

# Руководство по созданию модулей для OpenSCADA


## Оглавление

<a href="#">Руководство по созданию модулей для OpenSCADA</a> .....	1
<a href="#">Введение</a> .....	1
<a href="#">1. Создание нового модуля</a> .....	2
<a href="#">1.1. Создание в дереве исходных текстов проекта OpenSCADA</a> .....	2
<a href="#">1.2. Создание внешнего модуля к OpenSCADA</a> .....	4
<a href="#">2. API модуля</a> .....	5
<a href="#">2.1. Модуль подсистемы "Базы Данных (БД)"</a> .....	7
<a href="#">2.2. Модуль подсистемы "Транспорты"</a> .....	8
<a href="#">2.3. Модуль подсистемы "Транспортные протоколы"</a> .....	9
<a href="#">2.4. Модуль подсистемы "Сбор данных (DAQ)"</a> .....	9
<a href="#">2.5. Модуль подсистемы "Архивы"</a> .....	11
<a href="#">2.6. Модуль подсистемы "Пользовательские интерфейсы (UI)"</a> .....	12
<a href="#">2.7. Модуль подсистемы "Специальные"</a> .....	12

## Введение

Данное руководство призвано помочь в создании модулей для системы OpenSCADA. Создание модуля может потребоваться в случае желания добавить поддержку нового источника данных или другого расширения к системе OpenSCADA. Поскольку OpenSCADA является предельно модулей, то все интерфейсы взаимодействия с внешней средой осуществляются посредством расширения специализированными модулями типов:

- "Базы данных"
- "Коммуникационные интерфейсы, транспорты"
- "Протоколы коммуникационных интерфейсов"
- "Источники данных и сбор данных"
- "Архивы (сообщений и значений)"
- "Интерфейсы пользователя (GUI, TUI, WebGUI, speech, signal ...)"
- "Дополнительные модули, специальные"

 Для создания модулей OpenSCADA нужны знания в программировании на языке C/C++, сборочной системы [AutoTools](#), а также базовые знания ОС Linux и используемого дистрибутива Linux.

# 1. Создание нового модуля

Модули в OpenSCADA представляют из себя разделяемые библиотеки, которые подключаются к ядру OpenSCADA динамически, в момент работы программы. Многие модули в процессе работы могут быть отключены, подключены и обновлены из [менеджера модулей](#). Модули также могут быть включены в ядро OpenSCADA при сборке, посредством аргумента `--enable-{ModName}=incl` к скрипту конфигурации "configure", о чём можно узнать из [руководства по сборке](#). Модули OpenSCADA могут быть семи типов согласно присутствующим [модульным подсистемам](#). Сейчас модули к системе OpenSCADA пишутся на языке программирования "C++", хотя в дальнейшем возможно появление биндингов на другие языки.

Для облегчения создания новых модулей в дереве исходных текстов, в ветви каждой подсистемы, предусмотрена директория `"=Tmpl="` с шаблоном модуля соответствующей подсистемы. Разработчик нового модуля может взять эту директорию и скопировать её с именем своего нового модуля. Предусмотрена возможность создания модулей в дереве исходных текстов проекта OpenSCADA или как независимого проекта внешнего модуля к OpenSCADA.

## 1.1. Создание в дереве исходных текстов проекта OpenSCADA

Создавать новые модули в дереве исходных текстов проекта OpenSCADA имеет смысл в случае дальнейших планов передачи нового модуля проекту OpenSCADA. Поскольку модуль не должен противоречить духу открытого проекта и лицензии, на основе которой [разрабатывается и распространяется OpenSCADA](#); то лицензией нового модуля очевидно должна быть одна из свободных лицензий.

В целом процедура создания нового модуля с включением в дерево исходных текстов на основе шаблона является проще процедуры для внешнего модуля и включает в себя шаги:

1. Получение дерева исходных текстов проекта OpenSCADA.

