

Керівний документ по створенню модулів для OpenSCADA


Зміст

Керівний документ по створенню модулів для OpenSCADA	1
Вступ	1
1. Створення нового модуля	2
1.1. Створення у дереві вихідних текстів проекту OpenSCADA	2
1.2. Створення зовнішнього модуля до OpenSCADA	4
2. API модуля	5
2.1. Модуль підсистеми "Бази Даних (БД)"	7
2.2. Модуль підсистеми "Транспорти"	8
2.3. Модуль підсистеми "Транспортні протоколи"	9
2.4. Модуль підсистеми "Збір даних (DAQ)"	9
2.5. Модуль підсистеми "Архіви"	11
2.6. Модуль підсистеми "Користувацькі інтерфейси (UI)"	12
2.7. Модуль підсистеми "Спеціальні"	12

Вступ

Цей керівний документ покликано допомогти у створенні модулів для системи OpenSCADA. Створення модуля може знадобитися у випадку бажання додати підтримку нового джерела даних або іншого розширення до системи OpenSCADA. Оскільки OpenSCADA є гранично модульною то всі інтерфейси взаємодії з зовнішнім середовищем здійснюється за посередництвом розширення спеціалізованими модулями типів:

- "Бази даних"
- "Комунікаційні інтерфейси, транспорти"
- "Протоколи комунікаційних інтерфейсів"
- "Джерела даних та збір даних"
- "Архіви (повідомлень та значень)"
- "Інтерфейси користувача (GUI, TUI, WebGUI, speech, signal ...)"
- "Додаткові модулі, спеціальні"

 Для створення модулів OpenSCADA потрібні знання у програмуванні на мові C/C++, складальній системі [AutoTools](#), а також базові знання у ОС Linux та використовуюваного дистрибутиву Linux.

1. Створення нового модуля

Модулі в OpenSCADA представляють із себе поділювані бібліотеки, які підключаються до ядра OpenSCADA динамічно, під час роботи програми. Багато модулів у процесі роботи можуть бути відключені, підключені та оновлені із [менеджера модулів \(RU\)](#). Модулі також можуть бути включені у ядро OpenSCADA під час збірки, за посередництвом аргументу `--enable-{ModName}=incl` до скрипту конфігурації "configure", про що можна дізнатися із [керівного документа по збірці](#). Модулі OpenSCADA можуть бути семи типів, згідно присутнім [модульним підсистемам \(RU\)](#). Наразі модулі до системи OpenSCADA пишуться на мові програмування "C++", хоча у подальшому можлива поява біндингів на інші мови.

Для полегшення створення нових модулів у дереві вихідних текстів, у гілці кожної підсистеми, передбачено теку "`=Tmpl=`" з шаблоном модуля відповідної підсистеми. Розробник нового модуля може взяти цю теку та скопіювати її з ім'ям свого нового модуля. Передбачено можливість створення модулів у дереві вихідних текстів проекту OpenSCADA або як незалежного проекту зовнішнього модуля до OpenSCADA.

1.1. Створення у дереві вихідних текстів проекту OpenSCADA

Створювати нові модулі у дереві вихідних текстів проекту OpenSCADA має сенс у випадку подальших планів передачі нового модуля проекту OpenSCADA. Оскільки модуль не має суперечити духу відкритого проекту та ліцензії на основі якої [розробляється та розповсюджується OpenSCADA](#), то ліцензією нового модуля вочевидь має бути одна із вільних ліцензій.

В цілому процедура створення нового модуля з включенням до дерева вихідних текстів, на основі шаблону, є простішою за процедуру для зовнішнього модуля та включає в себе кроки:

1. Отримання дерева вихідних текстів проекту OpenSCADA.

Для робочої гілки:

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

Для гілки стабільного релізу (небажано оскільки до стабільних LTS релізів приймаються тільки виправлення):

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

2. Копіювання теки шаблону з ім'ям нового модуля "NewMod" (наприклад, для підсистеми "БД"):

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

3. Редагування файлу "module.cpp".

