А-Софт
Функциональные характеристики

АННОТАЦИЯ

Настоящий документ содержит описание внутренней архитектуры программной платформы «СКАДА А-СОФТ» (далее по тексту СКАДА).

СОДЕРЖАНИЕ

1 Общие сведения

ППО «СКАДА А-СОФТ» поставляется в виде изделия программного.

Программные требования, необходимые для функционирования ПП «СКАДА А-СОФТ»:

- операционная система Астра Linux SE Smolensk 1.4 (x64) и выше;

2 Функциональное назначение

ПП «СКАДА А-СОФТ» предназначена для разработки и исполнения прикладного программного обеспечения сбора, архивирования, визуализации информации, выдачи управляющих воздействий, а также других родственных операций, характерных для полнофункциональной СКАДА.

3 Описание логической структуры ПП «СКАДА А-СОФТ»

С целью наглядного и доступного восприятия архитектуры СКАДА в целом на рисунке 1 изображена статическая диаграмма классов системы на универсальном языке моделирования (UML). Исходя из диаграммы видно, что СКАДА содержит модульные подсистемы: «Архивы», «Базы данных», «Транспорты», «Транспортные протоколы», «Пользовательские интерфейсы», «Сбор данных» и «Специальные», а также подсистемы: «Безопасность» и «Управление модулями». На диаграмме наглядно представлены взаимосвязи между модульными подсистемами и модулями соответствующих типов.

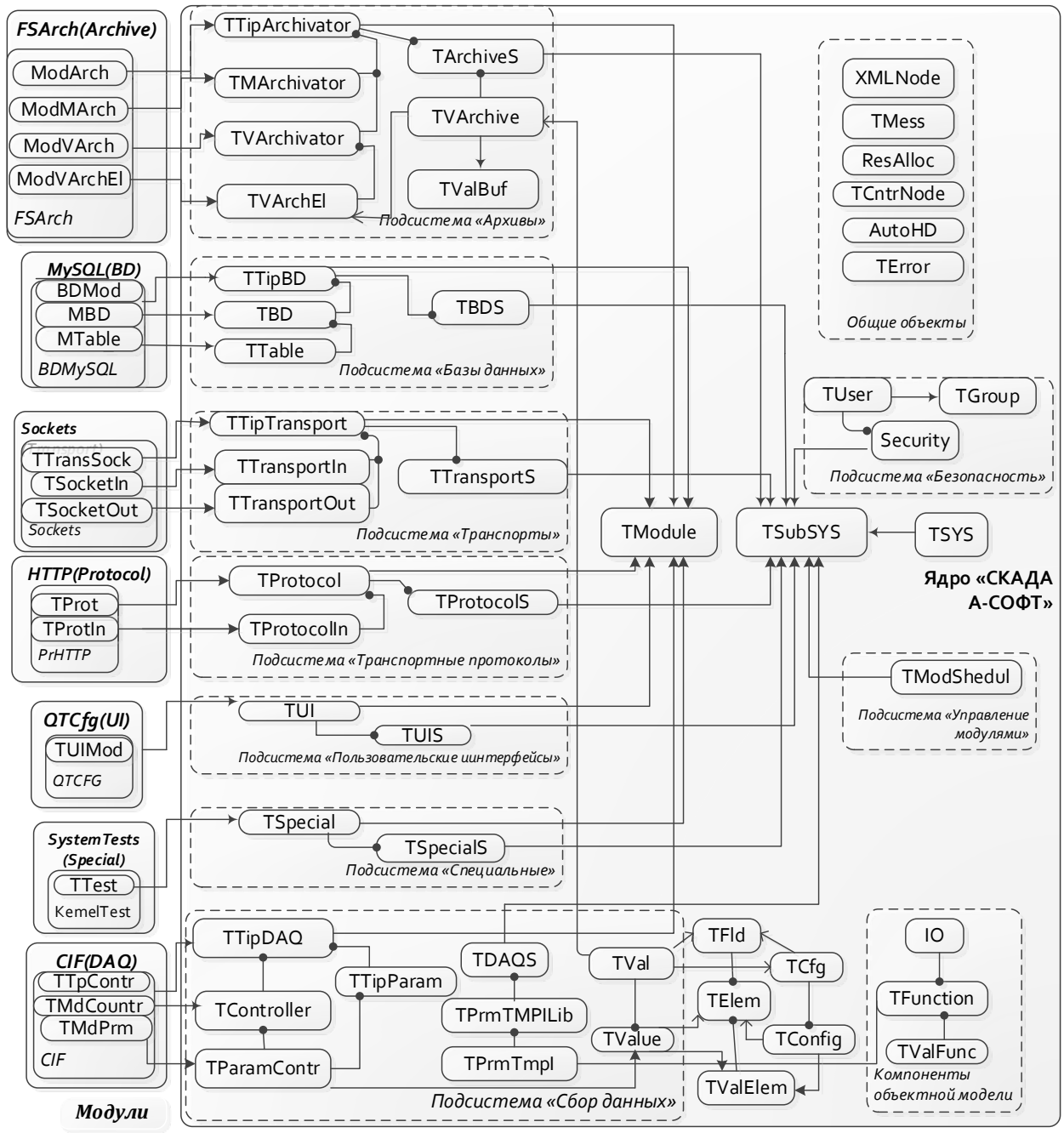

Рисунок 1 Статическая диаграмма классов

3.1 Общая структура системы. Модульность (TSubSYS, TModule)

Корнем, от которого строится вся система, является объект TSYS. Корень содержит подсистемы (TSubSYS). Подсистемы могут быть: обычными и модульными. Модульные подсистемы содержат список модульных объектов (TModule), например, подсистема Архивы (TArchiveS) содержит модульные объекты

тип архива (TTipArchivator), а подсистема безопасности TSecurity является обычной (рисунок 2).

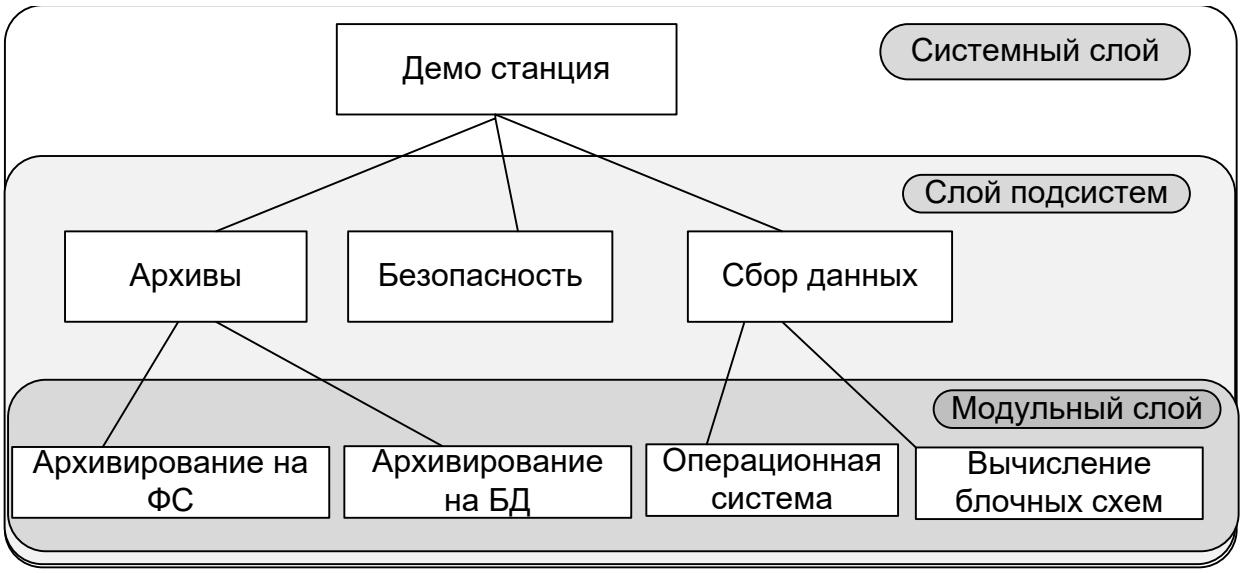

Рисунок 2 Иерархическая структура СКАДА.

В процессе инициализации корня (TSYS) определяется глобальная переменная SYS. Переменная SYS может использоваться для прямого обращения к корню системы из любого её узла. Инициализация корня выполняется единожды из главной вызывающей функции. После запуска управление захватывается объектом системы до остановки. Корневой объект концентрирует все общесистемные функции СКАДА.

Продолжением корневого объекта (TSYS), выполняющего функции обслуживания потока системных сообщений, выступает объект TMess. Объект доступен посредством глобальной переменной Mess, которая инициализируется корнем системы. Объект содержит функции кодирования, декодирования и локализации сообщений.

В подсистемах (TSubSYS) реализуются функции характерные для каждой подсистемы индивидуально, с общим для всех подсистем доступом, через объект TSubSYS. Модульная подсистема имеет возможность расширять функциональность посредством модулей, предоставляя доступ к модулям своего типа в виде

модульных объектов.

Модуль — составная часть модульной подсистемы. Для всех модулей и их подсистем, модуль предоставляет информацию о себе, своём происхождении и экспортируемых функциях. Отдельно взятый модуль реализует функциональность в соответствии со своими потребностями.

3.1.1 *Корневой объект системы (TSYS)*

Наследует: *TCntrNode*.

Данные:

Информационные переменные программы:

– PACKAGE_LICENSE – лицензия распространения программы;

– PACKAGE_DESCR – краткое описание программы;

– PACKAGE_AUTHOR – автор программы;

– PACKAGE_SITE - Web сайт поддержки программы.

Способы кодирования символьных последовательностей (enum — `TSYS::Code`):

– PathEl – элемент пути (символы: '/' и '%' к виду '%2f');

– HttpURL – адрес браузера (http url);

– Html – специальных символов для использования в html;

– JavaSc – символов конца строки для JavaScript;

– SQL – значения SQL-запросов;

– Custom – выборочное кодирование указанных символов;

– base64 – Mime кодирование в стандарте Base64;

– FormatPrint – кодирование/экранирование элементов форматирования для функций вроде "printf";

– oscdID – кодирование идентификаторов узлов;

– Bin – кодирование бинарных данных в текст и обратно;

– Reverse – инверсия порядка символов в строке;

– ShieldSimb – кодирование экранированных символов вроде "\n" в реальные коды.

Виды представления целого в функциях `TSYS::int2str()` и `TSYS::ll2str()` (enum — TSYS::IntView):

– Dec – десятичное;

– Oct – восьмеричное;

– Hex – шестнадцатеричное.

Стандартные коды ошибок в СКАДА (enum – TSYS::Errors):

– DBInit (1) – ошибка инициализации БД;

– DBConn (2) – ошибка подключения к БД;

– DBInernal (3) – внутрення ошибка БД;

– DBRequest (4) – ошибка в запросе к БД;

– DBOpen (5) – ошибка открытия БД;

– DBOpenTable (6) – ошибка открытия таблицы;

– DBClose (7) – ошибка закрытия БД;

– DBTableEmpty (8) – таблица БД пуста;

– DBRowNoPresent (9) – запись в таблице отсутствует.

Шаблоны:

– TO_FREE – значение свободного объекта (NULL).

– STR_BUF_LEN – стандартная длина строковых буферов в SCADA (3000);

– STD_WAIT_DELAY – стандартный квант времени циклов ожидания (100мс);

– STD_WAIT_TM – стандартный интервал ожидания события;

– __func__ – полное имя вызывающей функции;

– vmin(a,b) – определение минимального значения;

– vmax(a,b) – определение максимального значения.

Публичные методы:

– `TSYS(int argi, char **argb, char **env);` – инициализирующий конструктор;

– `int start();` – запуск системы (функция завершается только с завершением работы системы, возвращается код возврата);

– `void stop();` – команда остановки системы;

– `int stopSignal();` – код возврата в случае останова системы (может использоваться как признак «Останов системы» различными подсистемами);

– `string id();` – идентификатор станции;

– `string name();` – локализованное имя станции;

– `string user();` – системный пользователь, от имени которого запущена система;

– `string host();` – имя хоста, на котором исполняется станция;

– `void list(vector<string> &list);` – список подсистем зарегистрированных в системе;

– `bool present(const string &name);` – проверка на наличие указанной подсистемы;

– `void add(TSubSYS *sub);` – добавление/регистрация подсистемы;

– `void del(const string &name);` – удаление подсистемы;

– `AutoHD<TSubSYS> at(const string &name);` – подключение к указанной подсистеме;

– `AutoHD<TUIS> ui();` – прямой доступ к подсистеме «Пользовательские интерфейсы»;

– `AutoHD<TArchiveS> archive();` – прямой доступ к подсистеме «Архивы»;

– `AutoHD<TBDS> db();` – прямой доступ к подсистеме «Базы данных»;

– `AutoHD<TControllerS> daq();` – прямой доступ к подсистеме «Сбор данных»;

– `AutoHD<TProtocolS> protocol();` – прямой доступ к подсистеме «Протоколы»;

– `AutoHD<TTransportS> transport();` – прямой доступ к подсистеме «Транспорты»;

– `AutoHD<TSpecialS> special();` – прямой доступ к подсистеме «Специальные»;

– `AutoHD<TModSchedul> modSchedul();` – прямой доступ к подсистеме «Управление модулями»;

– `AutoHD<TSecurity> security();` – Прямой доступ к подсистеме «Безопасность»;

– `string workDir();` – рабочая директория станции;

– `string icoDir();` – директория иконок СКАДА;

– `string modDir();` – директория модулей СКАДА;

– `void setWorkDir(const string &wdi);` – установка рабочей директории станции;

– `void setIcoDir(const string &idir);` – установка директории иконок СКАДА;

– `void setModDir(const string &mdir);` – установка директории

модулей СКАДА;

– `string cfgFile();` – имя конфигурационного файла системы;

– `XMLNode &cfgRoot();` – разобранная структура конфигурационного файла;

– `XMLNode *cfgNode(const string &path, bool create = false);` – получение узла конфигурации по его пути <path> – создавать элементы пути в случае их отсутствии <create>;

– `void modifCfg();` – отметка модификации конфигурации, для последующего сохранения в файл;

– `string workDB();` – полное имя рабочей БД;

– `string selDB();` – выбранная БД (используется для избирательной загрузки из указанной БД, в подсистеме "БД");

– `bool chkSelDB(const string& wDB);` – функция проверки на соответствие указанной БД <wDB> выбранной в selDB();

– `void setWorkDB(const string &wdb);` – установка полного имени рабочей БД;

– `void setSelDB(const string &vl);` – установка выбранной БД для избирательной загрузки;

– `bool saveAtExit();` – признак "Сохранять конфигурацию системы при выходе";

– `void setSaveAtExit(bool vl);` – установка признака "Сохранять конфигурацию системы при выходе";

– `int savePeriod();` – периодичность автоматического сохранения станции в БД (секунд);

– `void setSavePeriod(int vl);` – установка периодичности автомати-ческого сохранения станции в БД (секунд);

– `string optDescr();` – локализованная помощь по опциям командной строки и параметрам конфигурационного файла;

– `static void sighandler(int signal);` – функция стандартного обработчика сигналов системы в целом;

– `unsigned long long sysClk();` – Расчётная частота процессора на котором функционирует система (Гц);

– `void clkCalc();` – Расчёт частоты процессора на котором работает

система; Вызывается периодически для систем с переменной частотой процессора;

– `unsigned long long shrtCnt();` – Функция замера малых интервалов времени по счетчику тактов процессора. Возвращает значение счетчика тактов процессора;

– `static long HZ();` – Время системного тика процессора;

– `bool cntrEmpty();` – Проверка на отсутствие счётчиков отладки;

– `double cntrGet(const string &id);` – Получение счётчика отладки <id>;

– `void cntrSet(const string &id, double vl);` – Установка счетчика отладки <id> в значение <vl>;

– `void cntrIter(const string &id, double vl);` – Итерация счётчика отладки <id> на значение <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);` – создание задачи (потока) с идентификатором <path>, приоритетом <priority> (-1...99), функцией задачи <start_routine> и её аргументов <arg>, а так-же ожиданием запуска пользовательской процедуры по признаку <startSt>;

– `void taskDestroy(const string &path, bool *endrunCntr = NULL, int wtm = 5, bool noSignal = false);` – удаление задачи с идентификатором <path> и командой останова <endrunCntr> (используйте <noSignal> для предотвращения отправки задаче сигнала SIGALRM);

– `static int sysSleep(float tm);` – функция системного засыпания потока на время в секундах, вплоть до наносекунд (1e-9).

– `static long long curTime();` – текущее время в микросекундах с начала эпохи (01.01.1970);

– `static void taskSleep(long long per, time_t cron = 0);` – функция засыпания потока по сетке абсолютного времени с периодом <per> в наносекундах или до запланированного времени <cron>;

– `static time_t cron(const string &vl, time_t base = 0);` – планирование времени исполнения по формату стандарта Cron <vl>, начиная от базового времени <base> или от текущего времени, если база не указана.

– `static bool eventWait(bool &m_mess_r_stat, bool`

exempl, const string &loc, time_t time = 0); – функция ожидания события <exempl> для переменной <m_mess_r_stat> в течение указанного интервала времени <time> для источника <loc>;

– static string int2str(int val, IntView view = Dec); – преобразование целого знакового в строку вида <view>;

– static string uint2str(unsigned val, IntView view = Dec); – преобразования целого беззнакового в строку вида <view>;

– static string ll2str(long long val, IntView view = Dec); – преобразования длинного целого (64бит) в строку вида view;

– static string real2str(double val, int prec = 15, char tp = 'g'); – преобразования вещественного с точностью <prec> знаков и типом <tp> в строку;

– static double realRound(double val, int dig = 0, bool toint = false); – округление вещественного числа до указанного знака <dig> после запятой с возможностью преобразования к целому после округления <toint>;

– static string time2str(time_t tm, const string &format); – преобразование UNIX времени <tm> в строку, в соответствии с форматом <format> POSIX-функции strftime();

– static string time2str(double utm); – преобразование интервала времени, в микросекундах, в строку вида "1час 23мин 10сек";

– static string cpct2str(double cnt); – преобразование счётчика трафика <cnt> (байт) в строки вида "12.5КиБ";

– static string addr2str(void *addr); – преобразование адреса в строку;

– static void *str2addr(const string &str); – преобразование строки в адрес;

– static string strNoSpace(const string &val); – удаляет из исходной строки <val> пустые символы в начале и в конце;

– static string strSepParse(const string &path, int level, char sep, int *off = NULL); – разбор строки <path> на составляющие, отделённые разделительным символом <sep>, начиная со смещения <off> и контролируя смещение конца элемента в нём же:

– static string strParse(const string &str, int level,

`const string &sep, int *off = NULL, bool mergeSepSymb = false);` – расширеная версия функции разбора строки strSepParse() позволяющая использовать многосимвольные разделители и объединять односимвольные.

– `static string strLine(const string &str, int level, int *off = NULL);` – разбор текста по строкам для разных способов окончания строки (CR, LF и CR/LF);

– `static string pathLev(const string &path, int level, bool encode = true, int *off = NULL);` – выделение элементов пути <path> с возможностью их декодирования, начиная со смещения <off> и контролируя смещение конца элемента в нём же;

– `static string path2sepstr(const string &path, char sep = '.');` – преобразование пути в строку с разделителем <sep> элементов;

– `static string sepstr2path(const string &str, char sep = '.');` – преобразование строки с разделителем <sep> элементов в путь;

– `static string strEncode(const string &in, Code tp, const string &symb = " \t\n");` – кодирование строки по указанному правилу <tp>;

– `static string strDecode(const string &in, Code tp = Custom);` – декодирование строки по указанному правилу <tp>;

– `static string strMess(const char *fmt, ...);` – формирование стоки по шаблону <fmt> и аргументам, (реализован на основе printf);

– `string strCompr(const string &in, int lev = -1);` – компрессия строки <in> с уровнем компрессии <lev>;

– `string strUncompr(const string &in);` – декомпрессия строки <in>;

– `static inline uint16_t getUnalign16(const void *p);` – невыравненное чтение беззнакового целого в 16-разрядов из буфера по смещению;

– `static inline uint32_t getUnalign32(const void *p);` – невыравненное чтение беззнакового целого в 32-разряда из буфера по смещению;

– `static inline uint64_t getUnalign64(const void *p);` – невыравненное чтение беззнакового целого в 64-разряда из буфера по смещению;

– `static inline int getUnalignInt(const void *p);` – невыравненное чтение знакового целого из буфера по смещению;

– `static inline float getUnalignFloat(const void *p);` – невыравненное чтение вещественного "float" из буфера по смещению;

– `static inline double getUnalignDbl(const void *p);` – невыравненное чтение вещественного "double" из буфера по смещению;

– `static float floatLE(float in);` – преобразование вещественного числа "float" из внутреннего представления в формат IEEE754 Little-Endian (LE);

– `static float floatLErev(float in);` – преобразование вещественного числа "float" из формата IEEE754 Little-Endian (LE) во внутреннее представление;

– `static double doubleLE(double in);` – преобразование вещественного числа "double" из внутреннего представления в формат IEEE754 Little-Endian (LE);

– `static double doubleLErev(double in);` – преобразование вещественного числа "double" из формата IEEE754 Little-Endian (LE) во внутреннее представление.

Публичные атрибуты:

static bool finalKill – признак "Финальное разрушение объектов" - используется для принудительного отключения заблокированных объектов на финальной стадии выключения;

const int argc – счётчик аргументов командной строки;

const char **argv – буфер аргументов командной строки;

const char **envp – указатель на список параметров окружения.

3.1.2 *Объект сообщений системы (TMess)*

Данные:

Типы (уровни) сообщений (enum – TMess::Type):

- Debug – отладка;
- Info – информация;
- Notice – замечание;
- Warning – предупреждение;
- Error – ошибка;
- Crit – критическая ситуация;

– Alert – тревога;

– Emerg – авария.

Структура сообщения (class – TMess::SRec):

– time_t time; – время сообщения;

– int utime; – микросекунды времени сообщения;

– string categ; – категория сообщения (обычно путь внутри системы);

– Type level; – уровень сообщения;

– string mess; – сообщение.

Шаблоны:

– _(mess) – Обёртка над функцией трансляции сообщений для предоставления принятого во многих программах вызова перевода сообщений;

– FTM(rec) – получения полного времени сообщения в микросекундах, используя два поля времени структуры сообщения;

– message(cat,lev,fmt,args...) – формироване полного сообщения;

– mess_debug(cat,fmt,args...) – формирование отладочного сообщения.

– mess_info(cat,fmt,args...) – формирование информационного сообщения;

– mess_note(cat,fmt,args...) – формирования сообщения - замечания;

– mess_warning(cat,fmt,args...) – формирование предупредительного сообщения;

– mess_err(cat,fmt,args...) – формирование сообщения ошибки;

– mess_crit(cat,fmt,args...) – формирование сообщения критического состояния;

– mess_alert(cat,fmt,args...) – формирование сообщения тревоги;

– mess_emerg(cat,fmt,args...) – формирование сообщения аварии.

Публичные методы:

– `void load();` – загрузка.

– `void save();` – сохранение.

– `string codeConv(const string &fromCH, const string &toCH, const string &mess);` – конвертация кодировки сообщения.

– `string codeConvIn(const string &fromCH, const string &mess);` – конвертация кодировки сообщения во внутреннюю кодировку системы.

– `string codeConvOut(const string &toCH, const string &mess);` – конвертация кодировки сообщения из внутренней кодировки системы.

– `static const char *I18N(const char *mess, const char *d_name = NULL);` – получение сообщения на языке системы.

– `static string I18Ns(const string &mess, const char *d_name = NULL);` – получение сообщения на языке системы.

– `string lang();` – язык системы, в виде en_US.UTF-8.

– `string lang2Code();` – язык системы в двухсимвольной кодировке (en).

– `string lang2CodeBase();` – язык базовых сообщений текстовых переменных в двухсимвольной кодировке (en).

– `string &charset();` – системная кодировка.

– `int logDirect();` – приемники, которым направляются системные сообщения (stdout, stderr, syslog, archive);

– `int messLevel();` – уровень, ниже которого сообщения игнорируются.

– `bool isUTF8();` – внутренняя кодировка UTF-8.

– `void setLang(const string &lang);` – установка языка системы (локализации).

– `void setLang2CodeBase(const string &vl);` – установка языка базовых сообщений текстовых переменных в двухсимвольной кодировке (en).

– `void setLogDirect(int dir);` – установка приемников которым направляются системные сообщения. Для <dir> используется битовая маска. Где:

1 – в syslog;

2 – в stdout;

4 – в stderr;

8 – в архив.

– `void setMessLevel(int level);` – установка минимального уровня обрабатываемых сообщений.

– `void put(const char *categ, int8_t level, const char *fmt, ...);` – сформировать сообщение за текущее время.

– `void get(time_t b_tm, time_t e_tm, vector<TMess::SRec> &recs, const string &category = "", int8_t level = Debug);` – запросить сообщения из архива за промежуток времени <b_tm> -<e_tm> в соответствии с шаблоном категории <category> и минимальным уровнем <level>.

3.1.3 *Объект подсистемы (TSubSYS)*

Наследует: *TCntrNode. TArchiveS, TProtocolS, TBDS, TFunctionS, TSecurity, TModShedul, TTransportS, TUIS.*

Наследуется:

TSpecialS, TControllerS.