*Для рабочей ветки:*

```
$ svn co svn://oscada.org/trunk/OpenSCADA
```

*Для ветки стабильного релиза (нежелательно поскольку к стабильным LTS релизам принимаются только исправления):*

```
$ svn co svn://oscada.org/tags/openscada_0.8.0
```

2. Копирование директории шаблона с именем нового модуля "NewMod" (например, для подсистемы "БД"):

```
$ cd OpenSCADA/src/moduls/bd
$ cp -r =Tmpl= NewMod; cd NewMod
$ rm -rf .svn po/.svn
```

3. Редактирование файла "module.cpp".

*Изменить имена функций включения модуля согласно имени нового модуля:*

```
"TModule::Sat bd_Tmpl_module( int n_mod )" — bd_NewMod_module
"TModule *bd_Tmpl_attach( const TModule::Sat &AtMod, const string &source )"
— bd_NewMod_attach
```

*Информация о модуле в файле "module.cpp", а именно участок:*

```
/* *****
/* Modul info! *
#define MOD_ID "NewMod"
#define MOD_NAME _ ("DB NewMod")
#define MOD_TYPE SDB_ID
#define VER_TYPE SDB_VER
#define MOD_VER "0.0.1"
#define AUTHORS _ ("MyName MyFamily")
#define DESCRIPTION _ ("BD NewMod description.")
#define MOD_LICENSE "GPL2"
```

4. Редактирование конфигурации сборки модуля в файле "Makefile.am" к такому виду:

```
EXTRA_DIST = *.h po/*

oscd_modul_LTLIBRARIES = db_NewMod.la
db_NewMod_la_CXXFLAGS = $(NewMod_CFLAGS)
db_NewMod_la_LDFLAGS = -module -avoid-version -no-undefined $(NewMod_LDFLAGS)
db_NewMod_la_SOURCES = module.cpp
db_NewMod_la_LIBTOOLFLAGS = --tag=disable-static

if NewModIncl
db_NewMod_la_CXXFLAGS += -DMOD_INCL
install:
endif

I18N_mod = $(oscd_modulpref)NewMod
```

5. Добавление записи нового модуля в конец секции подсистемы (у нас "> DB modules"), конфигурационного файла (OpenSCADA/configure.in) сборочной системы OpenSCADA:

```
AX_MOD_EN(NewMod, [disable or enable[=incl] build module
DB.NewMod], disable, incl,
[
AC_MSG_RESULT(Build module: DB.NewMod)
AC_CONFIG_FILES(src/moduls/bd/NewMod/Makefile)
DBSub_mod="${DBSub_mod}NewMod "
#>> Modules checkings
# Код проверки внешних библиотек модуля
if test $enable_NewMod = incl; then
LIB_CORE="${LIB_CORE} moduls/bd/NewMod/.libs/*.o "
ModsIncl="${ModsIncl}bd_NewMod "
fi
])
```

6. Теперь новый модуль можно собрать в составе OpenSCADA после переформирования сборочной системы:

```
$ autoreconf -if
$ configure --enable-NewMod
$ make
```

7. Публикация. Формирование патча с вашим модулем и отправка его разработчикам OpenSCADA:

```
$ cd OpenSCADA; make distclean; rm -f src/moduls/bd/NewMod/Makefile.in
$ svn add src/moduls/bd/NewMod
$ svn diff > NewMod.patch
```

## 1.2. Создание внешнего модуля к OpenSCADA

Создание внешнего модуля к OpenSCADA может иметь смысл в случае разработки интерфейса интеграции с коммерческими системами, требующими закрытия кода взаимодействия, а также в случае других реализаций коммерческих интерфейсов, при которых модуль к OpenSCADA приобретает статус отдельного проекта, распространяется и поддерживается независимо, часто в виде бинарных сборок под конкретную платформу и версию OpenSCADA. Лицензия таких модулей соответственно может быть любой.

