTable *openTable(const string &table, bool create);* — Open/create the table.
Table (child from TTable): • *virtual void fieldStruct(TConfig &cfg);* — Getting the table's structure. • *virtual bool fieldSeek(int row, TConfig &cfg);* — Sequential scan of the table fields. • *virtual void fieldGet(TConfig &cfg);* — Getting the specified field. • *virtual void fieldSet(TConfig &cfg);* — Setting the specified field. • *virtual void fieldDel(TConfig &cfg);* — Deleting the specified field.
API of the modules of subsystem "Transports".

Transport's type (child from TTipTransport):

- *virtual TTransportIn *In(const string &name, const string &db);* — Create/open new "incoming" transport.
- *virtual TTransportOut *Out(const string &name, const string &db);* — Create/open new "outgoing" transport.

Incoming transport (child from TTransportIn):

- *virtual string getStatus( );* — Interface status.
- *virtual void setAddr(const string &addr);* — The transport address set. It would be redefined for processing and checking for specific to the module transport's address format.
- *virtual void start();* — Transport's start.
- *virtual void stop();* — Transport's stop.

Outgoing transport (child from TTransportOut):

- *virtual string getStatus( );* — Interface status.
- *virtual void setAddr(const string &addr);* — The transport address set. It would be redefined for processing and checking for specific to the module transport's address format.
- *virtual void start( );* — Transport's start.
- *virtual void stop( );* — Transport's stop.
- *virtual int messIO(const char *obuf, int len_ob, char *ibuf = NULL, int len_ib = 0, int time = 0, bool noRes = false);* — Sending data via the transport. Timeout <time> of the connection is indicated in milliseconds. <noRes> is used by the protocols for exclusive block of the transport for the time of working with him and to avoid its own block by the function.

API of the modules of subsystem "Protocols".

Protocol (child from TProtocol):

- *virtual void itemListIn(vector<string> &ls, const string &curIt = "");* — The input protocol subelements list, at case the protocol allow. Used for select in object of input transport configuration.
- *virtual void outMess(XMLNode &io, TTransportOut &tro);* — Data transfer in the XML tree *<in>* to the remote system by means of transport *<tro>* and the current outgoing protocol.
- *virtual TProtocolIn *in_open(const string &name)* — Open/create the incoming protocol.

Incoming protocol (child from TProtocolIn):

- *virtual bool mess(const string &request, string &answer, const string &sender);* — Transfer of unstructured message to the protocol.

API of the modules of "Data acquisition" subsystem.

Controller's type (child from TTipDAQ):

- *virtual void compileFuncLangs(vector<string> &ls);* — The list of user programming languages that are supported by the module.
- *virtual void compileFuncSynthHighl(const string &lang, XMLNode &shgl);* — Syntax highlight rules *<shgl>* request for user programming language *<lang>*.
- *virtual string compileFunc(const string &lang, TFunction &fnc_cfg, const string &prog_text);* — Compilation of the user procedure and creation of the object of function's execution for the specified language of user programming.
- *virtual bool redntAllow( );* — The sign of support of the redundancy mechanisms by the module. Should by simply redefined and return true.
- *virtual TController *ContrAttach(const string &name, const string &daq_db);* — Opening/connection of the controller.

Controller (child from TController):

- *virtual string getStatus( );* — Call for specific controller status get.
- *virtual void enable_( );* — Controller enable.
- *virtual void disable_( );* — Controller disable.
- *virtual void start_( );* — Controller's start.
- *virtual void stop_( );* — Controller's stop.
- *virtual void redntDataUpdate(bool firstArchiveSync = false);* — An operation of the data receiving from the redundant station execution. It is called automatically by the service task of the redundancy scheme and before starting to synchronize the archives with the parameter *<firstArchiveSync>* set.
- *virtual TParamContr *ParamAttach(const string &name, int type);* — Creation/opening of the new parameter.