Публичные методы:

– `TSubSYS(const char *id, const char *name, bool mod = false);` – инициализирующий конструктор, признак <mod> указывает, что подсистема модульная;

– `string subId();` – идентификатор подсистемы;

– `string subName();` – локализованное имя подсистемы;

– `bool subStartStat();` – признак исполнения подсистемы;

– `bool subModule();` – признак модульности подсистемы;

– `virtual int subVer();` – версия подсистемы;

– `virtual void subStart();` – запуск подсистемы;

– `virtual void subStop();` – останов подсистемы;

– `void modList(vector<string> &list);` – список <list> модулей модульной подсистемы;

– `bool modPresent(const string &name);` – проверка на наличие указанного модуля <name>;

– `void modAdd(TModule *modul);` – добавление/регистрация модуля <modul>;

– `void modDel(const string &name);` – удаление модуля <name>;

– `AutoHD<TModule> modAt(const string &name);` – подключение к модулю <name>;

– `virtual void perSYSCall(unsigned int cnt);` – периодичный вызов из системного потока, с периодичностью 10 секунд и секундным счётчиком <cnt>;

– `TSYS &owner();` – система - владелец подсистемы.

3.1.4 *Объект модуля (TModule)*

Наследует:

TCntrNode.

Наследуется:

TProtocol, TTipBD, TTipArchive, TTipTransport, TUI, Tspecial, TTipController.

Данные:

Структура данных идентифицирующая модуль (class – TModule::SAt):

– `SAt(const string &iid, const string &itype = "", int itver = 0);` – инициализирующий конструктор;

– `bool operator==(const TModule::SAt &amst) const;` – функция сравнения идентификаторов модулей;

– `string id;` – идентификатор модуля;

– `string type;` – тип модуля (подсистема);

– `int t_ver;` – версия типа модуля (подсистемы) для которой модуль разработан.

Структура экспортируемых функций (class – TModule::ExpFunc):

– `string prot;` – прототип функции;

– `string dscr;` – локализованное описание функции;

– `void (TModule::*ptr)();` – относительный адрес функции (относительно объекта модуля).

Публичные методы:

– `TModule(const string &id);` – инициализирующий конструктор модуля <id>;

– `string modId();` – идентификатор модуля;

– `string modName();` – локализованное имя модуля;

– `virtual void modStart();` – запуск модуля;

– `virtual void modStop();` – останов модуля;

– `virtual void modInfo(vector<string> &list);` – список <list> информационных элементов модуля;

– `virtual string modInfo(const string &name);` – получение содержимого указанного информационного элемента <name>;

– `virtual void perSYSCall(unsigned int cnt);` – периодичный вызов из системного потока, с периодичностью 10 секунд и секундным счётчиком

<cnt>;

– `void modFuncList(vector<string> &list);` – список <list> экспортируемых функций модуля;

– `bool modFuncPresent(const string &prot);` – проверка на наличие указанной функции по её прототипу <prot>;

– `ExpFunc &modFunc(const string &prot);` – получить информацию об экспортируемой функции модуля <prot>;

– `void modFunc(const string &prot, void (TModule::**offptr)());` – получение относительного адреса <offptr> экспортируемой функции <prot>;

– `const char *I18N(const char *mess);` – локализация модульного сообщения <mess> в соответствии с текущей локалью;

– `TSubSYS &owner();` – подсистема - владелец модуля.

Защищённые атрибуты:

– string mName; – имя модуля;

– string mDescr; – описание модуля;

– string mType; – тип модуля;

– string mVers; – версия модуля;

– string mAutor; – автор модуля;

– string mLicense; – лицензия модуля;

– string mSource; – источник/происхождение модуля.

Защищённые методы:

`void modFuncReg(ExpFunc *func);` – регистрация экспортируемых модулем функций.

3.2 Подсистема “Базы Данных” (TBDS)

Подсистема «Базы Данных» представлена объектом TBDS, который содержит модульные объекты типов БД TtipBD. Каждый тип базы данных содержит объекты отдельно взятых баз данных данного типа TBD. Каждая БД, в свою очередь, содержит объекты своих таблиц TTable (рисунок 3).

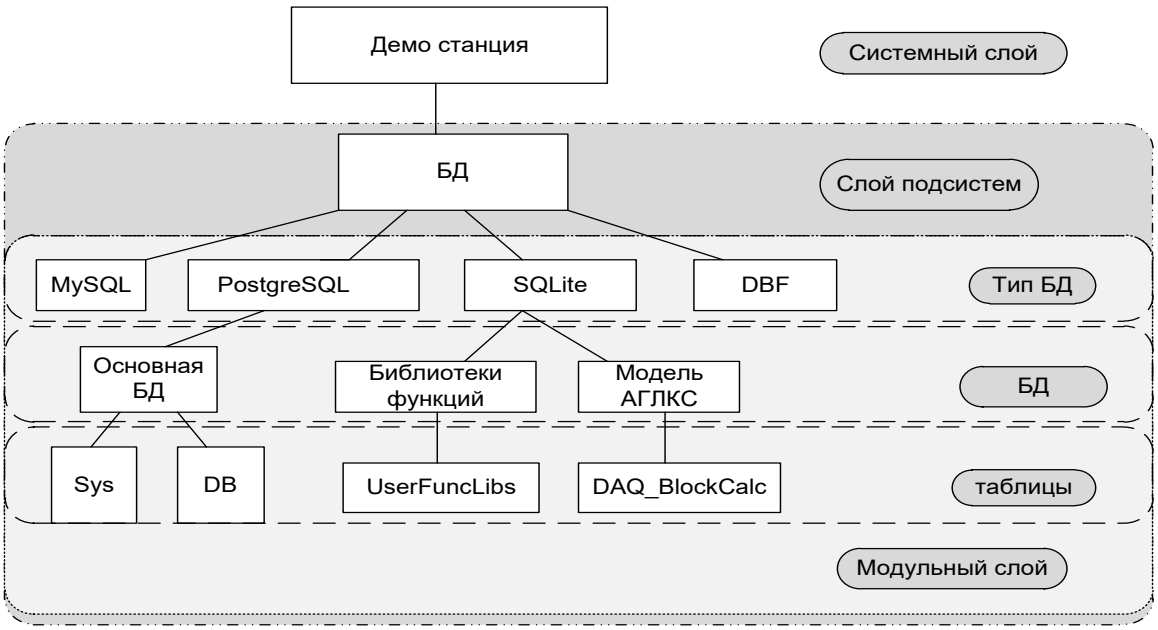

Рисунок 3. Иерархическая структура подсистемы БД

Подсистема представляет базовые функции для доступа к типам БД, а также обобщающие функции для работы с базами данных и таблицами. Так, для сокрытия источника данных, которым может быть и конфигурационный файл, используются функции абстрактного доступа к источнику данных. А для хранения общесистемных данных – системная таблица и функции абстрактного доступа к ней. Следовательно, общесистемные данные могут храниться как в конфигурационном файле, так и в таблице БД. Приоритетным источником в таком случае является таблица БД.

Являясь модульным объектом, тип БД (TTipBD) содержит доступ к реализации механизма той или иной БД. Доступ производится посредством открытых БД модуля отдельно взятого типа БД. Открываемые/регистрируемые БД описываются в таблице открываемых БД или в конфигурационном файле. Существует так называемая рабочая БД, которая открывается всегда и указывается в конфигурационном файле. БД, поддерживающие SQL-запросы, могут предоставлять доступ, основанный на прямых SQL-запросах.

В процессе использования компоненты СКАДА открывают таблицы (TTable) в доступных БД и работают с ними.

3.2.1 *Объект подсистемы «Базы Данных» (TBDS)*

Наследует:

TSubSYS, TElem;

Данные:

Флаги запросов к системной таблице (enum – `TDBS::ReqGen`):

– OnlyCfg – запрос только к конфигурационному файлу;

– UseTranslate – использовать перевод текстовой переменной;

Публичные методы:

– `int subVer();` – версия подсистемы;

– `static string realDBName(const string &bdn);` – преобразование полного шаблонного имени БД или таблицы (вида *;*;myTbl) в реальное имя, фактически выполняется замена специальных элементов '*' на элементы рабочей БД;

– `void dbList(vector<string> &ls, bool checkSel = false);` – список доступных БД, <checkSel> указывает на необходимость проверки факта загрузки из выбранной БД и вставки в список БД только выбранной;

– `AutoHD<TTable> open(const string &bdn, bool create = false);` – открытие таблицы <bdn> БД по её полному пути с созданием <create> в случае отсутствия;

– `void close(const string &bdn, bool del = false);` – закрытие таблицы <bdn> БД по её полному пути с возможностью удаления после закрытия <del>;

– `bool dataSeek(const string &bdn, const string &path, int lev, TConfig &cfg);` – общее сканирование записей источника данных. В качестве источника выступает конфигурационный файл или БД. В случае отсутствия БД используется конфигурационный файл. Если имя БД <bdn> или путь <path> конфигурационного файла не указаны, то их обработка пропускается.

– `bool dataGet(const string &bdn, const string &path, TConfig &cfg);` – получение записи из источника данных (БД или конфигурационный файл). Если имя БД <bdn> или путь <path> конфигурационного файла не указаны, то их обработка пропускается;

– `void dataSet(const string &bdn, const string &path,`

`TConfig &cfg);` – установить/сохранить запись в источнике данных (БД или конфигурационный файл). Если имя БД <bdn> или путь <path> конфигурационного файла не указаны, то их обработка пропускается;

– `bool dataDel(const string &bdn, const string &path, TConfig &cfg, bool useKeyAll = false);` – удаление записи из источника данных (БД или конфигурационный файл). Если имя БД <bdn> или путь <path> конфигурационного файла не указаны, то их обработка пропускается, <useKeyAll> используется для указания необходимости установки всех ключей для использования их при удалении с восстановлением исходного состояний выбора ключей при выходе из функции. Если этот флаг не установлен, то используются ранее выбранные ключи для выполнения операции;

– `static string genDBGet(const string &path, const string &oval = "", const string &user = "root", char rFlg = 0);` – получить общесистемные данные из конфигурационного файла или системной таблицы от имени пользователя <user>. Если данные отсутствуют, то возвращается значений <oval>;

– `static void genDBSet(const string &path, const string &val, const string &user = "root", char rFlg = 0);` – установить/сохранить общесистемные данные в конфигурационном файле или системной таблице от имени пользователя <user>;

– `string fullDBSYS();` – полное имя системной таблицы;

– `string fullDB();` – полное имя таблицы с описанием зарегистрированных БД;

– `TElem &openDB_E();` – структура таблицы зарегистрированных БД;

– `AutoHD<TTipBD> at(const string &iid);` – обращение к модулю БД (типу БД);

– `string optDescr();` – локализованная помощь по опциям командной строки и параметрам конфигурационного файла.

3.2.2 *Модульный объект типов баз данных (TTipBD)*

Наследует:

Tmodule.

Наследуется:

Корневыми объектами модулей подсистемы «БД».

Публичные методы:

– `bool fullDeleteDB();` – признак полного удаления БД.

– `void list(vector<string> &list);` – список зарегистрированных (открытых) БД;

– `bool openStat(const string &idb);` – проверка на наличие указанной открытой БД;

– `void open(const string &iid);` – открытие БД;

– `void close(const string &iid, bool erase = false);` – закрытие БД (если установлен признак <erase>, то БД будет полностью удалена);

– `AutoHD<TBD> at(const string &name);` – подключение к открытой БД;

– `TBDS &owner();` – подсистема - владелец модуля.

3.2.3 *Объект базы данных (TBD)*

Наследует:

TCntrNode, Tconfig.

Наследуется:

Объектами баз данных модулей подсистемы «БД».

Публичные методы:

– `TBD(const string &iid, TElem *cf_el);` – инициализирующий конструктор;

– `string id();` - идентификатор БД;

– `string name();` – имя БД;

– `string dscr();` – описание БД;

– `string addr();` – адрес БД (форма записи отлична для каждого типа БД);

– `string codePage();` – кодовая страница, в которой хранятся данные БД;

– `bool enableStat();` – состояние БД: "Включена";

– `bool toEnable();` – признак БД: "Включать";

– `void setName(const string &inm);` – установка имени БД;

– `void setDscr(const string &idscr);` – установка описания БД;

– `void setAddr(const string &iaddr);` – установка адреса БД;

– `void setCodePage(const string &icp);` – установка кодовой страницы хранения данных в БД;

– `void setToEnable(bool ivl);` – установка признака: "Включать";

– `virtual void enable();` – включение БД;

– `virtual void disable();` – отключение БД;

– `virtual void allowList(vector<string> &list);` – список таблиц, содержащихся в данной БД;

– `void list(vector<string> &list);` – список открытых таблиц;

– `bool openStat(const string &table);` – признак указывающий на то, что запрошенная таблица открыта;

– `void open(const string &table, bool create);` – открытие таблицы (если установлен признак <create>, то в случае отсутствия таблица будет создана);

– `void close(const string &table, bool del = false);` – закрытие таблицы (если установлен признак <del>, то таблица будет полностью удалена);

– `AutoHD<TTable> at(const string &name);` – подключение к таблице;

– `virtual void sqlReq(const string &req, vector<vector <string>> *tbl = NULL, char intoTrans = EVAL_BOOL);` – отправка SQL-запроса <req> на БД и получение результата в виде таблицы <tbl>. При установке <intoTrans> в true для запроса будет открыта транзакция, в false будет закрыта;

– `virtual void transCloseCheck();` – периодически вызываемая функция для проверки транзакций и закрытия старых или содержащих много запросов;

– `TTipBD &owner();` – тип базы данных – владелец данной БД;

Защищённые методы:

– `virtual TTable *openTable(const string &table, bool create);` – модульный метод открытия таблицы.

3.2.4 *Объект таблицы (TTable)*

Наследует:

TcntrNode.

Наследуется:

Объектами таблиц модулей подсистемы «БД».

Публичные методы:

– `TTable(const string &name);` – инициализирующий конструктор;

– `string name();` – имя таблицы;

– `virtual void fieldStruct(TConfig &cfg);` – получение структуры таблицы;

– `virtual bool fieldSeek(int row, TConfig &cfg);` – сканирование записей таблицы;

– `virtual void fieldGet(TConfig &cfg);` – запрос указанной записи, запрашиваемая запись определяется значениями ключевых ячеек исходной записи <cfg>;

– `virtual void fieldSet(TConfig &cfg);` – установка значений указанной записи, в случае отсутствия запись будет создана;

– `virtual void fieldDel(TConfig &cfg);` – удаление указанной записи;

– `TBD &owner();` – БД – владелец данной таблицы.

3.3 Подсистема “Сбор данных” (TDAQS)

Подсистема “Сбор данных” представлена объектом TDAQS, который содержит модульные объекты типов источников данных TTipDAQ и объекты библиотек шаблонов параметров подсистемы «Сбор данных» TprmTmplLib. Объект типов источников данных содержит объекты контроллеров TController и объекты типов параметров TtipParam. Объекты типов параметров предоставляются модулем контроллера и содержат структуру БД отдельных типов параметров (аналоговые, дискретные). Объекты контроллеров содержат объекты параметров TparamContr. Каждый параметр ассоциируется с одним из типов параметров. Для хранения атрибутов параметр наследуется от объекта значений TValue, который и содержит значения атрибутов Tval. Библиотека шаблонов параметров

данной подсистемы содержит объекты шаблонов TPrmTmpl. Пример описанной иерархической структуры приведён на рисунке 4.

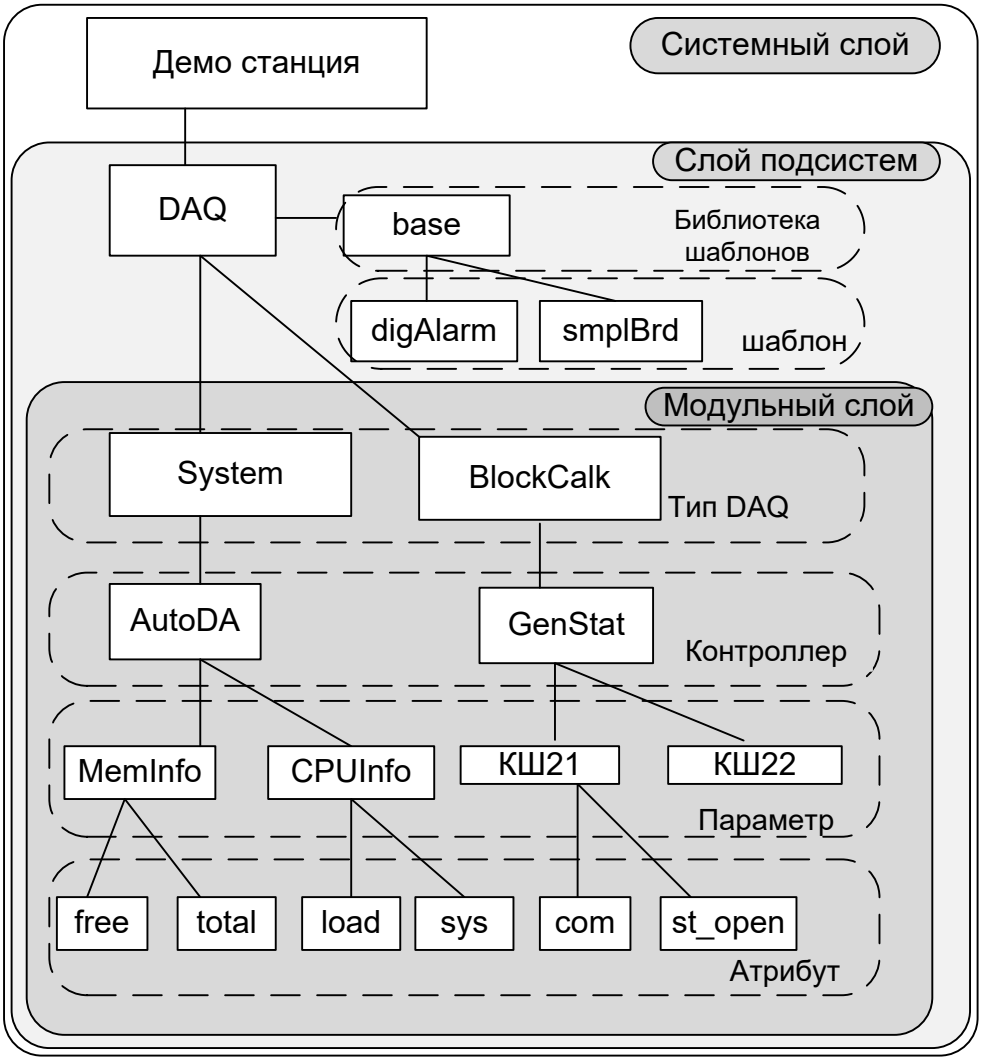

Рисунок 4. Иерархическая структура подсистемы сбора данных

Подсистема содержит типы источников данных. Тип источника может делиться на отдельные источники (контроллеры) в пределах конкретного типа. Например, если взять данные из операционной системы (ОС), то под отдельным источником можно понимать операционную систему отдельного ПК.

Источник данных (контроллер) далее делится, или содержит, параметры. Под параметром подразумевается какая-то часть источника данных. В случае с ОС это будет, например: расход оперативной памяти, частота процессора и т.д.

Параметр, в свою очередь, содержит атрибуты, которые и предоставляют данные. Кроме основных данных атрибутами могут предоставляться и сопутствующие или детализирующие данные. В случае ОС и расхода памяти, атрибутами может

предоставляться не только занятая память, а также и сколько её всего, сколько в swap и т.д.

Некоторые реализации источников данных могут предоставлять возможность формирования структуры параметра по ранее разработанным шаблонам параметров. Для этой цели подсистема содержит библиотеки шаблонов, которые, в свою очередь, содержат шаблоны параметров. В примере изображена библиотека шаблонов “base” с шаблонами “digAlrm” и “smplBrd”.

На уровне подсистемы предоставляются механизм резервирования источников данных. Резервирование подразумевает возможность согласованной работы нескольких станций СКАДА для выполнения общей задачи сбора данных в одноимённых источниках данных.

3.3.1 *Объект подсистемы «Сбор данных» (TDAQS)*

Наследует:

TSubSYS.

Публичные методы:

– `int subVer();` – версия подсистемы;

– `void subStart();` – запуск подсистемы;

– `void subStop();` – останов подсистемы;

– `AutoHD<TTipDAQ> at(const string &name);` – подключение к типу источника данных;

– `string tmplLibTable();` – имя таблицы для хранения шаблонов параметров подсистемы "Сбор данных";

– `void tmplLibList(vector<string> &list);` – список доступных шаблонов параметров;

– `bool tmplLibPresent(const string &id);` – проверка на наличие шаблона параметра <id>;

– `void tmplLibReg(TPrmTmplLib *lib);` – регистрация шаблона параметра <lib>;

– `void tmplLibUnreg(const string &id, int flg = 0);` – удаление/снятие с регистрации шаблона параметра <id>;

– `AutoHD<TPrmTmplLib> tmplLibAt(const string &id);` – подключение к шаблону параметра <id>;

– `bool rdActive();` – признак активности схемы резервирования.

указывает на факт наличия хотя-бы одной активной резервной станции;

– `int rdStLevel();` – уровень резервирования текущей станции;

– `void setRdStLevel(int vl);` – установить уровень резервирования текущей станции;

– `int rdTaskPer();` – периодичность задачи обслуживания резервирования, в секундах;

– `void setRdTaskPer(int vl);` – установить периодичность задачи обслуживания резервирования;

– `int rdRestConnTm();` – время повторения попытки восстановления связи с резервными станциями после её потери, в секундах;

– `void setRdRestConnTm(int vl);` – установка времени повторения попытки восстановления связи с резервными станциями;

– `float rdRestDtTm();` – максимальная глубина восстановления данных архивов при включении, в часах;

– `void setRdRestDtTm(float vl);` – установка максимальной глубины восстановления данных архивов при включении;

– `void rdStList(vector<string> &ls);` – список станций в резерве;

– `void rdActCntrList(vector<string> &ls, bool isRun = false);` – список активных контроллеров работающих в схеме резервирования, при указании <isRun> в список попадут только исполняющиеся на данной станции контроллеры;

– `string rdStRequest(const string &cntr, XMLNode &req, const string &prevSt = "", bool toRun = true);` – запрос <req> к резервной станции от имени контроллера <cntr>, станция для запроса выбирается после указанной в <prevSt>, для исполняемого удалённого контроллера при указании <toRun>;

– `TElem &elLib();` – структура таблицы библиотек шаблонов параметров;

– `TElem &tplE();` – структура таблицы шаблонов параметров;

– `TElem &tplIOE();` – структура атрибутов шаблонов параметров;

– `TElem &errE();` – структура атрибута(ов) ошибок параметров.

3.3.2 *Модульный объект типа контроллера (TTipDAQ)*

Наследует:

TModule, Telem.

Наследуется:

Корневыми объектами модулей подсистемы «Сбор данных».

Публичные методы:

– `string DAQPath();` – получение DAQ адреса элемента;

– `void modStart();` – запуск модуля;

– `void modStop();` – останов модуля;

– `void list(vector<string> &list);` – список контроллеров;

– `bool present(const string &name);` – проверка на наличие указанного контроллера;

– `void add(const string &name, const string &daq_db = "*;*");` – добавить контроллер;

– `void del(const string &name);` – удалить контроллер;

– `AutoHD<TController> at(const string &name, const string &who = "");` – подключиться к контроллеру;

– `bool tpPrmPresent(const string &name_t);` – проверка на наличие указанного типа параметра;

– `unsigned tpPrmToId(const string &name_t);` – получение индекса типа параметров по имени;

– `int tpParmAdd(const char *id, const char *n_db, const char *name);` – добавление/регистрация типа параметров;

– `unsigned tpPrmSize();` – количество типов параметров;

– `TTipParam &tpPrmAt(unsigned id);` – получить объект типа параметров;

– `virtual bool compileFuncLangs(vector<string> *ls = NULL);` – запрос перечня языков для которых реализована возможность формирования пользовательских процедур, в данном модуле, а также проверка факта их поддержки;

– `virtual void compileFuncSynthHighl(const string &lang, XMLNode &shgl);` – запрос правил подсветки синтаксиса для указанного языка;

– `virtual string compileFunc(const string &lang,`

TFunction &fnc_cfg, const string &prog_text, const string &usings = "", int maxCalcTm = 10); – компиляция/настройка пользовательской функции на поддерживаемом модулем языке программирования <lang> и исходном тексте процедуры <prog_text>, исходя из параметров процедуры <fnc_cfg>, результатом является адрес к подготовленному объекту функции;

– virtual bool redntAllow(); – признак поддержки механизмов резервирования модулем, должен просто переопределяться и возвращать true;

Защищённые методы:

virtual TController *ContrAttach(const string &name, const string &daq_db); – подключение контроллера - обязательно переопределяется в потомке модуля.

3.3.3 *Объект контроллера (TController)*

Наследует:

TCntrNode, Tconfig.

Наследуется:

Объектами контроллеров модулей подсистемы «Сбор данных».

Данные:

Режимы резервирования (enum TController::Redundant):

– Off – выключено;

– Asymmetric – асимметричный;

– Symmetric – симметричный.