Процедура создания нового внешнего модуля на основе шаблона во многом похожа на предыдущую процедуру и включает в себя шаги:

1. Получение исходных текстов проекта OpenSCADA. Для внешнего модуля в качестве источника шаблона можно использовать любые исходные файлы OpenSCADA версии более 0.8.0 поскольку из них нужно скопировать только директорию "=Tmp1=" и несколько файлов для сборки.
2. Копирование директории шаблона с именем нового модуля "NewMod" (например, для подсистемы "БД"). Создание и копирование нужных файлов для внешнего модуля. В дальнейшем информационные файлы проекта "COPYING", "NEWS", "README", "AUTHORS" и "ChangeLog" нужно заполнить согласно сути нового модуля.  
\$ cp -r OpenSCADA/src/moduls/bd/=Tmp1= NewMod  
\$ rm -rf NewMod/.svn NewMod/po/.svn  
\$ touch NewMod/{NEWS,README,AUTHORS,ChangeLog}  
\$ cp OpenSCADA/I18N.mk NewMod/
3. Редактирование информации о модуле в файле "module.cpp", аналогично этому пункту предыдущего раздела.
4. Редактирование конфигурации сборки модуля в файле "Makefile.am", аналогично этому пункту предыдущего раздела.
5. Редактирование файла конфигурации сборочной системы "configure.in":  
"AC\_INIT([DB.Tmp1],[0.0.1],[my@email.org])" — информация о модуле: имя, версия и email проекта.  
"AM\_CONDITIONAL([Tmp1Incl],[test])" — AM\_CONDITIONAL([NewModIncl],[test])
6. Установка пакета разработки OpenSCADA. Ввиду того, что модуль внешний и исходные файлы OpenSCADA нужны только на первом этапе создания модуля, необходимо установить пакет разработки OpenSCADA (openscada-devel), который содержит заголовочные файлы и библиотеки.
7. Теперь новый модуль можно собрать, после формирования сборочной системы:  
\$ autoreconf -if  
\$ configure  
\$ make

## 2. API модуля

API системы OpenSCADA для разработчика OpenSCADA и модулей к ней исчерпывающе, в формальной форме, описано в соответствующем документе [API системы OpenSCADA](#), который должен быть всегда под рукой при разработке для OpenSCADA. В данном же документе уклон сделан на детальное разъяснение основных моментов модульного API.

Общим для всех модулей является наследование корневого объекта-класса модуля от класса *TModule* посредством класса модульной подсистемы, а значит есть общая часть интерфейса модуля, которую рассмотрим ниже. В целом, для представления себе архитектуры модулей в контексте общей архитектуры OpenSCADA, настоятельно рекомендуется иметь перед глазами [общую диаграмму классов OpenSCADA!](#)

Точкой входа любого модуля являются функции:

- *TModule::SA module( int n\_mod )*, *TModule::SA bd\_DBF\_module( int n\_mod )* — используется для сканирования перечня и информации о всех модулях в библиотеке. Первая функция используется при реализации модулей во внешней разделяемой библиотеке, а вторая при линковке их в ядро OpenSCADA.
- *TModule \*attach( const TModule::SA &AtMod, const string &source )*, *TModule \*bd\_Tmpl\_attach( const TModule::SA &AtMod, const string &source )* — используется для непосредственного подключения-открытия выбранного модуля путём создания корневого объекта модуля, наследованного от *TModule*. Первая функция используется при реализации модулей во внешней разделяемой библиотеке, а вторая при линковке их в ядро OpenSCADA.

В конструкторе корневого объекта модуля, наследованного от *TModule*, необходимо определить общую мета информацию модуля в составе свойств:

- *mId* — идентификатор модуля, передаётся в аргументе конструктора;
- *mName* — имя модуля;
- *mDescr* — описание модуля;
- *mType* — тип модуля;
- *mVers* — версия модуля;
- *mAutor* — автор модуля;
- *mLicense* — лицензия распространения модуля;
- *mSource* — источник/происхождение модуля, обычно полный путь к файлу разделяемой библиотеки с кодом этого модуля.

А также инициировать окружение модуля с помощью функций:

- *void modFuncReg( ExpFunc \*func );* — Регистрация экспортируемой функции модуля. Эта функция часть механизма вызова межмодульного взаимодействия, которая регистрирует внутреннюю функцию модуля для внешнего вызова по имени функции и её указателю относительно объекта модуля.