Controller's parameter (child from TParamContr->TValue):

- *virtual void enable( );* — Enable parameter.
- *virtual void disable( );* — Disable parameter.
- *virtual void setType(const string &tpId);* — Is called to change the parameter type *<tpId>* and can be processed in the object module for self data change.
- *virtual TVal* vlNew( );* — Call at new attribute creation. Would be redefined for specific behavior into self, inherited from *TVal*, class at accessing to attribute.
- *virtual void vlSet(TVal &val, const TVariant &pvl);* — Call for attribute into direct write mode *TVal::DirWrite* (synchronous mode or writing to internal object's buffer) at a value set, for direct writing to physical controller or object's buffer.
- *virtual void vlGet(TVal &val);* — Call for attribute into direct read mode *TVal::DirRead* (synchronous mode or reading from internal object's buffer) at a value get, for direct reading from physical controller or object's buffer.
- *virtual void vlArchMake(TVal &val);* — Call at a value archive creation with attribute *<val>* as the source for quality properties initialization to archive's buffer according with specific of data source and its acquisition.

API of the modules of the "Archives" subsystem.

Archiver's type (child from TTipArchivator):

- *virtual TMArchivator *AMess(const string &id, const string &db);* — Creation of the messages' archive.
- *virtual TVArchivator *AVal(const string &id, const string &db);* — Creation of the messages' archiver.

Messages' archiver (child from TMArchivator):

- *virtual void start( );* — Archiver's start.
- *virtual void stop( );* — Archiver's stop.
- *virtual time_t begin( );* — The beginning of the data in the archive.
- *virtual time_t end( );* — The end of the data in the archive.
- *virtual void put(vector<TMess::SRec> &mess);* — Put the message to the archiver.
- *virtual void get(time_t b_tm, time_t e_tm, vector<TMess::SRec> &mess, const string &category = "", char level = 0, time_t upTo = 0);* — Get the message from the archiver.

Values' archiver (child from TVArchivator):

- *virtual void setValPeriod(double per);* — Set the periodicity of the archiver's values.
- *virtual void setArchPeriod(int per);* — Set the archiving periodicity.
- *virtual void start( );* — Start the archiver.
- *virtual void stop(bool full_del = false);* — Stop the archiver with the possibility of complete deleting if the *<full_del>* flag is set.
- *virtual TVArchEl *getArchEl(TVArchive &arch);* — Getting the *<arch>* archive, which is served by the archiver.

An archive element of values (child from TVArchEl):

- *virtual void fullErase( );* — Complete deleting of the part of archive in the archiver.
- *virtual int64_t end( );* — End time of the archive in the archiver.
- *virtual int64_t begin( );* — Start time of the archive in the archiver.
- *virtual TVariant getValProc(int64_t *tm, bool up_ord);* — Request processing function of the single value from the archive.
- *virtual void getValsProc(TValBuf &buf, int64_t beg, int64_t end);* — Function of the request processing by the module to get data group values <*buf*> for set time interval.
- *virtual void setValsProc(TValBuf &buf, int64_t beg, int64_t end);* — Function of the request processing by the module to set data values <*buf*> for set time interval.

API of the modules of subsystem "User interfaces".

User interface (child from TUI):
It does not contain specific functions!

API модулей подсистемы "Специальные”.

Specials (child from TSpecial):
It does not contain specific functions!

20. Debugging and Testing the OpenSCADA project

To monitor the quality of code and test the performance of various parts of the system the special modules are written that perform testing with the issue of testing protocol. These modules must be executed after the completion of any part of the project.

21. Rules for design and commenting of the sources of OpenSCADA and its modules

When writing and design of the sources of the OpenSCADA system and its modules you must to follow the rules:

- indent between the levels of enclosure: 4 characters;
- braces of opening and closing must be placed in separate lines at the level of the previous text;
- it is possible to write an enclosure in the single line with the previous level of enclosure, in case of increasing the readability of the code;;
- distance between the descriptions of the functions of at least one character;
- the distance between the definition of variables and the text of the program at least one symbol;
- it is possible to define variables in the text while saving the readability;
- to avoid long lines more than 100 characters;
- preprocessor commands must be located on the first level, regardless of the current level of the text;
- for the formatting the source code, inherited from the other free applications and examples, it is recommended to use the utility:

  indent -bli0 -i4 -l100 -npsl -npcs -prs -nsaf -nsai -ts8 <filename>.

Commenting rules of the OpenSCADA sources:

- obligatory commenting and thorough description is necessary for class declarations;
- declarations of public methods of the class should be thoroughly described with the individual description of each parameter;
- declarations of public attributes should also be thoroughly commented;
- text functions do not need to be thoroughly commented, but the implicit places are to be commented.

22. Conventional signs in the text and source code

???? — doubt in the usefulness of this section;

?!?! — the section is not fully implemented;

!!!! — the section requires a rethinking.