Публичные методы:

– TController(const string &name_c, const string &daq_db, TElem *cfgelem); – инициализирующий конструктор контроллера;

– string DAQPath(); – получение DAQ адреса элемента;

– string id(); – идентификатор контроллера;

– string workId(); – рабочий идентификатор контроллера, включая идентификатор модуля;

– string name(); – имя контроллера;

– string descr(); – описание контроллера;

– virtual string getStatus(); – функция запроса статуса контроллера;

– `string DB();` – имя БД экземпляра контроллера;

– `string tbl();` – имя таблицы базы данных экземпляра контроллера;

– `string fullDB();` – полное имя таблицы экземпляра контроллера;

– `void setName(const string &nm);` – установить имя контроллера;

– `void setDescr(const string &dscr);` – установить описание контроллера;

– `void setDB(const string &idb);` – установка имени БД экземпляра контроллера;

– `bool toEnable();` – признак «Включать контроллер»;

– `bool toStart();` – признак «Запускать контроллер»;

– `bool enableStat();` – состояние «Включен»;

– `bool startStat();` – состояние «Запущен»;

– `void start();` – запуск контроллера;

– `void stop();` – останов контроллера;

– `void enable();` – включение контроллера;

– `void disable();` – отключение контроллера;

– `void list(vector<string> &list);` – список параметров в контроллере;

– `bool present(const string &name);` – проверка на наличие параметра <name>;

– `void add(const string &name, unsigned type);` – добавление параметра <name> типа <type>;

– `void del(const string &name, bool full = false);` – удаление параметра <name>, если указано поле <full> , то контроллер будет удалён полностью;

– `AutoHD<TParamContr> at(const string &name, const string &who = "th_contr");` – подключение к параметру контроллера <name>;

– `bool redntUse();` – режим получения данных у резервной станции;

– `void setRedntUse(bool vl);` – смена режима получения данных у резервной станции;

– `Redundant redntMode();` – режим резервирования;

– `void setRedntMode(Redundant vl);` – установка режима резервирования;

– `string redntRun();` – конфигурация предпочтительного исполнения;

– `void setRedntRun(const string &vl);` – установка конфигурации предпочтительного исполнения;

– `virtual void redntDataUpdate(bool firstArchiveSync = false);` – выполнение операции получения данных из резервной станции. Вызывается автоматически задачей обслуживания схемы резервирования и перед запуском для синхронизации архивов, с установленным параметром <firstArchiveSync>;

– `void alarmSet(const string &mess, int lev = -TMess::Crit, const string &prm = "");` – формирование аварийной ситуации (нарушения) для объекта контроллера <prm>, или контроллера в целом если объект не указан, с сообщением <mess> и уровнем <lev>. Отрицательное значение уровня <lev> служит для установки, а положительное для снятия нарушения. Эта функция формирует нарушения и сообщения с категорией: al{ModId}:{CntrId}[:{PrmId}].

– `TTipDAQ &owner();` – Тип источника данных (модуль) - владелец данным контроллером;

Защищённые атрибуты:

– bool en_st; – признак «Включено»;

– bool run_st; – признак «Запущено»;

Защищённые методы:

– `virtual void enable_();` – включение контроллера (перехватывается потомком);

– `virtual void disable_();` – отключение контроллера (перехватывается потомком);

– `virtual void start_();` – запуск контроллера (перехватывается потомком);

– `virtual void stop_();` – останов контроллера (перехватывается потомком);

– `virtual TParamContr *ParamAttach(const string &name, int type);` – модульный метод создания/открытия нового параметра.

3.3.4 *Объект типа параметров (TTipParam)*

Наследует:

Telem.

Публичные методы:

`TTipParam(const char *id, const char *name, const char *db);` – инициализирующий конструктор.

Публичные атрибуты:

– string name; – имя типа параметра;

– string descr; – описание типа параметра;

– string db; – БД типа параметра.

3.3.5 *Объект параметра физического уровня (TParamContr)*

Наследует:

TConfig, Tvalue.

Наследуется:

Объектами параметров модулей подсистемы «Сбор данных».

Публичные методы:

– `TParamContr(const string &name, TTipParam *tpprm);` – инициализирующий конструктор;

– `string DAQPath();` – получение DAQ адреса элемента;

– `string id();` – идентификатор параметра (шифр);

– `string name();` – имя параметра;

– `string descr();` – описание параметра;

– `bool toEnable();` – признак «Включать параметр»;

– `bool enableStat();` – состояние «Включен»;

– `void setName(const string &inm);` – установка имени параметра;

– `void setDescr(const string &idsc);` – установка описания параметра;

– `void setToEnable(bool vl);` – установка признака «Включать параметр»;

– `TTipParam &type();` – тип параметра;

– `virtual void enable();` – включить параметр;

– `virtual void disable();` – отключить параметр;

– `bool operator==(TParamContr & PrmCntr);` – сравнение параметров;

– `TParamContr &operator=(TParamContr & PrmCntr);` – копирование параметра;

– `TController &owner();` – контроллер – владелец параметра.

Публичные атрибуты:

long long mRedntTmLast – время последних данных полученных через механизм резервирования.

Защищённые методы:

`virtual void setType(const string &tpId);` – вызывается для смены типа параметра <tpId> и может быть обработан в объекте модуля, для смены собственных данных.

3.3.6 *Объект значения (TValue)*

Наследует:

TCntrNode, TvalElem.

Наследуется:

TparamContr.

Публичные методы:

– `virtual string DAQPath();` – получение DAQ адреса элемента;

– `void vlList(vector<string> &list);` – получение списка атрибутов;

– `bool vlPresent(const string &name);` – проверка на наличия указанного атрибута;

– `AutoHD<TVal> vlAt(const string &name);` – подключение к атрибуту.

Защищённые методы:

– `TConfig *vlCfg()` – получение связанного объекта конфигурации, если возвращается NULL, то отсутствует связанный объект конфигурации;

– `void setVlCfg(TConfig *cfg);` – установка связанного объекта конфигурации <cfg>;

– `bool vlElemPresent(TElem *ValEl);` – проверка на наличие элемента атрибутов <ValEl>;

– `void vlElemAtt(TElem *ValEl);` – подключение структуры данных <ValEl>;

– `void vlElemDet(TElem *ValEl);` – отключение структуры данных <ValEl>;

– `TElem &vlElem(const string &name);` – получить структуру данных по её имени <name>;

– `virtual TVal* vlNew();` – создание экземпляра Tval, может переопределяться в модулях для создания производных объектов атрибутов параметров подсистемы "Сбор данных";

– `virtual void vlSet(TVal &val, const TVariant &pvl);` – упреждающая функция установки значения, используется для прямой (синхронной) записи с предыдущим значением в <pvl>;

– `virtual void vlGet(TVal &val);` – упреждающая функция получения значения, используется для прямого (синхронного) чтения;

– `virtual void vlArchMake(TVal &val);` – уведомляющая функция о создании архива для атрибута <val>, используется для настройки созданного архива, в соответствии с особенностями источника данных.

3.3.7 *Объект атрибута (TVal);*

Наследует:

TcntrNode.

Данные:

Дополнительные флаги к объекту TFld (enum TVal::AttrFlag):

– DirRead – флаг прямого чтения значения;

– DirWrite – флаг прямой записи значения.

Публичные методы:

– `TVal();` – конструктор по умолчанию;

– `TVal(TFld &fld);` – инициализация как хранилище динамических данных;

– `TVal(TCfg &cfg);` – инициализация как отражение статических данных (БД);

– `string DAQPath();` – получение DAQ адреса элемента;

– `void setFld(TFld &fld);` – инициализация как хранилище

динамических данных;

– `void setCfg(TCfg &cfg);` – инициализация как отражение статических данных (БД);

– `string name();` – имя атрибута;

– `int64_t time();` – метка времени последнего/текущего значения;

– `string getSEL(long long *tm = NULL, bool sys = false);` – запрос значения выборочного типа на указанное время <tm>, если NULL то возвратится последнее значение;

– `TVariant get(long long *tm = NULL, bool sys = false);` – запрос значения на указанное время <tm>, если NULL то возвратится последнее значение;

– `string getS(long long *tm = NULL, bool sys = false);` – запрос значения строкового типа на указанное время <tm>, если NULL, то возвратится последнее значение;

– `double getR(long long *tm = NULL, bool sys = false);` – запрос значения вещественного типа на указанное время <tm>, если NULL, то возвратится последнее значение;

– `int getI(long long *tm = NULL, bool sys = false);` – запрос значения целого типа на указанное время <tm>, если NULL, то возвратится последнее значение;

– `char getB(long long *tm = NULL, bool sys = false);` – запрос значения логического типа на указанное время <tm>, если NULL, то возвратится последнее значение;

– `AutoHD<TVarObj> getO(int64_t *tm = NULL, bool sys = false);` – запрос значения объектного типа;

– `void setSEL(const string &value, long long tm = 0, bool sys = false);` – установка значения выборочного типа <value>;

– `void set(const TVariant &value, long long tm = 0, bool sys = false);` – установка значения <value>;

– `void setS(const string &value, long long tm = 0, bool sys = false);` – установка значения строкового типа <value>;

– `void setR(double value, long long tm = 0, bool sys = false);` – установка значения вещественного типа <value>;

– `void setI(int value, long long tm = 0, bool sys = false);` – установка значения целого типа <value>;

– `void setB(char value, long long tm = 0, bool sys = false);` – установка значения логического типа <value>;

– `void setO(AutoHD<TVarObj> value, int64_t tm = 0, bool sys = false);` -установка значения объектного типа <value>;

– `AutoHD<TVArchive> arch();` – получение ассоциированного со значением архива;

– `void setArch(const AutoHD<TVArchive> &vl);` – установка ассоциированного со значением архива;

– `bool reqFlg(); bool resB1(); bool resB2();` – получить некоторые специфичные для реализации флаги;

– `void setReqFlg(bool vl); void setResB1(bool vl); void setResB2(bool vl);` – установить некоторые специфичные для реализации флаги;

– `TFld &fld();` – описатель структуры атрибута.

3.3.8 *Объект библиотеки шаблонов параметров подсистемы "DAQ" (TPrmTmplLib)*

Наследует:

TCntrNode, Tconfig.

Публичные методы:

– `TPrmTmplLib(const char *id, const char *name, const string &lib_db);` – инициализирующий конструктор;

– `string id();` – идентификатор библиотеки;

– `string name();` – имя библиотеки;

– `string descr();` – описание библиотеки;

– `string DB();` – БД экземпляра библиотеки;

– `string tbl();` – таблица БД экземпляра библиотеки;

– `string fullDB();` – полный адрес таблицы БД экземпляра библиотеки;

– `bool startStat();` – признак "Библиотека запущена";

– `void start(bool val);` – запуск/останов библиотеки;

– `void setName(const string &vl);` – установка имени библиотеки;

– `void setDescr(const string &vl);` – установка описания библиотеки;

– `void setFullDB(const string &vl);` – установка полного адреса таблицы БД экземпляра библиотеки;

– `void list(vector<string> &ls);` – список шаблонов в библиотеке;

– `bool present(const string &id);` – проверка на присутствие шаблона <id> в библиотеке;

– `AutoHD<TPrmTempl> at(const string &id);` – подключение к шаблону <id>;

– `void add(const char *id, const char *name = "");` – добавление шаблона <id.name> в библиотеку;

– `void del(const char *id, bool full_del = false);` – удаление шаблона <id> из библиотеки;

– `TDAQS &owner();` – объект - подсистема "DAQ", владелец библиотеки.

3.3.9 *Объект шаблона параметров подсистемы “DAQ” (TPrmTempl)*

Наследует:

TFunction, Tconfig.

Данные:

Дополнительные флаги к объекту атрибута функции IO (enum `TPrmTempl::IOTmplFlgs`):

– AttrRead – атрибут только на чтение;

– AttrFull – атрибут с полным доступом;

– CfgPublConst – публичная постоянная;

– CfgLink – внешняя связь;

– LockAttr – заблокированный атрибут.

Публичные методы:

– `TPrmTempl(const char *id, const char *name = "");` – инициализирующий конструктор шаблона;

– `string id();` – идентификатор шаблона параметра;

– `string name();` – имя шаблона параметра;

– `string descr();` – описание шаблона параметра;

– `int maxCalcTm();` – лимит на максимальное время вычисления процедуры шаблона;

– `string progLang();` – язык программирования шаблона параметра;

– `string prog();` – программа шаблона параметра;

– `void setName(const string &inm);` – установка имени шаблона параметра;

– `void setDescr(const string &idsc);` – установка описания шаблона параметра;

– `void setMaxCalcTm(int vl);` – установка лимита на максимальное время вычисления процедуры шаблона;

– `void setProgLang(const string &ilng);` – установка языка программирования шаблона параметра;

– `void setProg(const string &iprg);` – установка программы шаблона параметра;

– `void setStart(bool val);` – пуск/останов шаблона параметра;

– `AutoHD<TFunction> func();` – подключение к функции, сформированной шаблоном;

– `TPrmTmplLib &owner();` – объект библиотеки шаблонов - владелец шаблона.

3.4 Подсистема “Архивы” (TArchiveS)

Подсистема «Архивы» представлена объектом TArchiveS, который содержит на уровне подсистемы модульные объекты типов архиваторов TtipArchivator. Каждый объект типа архиватора содержит объекты архиваторов сообщений TMArchivator и архиваторов значений TVArchivator. Кроме этого, объект подсистемы архивы содержит методы архива сообщений и объекты архивов значений TVArchive. Объект архива значений TVArchive содержит буфер значений путём наследования объекта буфера TvalBuf. Для связи архива значений с архиваторами предназначен объект элемента значения TVArchEl. Этот объект содержится в архиваторе и на него ссылается архив. Структура подсистемы "Архивы" представлена на рисунке 5.

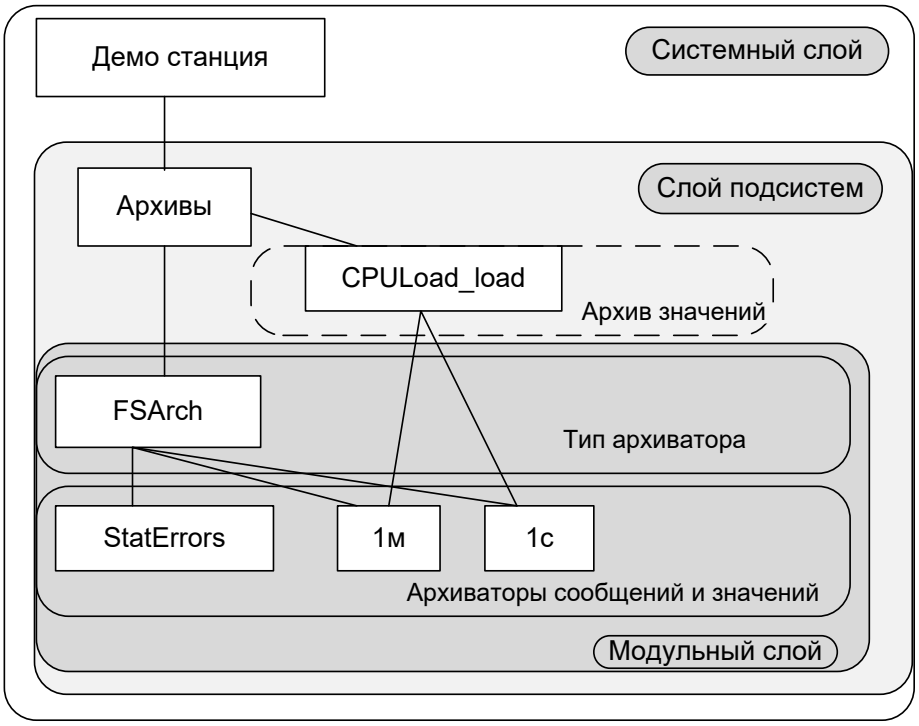

Рисунок 5. Иерархическая структура подсистемы архивов

Подсистема "Архивы" содержит механизмы архивирования сообщений и значений, архив сообщений вместе с его буфером, методы доступа к архивам значений и архиваторам значений и сообщений. Кроме этого, выполняет задачу активного сбора данных из источников значений для архивов значений, а также архивирование архива сообщений по архиваторам.

Архив значений (TVArchive) содержит буфер (TValBuf) для промежуточного накопления значений перед архивированием. Связывается с источником значений в лице параметров СКАДА в активном или пассивном режиме, а также с другими источниками в пассивном режиме. Для архивирования на физические хранилища связывается с архиваторами значений различных типов.

Объект буфера TValBuf содержит массив значений основных типов СКАДА: строка, целое, вещественное и логичное. Поддерживается хранение значений в режимах жесткой, мягкой сетки и режиме свободного доступа. Предусмотрен также режим времени высокого разрешения (микросекунды). Используется как для непосредственного хранения больших массивов значений, так и для обмена с большими массивами методом покадрового доступа.

Корневой объект модуля подсистемы "Архивы" (TTipArchivator) содержит информацию о конкретно взятом типе модуля. В рамках отдельных модулей может

реализовывать собственные общемодульные функции. В общем для модулей этого типа содержит методы доступа к хранилищам значений и сообщений.

Объект архиватора сообщений (TMArchivator) содержит конкретную реализацию хранилища сообщений. Для архиваторов сообщений предоставляется интерфейс доступа к реализации механизма архивирования в модулях.

Объект архиватора значений (TVArchivator) содержит конкретную реализацию хранилища значений. Для архиваторов значений предоставляется интерфейс доступа к реализации механизма архивирования и назначение архивов значений на обслуживание архиватором.

Объект элемента архива TVArchEl связывает объекты архивов с архиваторами. Используется для доступа к архиваторам из архива, а также к архивам из архиватора, т.е. для перекрёстных вызовов.

3.4.1 *Объект подсистемы «Архивы» (TArchiveS)*

Наследует:

SubSYS.

Публичные методы:

– `int subVer();` – версия подсистемы;

– `int messPeriod();` – период архивирование сообщений из буфера (секунд);

– `int valPeriod();` – период сбора значений для активных архиваторов (миллисекунд);

– `int valPrior();` – приоритет задачи сбора значений для активных архиваторов;

– `void setMessPeriod(int ivl);` – установка периода архивирования сообщений из буфера (секунд);

– `void setValPeriod(int ivl);` – установка периода сбора значений для активных архиваторов (миллисекунд);

– `void setValPrior(int ivl);` – установка приоритета задачи сбора значений для активных архиваторов;

– `void subStart();` – запуск подсистемы;

– `void subStop();` – останов подсистемы;

– `void valList(vector<string> &list);` – список архивов значений в подсистеме;

– `bool valPresent(const string &iid);` – проверка на наличие архива значений <iid>;

– `void valAdd(const string &iid, const string &idb = "*;*");` – добавление нового архива значений <iid>;

– `void valDel(const string &iid, bool db = false);` – удаление архива значений <iid>;

– `AutoHD<TVArchive> valAt(const string &iid);` – подключение/обращение к архиву значений <iid>;

– `void setActValArch(const string &id, bool val);` – установка архива <id> в активное состояние <val>. Активный архив будет обеспечиваться периодическим потоком значений (определяется периодичностью архива) подсистемой;

– `AutoHD<TTipArchivator> at(const string &name);` – подключение/обращение к типу архиватора (модулю) <name>;

– `void messPut(time_t tm, int utm, const string &categ, int8_t level, const string &mess);` – помещение значения <mess> с уровнем <level> категории <categ> и время <tm>+<utm> в буфер, а затем в архив сообщений;

– `void messPut(const vector<TMess::SRec> &recs);` – помещение группы значений <recs> в буфер, а затем в архив сообщений;

– `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);` – запрос значений <reqs> за указанный период времени <b_tm>, <e_tm> для указанной категории (по шаблону) <category> и уровня <level> из архиватора <arch>;

– `time_t messBeg(const string &arch = "");` – начало архива сообщений в целом или для указанного архиватора <arch>;

– `time_t messEnd(const string &arch = "");` – конец архива сообщений в целом или для указанного архиватора <arch>;

– `TElem &messE();` – структура БД архиваторов сообщений;

– `TElem &valE();` – структура БД архиваторов значений;

– `TElem &aValE();` – структура БД архивов значений;

Публичные атрибуты:

bool SubStarting; – признак запуска подсистемы.

3.4.2 *Объект архива значений (TVArchive)*

Наследует:

TCntrNode, TValBuf, Tconfig.

Данные:

Режим сбора данных/источник (struct – `TVArchive::SrcMode`):

– Passive – пассивный режим сбора данных, источник самостоятельно помещает данные в архив;

– PassiveAttr – пассивный режим сбора данных у атрибута параметра, атрибут параметра самостоятельно помещает данные в архив;

– ActiveAttr – активный режим сбора данных у атрибута параметра, атрибут параметра периодически опрашивается подсистемой "Архивы".

Публичные методы:

– `TVArchive(const string &id, const string &db, TElem *cf_el);` – инициализирующий конструктор архива, где <id> – идентификатор архива, <db> – БД для хранения и <cf_el> – структура БД архивов значений;

– `string id();` – идентификатор архива;

– `string name();` – имя архива;

– `string dscr();` – описание архива;

– `SrcMode srcMode();` – режим связывания с источником данных;

– `string srcData();` – параметры источника данных, в случае режима доступа к параметру это адрес параметра;

– `AutoHD<TVal> srcPAttr(bool force = false, const string &ipath = "");` – подключение к ассоциированному атрибуту параметра источника данных;

– `bool toStart();` – признак: "Запускать архив при включении";

– `bool startStat();` – состояние: "Архив запущен";

– `string DB();` – адрес БД архива значений;

– `string tbl();` – таблица БД архива значений;

– `string fullDB();` – полное имя таблицы БД архива значений;

– `long long end(const string &arch = BUF_ARCH_NM);` – время

окончания архива в целом (arch="") или указанного архиватора, буфера (arch="<bufer>");

– `long long begin(const string &arch = BUF_ARCH_NM);` – время начала архива в целом (arch="") или указанного архиватора, буфера (arch="<bufer>");

– `long long period(const string &arch = BUF_ARCH_NM);` – периодичность буфера архива или указанного архиватора (микросекунд);

– `TFld::Type valType();` – тип архивируемого значения;

– `bool hardGrid();` – использование жесткой сетки в буфере архива;

– `bool highResTm();` – использование высокого разрешения времени в буфере архива (микросекунды);

– `int size();` – размер буфера архива (единицы);

– `void setName(const string &inm);` – установка имени архива;

– `void setDscr(const string &idscr);` – установка описания архива;

– `void setSrcMode(SrcMode vl = SaveCur, const string &isrc = "<*>", bool noex = false);` – установка режима связывания с источником данных;

– `void setToStart(bool vl);` – установка признака: "Запускать архив при включении";

– `void setDB(const string &idb);` – установка адреса БД архива значений;

– `void setValType(TFld::Type vl);` – установка типа архивируемого значения;

– `void setHardGrid(bool vl);` – установка использования жесткой сетки в буфере архива;

– `void setHighResTm(bool vl);` – установка использования высокого разрешения времени в буфере архива (микросекунды);

– `void setSize(int vl);` – установка размера буфера архива (единиц);

– `void setPeriod(long long vl);` – установка периодичности буфера архива;

– `void start();` – запуск архива;

– `void stop(bool full_del = false);` – останов архива, с полным

удалением <full_del>;

– `TVariant getVal(long long *tm = NULL, bool up_ord = false, const string &arch = "", bool onlyLocal = false);` – запрос одного значения за время <tm> и признаком притягивания к верху <up_ord> из указанного архиватора <arch>, буфера (arch="<bufer>") или всех архиваторов по мере падения качества (arch=""). Для обработки запроса только локальной станцией устанавиливается <onlyLocal>;

– `void getVals(TValBuf &buf, long long beg = 0, long long end = 0, const string &arch = "", int limit = 100000, bool onlyLocal = false);` – запрос кадра значений <buf> за время от <beg> до <end> из указанного архиватора <arch>, буфера (arch="<bufer>") или всех архиваторов по мере падения качества (arch=""), с ограничением размера запроса в <limit> записей. Для запроса только локальных архивов, без компенсации пробелов архивов из резервных станций, устанавливается <onlyLocal>;

– `void setVals(TValBuf &buf, long long beg, long long end, const string &arch);` – загрузка кадра значений <buf> за время от <beg> до <end> в указанный архиватор <arch>, буфер (arch="<bufer>") или все архиваторы (arch="");

– `void getActiveData();` – опросить источник данных, используется подсистемой для периодического сбора данных активными архивами;

– `void archivatorList(vector<string> &ls);` – список архиваторов, которыми обслуживается архив;

– `bool archivatorPresent(const string &arch);` – проверка архиватора на обслуживание данного архива;

– `void archivatorAttach(const string &arch);` – подключение данного архива на обслуживание указанным архиватором;

– `void archivatorDetach(const string &arch, bool full = false);` – отключение данного архива от обслуживание указанным архиватором;

– `void archivatorSort();` – сортировка обслуживающих архиваторов в порядке ухудшения качества;

– `string makeTrendImg(long long beg, long long end, const string &arch, int hsz = 650, int vsz = 230);` – формирование изображения (pdf) тренда за указанный промежуток времени <beg>,

<end> и указанного архиватора <arch>;

– `TArchiveS &owner();` – подсистема "Архивы" - владелец архива значений.