Для удобства прямой адресации к корневному объекту модуля из любого объекта модуля ниже по иерархии рекомендуется определять глобальную переменную "mod" в области имён модуля с инициализацией её в конструкторе корневого объекта модуля. Также, для прозрачного перевода текстовых сообщений модуля рекомендуется определять шаблон функции вызова перевода сообщений модуля "**\_({Сообщение})**", как:

```
#undef _  
#define _(mess) mod->I18N(mess)
```

С целью общего управления модулем в классе *TModule* предусмотрен ряд виртуальных функций, которые могут быть определены в корневом объекте модуля с реализацией нужной реакции на команды ядра OpenSCADA к модулю:

- *void load\_();* — Загрузка модуля. Вызывается на стадии загрузки конфигурации модуля из конфигурационного файла или БД.

- `void save_( );` — Сохранение модуля. Вызывается на стадии сохранения конфигурации модуля в конфигурационном файле или БД обычно по инициативе пользователя.
- `void modStart( );` — Запуск модуля. Вызывается на стадии запуска задач исполнения фоновых функций модуля, если таковые модулем предоставляются.
- `void modStop( );` — Останов модуля. Вызывается на стадии останова задач исполнения фоновых функций модуля, если таковые модулем предоставляются.
- `void modInfo( vector<string> &list );` — Запрос списка информационных свойств модуля. Этой функцией класса *TModule* предоставляется стандартный набор свойств модуля ("Module", "Name", "Type", "Source", "Version", "Author", "Description", "License"), который может быть расширен дополнительными свойствами данного модуля.
- `string modInfo( const string &name );` — Запрос указанного элемента информации. Осуществляется обработка запросов к дополнительным свойствам данного модуля.
- `void postEnable( int flag );` — Подключение модуля к динамическому дереву объектов. Вызывается фактически после включения модуля.
- `void perSYSCall( unsigned int cnt );` — Вызов из системного потока с периодичностью 10 секунд и секундным счётчиком `<cnt>`. Может использоваться для выполнения периодических, редких, сервисных процедур.

Все интерфейсные объекты модулей наследуют класс узла *TCntrNode*, который предоставляет механизм [интерфейса управления](#), одной из задач которого является предоставление интерфейса конфигурации объекта в любом конфигураторе OpenSCADA. Для решения задач нового модуля может понадобиться расширение параметров конфигурации, что делается в виртуальной функции `void cntrCmdProc( XMLNode *opt );`. Содержимое этой функции, добавляющее свойство, в простейшем случае имеет вид:

```
void MBD::cntrCmdProc( XMLNode *opt )
{
    //> Get page info
    if(opt->name() == "info")
    {
        TBD::cntrCmdProc(opt);
        ctrMkNode("comm",opt,-1,"/prm/st/end_tr",_("Close opened
transaction"),RWRWRW,"root",SDB_ID);
        return;
    }
    //> Process command to page
    string a_path = opt->attr("path");
    if(a_path == "/prm/st/end_tr" &&
ctrChkNode(opt,"set",RWRWRW,"root",SDB_ID,SEC_WR)) transCommit();
    else TBD::cntrCmdProc(opt);
}
```

Первая половина этой функции обслуживает информационные запросы "info" с перечнем и свойствами полей конфигурации. Вторая половина обслуживает все остальные команды на получение, установку значения и другое. Вызов `TBD::cntrCmdProc(opt);` используется для получения наследованного интерфейса. Детальнее о назначении использованных функций смотрите в [интерфейса управления](#), а также в исходных текстах существующих модулей.

Кроме функции интерфейса управления объект *TCntrNode* предоставляет унифицированные механизмы контроля за модификацией конфигурации объекта, загрузки и сохранения конфигурации в хранилище. Для выполнения установки флага модификации данных объекта можно использовать функции `modif()` и `modifG()`, а специфические для модуля действия по загрузке и сохранению можно помещать в виртуальные функции:

- `void load_( );` — Загрузка объекта из хранилища.
- `void save_( );` — Сохранение объекта в хранилище.