Змінити імена функцій включення модуля згідно імені 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
```

Інформація про модуль у файлі "module.cpp", а саме ділянка:

```
/**
*****
**/
/* 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. Редагування конфігурації збірки модуля у файлі "Makefile.am", до такого вигляду:

```
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_LDLFLAGS)
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. Додання запису нового модуля в кінець секції підсистеми (у нас "> DB modules"), конфігураційного файлу (OpenSCADA/configure.in) складальної системи 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 перевірки зовнішніх бібліотек модуля
    if test $enable_NewMod = incl; then
        LIB_CORE="{LIB_CORE} moduls/bd/NewMod/.libs/*.o "
        ModsIncl="{ModsIncl}bd_NewMod "
    fi
]
)

```

6. Тепер новий модуль можна скласти у складі OpenSCADA, після переформування складальної системи:

```

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

```

7. Публікація. Формування патча з вашим модулем та відправка його розробникам 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. Створення зовнішнього модуля до OpenSCADA

Створення зовнішнього модуля до OpenSCADA може мати сенс у випадку розробки інтерфейсу інтеграції з комерційними системами, які вимагають закриття коду взаємодії, а також у випадку інших реалізацій комерційних інтерфейсів при яких модуль до OpenSCADA отримує статус окремого проекту, розповсюджується та підтримується незалежно, часто у вигляді бінарних збірок під конкретну платформу та версію OpenSCADA. Ліцензія таких модулів відповідно може бути будь-якою.

Процедура створення нового зовнішнього модуля, на основі шаблону, багато в чому схожа на попередню процедуру та включає в себе кроки:

1. Отримання вихідних текстів проекту OpenSCADA. Для зовнішнього модуля у якості джерела шаблону можна використати будь-які файли OpenSCADA версії більш 0.8.0 оскільки із них потрібно скопіювати тільки теку "=Tmpl=" та декілька файлів для збірки.
2. Копіювання теки шаблону з ім'ям нового модуля "NewMod" (наприклад, для підсистеми "БД"). Створення та копіювання потрібних файлів для зовнішнього модуля. В подальшому інформаційні файли проекту "COPYING", "NEWS", "README", "AUTHORS" та "ChangeLog" потрібно заповнити згідно сутності нового модуля.

```
$ cp -r OpenSCADA/src/moduls/bd/=Tmpl= NewMod
$ rm -rf NewMod/.svn NewMod/po/.svn
$ touch NewMod/{NEWS,README,AUTHORS,ChangeLog}
$ cp OpenSCADA/I18N.mk NewMod/
```
3. Редагування інформації про модуль у файлі "module.cpp", аналогічно цьому пункту попереднього розділу.
4. Редагування конфігурації збірки модуля у файлі "Makefile.am", аналогічно цьому пункту попереднього розділу.
5. Редагування файлу конфігурації складальної системи "configure.in":

```
"AC_INIT([DB.Tmpl],[0.0.1],[my@email.org])" — інформація про модуль: ім'я, версія та email проекту.
"AM_CONDITIONAL([TmplIncl],[test])" — AM_CONDITIONAL([NewModIncl],[test])
```
6. Встановлення пакету розробки OpenSCADA. У зв'язку з тим що модуль зовнішній та вихідні файли OpenSCADA потрібні тільки на першому етапі створення модуля то потрібно встановити пакет розробки OpenSCADA (openscada-devel), який містить заголовні файли та бібліотеки.
7. Тепер новий модуль можна скласти, після формування складальної системи:

```
$ autoreconf -if
$ configure
$ make
```

2. API модуля

API системи OpenSCADA для розробника OpenSCADA та модулів до неї вичерпно, у формальній формі, описано у відповідному документі [API системи OpenSCADA \(RU\)](#), який має бути завжди під рукою при розробці для OpenSCADA. В цьому ж документі ухил зроблено на детальне роз'яснення основних моментів модульного API.