3.4.3 *Объект буфера значений (TValBuf)*

Наследуется:

TVArchive.

Публичные методы:

– `TValBuf();` – инициализатор буфера с установками по умолчанию;

– `TValBuf(TFld::Type vtp, int isz, long long ipr, bool ihgrd = false, bool ihres = false);` – инициализатор буфера с указанными параметрами;

– `void clear();` – очистка буфера;

– `TFld::Type valType();` – тип значения, хранимого буфером;

– `bool hardGrid();` – работа буфера в режиме жесткой сетки;

– `bool highResTm();` – работа буфера в режиме времени высокого разрешения (микросекунды);

– `int size();` – максимальный размер буфера (едениц);

– `int realSize();` – реальный размер буфера (едениц);

– `long long period();` – периодичность значений в буфере (микросекунд), если периодичность нулевая, то буфер функционирует в режиме свободного доступа;

– `long long begin();` – время начала буфера (микросекунд);

– `long long end();` – время окончания буфера (микросекунд);

– `bool vOK(long long ibeg, long long iend);` – проверка наличия значений в буфере за указанный промежуток времени от <ibeg> до <iend>;

– `void setValType(TFld::Type vl);` – установка типа значения хранимого буфером;

– `void setHardGrid(bool vl);` – установка режима жесткой сетки;

– `void setHighResTm(bool vl);` – установка режима времени высокого разрешения (микросекунды);

– `void setSize(int vl);` – установка размера буфера (едениц);

– `void setPeriod(long long vl);` – установка периодичности

значений в буфере (микросекунд);

– `virtual void getVals(TValBuf &buf, long long beg = 0, long long end = 0);` – запрос кадра значений <buf> за время от <beg> до <end>;

– `virtual string getS(long long *tm = NULL, bool up_ord = false);` – запрос значения строкового типа за время <tm> и признаком притягивания к верху <up_ord>;

– `virtual double getR(long long *tm = NULL, bool up_ord = false);` – запрос значения вещественного типа за время <tm> и признаком притягивания к верху <up_ord>;

– `virtual int getI(long long *tm = NULL, bool up_ord = false);` – запрос значения целого типа за время <tm> и признаком притягивания к верху <up_ord>;

– `virtual char getB(long long *tm = NULL, bool up_ord = false);` – запрос значения логического типа за время <tm> и признаком притягивания к верху <up_ord>;

– `virtual void setVals(TValBuf &buf, long long beg = 0, long long end = 0);` – установка кадра значений из <buf> за время от <beg> до <end>;

– `virtual void setS(const string &value, long long tm = 0);` – установка значения строкового типа с временем <tm>;

– `virtual void setR(double value, long long tm = 0);` – установка значения вещественного типа с временем <tm>;

– `virtual void setI(int value, long long tm = 0);` – установка значения целого типа с временем <tm>;

– `virtual void setB(char value, long long tm = 0);` – установка значения логического типа с временем <tm>.

Защищённые методы:

`void makeBuf(TFld::Type v_tp, int isz, long long ipr, bool hd_grd, bool hg_res);` – пересоздание буфера для указанных параметров.

3.4.4 *Модульный объект типа архиватора (TTipArchivator)*

Наследует:

Tmodule.

Наследуется:

Корневыми объектами модулей подсистемы «Архивы».

Публичные методы:

– `void messList(vector<string> &list);` – список архиваторов сообщений;

– `bool messPresent(const string &iid);` – проверка на наличие указанного архиватора сообщений;

– `void messAdd(const string &iid, const string &idb = "*;*");` – добавление архиватора сообщений;

– `void messDel(const string &iid, bool full = false);` – удаление архиватора сообщений;

– `AutoHD<TMArchivator> messAt(const string &iid);` – подключение к архиватору сообщений;

– `void valList(vector<string> &list);` – список архиваторов значений;

– `bool valPresent(const string &iid);` – проверка на наличие указанного архиватора значений;

– `void valAdd(const string &iid, const string &idb = "*;*");` – добавление архиватора значений;

– `void valDel(const string &iid, bool full = false);` – удаление архиватора значений;

– `AutoHD<TVArchivator> valAt(const string &iid);` – подключение к архиватору значений;

– `TArchiveS &owner();` – Подсистема "Архивы" - владелец типа архиватора.

Защищённые методы:

– `virtual TMArchivator *AMess(const string &iid, const string &idb);` – Модульный метод создание архиватора сообщений;

– `virtual TVArchivator *AVal(const string &iid, const string &idb);` – Модульный метод создание архиватора значений.

3.4.5 *Объект архиватора сообщений (TMArchivator)*

Наследует:

TCntrNode, TConfig.

Наследуется:

Объектами архиваторов сообщений модулей подсистемы «Архивы».

Публичные методы:

– `TMArchivator(const string &id, const string &db, TElem *cf_el);` – инициализирующий конструктор архиватора сообщений с идентификатором <id>, для хранения на БД <db> со структурой <cf_el>;

– `string id();` – идентификатор архиватора;

– `string workId();` – рабочий идентификатор, включает имя модуля;

– `string name();` – имя архиватора;

– `string dscr();` – описание архиватора;

– `bool toStart();` – признак «Запускать архиватор»;

– `bool startStat();` – состояние архиватора «Запущен»;

– `string &addr();` – адрес хранилища архиватора;

– `int &level();` – уровень сообщений, обслуживаемых архиватором;

– `void categ(vector<string> &list);` – категории (шаблоны) сообщений, обслуживаемых архиватором;

– `string DB();` – адрес БД архиватора;

– `string tbl();` – адрес таблицы БД архиватора;

– `string fullDB();` – полный адрес таблицы БД архиватора;

– `void setName(const string &vl);` – установка имени архиватора;

– `void setDscr(const string &vl);` – установка описания архиватора;

– `void setToStart(bool vl);` – установка признака «Запускать архиватор»;

– `void setAddr(const string &vl);` – установка адреса хранилища архиватора;

– `void setLevel(int lev);` – установка уровня сообщений, обслуживаемых архиватором;

– `void setDB(const string &idb);` – установка адреса БД

архиватора;

– `virtual void start();` – запуск архиватора;

– `virtual void stop();` – останов архиватора;

– `virtual time_t begin();` – начало архива данного архиватора;

– `virtual time_t end();` – конец архива данного архиватора;

– `virtual void put(vector<TMess::SRec> &mess);` – поместить группу сообщений в архив сообщений данного архиватора;

– `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);` – получить сообщения из архива данного архиватора для указанных параметров фильтра;

– `TTipArchivator &owner();` – тип архиватора – владелец архиватора сообщений.

Защищённые атрибуты:

bool run_st; – признак «Запущен».

Защищённые методы:

`bool chkMessOK(const string &icateg, TMess::Type ilvl);` – проверка сообщения на соответствие условиям фильтра.

3.4.6 *Объект архиватора значений (TVArchivator)*

Наследует:

TCntrNode, Tconfig.

Наследуется:

Объектами архиваторов значений модулей подсистемы «Архивы».

Публичные методы:

– `TVArchivator(const string &id, const string &db, TElem *cf_el);` – инициализирующий конструктор архиватора значений с идентификатором <id>, для хранения на БД <db> со структурой <cf_el>;

– `string id();` – идентификатор архиватора;

– `string workId();` – рабочий идентификатор, включает имя модуля;

– `string name();` – имя архиватора;

– `string dscr();` – описание архиватора;

– `string addr();` – адрес хранилища архиватора;

– double valPeriod(); – периодичность значений архиватора (микросекунд);

– int archPeriod(); – периодичность архивирования значений архиватором (время через которое архиватор производит архивирование кадра значений из буфера архива);

– bool toStart(); – признак «Запускать архиватор»;

– bool startStat(); – состояние архиватора «Запущен»;

– string DB(); – адрес БД архиватора;

– string tbl(); – адрес таблицы БД архиватора;

– string fullDB(); – полный адрес таблицы БД архиватора;

– void setName(const string &inm); – установка имени архиватора;

– void setDscr(const string &idscr); – установка описания архиватора;

– void setAddr(const string &vl); – установка адреса хранилища архиватора;

– virtual void setValPeriod(double iper); – установка периодичности значений архиватора (микросекунд);

– virtual void setArchPeriod(int iper); – установка периодичности архивирования значений архиватором (время через которое архиватор производит архивирование кадра значений из буфера архива);

– void setToStart(bool vl); – установка признака "Запускать арзиватор";

– void setDB(const string &idb); – установка адреса БД архиватора;

– virtual void start(); – запуск архиватора;

– virtual void stop(bool full_del = false); – останов архиватора с возможностью полного удаления <full_del>;

– void archiveList(vector<string> &ls); – список архивов обслуживаемых архиватором;

– bool archivePresent(const string &iid); – проверка на обслуживаемость указанного архива архиватором;

– TTipArchivator &owner(); – тип архиватора – владелец архиватора

значений.

Защищённые методы:

– `TVArchEl *archivePlace(TVArchive &item);` – включить архив <item> в обработку архиватором;

– `void archiveRemove(const string &id, bool full = false);` – удалить архив <id> из обработки архиватором, с возможностью полного удаления <full>;

– `virtual TVArchEl *getArchEl(TVArchive &arch);` – получение объекта элемента архива для указанного архива;

Защищённые атрибуты:

– Res a_res – ресурс процесса архивирования;

– bool run_st – признак "Архив запущен";

– vector<TVArchEl *> arch_el; – массив элементов архивов.

3.4.7 *Объект элемента архива в архиваторе (TVArchEl)*

Наследуется:

Объектами архиваторов значений модулей подсистемы «Архивы».

Публичные методы:

– `TVArchEl(TVArchive &iachive, TVArchivator &iarchivator);` – инициализирующий конструктор для связи архива <iachive> с архиватором <iarchivator>;

– `virtual void fullErase();` – полное удаление элемента;

– `virtual long long end();` – время конца данных (микросекунды);

– `virtual long long begin();` – время начала данных (микросекунды);

– `long long lastGet();` – время последнего сброса данных из буфера в хранилище;

– `TVariant getVal(long long *tm, bool up_ord, bool onlyLocal = false);` – запрос значения за время <tm> и признаком притягивания к верху <up_ord>, с указанием запроса только локального архива в <onlyLocal>;

– `void getVals(TValBuf &buf, long long beg = 0, long long end = 0, bool onlyLocal = false);` – запрос кадра значений <buf>

за время от <beg> до <end>, с указанием запроса только локального архива в <onlyLocal>;

– `void setVals(TValBuf &buf, long long beg = 0, long long end = 0);` – установка кадра значений из <buf> за время от <beg> до <end>;

– `TVArchive &archive();` – архив элемента;

– `TVArchivator &archivator();` – архиватор элемента.

Публичные атрибуты:

– long long prev_tm; – время предыдущего значения (используется для усреднения);

– string prev_val; – предыдущее значение (используется для усреднения).

Защищённые методы:

– `virtual TVariant getValProc(long long *tm, bool up_ord);` – функция обработки запроса одного значения из архива;

– `virtual void getValsProc(TValBuf &buf, long long beg, long long end);` – функция обработки запроса модулем на получение данных;

– `virtual void setValsProc(TValBuf &buf, long long beg, long long end);` – функция обработки запроса модулем на установку данных.

3.5 Подсистема «Транспорты» (TTransportS)

Подсистема «Транспорты» представлена объектом TTransportS, который содержит на уровне подсистемы модульные объекты типов транспортов TtipTransport. Каждый тип транспорта содержит объекты входящих TTransportIn и исходящих TTransportOut транспортов. Общая структура подсистемы приведена на рисунке 6.

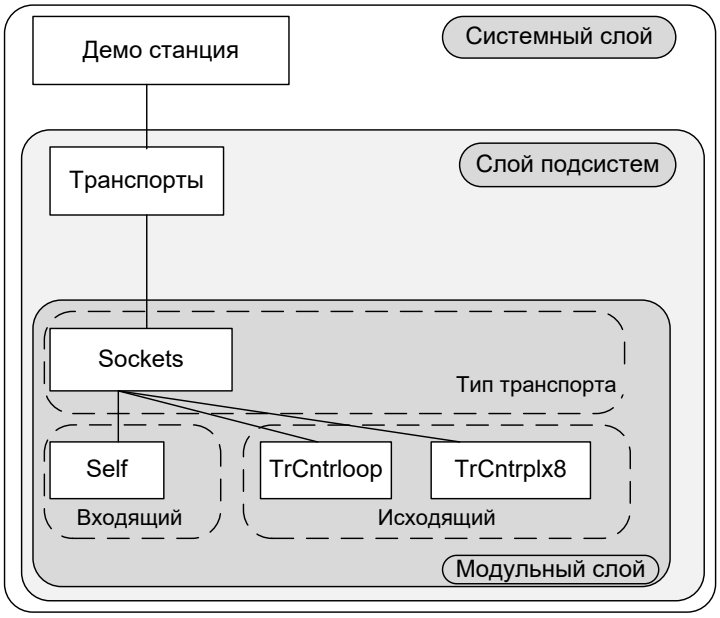

Рисунок 6. Иерархическая структура подсистемы транспорт

Корневой объект модуля подсистемы «Транспорты» содержит информацию о конкретно взятом типе модуля и внешних СКАДА хостах/станциях. В рамках отдельно взятого модуля может быть реализована собственная общемодульная функциональность. В общем, для всех модулей, содержатся методы доступа к входящим и исходящим транспортам конкретно взятого модуля.

Объект входящего транспорта TTransportIn предоставляет интерфейс к реализации модульного метода входящего транспорта.

Объект исходящего транспорта TTransportOut предоставляет интерфейс к реализации модульного метода исходящего транспорта.

3.5.1 *Объект подсистемы «Транспорты» (TTransportS)*

Наследует:

TSubSYS.

Данные:

Структура внешних хостов/станций

(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); – конструктор инициализации структуры;

string user_open; – пользователь, создавший запись про внешний хост/станцию;

string id; – идентификатор внешнего хоста/станции;

string name; – имя внешнего хоста/станции;

string transp; – транспорт, использующийся для доступа к внешнему хосту/станции;

string addr; – адрес для транспорта, который используется для доступа к внешнему хосту/станции;

string user; – пользователь внешнего хоста/станции;

string pass; – пароль пользователя внешнего хоста/станции;

bool link_ok; – признак "Связь с внешним хостом/станцией установлена".

Публичные методы:

– `int subVer();` – версия подсистемы;

– `void inTrList(vector<string> &ls);` – полный список входящих транспортов;

– `void outTrList(vector<string> &ls);` – полный список исходящих транспортов;

– `bool sysHost();` – признак "Отображать системные хосты";

– `void setSysHost(bool vl);` – установка признака "Отображать системные хосты";

– `string extHostsDB();` – БД хранения перечня внешних хостов;

– `void extHostList(const string &user, vector<string> &list);` – список внешних хостов;

– `bool extHostPresent(const string &user, const string &id);` – проверка наличия внешнего хоста <id> от имени пользователя <user> ("*" – для системных хостов);

– `AutoHD<TTransportOut> extHost(TTransportS::ExtHost host, const string &pref = "");` – создание - запрос исходящего транспорта для обслуживания внешнего хоста <host> с префиксом идентификации узла системы <pref>;

– `ExtHost extHostGet(const string &user, const string &id);` – получение информационного объекта внешнего хоста <id> от имени пользователя <user> ("*" - для системных хостов);

– `void extHostSet(const ExtHost &host);` – установка внешнего хоста/станции <host>;

– `void extHostDel(const string &user, const string &id);` – удаление внешнего хоста/станции <id> от имени пользователя <user> ("*" - для

системных хостов);

– `int cntrIfCmd(XMLNode &node, const string &senderPref, const string &user = "");` – передача запроса интерфейса управления СКАДА <node> к удалённой станции;

– `void subStart();` – запуск подсистемы;

– `void subStop();` – останов подсистемы;

– `TElem &inEl();` – структура БД входящих транспортов;

– `TElem &outEl();` – структура БД исходящих транспортов;

– `AutoHD<TTipTransport> at(const string &id);` – обращение/подключение к типу транспорта <id>.

3.5.2 *Модульный объект типа транспортов (TTipTransport)*

Наследует:

Tmodule.

Наследуется:

Корневыми объектами модулей подсистемы «Транспорты».

Публичные методы:

– `void inList(vector<string> &list);` – список входящих транспортов;

– `bool inPresent(const string &name);` – проверка на наличие входящего транспорта;

– `void inAdd(const string &name, const string &db = "*;*");` – добавление входящего транспорта;

– `void inDel(const string &name, bool complete = false);` – удаление входящего транспорта, возможно полное удаление, включающее и БД, путём установки признака <complete>;

– `AutoHD<TTransportIn> inAt(const string &name);` – подключение к входящему транспорту;

– `void outList(vector<string> &list);` – список исходящих транспортов;

– `bool outPresent(const string &name);` – проверка на наличие исходящего транспорта;

– `void outAdd(const string &name, const string &db =`

"*;*"); – добавление исходящего транспорта;

– void outDel(const string &name, bool complete = false); – удаление исходящего транспорта, возможно полное удаление, включающее и БД, путём установки признака <complete>;

– AutoHD<TTransportOut> outAt(const string &name) – подключение к исходящему транспорту;

– TTransportS &owner(); – подсистема "Транспорты" - владелец типом транспорта.

Защищённые методы:

– virtual TTransportIn *In(const string &name, const string &db); – модульный метод создания/открытия нового «входящего» транспорта;

– virtual TTransportOut *Out(const string &name, const string &db); – модульный метод создания/открытия нового «исходящего» транспорта.

3.5.3 *Объект входящих транспортов (TTransportIn)*

Наследует:

TCntrNode, Tconfig.

Наследуется:

Объектами входящих транспортов модулей подсистемы «Транспорты».

Публичные методы:

– TTransportIn(const string &id, const string &db, TElem *el); – инициализирующий конструктор;

– string id(); – идентификатор транспорта;

– string workId(); – полный идентификатор, включая идентификатор модуля;

– string name(); – имя транспорта;

– string dscr(); – описание транспорта;

– string addr(); – адрес;

– string protocol(); – связанный транспортный протокол;

– virtual string getStatus(); – получение статуса входящего транспорта;

– `bool toStart();` – признак «Запускать транспорт»;

– `bool startStat();` – состояние «Транспорт запущен»;

– `string DB();` – адрес БД транспорта;

– `string tbl();` – таблица БД транспорта;

– `string fullDB();` – полное имя таблицы БД транспорта;

– `void setName(const string &inm);` – установка имени транспорта в <inm>;

– `void setDscr(const string &idscr);` – установка описания транспорта в <idscr>;

– `virtual void setAddr(const string &addr);` – установка адреса транспорта в <addr>;

– `void setProtocol(const string &prt);` – установка связного транспортного протокола;

– `void setToStart(bool val);` – установка признака "Запускать транспорт";

– `void setDB(const string &vl);` – установка адреса БД транспорта;

– `virtual void start();` – запуск транспорта;

– `virtual void stop();` – останов транспорта;

– `TTipTransport &owner();` – тип транспорта – владелец входящим транспортом.

Защищённые атрибуты:

bool run_st; – признак «Запущен».

3.5.4 *Объект исходящих транспортов (TTransportOut)*

Наследует:

TCntrNode, Tconfig.

Наследуется:

Объектами исходящих транспортов модулей подсистемы «Транспорты».

Публичные методы:

TTransportOut(const string &id, const string &db, TElem *el); – инициализирующий конструктор;

– `string id();` – идентификатор транспорта;

– `string workId();` – полный идентификатор, включая идентификатор модуля;

– `string name();` – имя транспорта;

– `string dscr();` – описание транспорта;

– `string addr();` – адрес транспорта;

– `int prm1();` – первый резервный параметр;

– `int prm2();` – второй резервный параметр;

– `bool toStart();` – признак «Запускать транспорт»;

– `bool startStat();` – состояние «Транспорт запущен»;

– `virtual string getStatus();` – получение статуса транспорта;

– `string DB();` – адрес БД транспорта;

– `string tbl();` – таблица БД транспорта;

– `string fullDB();` – полное имя таблицы БД транспорта;

– `void setName(const string &inm);` – установка имени транспорта;

– `void setDscr(const string &idscr);` – установка описания транспорта;

– `virtual void setAddr(const string &addr);` – установка адреса транспорта;

– `void setPrm1(int vl);` – установка первого резервного параметра;

– `void setPrm2(int vl);` – установка второго резервного параметра;

– `void setToStart(bool val);` – установка признака «Запускать транспорт»;

– `void setDB(const string &vl);` – установка адреса БД транспорта;

– `virtual void start();` – запуск транспорта;

– `virtual void stop();` – останов транспорта;

– `virtual int messIO(const char *obuf, int len_ob, char *ibuf = NULL, int len_ib = 0, int time = 0, bool noRes = false);` – отправка данных через транспорт: время ожидания <time> соединения указывается в милисекундах, <noRes> используется протоколами для монопольного блокирования транспорта на время работы с ним и исключения собственной блокировки функцией;

– `void messProtIO(XMLNode &io, const string &prot);` –

отправка данных в дереве XML <in> через транспорт, используя транспортный протокол <prot>;

– `TTipTransport &owner();` – тип транспорта – владелец исходящим транспортом.

Защищённые атрибуты:

bool run_st; – Признак «Запущен».

3.6 Подсистема “Протоколы коммуникационных интерфейсов”(TProtocolS)

Подсистема «Протоколы коммуникационных интерфейсов» представлена объектом TProtocolS, который содержит на уровне подсистемы модульные объекты отдельных протоколов Tprotocol. Каждый протокол содержит объекты открытых сеансов входящих протоколов TprotocolIn.

Объект TProtocolS предоставляет доступ к входящим и исходящим протоколам отдельно взятых типов транспортных протоколов. Внутренняя сторона исходящего протокола строится по потоковому принципу с индивидуальной структурой потока для каждой реализации протокола;

3.6.1 *Объект подсистемы «Протоколы коммуникационных интерфейсов» (TProtocolS)*

Наследует:

TSubSYS.

Публичные методы:

– `int subVer();` – версия подсистемы;

– `AutoHD<TProtocol> at(const string &id);` – подключение к модулю протокола <id>;

– `string optDescr();` – локализованная помощь по опциям командной строки и параметрам конфигурационного файла.

3.6.2 *Модульный объект протокола (TProtocol)*

Наследует:

Tmodule.

Наследуется:

Корневыми объектами модулей подсистемы «Протоколы».

Публичные методы:

– `virtual void itemListIn(vector<string> &ls, const string &curIt = "");` – список элементов протокола для индивидуальной адресации в транспорте;

– `void list(vector<string> &list);` – список открытых входящих сеансов;

– `bool openStat(const string &name);` – проверка на открытость входящего сеанса с указанным именем;

– `void open(const string &name, const string &tr);` – открытие входящего сеанса от имени транспорта <tr>;

– `void close(const string &name);` – закрытие входящего сеанса;

– `AutoHD<TProtocolIn> at(const string &name);` – подключение к открытому входящему сеансу;

– `virtual void outMess(XMLNode &io, TTransportOut &tro);` – отправка данных в дереве XML <in> посредством данного протокола и транспорта <tro>.

3.6.3 *Объект сеанса входящего протокола (TProtocolIn)*

Наследует:

TcntrNode.

Наследуется:

Объектами сеанса входящего протокола модулей подсистемы «Протоколы».

Публичные методы:

– `TProtocolIn(const string &name);` – инициализирующий конструктор;

– `string name();` – имя входящего сеанса;

– `const string &srcTr();` – адрес транспорта-источника открытия данного сеанса входящего протокола;

– `void setSrcTr(const string &vl);` – установка адреса транспорта-источника открытия данного сеанса входящего протокола;

– `virtual bool mess(const string &request, string &answer, const string &sender);` – передача неструктурированных данных на обработку протоколу;

– `TProtocol &owner();` – протокол – владелец входящего сеанса.

3.7 Подсистема “Пользовательские интерфейсы” (TUIS)

Подсистема «Пользовательские интерфейсы» представлена объектом TUIS, который содержит на уровне подсистемы модульные объекты пользовательских интерфейсов TUI.

3.7.1 *Объект подсистемы «Пользовательские интерфейсы» (TUIS)*

Наследует:

TSubSYS.

Публичные методы:

– `int subVer();` – версия подсистемы;

– `void subStart();` – запуск подсистемы;

– `void subStop();` – останов подсистемы;

– `AutoHD<TUI> at(const string &name);` – подключение к модулю пользовательского интерфейса;

– `static bool icoPresent(const string &inm, string *tp = NULL);` – проверка на наличия иконки <inm> в стандартной директории, имя иконки указывается без расширения, расширение/тип загруженного изображения помещается в <tp>;

– `static string icoGet(const string &inm, string *tp = NULL);` – загрузка изображения иконки <inm> из стандартной директории;

– `static string icoPath(const string &ico, const string &tp = "png");` – полный путь к иконке, включая рабочую директорию.

3.7.2 *Модульный объект пользовательского интерфейса (TUI)*

Наследует:

Tmodule.

Наследуется:

Корневыми объектами модулей подсистемы «Пользовательские интерфейсы».

Защищённые методы:

`void cntrCmdProc(XMLNode *opt);` – обслуживание команд интерфейса управления системой.