Типично работа с конфигурацией осуществляется посредством объекта *TConfig*, который содержит набор указанных свойств. Для прямого отражения свойств объекта модуля он

наследуется от *TConfig*, а новые свойства добавляются командой:

```
fldAdd(new TFld("PRM_BD",_("Parameters cache table"),TFld::String,TFld::NoFlag,"30",""));
```

Загрузка и сохранение свойств, указанных в объекте *TConfig*, из/в хранилище осуществляется командами:

```
SYS->db().at().dataGet(fullDB(),owner().nodePath()+"DAQ",*this);  
SYS->db().at().dataSet(fullDB(),owner().nodePath()+"DAQ",*this);
```

Где:

- *fullDB()* — полное имя БД-хранилища в виде "{DBMod}.{DBName}.{Table}";
- *owner().nodePath()+"DAQ"* — суммарный путь к узлу объекта — представителя таблицы в конфигурационном файле;
- *\*this* — данный объект, наследованный от *TConfig*.

## 2.1. Модуль подсистемы "Базы Данных (БД)"

Модуль данного типа предназначен для интеграции OpenSCADA с СУБД, реализуемой модулем.

Интерфейс OpenSCADA для обслуживания запросов к БД представлен объектами и виртуальными функциями вызовов из ядра OpenSCADA:

- *TTipBD->TModule* — Корневой объект модуля подсистемы "БД":
  - *TBD \*openBD( const string &id );* — Вызывается при открытии или создании нового объекта БД с идентификатором *<id>* данным модулем.
- *TBD* — Объект базы данных:
  - *void enable();* — Включение БД.
  - *void disable();* — Отключение БД.
  - *void load\_();* — Загрузка БД из общего хранилища конфигурации.
  - *void save\_();* — Сохранение БД в общем хранилище конфигурации.
  - *void allowList( vector<string> &list );* — Запрос перечня *<list>* таблиц в БД.
  - *void sqlReq( const string &req, vector< vector<string> > \*tbl = NULL, char intoTrans = EVAL\_BOOL );* — Обработка SQL-запроса *<req>* к БД и получение результата в виде таблицы *<tbl>*, если запрос выборки и указатель ненулевой. При установке *<intoTrans>* в "true" для запроса должна быть открыта транзакция, в "false" закрыта. Данная функция должна реализоваться для СУБД, поддерживающих SQL-запросы.
  - *void transCloseCheck();* — Периодически вызываемая функция для проверки транзакций и закрытия старых или содержащих много запросов.
  - *TTable \*openTable( const string &table, bool create );* — Вызывается при открытии или создании нового объекта таблицы.
- *TTable* — Объект таблицы в базе данных:
  - *void fieldStruct( TConfig &cfg );* — Получение текущей структуры таблицы в объекте *TConfig*.
  - *bool fieldSeek( int row, TConfig &cfg );* — Последовательное сканирование записей таблицы перебором *<row>* и возврат "false" по окончанию с адресацией по активным, [keyUse\(\)](#), ключевым полям.
  - *void fieldGet( TConfig &cfg );* — Запрос указанной в объекте *TConfig* записи с адресацией по ключевым полям.
  - *void fieldSet( TConfig &cfg );* — Передача указанной в объекте *TConfig* записи с адресацией по ключевым полям.
  - *void fieldDel( TConfig &cfg );* — Удаление указанной записи по ключевым полям объекта *TConfig*.

## 2.2. Модуль подсистемы "Транспорты"

Модуль данного типа предназначен для обеспечения коммуникации OpenSCADA посредством интерфейса, часто сетевого, реализуемого модулем.