Загальним для всіх модулів є наслідування кореневого об'єкта-класу модуля від класу *TModule*, за посередництвом класу модульної підсистеми, а це означає що є загальна частина інтерфейсу модуля який розглянемо нижче. В цілому, для представлення собі архітектури модулів у контексті загальної архітектури OpenSCADA, наполегливо рекомендується мати перед очима [загальну діаграму класів OpenSCADA \(RU\)](#)!

Точкою входу будь-якого модуля є функції:

- *TModule::SAt module(int n_mod)*, *TModule::SAt bd_DBF_module(int n_mod)* — використовуються для сканування переліку та інформації про всі модулі у бібліотеці. Перша функція використовується під час реалізації модулів у зовнішній поділюваній бібліотеці, а друга під час лінування їх у ядро OpenSCADA.
- *TModule *attach(const TModule::SAt &AtMod, const string &source)*, *TModule *bd_Tmpl_attach(const TModule::SAt &AtMod, const string &source)* — використовується для безпосереднього підключення-відкриття обраного модуля, шляхом створення кореневого об'єкта модуля, успадкованого від *TModule*. Перша функція використовується під час реалізації модулів у зовнішній поділюваній бібліотеці, а друга під час лінування їх у ядро OpenSCADA.

У конструкторі кореневого об'єкта модуля, успадкованого від *TModule*, потрібно визначити загальну-мета інформацію модуля у складі властивостей:

- *mId* — ідентифікатор модуля, передається у аргументі конструктора;
- *mName* — ім'я модуля;
- *mDescr* — опис модуля;
- *mType* — тип модуля;
- *mVers* — версія модуля;
- *mAutor* — автор модуля;
- *mLicense* — ліцензія розповсюдження модуля;
- *mSource* — джерело/походження модуля, за звичай повний шлях до файлу поділюваної бібліотеки з кодом цього модуля.

А також ініціювати оточення модуля за допомогою функцій:

- *void modFuncReg(ExpFunc *func);* — Реєстрація експортованої функції модуля. Ця функція частина механізму виклику міжмодульної взаємодії, яка реєструє внутрішню функцію модуля для зовнішнього виклику за назвою функції та її вказівником відносно об'єкта модуля.

Для зручності прямої адресації до кореневого об'єкта модуля із будь-якого об'єкта модуля нижче за ієрархією рекомендується визначити глобальну змінну "mod", у області імен модуля, з ініціалізацією її у конструкторі кореневого об'єкта модуля. Також, для прозорого перекладу текстових повідомлень модуля, рекомендується визначити шаблон функції виклику перекладу повідомлень модуля "**_{Повідомлення}**", як:

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

З метою загального управління модулем у класі *TModule* передбачено низку віртуальних функцій, які можуть бути визначені у кореневому об'єкті модуля з реалізацією потрібної реакції на команди ядра OpenSCADA до модуля:

- `void load_();` — Завантаження модуля. Викликається на стадії завантаження конфігурації модуля із конфігураційного файлу або БД.
- `void save_();` — Збереження модуля. Викликається на стадії збереження конфігурації модуля у конфігураційному файлі або БД, зазвичай за ініціативою користувача.
- `void modStart();` — Запуск модуля. Викликається на стадії запуску задачі виконання фонових функцій модуля, якщо такі модулем надаються.
- `void modStop();` — Зупинка модуля. Викликається на стадії зупинки задач виконання фонових функцій модуля, якщо такі модулем надаються.
- `void modInfo(vector<string> &list);` — Запит переліку інформаційних властивостей модуля. Цією функцією класу *TModule* надається стандартний набір властивостей модуля ("Module", "Name", "Type", "Source", "Version", "Author", "Description", "License"), який може бути розширено додатковими властивостями цього модуля.
- `string modInfo(const string &name);` — Запит вказаного елемента інформації. Здійснюється обробка запитів по додатковим властивостям даного модуля.
- `void postEnable(int flag);` — Підключення модуля до динамічного дерева об'єктів. Викликається фактично після включення модуля.
- `void perSYSCall(unsigned int cnt);` — Виклик із системного потоку, з періодичністю 10 секунд та секундним лічильником `<cnt>`. Може використовуватися для виконання періодичних-рідких сервісних процедур.