Защищённые атрибуты:

bool run_st; – признак "Модуль запущен".

3.8 Подсистема “Специальные” (TSpecialS)

Подсистема «Специальные» представлена объектом TSpecialS, который содержит на уровне подсистемы модульные объекты специальных Tspecial.

3.8.1 *Объект подсистемы «Специальные» (TSpecialS)*

Наследует:

TSubSYS.

Публичные методы:

`int subVer();` – версия подсистемы.

3.8.2 *Модульный объект специальных (TSpecial)*

Наследует:

TModule.

Наследуется:

Корневыми объектами модулей подсистемы «Специальные».

Защищённые атрибуты:

`bool run_st;` – признак «Модуль запущен».

3.9 Подсистема “Безопасность” (TSecurity)

Подсистема безопасности представлена объектом TSecurity, который содержит объекты групп TGroup и пользователей TUser.

Объект пользователя TUser содержит пользовательскую информацию и выполняет проверку аутентичности пользователя в соответствии с указанным паролем.

Объект пользователя TGroup содержит информацию о группе пользователей и выполняет проверку на принадлежность пользователя к группе.

3.9.1 *Объект подсистемы «Безопасность» (TSecurity)*

Наследует:

TSubSYS.

Публичные методы:

– `bool access(const string &user, char mode, int owner, int group, int access);` – проверка доступа для пользователя <user> с правами <mode> к ресурсу с владельцем <owner> и группой <access>;

– `void usrList(vector<string> &list);` – список пользователей <list>;

– `void usrGrpList(const string &name, vector<string> &list);` – список групп пользователей <list>, в которые пользователь <name> включён;

– `bool usrPresent(const string &name);` – проверка на наличие указанного пользователя <name>;

– `int usrAdd(const string &name, const string &db = "*;*");` – добавление пользователя <name> с хранением в БД <db>;

– `void usrDel(const string &name, bool complete = false);` – удаление пользователя <name> с возможностью полного удаления <complete>;

– `AutoHD<TUser> usrAt(const string &name);` – подключение к пользователю <name>;

– `void grpList(vector<string> &list);` – список групп пользователей <list>;

– `bool grpPresent(const string &name);` – проверка на наличие указанной группы пользователей <name>;

– `int grpAdd(const string &name, const string &db = "*;*");` – добавление группы пользователей <name> с хранением в БД <db>;

– `void grpDel(const string &name, bool complete = false);` – удаление группы пользователей <name> с возможностью полного удаления <complete>;

– `AutoHD<TGroup> grpAt(const string &name);` – подключение к группе пользователей <name>.

3.9.2 *Объект пользователя (TUser)*

Наследует:

TCntrNode, Tconfig.

Публичные методы:

– `TUser(const string &name, const string &db, TElem *el);` – инициализирующий конструктор;

– `string name();` – имя пользователя;

– `string lName();` – полное имя пользователя;

– `string descr();` – описание пользователя;

– `string picture();` – изображение пользователя;

– `bool sysItem();` – признак системного пользователя;

– `bool auth(const string &pass);` – проверка аутентичности пользователя по паролю <pass>;

– `string DB();` – адрес БД пользователя;

– `string tbl();` – адрес таблицы БД пользователя;

– `string fullDB();` – полное имя таблицы БД пользователя;

– `void setLName(const string &nm);` – установка полного имени пользователя в <nm>;

– `void setDescr(const string &vl);` – установка описания пользователя в <vl>;

– `void setPicture(const string &pct);` – установка изображения пользователя в <pct>;

– `void setPass(const string &pass);` – установка пароля пользователя в <pass>;

– `void setSysItem(bool vl);` – установить признак системного пользователя в <vl>;

– `void setDB(const string &vl);` – установка адреса БД пользователя;

– `TSecurity &owner();` – подсистема «Безопасность» - владелец пользователя.

3.9.3 *Объект группы пользователей (TGroup)*

Наследует:

TCntrNode, Tconfig.

Публичные методы:

– `TGroup(const string &name, const string &db, TElem *el);` – инициализирующий конструктор;

– `string name();` – имя группы пользователей;

– `string lName();` – полное имя группы пользователей;

– `string descr();` – описание пользователя;

– `bool sysItem();` – признак системного пользователя;

– `string DB();` – адрес БД группы пользователей;

– `string tbl();` – адрес таблицы БД группы пользователей;

– `string fullDB();` – полное имя таблицы БД группы пользователей;

– `void setLName(const string &nm);` – установка полного имени группы пользователей в <nm>;

– `void setDescr(const string &vl);` – установка описания пользователя в <vl>;

– `void setSysItem(bool vl);` – установить признак системного пользователя в <vl>;

– `void setDB(const string &vl);` – установка адреса БД группы пользователей;

– `bool user(const string &name);` – проверка на принадлежность пользователя к группе <name>;

– `void userAdd(const string &name);` – добавление пользователя <name> в группу;

– `void userDel(const string &name);` – удаление пользователя <name> из группы;

– `TSecurity &owner();` – подсистема «Безопасность» - владелец группы пользователей.

3.10 Подсистема “Управление модулями” (TModSchedul)

Подсистема «Управление модулями» представлена объектом TmodSchedul. Подсистема содержит механизм управления модулями в разделяемых библиотеках.

3.10.1 *Объект подсистемы «Управление модулями» (TModSchedul)*

Наследует:

TSubSYS.

Данные:

Структура информации о разделяемой библиотеке (struct – `TModSchedul::SHD`):

– void *hd; – заголовок разделяемой библиотеки (если NULL, то библиотека присутствует, но не подключена);

– vector<string> use; – список подключенных модулей;

– time_t tm; – время модификации библиотеки;

– string name; – полное имя/путь разделяемой библиотеки.

Публичные методы:

– `string allowList();` – список разрешённых разделяемых библиотек (модулей);

– `string denyList();` – список запрещённых разделяемых библиотек (модулей);

– `int chkPer();` – периода проверки директории с модулями (сек);

– `void setAllowList(const string &vl);` – установка списка разрешённых разделяемых библиотек (модулей);

– `void setDenyList(const string &vl);` – установка списка запрещённых разделяемых библиотек (модулей);

– `void setChkPer(int per);` – установка периода проверки директории с модулями (сек) (если периодичность равна нуль, то проверка будет отключена);

– `void subStart();` – запуск подсистемы;

– `void subStop();` – останов подсистемы;

– `int loadLibS();` – загрузка разделяемых библиотек и инициализация модулей, возвращает количество загруженных модулей;

– `SHD &lib(const string &name);` – получение объекта разделяемой библиотеки <name>;

– `void libList(vector<string> &list);` – список разделяемых

библиотек <list>;

– `int libLoad(const string &path, bool full);` – загрузка разделяемых библиотек по указанному пути <path>, возвращает количество загруженных модулей;

– `void libAtt(const string &name, bool full = false);` – подключение указанной разделяемой библиотеке <name>;

– `void libDet(const string &name);` – отключение разделяемой библиотеки <name>.

3.11 Компоненты объектной модели СКАДА

Объектная модель СКАДА строится на основе объекта функции TFunction, параметрах функции IO и кадре значений функции TvalFunc. В последствии объекты функции включаются в объектное дерево, формируя объектную модель системы. Использование функций объектной модели производится путём связывания кадра значений TValFunc с функцией (рисунок 7) .

Рисунок 7. Основа среды программирования системы СКАДА

Объект функции (TFunction) предоставляет интерфейс для формирования параметров функции и алгоритма вычисления в объекте, наследующем его.

Объект параметра функции (IO) содержит конфигурацию отдельно взятого параметра.

Объект кадра значений (TValFunc) содержит значения в соответствии со структурой связанной функции. При исполнении алгоритма ассоциированой функции используются значения этого объекта.

3.11.1 *Объект функции (TFunction)*

Наследует:

TcntrNode.

Наследуется:

Модулями и узлами систем, содержащими функции для публикации в объектную модель системы.

Публичные методы:

– `TFunction(const string &iid);` – инициализирующий конструктор функции с идентификатором <id>;

– `TFunction &operator=(TFunction &func);` – копирование функций;

– `string id();` – идентификатор функции;

– `virtual string name();` – локализованное имя функции;

– `virtual string descr();` – описание функции;

– `bool startStat();` – состояние «Функция запущена»;

– `int use();` – счётчик использования функции;

– `virtual void setStart(bool val);` – запуск/останов <val> функции;

– `void ioList(vector<string> &list);` – список параметров функции <list>;

– `int ioId(const string &id);` – получение идентификатора параметра функции <id>;

– `int ioSize();` – количество параметров функции;

– `IO *io(int id);` – получение параметра функции по индексу <id>;

– `void ioAdd(IO *io);` – добавление параметра функции <io>;

– `int ioIns(IO *io, int pos);` – вставка параметра функции <io> в позицию <pos>, возвращает реальное положение нового параметра;

– `void ioDel(int pos);` – удаление параметра функции в позиции <pos>;

– `void ioMove(int pos, int to);` – перемещение параметра функции из позиции <pos> в позицию to>;

– `virtual void calc(TValFunc *val);` – вычисление алгоритма функции над указанными значениями <val>;

– `void valAtt(TValFunc *vfnc);` – вызывается объектом кадра значений <vfnc> в случае связывания с функцией;

– `void valDet(TValFunc *vfnc);` – вызывается объектом кадра значений <vfnc> в случае отвязывания от функции;

– `virtual void preIOCfgChange();` – вызывается перед изменением конфигурации функции;

– `virtual void postIOCfgChange();` – вызывается после изменения конфигурации функции.

Защищённые атрибуты:

– string mId; – идентификатор функции;

– bool run_st; – признак «Запущен»;

– TValFunc *mTVal; – ссылка на объект значений, используемый для тестирования функции. Может отсутствовать.

3.11.2 *Объект параметра функции (IO)*

Данные:

Типы параметра (enum – IO::Type):

– IO::String – строка/текст;

– IO::Integer – целое;

– IO::Real – вещественное;

– IO::Boolean – логический;

– IO::Object – объект;

Флаги параметра (enum – IO::IOFlgs):

– IO::Default – режим по умолчанию (вход);

– IO::Output – выход;

– IO::Return – возврат;

Публичные методы:

– `IO(const char *id, const char *name, IO::Type type, unsigned flgs, const char *def = "", bool hide = false, const char *rez = "");` – инициализирующий конструктор;

– `IO &operator=(IO &iio);` – копирование параметров функции;

– `const string &id();` – идентификатор параметра функции;

– `const string &name();` – локализованное имя параметра функции;

– `const Type &type();` – тип параметра функции;

– `unsigned flg();` – флаги параметра функции;

– `const string &def();` – значение по умолчанию;

– `bool hide();` – признак «Скрыто»;

– `const string &rez();` – резервное свойство параметра функции;

– `void setId(const string &val);` – установить идентификатор в <val>;

– `void setName(const string &val);` – установить имя в <val>;

– `void setType(Type val);` – установить тип в <val>;

– `void setFlg(unsigned val);` – установить флаги в <val>;

– `void setDef(const string &val);` – установить значение по умолчанию в <val>;

– `void setHide(bool val);` – установить/снять признак «Скрыто»;

– `void setRez(const string &val);` – установить резервное свойство в <val>.

3.11.3 *Объект значения функции (TValFunc);*

Публичные методы:

– `TValFunc(const string &iname = "", TFunction *ifunc = NULL, bool iblk = true);` – инициализирующий конструктор;

– `string user();` – пользователь, от имени которого исполняется функция;

– `const string &vfName();` – имя объекта значений;

– `bool blk();` – признак "Блокирование изменений параметров функции";

– `bool dimens();` – признак "Измерять время вычисления";

– `bool mdfChk();` – признак "Проверка атрибутов на модификацию";

– `void setUser(const string &iuser);` – установить пользователя, от имени которого будет исполнятся функция;

– `void setVfName(const string &nm);` – установить имя объекта значений в <nm>;

– `void setDimens(bool set)` – установка/сброс <set> признака «Измерять время вычисления»;

– `void setMdfChk(bool set);` – становка/сброс <set> признака "Проверка атрибутов на модификацию";

– `void ioList(vector<string> &list);` – список параметров функции <list>;

– `int ioId(const string &id);` – получение индекса параметра <id>;

– `int ioSize();` – общее количество параметров;

– `IO::Type ioType(unsigned id);` – тип параметра <id>;

– `unsigned ioFlg(unsigned id);` – флаги параметра <id>;

– `bool ioHide(unsigned id);` – признак параметра <id> - «Скрыт»;

– `bool ioMdf(unsigned id);` – признак параметра <id> - «Модифицирован»;

– `TVariant get(unsigned id);` – получить значение параметра <id>;

– `string getS(unsigned id);` – получить значение (строка) параметра <id>;

– `int getI(unsigned id);` – получить значение (целое) параметра <id>;

– `double getR(unsigned id);` – получить значение (вещественное) параметра <id>;

– `bool getB(unsigned id);` – получить значение (логическое) параметра <id>;

– `AutoHD<TVarObj> getO(unsigned id);` – получить значение объекта параметра <id>;

– `void set(unsigned id, const TVariant &val);` – установить значение <val> параметра <id>;

– `void setS(unsigned id, const string &val);` – установить

значение <val> (строка) параметра <id>;

– `void setI(unsigned id, int val);` – установить значение <val> (целое) параметра <id>;

– `void setR(unsigned id, double val);` – установить значение <val> (вещественное) параметра <id>;

– `void setB(unsigned id, bool val);` – установить значение <val> (логическое) параметра <id>;

– `void setO(unsigned id, AutoHD<TVarObj> val);` – установить значение <val> объекта параметра <id>;

– `virtual void calc(const string &user = "");` – вычислить от имени пользователя <user> или пользователя, установленного ранее;

– `double calcTm();` – время вычисления;

– `void setCalcTm(double vl);` – инициализация времени вычисления значением <vl>;

– `TFunction *func();` – связанная функция;

– `void setFunc(TFunction *func, bool att_det = true);` – связать с указанной функцией <func>;

– `virtual void preIOCfgChange();` – вызывается перед изменением конфигурации;

– `virtual void postIOCfgChange();` – вызывается после изменения конфигурации;

– `TValFunc *ctxGet(int key);` – получение контекста для ключа <key>;

– `void ctxSet(int key, TValFunc *val);` – установка контекста для ключа <key>;

– `void ctxClear();` – очистка контекста вызовов внешних функций.

3.12 Данные в СКАДА и их хранение в БД (TConfig)

Хранение данных в системе основано на объектах TConfig и TElem. Эти объекты хранят структуру и значения полей БД, что позволяет выполнять прямую загрузку и сохранение конфигурации через подсистему «БД». Для специализированного хранения данных разных типов предусмотрен объект TVariant.

Объект TElem содержит структуру записи БД. Структура записи содержит исчерпывающую информацию об элементах, их типах, размерах и остальных параметрах. Информации в данной структуре достаточно для создания, контроля и управления реальной структурой БД. Элементарной единицей записи является ячейка Tfld.

Объект TConfig является наследником от TElem и содержит реальные значения элементов. TConfig используется в качестве параметра в функциях манипуляции с записями таблиц в подсистеме «БД». Элементарной единицей записи является ячейка TCfg.

Для предоставления возможности предупреждения хранилища данных о смене структуры предусмотрен объект TValElem, от которого наследуется хранилище TConfig и список которых содержится в структуре TElem.

3.12.1 *Объект данных (TConfig)*

Наследует:

TValElem TParamContr, TController, TMArchivator, TPrmTempl, TPrmTmplLib, TUser, TGroup

Наследуется:

TTransportIn, TTransportOut, TBD, TVArchive, TVArchivator, а также модульные объекты хранящие свои данные в БД.

Публичные методы:

– `TConfig(TElem *Elements = NULL);` – инициализирующий конструктор;

– `TConfig &operator=(TConfig &cfg);` – копирование из <cfg>;

– `void cfgList(vector<string> &list);` – список элементов <list>;

– `bool cfgPresent(const string &n_val);` – проверка на наличие элемента <n_val>;

– `TCfg &cfg(const string &n_val);` – получение элемента <n_val>;

– `TCfg *at(const string &n_val, bool noExpt = false);` – получение указателя на элемент <n_val>. В случае отсутствия элемента генерируется исключение или возвращается нулевой указатель при установке <noExpt>;

– `void cfgViewAll(bool val = true);` – установка/снятие признака видимости для всех элементов;

– `void cfgKeyUseAll(bool val);` – установка/снятие признака

использования ключа для всех элементов;

– `TElem &elem();` – используемая структура;

– `void setElem(TElem *Elements, bool first = false);` – назначение структуры в <Elements>;

– `void cntrCmdMake(XMLNode *fld, const string &path, int pos, const string &user = "root", const string &grp = "root", int perm = 0664);` – формирование информационного описания элементов конфигурации для интерфейса управления СКАДА;

– `void cntrCmdProc(XMLNode *fld, const string &elem, const string &user = "root", const string &grp = "root", int perm = 0664);` – обработка запросов интерфейса управления СКАДА к элементам конфигурации;

– `bool noTransl();` – признак: «Не выполнять трансляцию сообщений при работе с БД»;

– `void setNoTransl(bool vl);` – установка признака выполнения трансляции сообщений;

– `TVariant objFunc(const string &id, vector<TVariant> &prms, const string &user);` – объектные функции доступа к конфигурации.

Защищённые методы:

`virtual bool cfgChange(TCfg &cfg);` – вызывается в случае изменения содержимого элемента конфигурации.

3.12.2 *Ячейка данных (TCfg)*

Наследует:

TVariant

Данные:

Дополнительные флаги к TFld (enum – TCfg::AttrFlg):

- TCfg::TransltText – переводить текстовые переменные записи;
- TCfg::NoVal – не отражать элемент на значение объекта TValue;
- TCfg::Key – ключевое поле;
- TCfg::Hide – атрибут скрыт.

Флаги запросов (enum – ReqFlg):

TFld::ForceUse – форсирование установки флага использования элемента при установке его значения.

Публичные методы:

– `TCfg(TFld &fld, TConfig &owner);` – инициализирующий конструктор;

– `const string &name();` – имя ячейки;

– `bool operator==(TCfg &cfg);` – сравнение ячеек;

– `TCfg &operator=(TCfg &cfg);` – копирование ячеек;

– `bool view();` – признак «Ячейка видима»;

– `bool keyUse();` – признак «Использовать ключ», для запросов dataSeek() и dataDel();

– `bool noTransl();` – признак «Отключить перевод» предназначен для отключения механизма перевода текстовых переменных для данной записи на время одного запроса;

– `void setView(bool vw);` – установка признака «Ячейка видима» в <vw>;

– `void setKeyUse(bool vl);` – установка признака «Использовать ключ» в <vl>;

– `void setNoTransl(bool vl);` – установка признака «Отключить перевод»;

– `TFld &fld();` – конфигурация ячейки;

– `string getSEL();` – получить значение выборочного типа;

– `string getS();` – получить значение строкового типа;

– `const char *getSd();` – получить прямой доступ к значению строкового типа, только для чтения;

– `double &getRd();` – получить прямой доступ к значению вещественного типа;

– `int &getId();` – получить прямой доступ к значению целого типа;

– `char &getBd();` – получить прямой доступ к значению логического типа;

– `void setSEL(const string &val, char RqFlg = 0);` – установить значение выборочного типа в <val> с флагами запроса <RqFlg>;

– `void setS(const string &val); TCfg &operator=(const`

`string &vl); TCfg &operator=(const char *vl); void setS(const string &val, char RqFlg);` – установить значение строкового типа в <val> с флагами запроса <RqFlg>;

– `void setR(double val); TCfg &operator=(double vl); void setR(double val, char RqFlg);` – установить значение вещественного типа в <val> с флагами запроса <RqFlg>;

– `void setI(int val); TCfg &operator=(int vl); void setI(int val, char RqFlg);` – установить значение целого типа в <val> с флагами запроса <RqFlg>;

– `void setB(char val); TCfg &operator=(bool vl); void setB(char val, char RqFlg);` – установить значение логического типа в <val> с флагами запроса <RqFlg>.

3.12.3 *Объект структуры данных (TElem)*

Наследуется:

TTipParam, TControllerS, TTipController, а также модульными объектами, совмещающими функции хранения структуры.

Публичные методы:

– `TElem(const string &name = "");` – инициализация структуры с указанным именем <name>;

– `string &elName();` – имя структуры;

– `void fldList(vector<string> &list);` – список ячеек в структуре <list>;

– `unsigned fldSize();` – количество ячеек в структуре;

– `unsigned fldId(const string &name, bool noex = false);` – получение индекса ячейки по её идентификатору <name>. В случае установки <noex>, при отсутствии ячейки будет возвращаться общее количество ячеек, а не осуществляться генерация исключения.

– `bool fldPresent(const string &name);` – проверка на наличие указанной ячейки <name>;

– `int fldAdd(TFld *fld, int id = -1);` – добавление/вставка ячейки <fld> в позицию <id> (-1 – вставка в конец);

– `void fldDel(unsigned int id);` – удаление ячейки <id>;

– `TFld &fldAt(unsigned int id);` – получение ячейки <id>;

– `void valAtt(TValElem *cnt);` – вызывается автоматически в случае подключения структуры к хранилищу данных <cnt>;

– `void valDet(TValElem *cnt);` – вызывается автоматически в случае отключения структуры от хранилища данных <cnt>.

3.12.4 *Ячейка структуры данных (TFld)*

Данные:

Тип ячейки (enum – TFld::Type):

– TFld::Boolean(0) – логический тип;

– TFld::Integer(1) – целочисленный тип;

– TFld::Real(4) – вещественный тип;

– TFld::String(5) – строковый тип.

Флаги ячейки (enum – TFld::AttrFlg):

– TFld::NoFlag(0x00) – флаги отсутствуют;

– TFld::Selected(0x01) – режим выборки из доступных значений, выборочный тип;

– TFld::SelfFld(0x02) – создавать собственную копию этой ячейки;

– TFld::NoWrite(0x04) – недоступна для записи;

– TFld::HexDec(0x08) – целый тип: шестнадцатеричное представление;

– TFld::OctDec(0x10) – целый тип: восьмеричное представление;

– TFld::DateTimeDec(0x20) – целый тип: содержит дату в UTC;

– TFld::FullText(0x08) – полнотекстовый, многострочный, режим тестового типа;

– TFld::NoStrTransl(0x10) – не выполнять перевод строковых переменных.

Публичные методы:

– `TFld();` – инициализация по умолчанию;

– `TFld(TFld &ifld);` – копирующий конструктор;

– `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 = "");` – инициализация с указанной конфигурацией;

– `TFld &operator=(TFld &fld);` – копирование ячейки из <fld>;

– `const string &name();` – имя ячейки;

– `const string &descr();` – описание ячейки;

– `int len();` – размер значения ячейки (символов в символьном представлении);

– `int dec();` – размер дробной части для вещественного (символов в символьном представлении);

– `Type type();` – тип ячейки;

– `static Type type(IO::Type tp);` – тип ячейки из IO::Type типа;

– `IO::Type typeIO();` – тип ячейки в IO::Type;

– `unsigned flg();` – флаги ячейки;

– `const string &def();` – значение по умолчанию;

– `string values();` – рабочий диапазон значения или перечень возможных значений для выборочного типа (в виде – "vl1;vl2;vl3");

– `string selNames();` – перечень имён значений для выборочного типа (в виде – "Value 1;Value 2;Value 3");

– `const string &reserve();` – резервный параметр;

– `void setDescr(const string &dscr);` – установка описания в <dscr>;

– `void setLen(int vl);` – установка размера ячейки в <vl>;

– `void setDec(int vl);` – установка дробной части для вещественного, в <vl>;

– `void setDef(const string &def);` – установка значения по умолчанию в <def>;

– `void setFlg(unsigned flg);` – установка флагов в <flg>;

– `void setValues(const string &vls);` – установка рабочего диапазона значения или перечня возможных значений для выборочного типа (в виде – "vl1;vl2;vl3") в <vls>;

– `void setSelNames(const string &slnms);` – установка перечня имён значений для выборочного типа (в виде – "Value 1;Value 2;Value 3") в <slnms>;

– `void setReserve(const string &ires);` – установка резервного параметра в <res>;

– `const vector<string> &selValS();` – список вариантов значений для строкового типа;

– `const vector<int> &selValI();` – список вариантов значений для целого типа;

– `const vector<double> &selValR();` – список вариантов значений для вещественного типа;

– `const vector<bool> &selValB();` – список вариантов значений для логического типа;

– `const vector<string> &selNm();` – список имён вариантов значений;

– `string selVl2Nm(const string &val);` – получить выбранное имя по значению <val> строкового типа;

– `string selVl2Nm(int val);` – получить выбранное имя по значению целого <val> типа;

– `string selVl2Nm(double val);` – получить выбранное имя по значению <val> вещественного типа;

– `string selVl2Nm(bool val);` – получить выбранное имя по значению <val> логического типа;

– `string selNm2VlS(const string &name);` – получить значение строкового типа по выбранному имени <name>;

– `int selNm2VlI(const string &name);` – получить значение целого типа по выбранному имени <name>;

– `double selNm2VlR(const string &name);` – получить значение вещественного типа по выбранному имени <name>;

– `bool selNm2VlB(const string &name);` – получить значение логического типа по выбранному имени <name>;

– `XMLNode *cntrCmdMake(XMLNode *opt, const string &path, int pos, const string &user = "root", const string &grp = "root", int perm = 0664);` – создать элемент формы в соответствии с параметрами ячейки.