Программный интерфейс OpenSCADA для обслуживания входящих и исходящих запросов через сетевой интерфейс представлен объектами и виртуальными функциями вызовов из ядра OpenSCADA:

- *TTipTransport->TModule* — Корневой объект модуля подсистемы "Транспорты":
  - *TTransportIn \*In( const string &name, const string &db );* — Вызывается при открытии или создании нового объекта входящего транспорта *<name>* данным модулем с хранилищем в *<db>*.
  - *TTransportOut \*Out( const string &name, const string &db );* — Вызывается при открытии или создании нового объекта исходящего транспорта *<name>* данным модулем с хранилищем в *<db>*.
- *TTransportIn* — Объект транспорта обработки входящих запросов, функция сервера. Входящие запросы, полученные модулем через реализацию сетевого интерфейса, должны направляться к указанному в конфигурации входящему протоколу *protocol()* посредством функции *mess()*:
  - *string getStatus();* — Вызов для получения специфического статуса интерфейса.
  - *void setAddr( const string &addr );* — Установка адреса транспорта. Может переопределяться для обработки и проверки специфического для модуля формата адреса транспорта.
  - *void start();* — Запуск транспорта. При запуске входящего транспорта обычно создаётся задача, которая ожидает запросов извне.
  - *void stop();* — Останов транспорта.
- *TTransportOut* — Объект транспорта обработки исходящих запросов, функция клиента:
  - *string getStatus();* — Вызов для получения специфического статуса интерфейса.
  - *void setAddr( const string &addr );* — Установка адреса транспорта. Может переопределяться для обработки и проверки специфического для модуля формата адреса транспорта.
  - *void start();* — Запуск транспорта. При запуске исходящего транспорта осуществляется фактическое подключение к удалённой станции для интерфейсов работающих по подключению. В этот момент возможны ошибки, если подключение невозможно, и транспорт должен вернуться в остановленное состояние.
  - *void stop();* — Останов транспорта.
  - *int messIO( const char \*obuf, int len\_ob, char \*ibuf = NULL, int len\_ib = 0, int time = 0, bool noRes = false );* — Обслуживание запросов из ядра OpenSCADA на отправку данных через транспорт. Время ожидания *<time>* соединения указывается в миллисекундах, при ненулевом значении должно замещать одноимённый таймаут транспорта в его общих настройках. *<noRes>* используется протоколами для монопольного блокирования транспорта на время работы с ним и исключения собственной блокировки функцией. Пакет для отправки указывается в буфере *<obuf>* длиной *<len\_ob>*, а в *<ibuf>* и *<len\_ib>* указывается буфер и его размер для ответа. Исходящий буфер *<obuf>* может быть пуст (NULL) если нужно проверить наличие продолжения ответа или ответов, поступающих без запроса, режим вещания. Если не указан буфер для ответа (NULL) то ожидание ответа не будет осуществляться.



### 2.3. Модуль подсистемы "Транспортные протоколы"

Модуль данного типа предназначен для обеспечения протокольного слоя коммуникаций OpenSCADA, реализуемого модулем, как для доступа к данным внешних систем, так и к данным OpenSCADA из внешних систем.

Программный интерфейс OpenSCADA для реализации протокольного слоя представлен объектами и виртуальными функциями вызовов из ядра OpenSCADA:

- *TProtocol->TModule* — Корневой объект модуля подсистемы "Протоколы":
  - *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.
  - *void outMess( XMLNode &io, TTransportOut &tro );* — Реализуемая передача данных объектами ядра OpenSCADA в дереве XML *<in>* удалённой системе посредством транспорта *<tro>* и текущего исходящего протокола. Представление данных в дереве XML *<in>* не унифицировано и специфично логической структуре протокола. Эти данные сериализуются (переводятся в последовательность байтов согласно протоколу) и отправляются через указанный исходящий транспорт *<tro>* функцией *messIO()* выше.
  - *TProtocolIn \*in\_open( const string &name )* — Вызывается при открытии или создании нового объекта входящего транспортного протокола *<name>* данным модулем.
- *TProtocolIn* — Объект протокола обработки входящих запросов из объекта входящего транспорта *TTransportIn* выше. На каждый сеанс входящего запроса создаётся объект связанного входящего протокола, который остаётся жив до момента завершения полного сеанса "Запрос->Ответ". Адрес транспорта, открывшего экземпляр протокола, указан в *srcTr()*:
  - *bool mess( const string &request, string &answer, const string &sender );* — Передача последовательности данных *<request>* объекту протокола для их разбора согласно реализации протокола с указанием адреса запросившего объекта в *<sender>*. Данная функция протокола должна обработать запрос, сформировать ответ в *<answer>* и вернуть "false" в случае полноты запроса. В случае если запрос поступил не весь, нужно возвращать "true" для индикации транспорту "ожидать завершения", при этом первую часть запроса нужно сохранять в контексте объекта протокола.