Всі інтерфейсні об'єкти модулів успадковують клас вузла *TCntrNode*, який надає механізм [інтерфейсу управління \(RU\)](#), однієї із задач якого є надання інтерфейсу конфігурації об'єкту в будь-якому конфігураторі OpenSCADA. Для вирішення задач нового модуля може знадобитися розширення параметрів конфігурації, що робиться у віртуальній функції `void cntrCmdProc(XMLNode *opt);`. Вміст цієї функції, який додає властивість, у найпростішому випадку має вигляд:

```
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);
}
```

Перша половина цієї функції обслуговує інформаційні запити "info", з переліком та властивостями полів конфігурації. Друга половина обслуговує всі інші команди, на отримання, встановлення значення та інше. Виклик `TBD::cntrCmdProc(opt)`; використовується для отримання успадкованого інтерфейсу. Детальніше про призначення використаних функцій дивіться у [інтерфейсі управління \(RU\)](#), а також у вихідних текстах існуючих модулів.

Крім функції інтерфейсу управління об'єкт *TCntrNode* надає уніфіковані механізми контролю за модифікацією конфігурації об'єкта, завантаження та збереження конфігурації у сховищі. Для здійснення встановлення прапорця модифікації даних об'єкта можна використовувати функції `modif()` та `modifG()`, а специфічні для модуля дії по завантаженню та збереженню можна розташовувати у віртуальні функції:

- `void load_();` — Завантаження об'єкта із сховища.
- `void save_();` — Збереження об'єкта у сховищі.

Типово, робота з конфігурацією здійснюється за посередництвом об'єкта *TConfig (RU)*, яких містить набір вказаних властивостей. Для прямого відображення властивостей об'єкта модуля він успадковується від *TConfig*, а нові властивості додаються командою:

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

Завантаження та збереження властивостей, вказаних у об'єкті *TConfig*, із/у сховище здійснюється командами:

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

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

Де:

- *fullDB()* — повне ім'я БД-сховища, у вигляді "**{DBMod}.{DBName}.{Table}**";
- *owner().nodePath()+"DAQ"* — сумарний шлях до вузла об'єкта — представника таблиці у конфігураційному файлі;
- **this* — цей об'єкт, успадковується від *TConfig*.

2.1. Модуль підсистеми "Бази Даних (БД)"

Модуль даного типу призначено для інтеграції OpenSCADA зі СУБД, яку реалізується модулем.

Інтерфейс OpenSCADA для обслуговування запитів до БД представлено об'єктами та віртуальними функціями викликів із ядра OpenSCADA:

- *TTipBD->TModule* — Кореневий об'єкт модуля підсистеми "БД":
 - *TBD *openBD(const string &id);* — Викликається при відкритті або створенні нового об'єкта БД з ідентифікатором *<id>* даним модулем.
- *TBD* — Об'єкт бази даних:
 - *void enable();* — Включення БД.
 - *void disable();* — Відключення БД.
 - *void load_();* — Завантаження БД із загального сховища конфігурації.
 - *void save_();* — Збереження БД у загальному сховищі конфігурації.
 - *void allowList(vector<string> &list);* — Запит переліку *<list>* таблиць у БД.
 - *void sqlReq(const string &req, vector< vector<string> > *tbl = NULL, char intoTrans = EVAL_BOOL);* — Обробка SQL-запиту *<req>* до БД та отримання результату у вигляді таблиці *<tbl>*, якщо запит вибірки та вказівник ненульовий. При встановленні *<intoTrans>* у "true" для запиту мусить бути відкрита транзакція, у "false" закрита. Ця функція має реалізовуватися для СУБД, які підтримують SQL-запити.
 - *void transCloseCheck();* — Періодично викликається функція для перевірки транзакцій та закриття старих або які містять багато запитів.
 - *TTable *openTable(const string &table, bool create);* — Викликається при відкритті або створенні нового об'єкта таблиці.
- *TTable* — Об'єкт таблиці у базі даних:
 - *void fieldStruct(TConfig &cfg);* — Отримання поточної структури таблиці у об'єкті *TConfig*.
 - *bool fieldSeek(int row, TConfig &cfg);* — Послідовне сканування записів таблиці, перебором *<row>* та повернення "false" по закінченню, з адресацією по активним, [keyUse\(\) \(RU\)](#), ключовим полям.
 - *void fieldGet(TConfig &cfg);* — Запит вказаного у об'єкті *TConfig* запису, з адресацією по ключовим полям.
 - *void fieldSet(TConfig &cfg);* — Передача вказаного у об'єкті *TConfig* запису, з адресацією по ключовим полям.
 - *void fieldDel(TConfig &cfg);* — Видалення вказаного запису по ключовим полям об'єкта *TConfig*.

2.2. Модуль підсистеми "Транспорти"

Модуль даного типу призначено для забезпечення комунікації OpenSCADA за посередництвом інтерфейсу, часто мережевого, який реалізується модулем.

Програмний інтерфейс OpenSCADA, для обслуговування вхідних та вихідних запитів через мережевий інтерфейс, представлено об'єктами та віртуальними функціями викликів із ядра OpenSCADA:

- *TTipTransport->TModule* — Кореневий об'єкт модуля підсистеми "Транспорти":
 - *TTransportIn *In(const string &name, const string &db);* — Викликаються при відкритті або створенні нового об'єкта вхідного транспорту *<name>*, даним модулем, зі сховищем у *<db>*.
 - *TTransportOut *Out(const string &name, const string &db);* — Викликаються при відкритті або створенні нового об'єкта вихідного транспорту *<name>*, даним модулем, зі сховищем у *<db>*.
- *TTransportIn* — Об'єкт транспорту обробки вхідних запитів, функція серверу. Вхідні запити, отримані модулем через реалізацію мережевого інтерфейсу повинні направлятися до вказаного у конфігурації вхідного протоколу *protocol()*, за посередництвом функції [*mess\(\)*](#) (*RU*):
 - *string getStatus();* — Виклики для отримання специфічного статусу інтерфейсу.
 - *void setAddr(const string &addr);* — Встановлення адреси транспорту. Може перевизначатися для обробки та перевірки специфічного для модуля формату адреси транспорту.
 - *void start();* — Запуск транспорту. При запуску вхідного транспорту зазвичай створюється задача, яка очікує запитів ззовні.
 - *void stop();* — Зупинка транспорту.
- *TTransportOut* — Об'єкт транспорту обробки вихідних запитів, функція клієнта:
 - *string getStatus();* — Виклик для отримання специфічного статусу інтерфейсу.
 - *void setAddr(const string &addr);* — Встановлення адреси транспорту. Може перевизначатися для обробки та перевірки специфічного для модуля формату адреси транспорту.
 - *void start();* — Запуск транспорту. Під час запуску вихідного транспорту здійснюється фактичне підключення до віддаленої станції, для інтерфейсів які працюють по підключенню. У цей час можливі помилки, якщо підключення не можливе, та транспорт мусить повернутися у зупинений стан.
 - *void stop();* — Зупинка транспорту.
 - *int messIO(const char *obuf, int len_ob, char *ibuf = NULL, int len_ib = 0, int time = 0, bool noRes = false);* — Обслуговування запитів із ядра OpenSCADA на відправку даних через транспорт. Час очікування *<time>* з'єднання визначається у мілісекундах, при ненульовому значенні має заміщувати однойменний таймаут транспорту в його загальних налаштуваннях. *<noRes>* використовується протоколами для монопольного блокування транспорту на час роботи з ним та попередження власного блокування функцією. Пакет для відправки вказується у буфері *<obuf>* довжиною *<len_ob>*, а у *<ibuf>* та *<len_ib>* вказується буфер та його розмір для відповіді. Вихідний буфер *<obuf>* може бути порожнім (NULL) якщо потрібно перевірити наявність продовження відповіді або відповідей, які надходять без запитів, режим віщання. Якщо не вказано буфер для відповіді (NULL) то очікування відповіді не буде здійснюватися.