3.12.5 *Объект упреждения про смену структуры (TValElem)*

Наследуется:

TValue, TConfig.

Защищённые методы:

– `virtual void detElem(TElem *el);` – уведомление элементом <el> контейнера про желание отключиться;

– `virtual void addFld(TElem *el, unsigned id) = 0;` – уведомление про добавление ячейки <id> елемента <el>;

– `virtual void delFld(TElem *el, unsigned id) = 0;` – уведомление про удаление ячейки <id> елемента <el>.

3.12.6 *Ячейка данных (TVariant)*

Данные:

Значения ошибки для различных типов данных (define):

– EVAL_BOOL – значение ошибки логического (2);

– EVAL_INT – значение ошибки целого (-2147483647);

– EVAL_REAL – значение ошибки вещественного (-3;3E308);

– EVAL_STR – значение ошибки строкового ("<EVAL>").

Типы данных (enum – TVariant::Type):

– TVariant::Null – тип данных и данные не установлены;

– TVariant::Boolean – логический тип (boolean, 8бит);

– TVariant::Integer – целочисленный тип (integer, 32бит);

– TVariant::Real – вещественный тип (double);

– TVariant::String – строка;

– TVariant::Object – объект.

Публичные методы:

– `TVariant();` – конструктор по умолчанию;

– `TVariant(char ivl);` – конструктор для логического типа;

– `TVariant(int ivl);` – конструктор для целого типа;

– `TVariant(double ivl);` – конструктор для вещественного типа;

– `TVariant(string ivl);` – конструктор для строки;

– `TVariant(const char *var);` – конструктор для строки;

– `TVariant(AutoHD<TVarObj> ivl); TVariant(TVarObj *ivl);` – конструктор для объекта;

– `TVariant(const TVariant &var);` – копирующий конструктор;

– `bool operator==(const TVariant &vr);` – сравнение объекта на равенство;

– `bool operator!=(const TVariant &vr);` – сравнение объекта на неравенство;

– `TVariant &operator=(const TVariant &vr);` – копирование объекта;

– `bool isNull() const;` – признак того, что объект не инициирован;

– `bool isEVal() const;` – признак того, что содержится ошибочное значение;

– `Type type() const;` – тип значения;

– `void setType(Type tp);` – установка типа;

– `bool isModify();` – флаг модификации: служит в объектных функциях для индикации модификации переменных;

– `void setModify(bool vl = true);` – установка флага модификации;

– `virtual char getB() const; operator char();` – получение значения как логическое;

– `virtual int getI() const; operator int();` – получение значения как целого;

– `virtual double getR() const; operator double();` – получение значения как вещественного;

– `virtual string getS() const; operator string();` – получение значения как строки;

– `virtual AutoHD<TVarObj> getO(bool noex = false) const; operator AutoHD<TVarObj>();` – получение объекта, установить <noex> для выключения генерации исключения при получении нулевого объекта;

– `virtual void setB(char val);` – установка в значение логического;

– `virtual void setI(int val);` – установка в значение целого;

– `virtual void setR(double val);` – установка в значение вещественного;

– `virtual void setS(const string &val);` – установка в значение строки;

– `virtual void setO(AutoHD<TVarObj> val);` – установка объекта.

3.12.7 *Пользовательский объект (TVarObj)*

Наследуется:

TArrayObj

Публичные методы:

– `TVarObj();` – конструктор;

– `int connect();` – подключение к объекту;

– `int disconnect();` – отключение от объекта;

– `virtual void propList(vector<string> &ls);` – список свойств <ls> объекта;

– `virtual TVariant propGet(const string &id);` – запрос свойства объекта с идентификатором <id>;

– `virtual void propSet(const string &id, TVariant val);` – установка свойства объекта с идентификатором <id> в значение <val>;

– `virtual string getStrXML(const string &oid = "");` – преобразование содержимого объекта в поток XML;

– `static AutoHD<TVarObj> parseStrXML(const string &str, XMLNode *nd = NULL, AutoHD<TVarObj> prev = NULL);` – обратное преобразование XML-потока в объект;

– `virtual TVariant funcCall(const string &id, vector<TVariant> &prms);` – вызов функции объекта с идентификатором <id> и параметрами <prms>.

3.13 Интерфейс управления системой и динамическое дерево объектов системы (TCntrNode)

Для полного покрытия ключевых компонентов системы сетью объектов единой структуры предназначен объект узла динамического дерева TcntrNode. На этот объект возлагаются следующие функции:

– единообразного доступа к компонентам системы, включая блокировки динамического доступа;

– построение распределённого интерфейса управления.

Любой объект, предоставляющий динамический доступ к себе или своим компонентам, должен наследоваться от объекта узла динамического дерева TcntrNode. Это автоматически включает узел в динамическое дерево объектов, охваченное как

прямой, так и обратной связью, а также предоставляет возможность создания контейнеров под собственные дочерние узлы. Кроме того, узел получает возможность упреждения включения и исключения/удаления узла из дерева с возможностью отказа от исключения/удаления.

Интерфейс управления системы включён в состав объекта TCntrNode и, соответственно, охватывает все узлы динамического дерева системы, позволяя единообразно управлять системой вне зависимости от используемого клиентского инструмента. Интерфейс управления системой выполнен на основе языка разметки XML. Можно придумать множество способов использования интерфейса управления системой, например:

– GUI интерфейс конфигурации (QT, GTK+);

– отражение конфигурации в сеть для распределённого управления множеством СКАДА-станций из единой среды администрирования;

– использование в роли протокола для доступа к данным объектов из сети;

– предоставление сервисных функций для доступа третьих приложений и отдельных компонентов СКАДА к внутренним данным.

Интерфейс управления системой реализован посредством составляющих:

– информационной структуры конфигурационной пользовательской страницы;

– динамической части в виде запросов на получение, модификацию данных и сервисных запросов;

– контейнера или группы вышеуказанных элементов.

Информационная иерархическая структура содержит информацию о публичных элементах управления и может быть использована для построения пользовательских диалогов управления узлами системы.

Динамическая часть содержит сценарии обслуживания запросов к элементам управления, описанным в информационной структуре, а также скрытые элементы управления в виде сервисных функций, используемые для унифицированного доступа к узлу.

Контейнер позволяет собрать в один запрос несколько информационных структур и динамических частей, оптимизируя тем самым время запроса особенно на сетевых высоколатентных интерфейсах.

Общий интерфейс управления строится из отдельных узлов динамического дерева. Иерархическое наследование от объекта TCntrNode позволяет реализовывать

многоуровневое дополнение конфигурации интерфейса управления. Общий вид динамического дерева узлов представлен на рисунке 8.

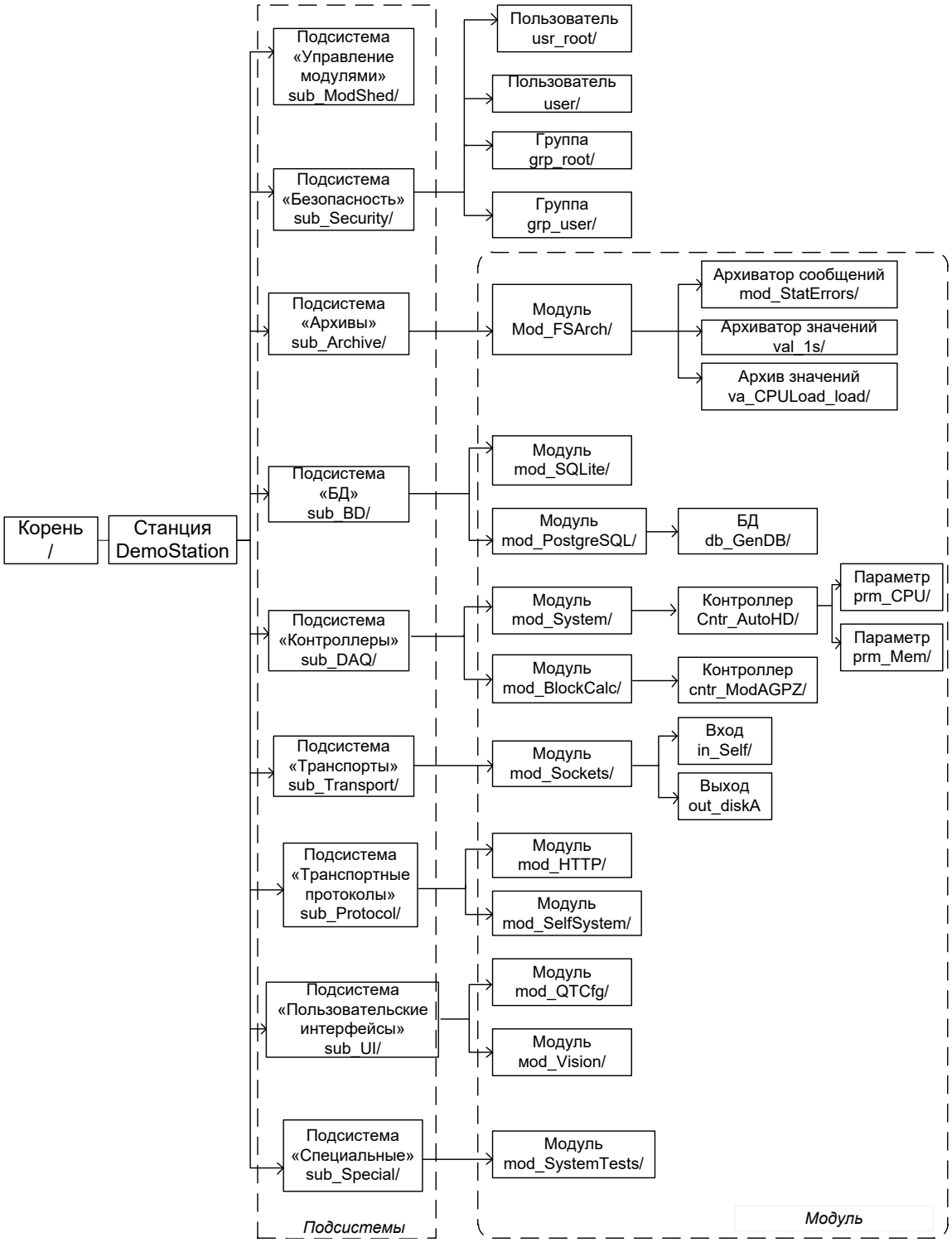

Рисунок 8. Пример динамического дерева узлов СКАДА

Узлы системы, содержащие данные для интерфейса управления системой также должны подключаться в динамическое дерево объектов.

Подключение узла в динамическое дерево производится следующим образом:

– наследование объекта TCntrNode или его потомка;

– формирование информационной структуры;

– обслуживание запросов к динамическим данным.

3.13.1 *Синтаксис запроса и ответа интерфейса управления*

Весь обмен с интерфейсом управления производится посредством языка XML. При этом внутренний обмен выполняется разобранной структурой языка XML (DOM), а внешний – посредством преобразования в поток символов сплошного XML-файла и обратно.

Запрос выполняется посредством отправки одного контейнера с некоторыми параметрами в атрибутах. Результат помещается в полученный контейнер с изменением некоторых атрибутов корневого контейнера. В общем виде контейнер запроса можно записать следующим образом:

`<cmd path="/TreePath" user="user" force="1"/>`, где:

– cmd – команда запроса;

– path – путь к узлу или ветви узла;

– user – пользователь системы от имени которого направлен запрос;

– force – признак выполнить запрос без предупреждения.

В подтверждение результата запроса устанавливается атрибут результата <rez> в значения: 0 - запрос прошел, 1 - предупреждение (с возможностью выполнения), 2 – ошибка. В случае ошибки и предупреждения сообщения записываются в текст контейнера и атрибут mcat (категория сообщения):

`<cmd path="/TreePath" user="user" force="1" rez="2" mcat="sub_DAQ/mod_BlockCalc>.` Невозможно удалить узел</cmd>.

Группирующий запрос <CntrReqs> обрабатываются на уровне API узла и не требуют отдельной обработки в пользовательском коде. Фактически в тег <CntrReqs> могут помещаться любые другие запросы с возможностью иерархической группировки путём включения внутренних тегов <CntrReqs>. Единственным атрибутом этого тега является атрибут path, который указывает путь к узлу и является основой для внутренних запросов.

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

3.13.2 *Тег информационной структуры для описания групп дочерних веток страницы*

Каждая страница может содержать группы дочерних ветвей. Для описания групп ветвей предусмотрен тег <branches>. Тег содержит описание групп ветвей посредством вложенных тегов <grp>. К тегу группы возможен доступ как на «чтение» (видимость) так и на модификацию (выполнение команд добавления и удаления элементов группы), следовательно элемент триады доступа может принимать значения:

00 – доступ вообще отсутствует;

04 – присутствует доступ только на чтение;

02 – присутствует доступ только на запись, обычно такое значение не имеет смысла поскольку доступ на запись подразумевает и доступ на чтение;

06 – присутствует доступ и на чтение, и на запись.

```
<branches id='br'>
    <grp id='/br/in_' descr='Входной транспорт' acs='04'/>
    <grp id='/br/out_' descr='Выходной транспорт' acs='04'/>
</branches>
```

Действия над группой элементов полностью совпадают с действиями над списком визуальных элементов "list", о котором написано ниже.

3.13.3 *Теги описания информационной структуры интерфейса управления*

Информационные теги для языка XML составляют алфавит формирования описания конфигурационных диалогов. Команда запроса информационной части имеет вид: `<info path="/TreePath" user="user"/>`

В результате запроса будет получена информационная структура страницы в соответствии с привилегиями указанного пользователя.

3.13.3.1 Тег области <area>

Области описываются тегом <area> и предназначены для группировки элементов по различным признакам. Область может включать другие элементы и области. Корневые области формируют закладки в представлении пользовательского интерфейса. К тегу возможен доступ только на "чтение" или видимость, следовательно элемент триады доступа может принимать значение 00, если доступ отсутствует, или 04, если присутствует;

```
<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>
```

3.13.3.2 Теги данных

Теги, описывающие данные, сведены в таблице 1.

Таблица 1. Теги описывающие данные

Тег	Описание
<fld>	Простейшие данные строкового, целого, вещественного и логического типов
<list>	Списки с данными строкового, целого, вещественного и логического типов
<table>	Таблицы с данными в ячейках строкового, целого, вещественного и логического типов
<img>	Изображения

1) Тег <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'/>
```

К тегу возможен доступ как на "чтение", так и на "запись", следовательно элемент триады доступа может принимать значения:

00 – доступ вообще отсутствует;

04 – присутствует доступ только на чтение;

02 – присутствует доступ только на запись, обычно такое значение не имеет смысла поскольку доступ на запись подразумевает и доступ на чтение;

06 – присутствует доступ и на чтение и на запись.

Тип элемента, описываемого тегом <fld>, указывается атрибутом <tp> (таблица 2).

Таблица 2. Значения атрибута <tp> тега <fld>

Тег <tp>	**Описание**
str	Строковый тип: `<fld id='host' dscr='Host name' tp='str'/>`
dec	Целое число в десятичном представлении: `<fld id='debug' dscr='Debug level' tp='dec'/>`
oct	Целое число в восьмеричном представлении: `<fld id= 'cr_file_perm' dscr='Make files permissions(default 0644)' tp='oct' len='3'/>`
hex	Целое число в шестнадцатеричном представлении.
real	Вещественное число.
bool	Логический признак («false»|"true"): `<fld id='log_sysl' dscr='Direct messages to syslog' tp='bool'/>`
time	Время в секундах (от 01/01/1970): `<fld id='v_beg' dscr='Start time' tp='time'/>`

Таблица 3. Действия над элементом, описанным тегом <fld>

Операция	**Действие**
Опрос	*Запрос:* команда «get»: `<get path="/fld_teg" user="user"/>` *Результат:* подтверждение со значением в тексте тега или сообщение об ошибке.
Модификация	*Запрос:* команда "set": `<set path="/fld_teg" user="user"> value</set>` *Результат:* подтверждение или сообщение об ошибке.

Запрос правил подсветки синтаксиса, для текстовых полей (определён атрибут "rows");	*Запрос:* команда
	`"SnthHgl": <SnthHgl path="/fld_teg" user="user"/>`
	Результат: подтверждение со списком правил подсветки синтаксиса:
	`<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"/>`

2) Тег <list>

```
<list id='mod_auto' dscr='List of shared libs(modules)' tp='str'
```

dest='file'/>

К тегу возможен доступ как на "чтение", так и на "запись"(модификацию), следовательно элемент триады доступа может принимать значения:

00 – доступ вообще отсутствует;

04 – присутствует доступ только на чтение;

02 – присутствует доступ только на запись, обычно такое значение не имеет смысла поскольку доступ на запись подразумевает и доступ на чтение;

06 – присутствует доступ и на чтение и на запись.

Тип элементов в списке указывается атрибутом <tp>. Значения атрибута <tp> приведены в таблице 4.

Таблица 4. Действия над списком

Операция	Действие

Опрос	*Запрос:* команда "get": `<get path="/fld_teg" user="user"/>`
	Результат: подтверждение с результатом в тексте тега или сообщение об ошибке.
	Результат формируется в виде:
	`<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>`

Добавление строки	*Запрос:* команда "add": `<add path="/fld_teg"user="user" id="tst">Test</add>` Читается, как добавить строку с идентификатором «tst» и значением «Test». Если список не индексированный, то атрибут id отсутствует. *Результат:* подтверждение или сообщение об ошибке.
Вставка строки	*Запрос:* команда "ins": `<ins path="/fld_teg" user="user" pos="3" p_id="tst1" id="tst">Test</ins>` Читается, как вставить строку с идентификатором «tst» и значением «Test» в позицию 3 со строкой «tst1». В случае индексного списка атрибут p_id содержит идентификатор, иначе – текст строки. Если список не индексирован, то атрибут id отсутствует. *Результат:* подтверждение или сообщение об ошибке.
Удаление строки	*Запрос:* команда "del": `<del path="/fld_teg" user="user" pos='3' id='tst'>Test</del>` Читается, как удалить строку с идентификатором «tst» и значением «Test» в позиции 3. Если список не индексирован, то атрибут id отсутствует. *Результат:* подтверждение или сообщение об ошибке.
Изменение строки	*Запрос:* команда "edit": `<edit path="/fld_teg" user="user" pos='3' p_id='tst1' id='tst'>Test</edit>` Читается, как заменить строку в позиции 3 с идентификатором "tst1" на строку с идентификатором «tst» и значением «Test». В случае индексированного списка атрибут p_id содержит идентификатор, иначе – текст строки. Если список не индексирован, то атрибут id отсутствует. *Результат:* подтверждение или сообщение об ошибке.

Операция	Действие
Перемещение строки	*Запрос:* команда “move”: <move path="/fld_teg” user="user” pos='3' to='5'/> Читается, как переместить строку с позиции 3 в позицию 5. *Результат:* подтверждение или сообщение об ошибке.

3) Тег <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>
```

К тегу таблицы и колонкам отдельно возможен доступ как на "чтение", так и на "запись" (модификацию), следовательно элемент триады доступа может принимать значения:

00 – доступ вообще отсутствует;

04 – присутствует доступ только на чтение;

02 – присутствует доступ только на запись, обычно такое значение не имеет смысла поскольку доступ на запись подразумевает и доступ на чтение;

06 – присутствует доступ и на чтение и на запись.

Если указан атрибут <key> и в нём перечислены ключевые колонки, то работа с таблицей переходит в режим адресации по идентификаторам колонок и ключам.

Таблица 5. Действия над таблицей

Операция	Действие
Опрос	*Запрос:* команда “get”: <get path="/fld_teg” user="user” cols="0;2” rows="100;1000”/> Читается, как получить колонки 0-2 и строки в них с 100 по 1000 таблицы. *Результат:* Подтверждение с данными таблицы или сообщение об ошибке. Результат формируется в виде: <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>

Операция	Действие
Добавление строки	*Запрос:* команда “add”: `<add path="/fld_teg” user="user”/>` *Результат:* подтверждение или сообщение об ошибке.
Вставка строки	*Запрос:* команда “ins”: `<ins path="/fld_teg” user="user” row='3'/>` Читается, как вставить строку в позицию 3. Команда не работает при установленном атрибуте <key>! *Результат:* подтверждение или сообщение об ошибке.
Удаление строки	*Запрос:* команда “del”: `<del path="/fld_teg” user="user” row='3'/>` или `<del path="/fld_teg” user="user” key_id='Test'/>` для ключевого режима. Читается, как удалить строку в позиции 3 или строку в позиции, где значение колонки <id> равно «Test». *Результат:* подтверждение или сообщение об ошибке.
Перемещение строки	*Запрос:* команда “move”: `<move path="/fld_teg” user="user” row='3' to='5'/>` Читается, как переместить строку с позиции 3 в позицию 5. Данная команда не работает при установленном атрибуте <key>! *Результат:* подтверждение или сообщение об ошибке.
Изменить ячейку	*Запрос:* команда “set”: `<set path="/fld_teg” user="user” row='3' col='id'>Test</set>` или `<set path="/fld_teg” user="user” key_id='Test' col='id'> Test1</set>` для ключевого режима. Читается как – установить значение ячейки в строке 3 и колонке 'id' в «Test» или установка колонки с именем 'id' строки в позиции, где значение колонки <id> равно 'Test' в значение 'Test1'. Практически данная команда переименовывает ключевой элемент указанной строки. *Результат:* подтверждение или сообщение об ошибке.

4) Тег <img>

```
<img id='ico' descr='Иконка страницы'/>
```

К тегу возможен доступ как на "чтение", так и на "запись", следовательно элемент триады доступа может принимать значения:

00 – доступ вообще отсутствует;

04 – присутствует доступ только на чтение;

02 – присутствует доступ только на запись, обычно такое значение не имеет смысла поскольку доступ на запись подразумевает и доступ на чтение;

06 – присутствует доступ и на чтение и на запись.

Тег предназначен для передачи изображений клиентам интерфейса управления. Под изображением могут выступать: иконки страниц, графики массивов значений и другие данные, которые можно представить в графическом виде.

Поддерживаются команды запросов:

`<get path="/fld_teg" user="user"/>` – запрос изображения.

Результатом является подтверждение с данными изображения или сообщение об ошибке.

<set path="/fld_teg" user="user">img</set> – загрузка изображения.

Результатом является подтверждение или сообщение об ошибке.