### 2.4. Модуль подсистемы "Сбор данных (DAQ)"

Модуль этого типа предназначен для получения данных реального времени внешних систем или их формирования в вычислителях, реализуемых модулем.

Программный интерфейс OpenSCADA для реализации доступа к данным реального времени представлен объектами и виртуальными функциями вызовов из ядра OpenSCADA:

- *TTipDAQ->TModule* — Корневой объект модуля подсистемы "Сбор данных":
  - *void compileFuncLangs( vector<string> &ls );* — Запрос перечня языков пользовательского программирования, поддерживаемых модулем в *<ls>*.
  - *void compileFuncSynthHighl( const string &lang, XMLNode &shgl );* — Запрос правил подсветки синтаксиса *<shgl>* указанного языка пользовательского программирования *<lang>*.
  - *string compileFunc( const string &lang, TFunction &fnc\_cfg, const string &prog\_text );* — Вызов компиляции пользовательской процедуры в *<prog\_text>* и создания объекта исполнения функции на основе *<fnc\_cfg>* для указанного языка пользовательского программирования *<lang>* этого модуля. Возвращается адрес к скомпилированному объекту функции, готовому для исполнения.
  - *bool redntAllow( );* — Признак поддержки механизмов резервирования модулем. Должен переопределяться и возвращать "true" в случае поддержки, иначе "false".
  - *TController \*ContrAttach( const string &name, const string &daq\_db );* — Вызывается при

открытии или создании нового объекта контроллера *<name>* данным модулем с хранилищем в *<db>*.

- *TController* — Объект контроллера источника данных. В контексте данного объекта обычно запускается задача периодического или по расписанию опроса данных реального времени одного физического контроллера или физически выделенного блока данных. В случае получения данных пакетами они помещаются непосредственно в архив, связанный с атрибутом параметра *TVal::arch()*, а текущее значение устанавливается функцией *TVal::set()* с атрибутом "sys"=true:
  - *string getStatus()*; — Вызов для получения специфического статуса контроллера.
  - *void enable\_()*; — Включение контроллера. Обычно здесь осуществляется инициализация объектов параметров и их интерфейса в виде атрибутов, которые иногда могут запрашиваться у ассоциированного удалённого источника.
  - *void disable\_()*; — Отключение контроллера.
  - *void start\_()*; — Запуск контроллера. Обычно здесь создаётся и запускается задача периодического или по расписанию опроса.
  - *void stop\_()*; — Останов контроллера.
  - *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>*.
  - *TParamContr \*ParamAttach( const string &name, int type );* — Вызывается при открытии или создании нового объекта параметра *<name>* с типом *<type>*.
- *TParamContr->TValue* — Объект параметра контроллера источника данных. Содержит атрибуты с реальными данными в наборе, определённом физически доступными данными. Значения в атрибуты попадают из задачи опроса контроллера при асинхронном режиме или запрашиваются в момент обращения при синхронном режиме посредством методов наследованного типа *TValue* данного объекта:
  - *void enable()*; — Включить параметр. Осуществляется формирование набора атрибутов и заполнение их значением достоверности.
  - *void disable()*; — Отключить параметр.
  - *void setType( const string &tpId );* — Вызывается для смены типа параметра *<tpId>* и может быть обработан в объекте модуля для смены собственных данных.
  - *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.
  - *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.
  - *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.
  - *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.

## 2.5. Модуль подсистемы "Архивы"

Модуль этого типа предназначен для архивирования и ведения истории сообщений OpenSCADA и данных реального времени, полученных в подсистеме "Сбор данных" реализуемым модулем способом.