2.3. Модуль підсистеми "Транспортні протоколи"

Модуль даного типу призначено для забезпечення протокольного шару комунікацій OpenSCADA, який реалізується модулем, як для доступу к даним зовнішніх систем, так і даних OpenSCADA, із зовнішніх систем.

Програмний інтерфейс OpenSCADA, для реалізації протокольного шару, представлено об'єктами та віртуальними функціями викликів із ядра OpenSCADA:

- *TProtocol->TModule* — Кореневий об'єкт модуля підсистеми "Протоколи":
 - *void itemListIn(vector<string> &ls, const string &curlt = "");* — Перелік піделементів у вхідному протоколу, якщо протокол їх передбачає. Використовується при обранні у конфігурації об'єкту вхідного транспорту.
 - *void outMess(XMLNode &io, TTransportOut &tro);* — Реалізована передача даних об'єктами ядра OpenSCADA у дереві XML *<in>* віддаленій системі за посередництвом транспорту *<tro>* и поточного вихідного протоколу. Надання даних у дереві XML *<in>* не уніфіковано та специфічно до логічної структури протоколу. Ці дані серіалізуються (переводяться у послідовність байтів згідно протоколу) та надсилаються через вказаний вихідний транспорт *<tro>*, функцією *messIO(RU)*, вище.
 - *TProtocolIn *in_open(const string &name)* — Викликається при відкритті або створенні нового об'єкта вхідного транспортного протоколу *<name>*, цим модулем.
- *TProtocolIn* — Об'єкт протоколу обробки вхідних запитів із об'єкта вхідного транспорту *TTransportIn (RU)*, вище. На кожний сеанс вхідного запиту створюється об'єкт пов'язаного вхідного протоколу, який залишається живим до моменту завершення повного сеансу "Запит->Відповідь". Адреса транспорту, який відкрив екземпляр протоколу, вказано у *srcTr()*:
 - *bool mess(const string &request, string &answer, const string &sender);* — Передача послідовності даних *<request>* об'єкту протоколу для їх розбору згідно з реалізацією протоколу, з вказанням адреси того хто запитав у *<sender>*. Ця функція протоколу мусить обробити запит, сформувавши відповідь у *<answer>* та повернути "false" у випадку повноти запиту. У випадку якщо запит поступив не весь потрібно повертати "true" для індикації транспорту "очікувати завершення", при цьому першу частину запиту потрібно зберігати у контексті об'єкта протоколу.

2.4. Модуль підсистеми "Збір даних (DAQ)"

Модуль цього типу призначено для отримання даних реального часу зовнішніх систем або їх формування, у обчислювачах, які реалізуються модулем.

Програмний інтерфейс OpenSCADA для реалізації доступу до даних реального часу представлено об'єктами та віртуальними функціями викликів із ядра OpenSCADA:

- *TTipDAQ->TModule* — Кореневий об'єкт модуля підсистеми "Збір даних":
 - *void compileFuncLangs(vector<string> &ls);* — Запит переліку мов користувацького програмування, які підтримуються модулем в *<ls>*.
 - *void compileFuncSynthHigh(const string &lang, XMLNode &shgl);* — Запит правил підсвічення синтаксису *<shgl>* вказаної мови користувацького програмування *<lang>*.
 - *string compileFunc(const string &lang, TFunction &func_cfg, const string &prog_text);* — Виклик компіляції користувацької процедури у *<prog_text>* та створення об'єкту виконання функції на основі *<func_cfg>* для вказаної мови користувацького програмування *<lang>* цього модуля. Повертається адреса до скомпільованого об'єкту функції, готового для виконання.
 - *bool redntAllow();* — Ознака підтримки механізмів резервування модулів. Повинен перевизначатися та повертати "true" у випадку підтримки, інакше "false".