5) Команды с параметрами. Тег <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>
```

К тегу возможен доступ как на "чтение" или видимость+обслуживание запросов, так и на модификацию или выполнение команды, следовательно элемент триады доступа может принимать значение 00, если доступ отсутствует вообще; 04, если команду можно увидеть; и 06, если команду можно инициировать.

Предназначен для передачи команд и действий узлу, а также может использоваться для создания ссылок на другие страницы. Команды могут включать параметры. Параметры описываются тегом <fld>.

Поддерживаются команды запросов:

Вызов команды:

```
<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>
```

Загрузка ссылки на другую страницу:

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

Результатом является подтверждение или сообщение об ошибке.

6) Ветки (дочерние узлы)

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

Ветки описываются обычным списком <list> со специальными атрибутами tp='br'. Методика запроса и модификации веток полностью совпадает с методикой работы со списком <list>.

3.13.4 *Иерархические зависимости информационных элементов языка управления*

Пример страницы узла языка управления:

```
<scada_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>
</scada_cntr>
```

Таблица 6. Иерархические зависимости информационных элементов языка

Тег	**Описание**	**Атрибуты**	**Содержимое**
oscada_cntr	Корневой элемент страницы. Является единственным и служит для идентификации принадлежности к языку интерфейса управления	*id* – идентификатор; *dscr* – описание;	area, img, branches
branches	Контейнер групп дочерних веток узла	*id* – идентификатор контейнера. Равен: *br*.	grp
grp	Группа дочерних узлов	*id* – префикс группы дочерних узлов в системе; *dscr* – описание группы веток; *acs* – опции доступа	
area	Группировка стандартных тегов	*id* – идентификатор; *dscr* – описание; *acs* – опции доступа	area, fld, list, table, comm, img
comm	Команды узлу	*id* – идентификатор; *dscr* – описание; *help* – помощь по команде; *tp* – тип команды (*lnk* – ссылка); *acs* – опции доступа	fld

Тег	Описание	Атрибуты	Содержимое
fld	Описание данных стандартных типов	*id* – идентификатор; *dscr* – описание; *help* – помощь; *acs* – опции доступа; *tp* – тип элемента: - *str(len, dest, cols, rows (SnthHgl))* – строковый элемент; - *dec(len, max, min, dest)* – целое число в десятичном представлении; - *oct(len, max, min, dest)* – целое число в восьмеричном представлении; - *hex(len, max, min, dest)* – целое число в шестнадцатеричном; - *real(len, max, min, dest)* – вещественное число; - *bool* – логический признак; - *time* – время/дата в секундах (от 01/01/1970)/ **Связные:** *len* – длина значения (симв.); *min* – минимум значения; *max* – максимум значения; *cols* – количество колонок; *rows* – количество строк; *dest* – способ ввода: - *data* – источник бинарных данных (base64); - *select(select)* – выборный тип; - *sel_ed(select)* – выборный тип с возможностью редактирования; - *select* – путь к скрытому списку; - *sel_list* – статический список (разделитель ';'); - *sel_id* – статический список идентификаторов (разделитель';')	
list	Список данных стандартных типов	*id* – идентификатор; *dscr* – описание;	

Тег	Описание	Атрибуты	Содержимое
		help – помощь по списку;	
table	Таблица данных стандартных типов	*acs* – опции доступа; *tp* – как в <fld> кроме: - *br(br_pref)* – дочерние узлы; *idm* – индексированный список(0|1); *s_com* – способы модификации списка [add][,ins] [,edit][,del]: - *add* – добавлять строки; - *ins* – вставлять строки; - *edit* – модифицировать строки; - *del* – удалять строки. **Связные:** *br_pref* – префикс дочерних узлов; *dest* – как в *<fld>* *id* – идентификатор; *dscr* – описание; *help* – помощь по таблице; *acs* – опции доступа; *key* – ключевые колонки (key=«id,name,per»); *cols* – перечень колонок в атрибуте запроса; *rows* – диапазон строк в атрибуте запроса; *s_com* – способы модификации таблицы [add][,del] [,ins][,move]: - *add* – добавлять строки; - *ins* – вставлять строки; - *del* – удалять строки; *move* – перемещать строки	list
img	Изображение	*id* – идентификатор; *dscr* – описание; *help* – помощь по изображению; *acs* – опции доступа; *h_sz* – горизонтальное ограничение; *v_sz* – вертикальное ограничение	

3.13.5 *Объект узла динамического дерева (TCntrNode)*

Наследуется:

Всеми динамическими и управляемыми объектами прямо или через потомков.

Данные:

Именованные права доступа к элементам управления (define):

– R_R_R_ (0444) – доступ всем только на чтение;

– R_R___ (0440) – доступ на чтение только владельцу и группе;

– R_____ (0400) – доступ на чтение только владельцу;

– RWRWRW (0666) – полный доступ для всех;

– RWRWR_ (0664) – полный доступ владельцу и группе, а всем остальным только на чтение;

– RWR_R_ (0644) – полный доступ владельцу, а группе и всем остальным только на чтение;

– RWR___ (0640) – полный доступ владельцу, только на чтение – группе и закрыт всем остальным;

– RW____ (0600) – полный доступ владельцу, а группе и всем остальным закрыт.

Флаги динамического узла (enum TCntrNode::Flag):

– TCntrNode::MkDisable – отключение (0);

– TCntrNode::Disable – отключен (1);

– TCntrNode::MkEnable – включение (2);

– TCntrNode::Enable – включен (3);

– TCntrNode::SelfModify – признак модификации узла (0x04);

– TCntrNode::SelfModifyS – признак сохранения модификации узла в load_() (0x08);

Флаги режимов включения/отключения узла (enum TCntrNode::Flag):

– TCntrNode::NodeConnect – подключение узла;

– TCntrNode::NodeRestore – восстановление подключения узла;

– TCntrNode::NodeShiftDel – признак удаления узла отложено.

Флаги модификации узла (enum TCntrNode::ModifFlag):

– TCntrNode::Self – данный узел модифицирован;

– TCntrNode::Child – модифицированы дочерние узлы;

– TCntrNode::All – модифицирован данный узел и дочерние.

Публичные методы:

– `TCntrNode(TCntrNode *prev = NULL);` – инициализация с указанием родительского узла <prev>;

– `virtual string objName();` – имя объекта, объект-наследник

должен доопределить суммарное имя объекта;

– `virtual TCntrNode &operator=(TCntrNode &node);` – виртуальная функция копирования узлов динамического дерева;

– `void cntrCmd(XMLNode *opt, int lev = 0, const string &path = "", int off = 0);` – команда работы с интерфейсом управления системы. Поддерживаются транспортные переходы по полному пути вида: </sub_Security/usr_root/%2fgen>, где %2fgen закодированный вложенный путь к конкретному полю страницы (/gen);

– `static XMLNode *ctrId(XMLNode *inf, const string &n_id, bool noex = false);` – получение узла XML по значению атрибута 'id' <n_id>. Поддерживаются запросы XML узла по полному пути к нему вида (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, …);` – добавление элемента управления на страницу. Возможно указание множества дополнительных атрибутов в количестве <n_attr> в виде: <атрибут1>,<значений1>,<атрибут2>,<значений2>,… ;

– `bool ctrRemoveNode(XMLNode *nd, const char *path);` – удаление элемента управления <path> на странице <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);` – проверка на получение динамической команды <cmd> и наличие прав на её исполнение;

– `virtual Res &nodeRes();` – ресурс использования узла;

– `virtual const char *nodeName();` – имя узла;

– `string nodePath(char sep = 0, bool from_root = false);` – получение полного пути к узлу, начиная с корня <from_root>, и используя разделитель <sep> или обычную запись пути;

– `void nodeList(vector<string> &list, const string& gid = "");` – список дочерних узлов <list> в указанной группе <gid>;

– `AutoHD<TCntrNode> nodeAt(const string &path, int lev = 0, char sep = 0, int off = 0, bool noex = false);` – подключение к

дочернему узлу;

– `void nodeDel(const string &path, char sep = 0, int flag = 0);` – удаление узла по его полному пути;

– `static void nodeCopy(const string &src, const string &dst, const string &user = "root");` – копирование узлов динамического дерева;

– `TCntrNode *nodePrev(bool noex = false);` – адрес родительского узла;

– `char nodeFlg();` – флаги узла;

– `char nodeMode();` – состояние узла;

– `unsigned nodeUse(bool selfOnly);` – количество подключений к узлу, <selfOnly> – только собственных подключений, без дочерних;

– `unsigned nodePos();` – положение данного узла в контейнере узла-владельца, достоверно только для упорядоченных контейнеров;

– `int isModify(int mflg = TCntrNode::All);` – проверка факта модификации узла или ветви узлов;

– `void modif(bool save = false);` – установка признака модифицированности узла, с признаком сохранения <save>;

– `void modifG();` – установка признака модифицированности ветви узлов;

– `void modifClr(bool save = false);` – очистка признака модифицированности узла или признака сохранения <save>;

– `void modifGClr();` – очистка признака модифицированности ветви узлов;

– `void load(bool force = false);` – загрузка узла динамического дерева;

– `void save();` – сохранение узла динамического дерева;

– `virtual void AHDConnect();` – подключение к узлу (захват ресурса);

– `virtual bool AHDDisConnect();` – отключение от узла (освобождение ресурса), удаление при нулевом результате;

– `virtual TVariant objPropGet(const string &id);` – запрос свойства узла как пользовательского объекта;

– `virtual void objPropSet(const string &id, TVariant val);` – установка свойства узла как пользовательского объекта;

– `virtual TVariant objFuncCall(const string &id, vector<TVariant> &prms);` – вызов функции узла как пользовательского объекта;

– `virtual AutoHD<TCntrNode> chldAt(int8_t igr, const string &name, const string &user = "");` – подключение к дочернему узлу <name> контейнера <gr> пользователя <user>;

– `void chldList(int8_t igr, vector<string> &list);` – список дочерних узлов <list> в указанном контейнере <gr>;

– `bool chldPresent(int8_t igr, const string &name);` – проверка на присутствие указанного дочернего узла <name> в контейнере <gr>.

Защищённые методы:

– `virtual void cntrCmdProc(XMLNode *req);` – функция обслуживания запросов интерфейса управления, должна переопределяться у потомка;

– `void nodeEn(int flag = 0);` – включение узла;

– `void nodeDis(long tm = 0, int flag = 0);` – отключение узла с передачей флага;

– `void nodeDelAll();` – очистка всех контейнеров с дочерними узлами;

– `void setNodePrev(TCntrNode *node);` – установка родительского узла в <node>;

– `void setNodeMode(char mode);` – установка состояния узла;

– `virtual void chldAdd(int8_t igr, TCntrNode *node, int pos = -1);` – добавление дочернего узла <node> в контейнер <gr> и позицию <pos>;

– `void chldDel(int8_t igr, const string &name, long tm = -1, int flag = 0, bool shDel = false);` – удаление дочернего узла <name> из контейнера <gr> с флагом <flag>;

– `int8_t grpSize();` – количество контейнеров с дочерними узлами;

– `int8_t grpId(const string &sid);` – получение индекса группы по её идентификатору;

– `GrpEl &grpAt(int8_t id);` – доступ к структуре группы;

– `unsigned grpAdd(const string &id, bool ordered =`

`false);` – добавление контейнера дочерних узлов с префиксом <id> и возможностью упорядоченного хранения <ordered>. Возвращает идентификатор нового контейнера;

– `virtual void preEnable(int flag);` – упреждение о подключении, вызывается перед реальным подключением;

– `virtual void postEnable(int flag);` – упреждение о подключении, вызывается после реального подключения;

– `virtual void preDisable(int flag);` – упреждение о отключении, вызывается перед реальным отключением;

– `virtual void postDisable(int flag);` – упреждение о отключении, вызывается после реального отключения (перед удалением);

– `virtual void load_();` – функция вызова загрузки узла у потомка;

– `virtual void save_();` – функция вызова сохранения узла у потомка.

3.14 XML в СКАДА (XMLNode)

XML в системе СКАДА представлен объектом XML-тега – XMLNode.

3.14.1 *XML-тег (XMLNode)*

Данные:

Опции функции генерации XML-файла (enum – XMLNode::SaveView):

– XMLNode::BrOpenPrev – вставлять конец строки перед тегом открытия;

– XMLNode::BrOpenPast – вставлять конец строки после тега открытия;

– XMLNode::BrClosePast – вставлять конец строки после тега закрытия;

– XMLNode::BrTextPast – вставлять конец строки после текста тега;

– XMLNode::BrSpecBlkPast – вставлять конец строки после вычислительной инструкции;

– XMLNode::BrAllPast – вставлять конец строки после всех элементов;

– XMLNode::XMLHeader – вставлять стандартный xml-заголовок;

– XMLNode::Clean – очистка результата от комментариев и процедурных вставок;

– XMLNode::XHTMLHeader – вставлять стандартный XHTML-заголовок;

– XMLNode::MissTagEnc – пропускать кодирование имён тегов;

– XMLNode::MissAttrEnc – пропускать кодирование имён атрибутов.

Публичные методы:

– `XMLNode(const string &name = "");` – инициализация тега с именем <name>;

– `XMLNode(const XMLNode &nd);` – копирующий конструктор;

– `XMLNode &operator=(const XMLNode &prm);` – копирование ветки XML-дерева из <prm>;

– `string name() const;` – имя тега;

– `XMLNode* setName(const string &s);` – установка имени тега в <s>;

– `string text(bool childs = false, bool recursive = false) const;` – текст тега. <childs> – для получения текст из специализированных узлов текста;

– `XMLNode* setText(const string &s, bool childs = false);` – установка текста тега в <s>. <childs> – для установки текста в специализированный узел текста;

– `void attrList(vector<string> & list) const;` – список атрибутов <list> в теге;

– `XMLNode* attrDel(const string &name);` – удаление атрибута <name>;

– `void attrClear();` – очистка атрибутов тега;

– `string attr(const string &name, bool caseSens = true) const;` – получение атрибута <name>;

– `XMLNode* setAttr(const string &name, const string &val);` – установка/создание атрибута <name> со значением <val>;

– `void load(const string &vl, bool full = false, const string &cp = "UTF-8");` – загрузка/парсинг XML-файла, с кодировкой <cp>; <full> – для полной загрузки XML с блоками текстов и комментариев в специализированных узлах;

– `string save(unsigned flgs = 0, const string &cp = "UTF-8");` – сохранение/создание XML-файла с параметрами форматирования <flgs> и кодировкой <cp>;

– `XMLNode* clear();` – очистка тега (рекурсивно, включая все вложения);

– `bool childEmpty() const;` – проверка на отсутствие вложенных тегов;

– `unsigned childSize() const;` – количество вложенных тегов;

– `void childAdd(XMLNode *nd);` – добавление вложенного тега;

– `XMLNode* childAdd(const string &name = "");` – добавление вложенного тега;

– `int childIns(unsigned id, XMLNode *nd);` – вставка вложенного тега в позицию <id>;

– `XMLNode* childIns(unsigned id, const string &name = "");` – вставка вложенного тега с именем <name> в позицию <id>;

– `void childDel(const unsigned id);` – удаление вложенного тега <id>;

– `void childDel(XMLNode *nd);` – удаление вложенного тега по его адресу <nd>;

– `void childClear(const string &name = "");` – очистка вложенного тега <name>;

– `XMLNode* childGet(const int, bool noex = false) const;` – получение вложенного тега по порядковому номеру;

– `XMLNode* childGet(const string &name, const int numb = 0, bool noex = false) const;` – получение вложенного <numb> порядкового тега по имени тега <name>.<noex> указывает на запрет генерации исключения в случае отсутствия тега;

– `XMLNode* childGet(const string &attr, const string &name, bool noex = false) const;` – получение вложенного <numb> порядкового тега по значению <name> атрибута <attr>.<noex> указывает на запрет генерации исключения в случае отсутствия тега;

– `XMLNode* parent();` – родительский тег данного тега.

3.15 Ресурсы в системе СКАДА (Res, ResAlloc, AutoHD)

Большинство узлов и подсистем СКАДА являются динамическими, т.е. допускают создание/удаление/конфигурацию в процессе функционирования системы. Учитывая многопоточность системы, данная функциональность накладывает жесткие требования к

синхронизации потоков. Для синхронизации в системе используются ресурсы, функции которых локализованы в объектах <Res> и <ResAlloc>. Объект <Res> предоставляет хранилище ресурса, предусматривающего функции захвата/освобождения на чтение и запись. В объекте <ResAlloc> реализованы функции автоматического освобождения ресурса. Автоматический ресурс подразумевает создание локального объекта ресурса с автоматическим его освобождением при разрушении (в деструкторе). Использование автоматических ресурсов значительно упрощает работу с ресурсами при использовании исключений.

Любой динамический объект системы наследуется от объекта TCntrNode, который содержит механизм подключения через шаблон AutoHD. Основной функцией шаблона является хранение ссылки на объект и захват ресурса, исключающего удаление объекта на момент использования. Шаблон поддерживает копирование ресурса и автоматическое его освобождение в случае разрушения объекта шаблона. Для наглядности доступа к объектам порождённым от TCntrNode шаблон AutoHD поддерживает приведение типов, основанное на динамическом приведении.

3.15.1 *Объект ресурса (Res)*

Публичные методы:

– `Res();` – инициализация ресурса;

– `static void resRequestW(long tm = 0);` – запрос ресурса на запись/модификацию с таймаутом ожидания <tm> (в миллисекундах);

– `bool resTryW();` – попытка захвата ресурса на запись/модификацию, при успешном захвате возвращает "true" иначе "false";

– `static void resRequestR(long tm = 0);` – запрос ресурса на чтение с таймаутом ожидания <tm> (в миллисекундах);

– `bool resTryR();` – попытка захвата ресурса на чтение, при успешном захвате возвращает "true" иначе "false";

– `static void resRelease();` – освобождение ресурса.

3.15.2 *Объект ресурса (ResAlloc)*

Публичные методы:

– `ResAlloc(Res &rid);` – инициализация автоматически освобождающегося ресурса для ранее выделенного идентификатора <rid>;

– `ResAlloc(Res &rid, bool write, long tm = 0);` –

инициализация автоматически освобождающегося ресурса для ранее выделенного идентификатора <rid>, с указанием типа ресурса <write> (чтение/запись);

– `void request(bool write = false, long tm = 0);` – запрос ресурса в указанном режиме <write> (чтение/запись);

– `void release();` – освобождение ресурса.

3.15.3 *Шаблон (AutoHD)*

Публичные методы:

– `AutoHD();` – инициализация без привязки к объекту;

– `AutoHD(ORes *node, const string &who = "");` – инициализация с привязкой к объекту <node>, объект должен содержать функцию AHDConnect() и AHDDisConnect();

– `AutoHD(const AutoHD &hd);` – копирующий конструктор;

– `template <class ORes1> AutoHD(const AutoHD<ORes1> &hd_s);` – конструктор приведения типов в безопасном режиме приведения (посредством dynamic_cast). Возвращает свободный ресурс в случае невозможности приведения;

– `ORes &at() const;` – получение объекта за ресурсом;

– `AutoHD &operator=(const AutoHD &hd);` – копирование ресурсов;

– `bool operator==(const AutoHD &hd);` – сравнение объектов за ресурсом по указателю;

– `void free();` – освобождение ресурса;

– `bool freeStat() const;` – признак «Ресурс свободен».

3.15.4 *Объект строки с доступом, разделённым ресурсом (ResString)*

Публичные методы:

– `explicit ResString(const string &vl = "");` – инициализация строки с указанным значением <vl>;

– `ResString &operator=(const string &val);` – функция неявного преобразования из std::string;

– `operator string();` – функция неявного преобразования в std::string;

– `size_t size();` – размер строки;

– `bool empty();` – строка пуста;

– `void setVal(const string &vl);` – установка значения строки в <vl>;

– `const string &getVal();` – получение значения строки;

– `const string &getValRef();` – получение прямой ссылки на строку std::string; Ресурс игнорируется в случае использования этой функции!

3.16 Организация и структура базы данных компонентов системы

Узлы и подсистемы СКАДА могут иметь собственные таблицы в БД для хранения своих данных. При этом структура таблиц индивидуальна и определяется объектом <TConfig>. Узлы и подсистемы должны создавать и конфигурировать объект <TConfig> под свои требования.

3.16.1 *Системные таблицы*

СКАДА имеет две системные таблицы BD и SYS. Таблица BD содержит записи зарегистрированных БД, а таблица SYS содержит данные общесистемных параметров.

Таблица 7. Структура таблицы общесистемных параметров (SYS)

Поль зователь <user>	**Идентификатор параметра <id>**	**Значение параметра <val>**
root	/DemoStation/MessLev	0
user	/DemoStation/Workdir	/mnt/home/user/work/ScadaD/share/Scada
user	/DemoStation/UI/QTStarter/StartMod	QTCfg

Таблица 8. Структура таблицы зарегистрированных БД

Идентификатор <ID>	**Тип БД <TYPE>**	**Имя <NAME>**	**Описание <DESCR>**	**Адрес <ADDR>**	**Кодировка содержимого БД <CODEPAGE>**	**Включать <EN>**
LibBD	MySQL	Библиотека функций		Server.diya.org;roman;123456;oscadaUserLibs	KOI8-U	1
AnastModel	SQLite	Модель АГЛКС		./DATA/AGLKSModel.db	UTF8	1
GenDB	MySQL	Основная БД		Server.diya.org;roman;123456;scadaDemoSt	KOI8-U	1

3.16.2 *Таблицы подсистемы «Сбор данных»*

Контроллеры (источники данных) подсистемы «Сбор данных» хранятся в таблицах своих подсистем с именем DQA_<ModName>. Структуры этих таблиц могут значительно отличаться, однако у всех присутствуют обязательные поля. Общая структура таблиц контроллеров представлена в таблице 9.

Таблица 9. Общая структура таблиц контроллеров подсистемы «Сбор данных» (DQA_<ModName>)

Идентификатор <ID>	**Имя контроллера <NAME>**	**Описание <DESCR>**	**Включать <ENABLE>**	**Запускать <START>**	**Индивидуальные параметры**
AutoDA	Автоматический источник	Сбор данных из активных источников с автоматическим их выявлением	1	1	…

Также как и таблица контроллеров, таблицы параметров для различных типов источников данных могут значительно отличаться, но также имеют обязательные поля. Кроме отличия, характерного для типа источника данных, таблицы параметров ещё могут быть разными для различных типов параметров. Общая структура таблицы параметров приведена в таблице 10.

Таблица 10. Общая структура таблиц параметров подсистемы «Сбор данных»

Шифр параметра <SHIFR>	**Имя параметра <NAME>**	**Описание параметра <DESCR>**	**Включать <EN>**	**Индивидуальные параметры**
P3	P3	Давление на диафрагме	1	…

Кроме контроллеров и параметров подсистема «Сбор данных» содержит шаблоны параметров. Шаблоны параметров сгруппированы по библиотекам шаблонов и хранятся в таблицах трёх типов: таблица библиотек шаблонов (ParamTemplLibs) таблица 11, таблица шаблонов параметров таблица 12 и таблица параметров шаблонов таблица 13.

Таблица 11. Структура таблицы библиотек шаблонов

Идентификатор <ID>	**Имя <NAME>**	**Описание <DESCR>**	**Таблица БД библиотеки <DB>**
base	Базовые шаблоны	Библиотека базовых шаблонов	tmplib_base
S7		Библиотека шаблонов для контроллеров серии Siemens S7	tmplib_S7

Таблица 12. Структура таблицы шаблонов

Идентификатор <ID>	**Имя <NAME>**	**Описание <DESCR>**	**Текст процедуры шаблона <PROGRAM>**
digAlarm	Дискр. сигнал	Сигнализация по дискретному параметру	JavaLikeCalc.JavaScript
simpleBoard	Простые границы	Формирование простых границ для аналогового сигнала	JavaLikeCalc.JavaScript

Таблица 13. Структура таблицы параметров шаблонов

Идентификатор шаблона <TMPL_ID>	**Идентификатор параметра <ID>**	**Имя <NAME>**	**Тип <TYPE>**	**Флаги <FLAGS>**	**Значение <VALUE>**	**Позиция <POS>**
digAlarm	in	Вход	3	144		2
digitBlock	cmdOpen	Команда открытия	3	161	Параметр:com	0

3.16.3 *Таблицы подсистемы “Транспорты”*

Подсистема “Транспорты” делится на входящие и исходящие транспорты. Для каждого типа транспортов существует своя таблица с собственной структурой. Имена таблиц соответственно: Transport_In и Transport_Out. Таблицы могут дополняться полями, характерными для типа транспорта.

Таблица 14. Структура таблицы входящих транспортов (Transport_in)

Идентификатор <ID>	**Тип <MODULE>**	**Имя <NAME>**	**Описание <DESCRIPT>**	**Адрес <ADDR>**	**Протокол <PROT>**	**Запускать <START>**	**Индивидуальные поля типов транспортов**
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	…

Таблица 15. Структура таблицы исходящих транспортов (Transport_out);

Идентификатор <ID>	**Тип <MODULE>**	**Имя <NAME>**	**Описание <DESCRIPT>**	**Адрес <ADDR>**	**Запускать <START>**	**Индивидуальные поля типов транспортов**
tcp_o1	Sockets	TCP Out 1	Output TCP transport 1	TCP::10001	1	…

Для централизованного описания перечня внешних СКАДА станций используется таблица внешних хостов (CfgExtHosts). Структура этой таблицы приведена в таблице 16.

Таблица 16. Структура таблицы внешних СКАДА хостов (CfgExtHosts);

Пользователь системы <OP_USER>	**Идентификатор <ID>**	**Имя <NAME>**	**Транспорт <TRANSP>**	**Адрес удалённого хоста <ADDR>**	**Пользователь внешнего хоста <USER>**	**Пароль пользователя внешнего хоста <PASS>**
tcp_o1	Sockets	TCP Out 1	Output TCP transport 1	TCP::10001	1	...

3.16.4 *Таблицы подсистемы "Архивы"*

Подсистема "Архивы" содержит три таблицы с фиксированными именами:

– Архивы значений: Archive_val;

– Архиваторы значений: Archive_val_proc;

– Архиваторы сообщений: Archive_mess_proc;

– Таблицы архиваторов могут дополняться полями, характерными для каждого типа архиватора.

Таблица 17. Структура таблицы архивов значений (Archive_val)

Идентификатор <ID>	**Имя <NAME>**	**Описание <DESCR>**	**Запускать <START>**	**Режим источника значений <SrcMode>**	**Источник значений <Source>**	**Тип значений <VTYPE>**	**Периодичность буфера <BPER>**	**Размер буфера <BSIZE>**	**Перечень обслуживающих архиваторов <ArchS>**
CPULoad_load			1	1	DAQ.System.AutoDA.CPULoad.load	4	1	100	FSArch.1s; DBArch.1m; FSArch.1m;
ai1_dP			0	0		4	0;0001	100	FSArch.POMP_20070301; FSArch.1s

Таблица 18. Структура таблицы архиваторов значений (Archive_val_proc)

Идентификатор <ID>	Тип архиватора <MODUL>	Имя <NAME>	Описание <DESCR>	Запускать <START>	Адрес <ADDR>	Период значений <V_PER>	Период архивирования	Индивидуальные поля типов архиваторов
1s	FSArch		Односекундный	1	ARCHIVES/VAL/1s	1	60	...
POMP_20070301	FSArch			0	ARCHIVES/VAL/POMP_20070301	0.0001	60	...

Таблица 19. Структура таблицы архиваторов сообщений (Archive_mess_proc)

Идентификатор <ID>	Тип архиватора <MODUL>	Имя <NAME>	Описание <DESCR>	Запускать <START>	Шаблон категории сообщений <CATEG>	Уровень сообщений <LEVEL>	Адрес <ADDR>	Индивидуальные поля типов архиваторов
StatErrors	FSArh	Ошибки станции		1	/DemoStation*	4	ARCHIVES/MESS/stError/	...
NetRequsts	FSArh	Сетевые запросы		1	/ DemoStation/Transport/Sockets*	1	ARCHIVES/MESS/Net/	...

3.16.5 *Таблицы подсистемы "Безопасность"*

Подсистема "Безопасность" содержит две таблицы: таблица пользователей системы (Security_user) и групп системы (Security_grp).

Таблица 20. Структура таблицы пользователей системы (Security_user)

Имя <NAME>	Описание <DESCR>	Пароль <PASS>	Изображение <PICTURE>
root	Суперпользователь	scada	
user	Пользователь	user	

Таблица 21. Структура таблицы групп пользователей системы (Security_grp)

Имя <NAME>	Описание <DESCR>	Пользователи в группе <USERS>
root	Группа суперпользователей	root;user
users	Группа пользователей	root;user

3.16.6 *Структура баз данных модулей*

Каждый модуль может иметь собственные таблицы БД для хранения индивидуальных данных. Структура таблиц БД модулей может формироваться свободно, исходя из внутренних потребностей.

3.17 Сервисные функции интерфейса управления СКАДА

Сервисные функции – это интерфейс доступа к СКАДА посредством интерфейса управления СКАДА из внешних систем. Данный механизм положен в основу всех механизмов обмена внутри СКАДА, реализованных посредством слабых связей и стандартного протокола обмена СКАДА. Основным его преимуществом является приоритетная обработка и возможность использования нестандартной упаковки данных. Для доступа к обычным данным можно использовать стандартные команды интерфейса управления.

3.17.1 *Групповой доступ к значениям атрибутов параметров подсистемы «Сбор данных», а также к детальной информации*

Данные запросы позволяют получить детальную информацию о параметрах подсистемы "Сбор данных", запросить значения всех атрибутов параметров, а также выполнить групповую установку. Детальная информация о запросах приведена в таблице 22.

Таблица 22. Атрибуты команд запроса атрибутов параметров подсистемы «Сбор данных»

Id	Имя	Значение
Команда запроса информации об атрибутах параметра: `<list path="/sub_DAQ/mod_{SRC}/cntr_{CNTR}/prm_{PRM}/%2fserv%2fattr"/>`		
`<el id="iatr"/>`	Информация атрибутов	В отдельных тегах возвращается информация об атрибуте
`id`	Идентификатор атрибута	Символьный идентификатор отдельно взятого атрибута
`nm`	Имя атрибута	Имя отдельно взятого атрибута
`flg`	Флаги атрибута	Флаги отдельно взятого атрибута
`tp`	Тип атрибута	Тип отдельно взятого атрибута
`vals`	Область значений атрибута	Область значений отдельно взятого атрибута

names	Имена значений атрибута для выборочного типа;	Имена значений отдельно взятого атрибута для выборочного типа
Команда запроса значений всех атрибутов параметра: <get path="/sub_DAQ/mod_{SRC}/cntr_{CNTR}/prm_{PRM}/%2fserv%2fattr"/>		
<el id="iatr">val</el>	Значения атрибутов	В отдельных тегах возвращается значение атрибутов
Команда установки значений указанных атрибутов параметра: <set path="/sub_DAQ/mod_{SRC}/cntr_{CNTR}/prm_{PRM}/%2fserv%2fattr"/>		
<el id="iatr">val</el>	Указываются значения атрибутов	В отдельных тегах указываются значение атрибутов

3.17.2 *Доступ к архивным данным архивов сообщений*

Для запроса и установки данных архива в подсистеме архивирования объекта архива сообщений предусмотрена команда <info path="/sub_Archive/%2fserv%2fmess"/>, <get path="/sub_Archive/ %2fserv%2fmess"/> и <set path="/sub_Archive/%2fserv%2fmess"/> для запроса информации об архиве, значений архива и установки значений в архив, соответственно. Детальное описание данных команд представлено в таблице 23.

Таблица 23. Атрибуты команд запроса информации об архиве значений и архивных данных

Id	Имя	Значение
Команда запроса информации об архиве: <info path="/sub_Archive/%2fserv%2fmess"/>		
arch	Установка имени архиватора архива	Архиватор архива, для которого определять параметры
end	Контроль вершины архива в данном архиваторе	В результате запроса указывает на реальную вершину архива сообщений в данном архиваторе <arch>
beg	Контроль глубины архива в данном архиваторе	В результате запроса указывает на реальную глубину архива сообщений в данном архиваторе <arch>
Команда запроса архивных и/или текущих данных: <get path="/sub_Archive/%2fserv%2fmess"/>		
tm	Установка времени	Время вершины блока архива сообщений
tm_grnd	Установка времени основания/начала архива	Указывает основание/начало архива

Id	Имя	Значение
arch	Установка архиватора архива	Указывает, у какого архиватора запрашивать значения. Если архиватор не указан, то запрос будет производиться последовательно у всех архиваторов с исключением дубликатов
cat	Установка категории сообщений	Указывает категорию/шаблоны запрашиваемых сообщений
lev	Установка уровня важности	Указывает уровень важности сообщений, для которого и выше получать сообщения
<el>{mess}</el>	Сообщения	В отдельных тегах возвращаются сообщения
time, utime	Время сообщения (секунды, микросекунды)	Время отдельно взятого сообщения
cat	Категория сообщения	Категория отдельно взятого сообщения
lev	Уровень сообщения	Уровень отдельно взятого сообщения
Команда установки архивных данных: <set path="/sub_Archive/%2fserv%2fmess"/>		
<el>{mess}</el>	Сообщения	В отдельных тегах указываются сообщения для установки
time, utime	Время сообщения (секунды, микросекунды)	Время отдельно взятого сообщения
cat	Категория сообщения	Категория отдельно взятого сообщения
lev	Уровень сообщения	Уровень отдельно взятого сообщения

3.17.3 *Доступ к архивным данным архивов значений*

Для запроса данных архива в подсистеме архивирования объекта архива значения и объекте атрибута параметра подсистемы сбора данных предусмотрена команда `<info path="{a_p_addr}/ %2fserv%2fval"/>` и `<get path="{a_p_addr}/%2fserv%2fval"/>` для запроса информации об архиве и значений архива соответственно. Атрибуты данных команд, предусматривающие различные механизмы запроса, представлены в таблице 24.

Таблица 24. Атрибуты команд запроса информации об архиве значений и архивных данных

Id	Имя	Значение
Команда запроса информации об архиве: <info path="{a_p_addr}/%2fserv%2fval"/>		

Id	Имя	Значение
arch	Установка имени архиватора архива	Архиватор архива, для которого определять параметры;
end	Контроль вершины архива в данном архиваторе	В результате запроса указывает на реальную вершину архива значений в данном архиваторе <arch>. В случае отсутствия архива атрибут устанавливается в “0”.
beg	Контроль глубины архива в данном архиваторе	В результате запроса указывает на реальную глубину архива значений в данном архиваторе <arch>. В случае отсутствия архива атрибут устанавливается в “0”.
per	Контроль периодичности архива в данном архиваторе	В результате запроса указывает на реальную периодичность значений в данном архиваторе <arch>. В случае отсутствия архива атрибут устанавливается в “0”.
vtp	Контроль типа значений архива/параметра	Возвращает код типа значений архивных данных: 0 – Boolean, 1 – Integer, 4 – Real, 5 – String. Только данное свойство доступно в случае запроса у параметра без архива!
Команда запроса архивных и/или текущих данных: `<get path="{a_p_addr}/%2fserv%2fval"/>`		
tm	Установка и контроль времени	Время запрашиваемого значения или вершины блока архива значений. Если атрибут не указан или равен “0”, то возвращается последнее значение. Значение данного атрибута устанавливается в значение времени полученного значения или вершины блока архивных данных.
tm_grnd	Установка и контроль времени основания/начала архива	Указывает основание/начало архива. Если атрибут не указан или равен “0”, то производится запрос одного значения. Значение данного атрибута устанавливается в значение времени основания блока архивных данных.
per	Установка и контроль периодичности получаемых значений	Периодичность запрашиваемых значений. Если атрибут не указан, равен “0” или меньше чем реальная периодичность архива, то значения будут возвращаться с реальной периодичностью архива и значение данного атрибута установится в значение реальной периодичности. Если значение атрибута больше реальной периодичности, то значения архива будут усредняться к указанной периодичности.

Id	Имя	Значение
arch	Установка и контроль архиватора архива	Указывает, у какого архиватора запрашивать значения. Если архиватор не указан, то запрос будет производиться последовательно с приоритетностью от лучшего к худшему. Имя архиватора, у которого получены значения, будет указанно в данном атрибуте при возврате результата. Если архиватор для данного параметра не установлен или такого архиватора нет, то значение данного атрибута будет обнулено. В случае выполнение запроса, пересекающего несколько архиваторов, выполняется обработка только первого архиватора с указанием его имени и размерности в соответствующих атрибутах. Для запроса данных последующих архиваторов запрос повторяется отталкиваясь от размерности предыдущего запроса.
mode	Установка и контроль режима передачи данных архива	Указывается форма, в которой желательно получать данные архива. Предусматриваются следующие режимы/формы передачи данных архивов: *0* – простая запись: одно значение - одна строка без упаковки смежных значений. В случае архива строк значения кодируются на предмет исключения символов перевода строки. Пример формы записи: 34.5678 23.6543 65.8754 34.6523 *1* – Упакованная запись по принципу сворачивания смежных значений: одно значение - одна запись. Перед каждым значением указывается его позиционный номер, отделённый пробелом: 0 34.5678 1 23.6543 4 65.8754 6 34.6523 *2* – Массив бинарных неупакованных значений, кодированный Mime Base64. Не может быть использован для строковых архивов.
real_prec	Установка точности вещественных чисел	Указывает, с какой точностью передавать данные значений вещественного типа в режиме "0" и "1". По умолчанию эта точность составляет 6 значащих цифр
round_perc	Установка процента округления	Указывает процент округления смежных числовых значений, для режима “1”.

3.18 API модулей модульных подсистем

Модули в системе СКАДА реализуются в виде разделяемых библиотек. Каждая библиотека может содержать множество модулей подсистем СКАДА, фактически выступая в роли контейнера.

Первым шагом при подключении разделяемых (SO – shared object) библиотек является подключение функций инициализации. Эти функции должны быть определены как обычные “C” функции для исключения искажения имен функций. Обычно это делается следующим образом:

```
//================== CUT ========================
extern "C"
{
#ifdef MOD_INCL
//Нужно указывать только в случае поддержки модулем возможности
//влинковки в ядро СКАДА
TModule::SAt bd_DBF_module(int n_mod)
#else
TModule::SAt module(int n_mod)
#endif
{
//Формирование описателей модулей из списка
//доступных в разделяемой библиотеке;
}
#ifdef MOD_INCL
//Нужно указывать только в случае поддержки модулем возможности
//влинковки в ядро СКАДА
TModule *bd_Tmpl_attach(const TModule::SAt &AtMod, const string &source)
#else
TModule *attach(const TModule::SAt &AtMod, const string &source)
#endif
{
//Подключить указанный модуль
}
}
//================== CUT ========================
```

Функции для работы с разделяемой библиотекой:

`TModule::SAt module(int n_mod);` - функция предназначена для последовательного опроса информации о модулях, содержащихся в SO библиотеке. Параметр <n_mod> указывает на порядковый номер запрашиваемого модуля и должен перебираться, начиная с нуля. В случае отсутствия модуля с данным идентификатором функция должна возвращать структуру с именем модуля нулевой длины, что и служит окончанием процесса сканирования.

`TModule *attach(const TModule::SAt &AtMod, const string &source);` - подключение к указанному модулю.

Практически все функции и данные системы сведены в API-системы, описываемого в данном документе. Однако для упрощения и ускорения процесса

написания модулей основные функции, переопределяемые в модулях, приведены в таблице 25.

Таблица 25. Основные функции, использующиеся при создании модулей

API объекта (TCntrNode): `virtual void load_();` – загрузка объекта из хранилища; `virtual void save_();` – сохранение объекта в хранилище.
Общее API модулей (TModule): Атрибуты: *string mId;* – Идентификатор модуля; *string mName;* – Имя модуля; *string mDescr;* – Описание модуля; *string mType;* – Тип модуля; *string mVers;* – Версия модуля; *string mAutor;* – Автор модуля; *string mLicense;* – Лицензия модуля; *string mSource;* – Источник/происхождение модуля; Методы: `virtual void modStart();` – Запуск модуля; `virtual void modStop();` – Останов модуля; `virtual void modInfo(vector<string> &list);` – Список доступных элементов информации о модуле. Предусмотрены следующие информационные элементы: Module – идентификатор модуля; Name – локализованное имя модуля; Type – тип модуля; Source – источник модуля (контейнер); Version – версия модуля; Author – автора модуля; Description – описание модуля; License – лицензия модуля; `virtual string modInfo(const string &name);` – Запрос указанного элемента информации; `virtual void perSYSCall(unsigned int cnt);` – Вызов из системного потока, с периодичностью 10 секунд и секундным счётчиком <*cnt*>; `void postEnable(int flag);` – Подключение модуля к динамическому дереву объектов; `void modFuncReg(ExpFunc *func);` – Регистрация экспортируемой функции.
API модулей подсистемы “БД”
Тип БД (потомок от TTipBD): `virtual TBD *openBD(const string &id);` – Открыть/создать БД;
БД (потомок от TBD): `virtual void enable();` – Включение БД; `virtual void disable();` – Отключение БД; `virtual void allowList(vector<string> &list);` – Перечень таблиц в БД; `virtual void sqlReq(const string &req, vector < vector<string> > *tbl = NULL, char intoTrans = EVAL_BOOL);` – Отправка SQL-запроса <req> на БД и получение результата в виде таблицы <tbl>; При установке <intoTrans> в true для запроса будет открыта транзакция, в false будет закрыта; `virtual void transCloseCheck();` – Периодически вызываемая функция для

проверки транзакций и закрытия старых или содержащих много запросов;
`virtual TTable *openTable(const string &table, bool create);` – Открыть/создать таблицу.

Таблица (потомок от TTable):
`virtual void fieldStruct(TConfig &cfg);` – Получение структуры таблицы;
`virtual bool fieldSeek(int row, TConfig &cfg);` – Последовательное сканирование записей таблицы;
`virtual void fieldGet(TConfig &cfg);` – Получение указанной записи;
`virtual void fieldSet(TConfig &cfg);` – Установка указанной записи;
`virtual void fieldDel(TConfig &cfg);` – Удаление указанной записи.

API модулей подсистемы "Транспорты"

Тип транспорта (потомок от TTipTransport):
`virtual TTransportIn *In(const string &name, const string &db);` – Создать/открыть новый «входящий» транспорт;
`virtual TTransportOut *Out(const string &name, const string &db);` – Создать/открыть новый «исходящий» транспорт

Входящий транспорт (потомок от TTransportIn):
`virtual string getStatus();` – Статус интерфейса;
`virtual void setAddr(const string &addr);` – Установка адреса транспорта. Может переопределяться для обработки и проверки специфического для модуля формата адреса транспорта;
`virtual void start();` – Запуск транспорта;
`virtual void stop();` – Останов транспорта

Исходящий транспорт (потомок от TTransportOut):
`virtual string getStatus();` – Статус интерфейса;
`virtual void setAddr(const string &addr);` – Установка адреса транспорта. Может переопределяться для обработки и проверки специфического для модуля формата адреса транспорта;
`virtual void start();` – Запуск транспорта;
`virtual void stop();` – Останов транспорта;
`virtual int messIO(const char *obuf, int len_ob, char *ibuf = NULL, int len_ib = 0, int time = 0, bool noRes = false);` – Отправка данных через транспорт. Время ожидания <time> соединения указывается в милисекундах; <noRes> используется протоколами для монопольного блокирования транспорта на время работы с ним и исключения собственной блокировки функцией

API модулей подсистемы "Протоколы";

Протокол (потомок от TProtocol):

virtual void itemListIn(vector<string> &ls, const string &curIt = ""); – Перечень подэлементов у входящего протокола, если протокол их предусматривает. Используется при выборе в конфигурации объекта входящего транспорта;

virtual void outMess(XMLNode &io, TTransportOut &tro); – Передача данных в дереве XML <*in*> удалённой системе посредством транспорта <*tro*> и текущего исходящего протокола;

*virtual TProtocolIn *in_open(const string &name);* – Открыть/создать входной протокол

Входной протокол (потомок от TProtocolIn):

`virtual bool mess(const string &request, string &answer, const string &sender);` – Передача неструктурированного сообщения в протокол

API модулей подсистемы “Сбор данных”

Тип контроллера (потомок от TTipDAQ):

`virtual void compileFuncLangs(vector<string> &ls);` – Перечень языков пользовательского программирования, поддерживаемых модулем;

`virtual void compileFuncSynthHighl(const string &lang, XMLNode &shgl);` – Запрос правил подсветки синтаксиса <*shgl*> указанного языка пользовательского программирования <*lang*>;

`virtual string compileFunc(const string &lang, TFunction &fnc_cfg, const string &prog_text);` – Компиляция пользовательской процедуры и создания объекта исполнения функции для указанного языка пользовательского программирования;

`virtual bool redntAllow();` – Признак поддержки механизмов резервирования модулем. Должен просто переопределяться и возвращать true;

`virtual TController *ContrAttach(const string &name, const string &daq_db);` – Открытие/подключение контроллера

Контроллер (потомок от TController):

`virtual string getStatus();` – Вызов для получения специфического статуса контроллера;

`virtual void enable_();` – Включение контроллера;

`virtual void disable_();` – Отключение контроллера;

`virtual void start_();` – Запуск контроллера;

`virtual void stop_();` – Останов контроллера;

`virtual void redntDataUpdate(bool firstArchiveSync = false);` – Выполнение операции получения данных из резервной станции. Вызывается автоматически задачей обслуживания схемы резервирования и перед запуском для синхронизации архивов с установленным параметром <*firstArchiveSync*>;

`virtual TParamContr *ParamAttach(const string &name, int type);` – Создание/открытие нового параметра.

Параметр контроллера (потомок от TParamContr->TValue):

`virtual void enable();` – Включить параметр;

`virtual void disable();` – Отключить параметр;

`virtual void setType(const string &tpId);` – Вызывается для смены типа параметра <*tpId*> и может быть обработан в объекте модуля, для смены собственных данных;

`virtual TVal* vlNew();` – Вызывается при создании нового атрибута. Может быть переопределён для реализации особого поведения в рамках своего, наследованного от *TVal*, класса при доступе к атрибуту;

`virtual void vlSet(TVal &val, const TVariant &pvl);` – Вызывается для атрибута с прямым режимом записи `TVal::DirWrite` (синхронный режим или запись во внутренний буфер объекта) при установке значения, с целью непосредственной записи значения в физический контроллер или буфер объекта;

`virtual void vlGet(TVal &val);` – Вызывается для атрибута с прямым режимом чтения *TVal::DirRead* (синхронный режим или чтение из внутреннего буфера объекта) при чтении значения, с целью непосредственного чтения значения из физического контроллера или буфера объекта;

`virtual void vlArchMake(TVal &val);` – Вызывается при создании архива значений с атрибутом <*val*> в качестве источника с целью инициализации качественных характеристик буфера архива согласно особенностям источника данных и их опроса

API модулей подсистемы "Архивы"

Тип архиватора (потомок от TTipArchivator):

`virtual TMArchivator *AMess(const string &id, const string &db);` – Создание архиватора сообщений;

`virtual TVArchivator *AVal(const string &id, const string &db);` – Создание архиватора значений

Архиватор сообщений (потомок от TMArchivator):

`virtual void start();` – Старт архиватора;

`virtual void stop();` – Останов архиватора;

`virtual time_t begin();` – Начало данных в архиваторе;

`virtual time_t end();` – Конец данных в архиваторе;

`virtual void put(vector<TMess::SRec> &mess);` – Передать сообщение архиватору;

`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);` – Запросить сообщение из архиватора

Архиватор значений (потомок от TVArchivator):

`virtual void setValPeriod(double per);` – Установить периодичность значений архиватора;

`virtual void setArchPeriod(int per);` – Установить периодичность архивирования;

`virtual void start();` – Запустить архиватор;

`virtual void stop(bool full_del = false);` – Остановить архиватор с возможностью полного удаления, если установлен <*full_del*>;

`virtual TVArchEl *getArchEl(TVArchive &arch);` – Получение архива <*arch*>, обслуживаемого архиватором

Архивный элемент значений (потомок от TVArchEl): `virtual void fullErase();` – Полное удаление части архива в архиваторе; `virtual int64_t end();` – Время окончания архива в архиваторе; `virtual int64_t begin();` – Время начала архива в архиваторе; `virtual TVariant getValProc(long long *tm, bool up_ord);` - Функция обработки запроса одного значения из архива; `virtual void getValsProc(TValBuf &buf, int64_t beg, int64_t end);` – Функция обработки запроса модулем на получение данных группы значений <*buf*> за указанный промежуток времени; `virtual void setValsProc(TValBuf &buf, int64_t beg, int64_t end);` – Функция обработки запроса модулем на установку данных группы значений <*buf*> за указанный промежуток времени
API модулей подсистемы "Пользовательские интерфейсы"
Пользовательский интерфейс (потомок от TUI): Не содержит специфических функций
API модулей подсистемы "Специальные”
Специальные (потомок от TSpecial): Не содержит специфических функций

3.19 Отладка и тестирование проекта СКАДА

Для контроля за качеством кода и проверки работоспособности различных участков системы пишутся специальные модули, выполняющие процедуру тестирования с выдачей протокола тестирования. Данные модули необходимо выполнять после завершения работы над любым участком проекта.

3.20 Правила оформления и комментирования исходных текстов СКАДА и его модулей

При написании и оформлении исходных текстов СКАДА и его модулей необходимо придерживаться следующих правил:

– отступ между уровнями вложений: 4 символа;

– фигурные скобки открытия и закрытия должны располагаться в отдельных строках на уровне предыдущего текста;

– возможно написание вложений в одной строке с предыдущим уровнем вложения, в случае повышения читабельности кода;

– расстояние между описаниями функций не менее одного символа;

– расстояние между определением переменных и текстом программы не менее одного символа;

– допускается определение переменных в тексте при сохранении читабельности;

– избегать длины строки более 100 символов;

– команды препроцессора располагать на первом уровне вне зависимости от текущего уровня текста;

– для форматирования исходного текста, наследованного у других свободных приложений и примеров, рекомендуется использовать утилиту:

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

Правила комментирования исходных текстов СКАДА:

– обязательному комментированию и тщательному описанию подлежат объявления классов;

– объявления публичных методов классов должны быть тщательно описаны с индивидуальным описанием каждого параметра;

– объявления публичных атрибутов также необходимо тщательно комментировать;

– текст функций не нуждаются в тщательном комментировании, однако неявные места желательно комментировать.

4 Используемые технические средства

Аппаратные требования СКАДА-системы для её исполнения в зависисмости от сложности прокекта приведены в таблице 26.

Таблица 26.

	Требования к аппаратной части
Проекты более 30000 точек	
АРМ	Процессор:Intel Core i3 2 ГГц; ОЗУ: 8 Гб; Жесткий диск: 40 Гб
Сервер	Процессор: Intel Xeon E5; ОЗУ: 128 Гб; Жесткий диск: 40 Гб
Сервер БД	Процессор: Intel Xeon E5; ОЗУ: 64 Гб; Жесткий диск: 1 Тб, RAID 10
Шлюз	Процессор: Intel Core i3 2 ГГц; ОЗУ: 2 Гб; Жесткий диск: 40 ГБ
Проекты до 5000 точек	
АРМ-сервер	Процессор: Intel Core i7 3 ГГц; ОЗУ: 16 Гб; Жесткий диск: 300 Гб, RAID 5
Шлюз	Процессор: Intel Core i3 2 ГГц; ОЗУ: 2 Гб; Жесткий диск: 40 ГБ
Проекты до 500 точек	
АРМ-сервер	Процессор: Intel Core i5 2 ГГц; ОЗУ: 16 Гб; Жесткий диск: 300 ГБ

5 Вызов и загрузка

Запуск программы осуществляется командой `scada` из командной строки терминала Fly или из графического меню «Пуск» → «Графика» → «SCADA» с помощью мыши.

После регистрации в СКАДА (пользователь: root, пароль: root) на экране появится окно системного конфигуратора СКАДА.

Загрузка ПП «СКАДА А-СОФТ» считается корректной, если в ходе ее запуска не выдается ошибок и на экране появляется окно системного конфигуратора.

6 Входные и выходные данные

6.1 Входные данные

Входными данными для ПП «СКАДА А-СОФТ» являются сигналы от низовых ПТК.

6.2 Выходные данные

Выходными данными ПП «СКАДА А-СОФТ» являются пиктограммы элементов на мнемосхемах (видеокадрах), отображающие состояние технологического оборудования, архивы технологических параметров и команды оператора.

Перечень принятых сокращений

БД	база данных
ОЗУ	оперативное запоминающее устройство
ОС	операционная система
ПО	программное обеспечение
ППО	прикладное программное обеспечение
SCADA	диспетчерское управление и сбор данных (Supervisory Control And Data Acquisition)

Лист регистрации изменений

Изм;	Номера листов (страниц)				Всего листов (страниц) в докум;	№ документа	Входящий № сопроводи тельного докум; и дата	Подп;	Дата
	Измененных	Замененных	новых	Аннулированных					