Программный интерфейс OpenSCADA для реализации доступа к архивным данным представлен объектами и виртуальными функциями вызовов из ядра OpenSCADA:

- *TTipArchivator*->*TModule* — Корневой объект модуля подсистемы "Архивы":
  - *TMArchivator \*AMess(const string &id, const string &db );* — Вызывается при открытии или создании нового объекта архиватора сообщений *<id>* данным модулем с хранилищем в *<db>*.
  - *TVArchivator \*AVal(const string &id, const string &db );* — Вызывается при открытии или создании нового объекта архиватора значений *<id>* данным модулем с хранилищем в *<db>*.
- *TMArchivator* — Объект архиватора сообщений с реализуемым способом архивирования и расположением хранилища:
  - *void start( );* — Запуск архиватора. Архиватор начинает принимать сообщения и размещать их в хранилище.
  - *void stop( );* — Останов архиватора.
  - *time\_t begin( );* — Начало данных в архиваторе согласно текущему состоянию хранилища.
  - *time\_t end( );* — Конец данных в архиваторе согласно текущему состоянию хранилища.
  - *void put( vector<TMess::SRec> &mess );* — Вызов на размещение сообщений *<mess>* в хранилище.
  - *void get( time\_t b\_tm, time\_t e\_tm, vector<TMess::SRec> &mess, const string &category = "", char level = 0, time\_t upTo = 0 );* — Запрос сообщений *<mess>* в архиве за промежуток времени *<b\_tm> ... <e\_tm>* согласно шаблону категории *<category>* и уровню с ограничением на время запроса до *<upTo>*.
- *TVArchivator* — Объект архиватора значений с реализуемым способом архивирования и расположением хранилища:
  - *void setValPeriod( double per );* — Вызывается при смене периодичности значений архиватора.
  - *void setArchPeriod( int per );* — Вызывается при смене периодичности архивирования.
  - *void start( );* — Запуск архиватора. Архиватор начинает принимать сообщения и размещать их в хранилище.
  - *void stop( bool full\_del = false );* — Останов архиватора с возможностью полного удаления его данных в хранилище, если установлен *<full\_del>*.
  - *TVArchEl \*getArchEl( TVArchive &arch );* — Запрос объекта-представителя архива *<arch>*, обслуживаемого архиватором.
- *TVArchEl* — Объект представителя архива значений в хранилище архиватора:
  - *void fullErase( );* — Вызывается для полного удаления части архива в архиваторе.
  - *int64\_t end( );* — Время окончания архива в архиваторе.
  - *int64\_t begin( );* — Время начала архива в архиваторе.
  - *TVariant getValProc( int64\_t \*tm, bool up\_ord );* — Запрос на обработку получения одного значения из архива за время *<tm>* и доводкой к верхнему значению в сетке дискретизации *<up\_ord>*.
  - *void getValsProc( TValBuf &buf, int64\_t beg, int64\_t end );* — Запрос на обработку модулем получения данных группы значений *<buf>* за указанный промежуток времени.
  - *void setValsProc( TValBuf &buf, int64\_t beg, int64\_t end );* — Запрос на обработку модулем размещения данных группы значений *<buf>* за указанный промежуток времени.

## 2.6. Модуль подсистемы "Пользовательские интерфейсы (UI)"

Модуль этого типа предназначен для предоставления пользовательского интерфейса реализуемым модулем способом. Корневым объектом модуля данной подсистемы является *TUI->TModule*, который не содержит специфических интерфейсов, а пользовательский интерфейс формируется согласно с реализуемой концепцией и механизмами, например, библиотеки графических примитивов.

## 2.7. Модуль подсистемы "Специальные"

Модуль этого типа предназначен для реализации специфических функций, не вошедших ни в одну из вышеперечисленных подсистем, реализуемым модулем способом. Корневым объектом модуля данной подсистемы является *TSpecial->TModule*, который не содержит специфических интерфейсов, а специфические функции формируются согласно их требованиям с использованием всех возможностей API OpenSCADA.