- *TController *ContrAttach(const string &name, const string &daq_db);* — Викликається при відкритті або створенні нового об'єкта контролера *<name>* цим модулем зі сховищем у *<db>*.
- *TController* — Об'єкт контролера джерела даних. У контексті даного об'єкту за звичай запускається задача періодичного або за розкладом опиту даних реального часу одного фізичного контролера або фізично відокремленого блоку даних. У випадку отримання даних пакетами вони поміщаються безпосередньо до архіву, пов'язаного з атрибутом параметра *TVal::arch() (RU)*, а поточне значення встановлюється функцією *TVal::set() (RU)*, с атрибутом "sys"=true:
 - *string getStatus();* — Виклик для отримання специфічного стану контролера.
 - *void enable_();* — Включення контролеру. Зазвичай тут здійснюється ініціалізація об'єктів параметрів та їх інтерфейсу, в особі атрибутів, які інколи можуть запитуватися у асоційованого віддаленого джерела.
 - *void disable_();* — Відключення контролеру.
 - *void start_();* — Запуск контролеру. Зазвичай тут створюється та запускається задача періодичного або за розкладом опитування.
 - *void stop_();* — Зупинка контролеру.
 - *void redntDataUpdate(bool firstArchiveSync = false);* — Виконання операції отримання даних із резервної станції. Викликається автоматично завданням обслуговування схеми резервування та перед запуском для синхронізації архівів зі встановленим параметром *<firstArchiveSync>*.
- *TParamContr->TValue* — Об'єкт параметра контролера джерела даних. Містить атрибути з реальними даними у наборі визначеному фізично доступними даними. Значення у атрибути попадають із задачі опитування контролера, при асинхронному режимі, або запитуються під час звернення, при синхронному режимі, за посередництвом методів успадкованого типу *TValue (RU)*, даного об'єкта:
 - *void enable();* — Включити параметр. Здійснюється формування набору атрибутів та заповнення їх значенням недостовірності.
 - *void disable();* — Відключити параметр.
 - *void setType(const string &tpld);* — Викликається для зміни типу параметра *<tpld>* та може бути оброблено у об'єкті модуля, для зміни власних даних.
 - *TVal* vlNew();* — Викликається при створенні нового атрибута. Може бути перевизначено для реалізації особливої поведінки у межах свого, успадкованого від *TVal*, класу при доступі до атрибута.
 - *void vlSet(TVal &val, const TVariant &pvl);* — Викликається для атрибута з прямим режимом запису *TVal::DirWrite* (синхронний режим або запис до внутрішнього буфера об'єкта) при встановленні значення з метою безпосереднього запису значення до фізичного контролера або буфера об'єкта.
 - *void vlGet(TVal &val);* — Викликається для атрибута з прямим режимом читання *TVal::DirRead* (синхронний режим або читання із внутрішнього буфера об'єкта) при читанні значень з метою безпосереднього читання значення із фізичного контролера або буфера об'єкта.
 - *void vlArchMake(TVal &val);* — Викликається при створенні архіву значень з атрибутом *<val>* у якості джерела з метою ініціалізації якісних характеристик буфера архіву згідно особливостям джерела даних та їх опитування.

2.5. Модуль підсистеми "Архіви"

Модуль цього типу призначено для архівування та ведення історії, повідомлень OpenSCADA та даних реального часу, отриманих у підсистемі "Збір даних", засобом який реалізується модулем.

Програмний інтерфейс OpenSCADA, для реалізації доступу до архівних даних, представлено об'єктами та віртуальними функціями викликів із ядра OpenSCADA:

- *TTipArchivator*->*TModule* — Кореневий об'єкт модуля підсистеми "Архіви":
 - *TMArchivator *AMess(const string &id, const string &db);* — Викликається при відкритті або створенні нового об'єкта архіватора повідомлень *<id>* даним модулем зі сховищем у *<db>*.
 - *TVArchivator *AVal(const string &id, const string &db);* — Викликається при відкритті або створенні нового об'єкта архіватора значень *<id>* даним модулем зі сховищем у *<db>*.
- *TMArchivator* — Об'єкт архіватора повідомлень з реалізованим засобом архівування та розташуванням сховища:
 - *void start();* — Запуск архіватора. Архіватор починає приймати повідомлення та розташовувати їх у сховищі.
 - *void stop();* — Зупинка архіватора.
 - *time_t begin();* — Початок даних у архіваторі, згідно поточного стану сховища.
 - *time_t end();* — Кінець даних у архіваторі, згідно поточному стану сховища.
 - *void put(vector<TMess::SRec> &mess);* — Виклик на розташування повідомлень *<mess>* у сховищі.
 - *void get(time_t b_tm, time_t e_tm, vector<TMess::SRec> &mess, const string &category = "", char level = 0, time_t upTo = 0);* — Запит повідомлень *<mess>* у архіві за проміжок часу *<b_tm> ... <e_tm>*, згідно шаблону категорії *<category>* та рівню з обмеженням на час запити до *<upTo>*.
- *TVArchivator* — Об'єкт архіватора значень з реалізованим способом архівування та розташуванням сховища:
 - *void setValPeriod(double per);* — Викликається при зміні періодичності значень архіватора.
 - *void setArchPeriod(int per);* — Викликається при зміні періодичності архівування.
 - *void start();* — Запуск архіватора. Архіватор починає приймати повідомлення та розташовувати їх у сховищі.
 - *void stop(bool full_del = false);* — Зупинка архіватора з можливістю повного видалення його даних у сховищі, якщо встановлено *<full_del>*.
 - *TVArchEl *getArchEl(TVArchive &arch);* — Запит об'єкта-представника архіву *<arch>*, який обслуговується архіватором.
- *TVArchEl* — Об'єкт представника архіву значень у сховищі архіватора:
 - *void fullErase();* — Викликається для повного видалення частини архіву у архіваторі.
 - *int64_t end();* — Час закінчення архіву у архіваторі.
 - *int64_t begin();* — Час початку архіву в архіваторі.
 - *TVariant getValProc(int64_t *tm, bool up_ord);* — Запит на обробку отримання одного значення із архіву, за час *<tm>* та доведенням до верхнього значення у гратці дискретизації *<up_ord>*.
 - *void getValsProc(TValBuf &buf, int64_t beg, int64_t end);* — Запит на обробку модулем отримання даних групи значень *<buf>* за визначений проміжок часу.
 - *void setValsProc(TValBuf &buf, int64_t beg, int64_t end);* — Запит на обробку модулем розташування даних групи значень *<buf>* за визначений проміжок часу.

2.6. Модуль підсистеми "Користувацькі інтерфейси (UI)"

Модуль цього типу призначено для надання користувацького інтерфейсу, засобом який реалізується модулем. Кореневим об'єктом модуля даної підсистеми є *TUI->TModule*, який не містить специфічних інтерфейсів, а користувацький інтерфейс формується згідно із реалізованою концепцією та механізмами, наприклад, бібліотеки графічних примітивів.

2.7. Модуль підсистеми "Спеціальні"

Модуль цього типу призначено для реалізації специфічних функцій, які не увійшли до жодної з вищеперелічених підсистем, реалізованим модулем засобом. Кореневим об'єктом модуля даної підсистеми є *TSpecial->TModule*, який не містить специфічних інтерфейсів, а специфічні функції формуються згідно їх вимогами з використанням всіх можливостей API OpenSCADA.