

**НАСТРОЙКА  
ОБМЕНА ДАННЫМИ  
ПО СТАНДАРТУ IEC 61850  
НА КОНТРОЛЛЕРАХ СЕРИИ  
REGUL RX00**

**Руководство пользователя**

**DPA-302.14**

**Версия ПО 1.7.2.0**

**Версия 1.0**

**Сентябрь 2024**

### История изменений руководства пользователя

Версия документа	Описание изменения
1.0	Релизная версия

## АННОТАЦИЯ

Настоящий документ содержит сведения о настройке передачи данных по стандарту IEC 61850 на промышленных логических контроллерах серии Regul RX00. Настройка осуществляется с помощью программного обеспечения Astra.IDE.

Данное руководство предназначено для эксплуатационного персонала и инженеров-проектировщиков АСУ ТП, которые должны:

- иметь, как минимум, среднее техническое образование;
- приступить к работе только после изучения данного руководства.


### Обновление информации в Руководстве

Производитель ООО «РегЛаб» оставляет за собой право изменять информацию в настоящем Руководстве и обязуется публиковать более новые версии с внесенными изменениями. Обновленная версия Руководства доступна для скачивания на официальном сайте Производителя: <https://reglab.ru/>.


Для своевременного отслеживания выхода новой версии Руководства рекомендуется оформить подписку на обновление документа. Для этого необходимо на сайте Производителя: <https://reglab.ru/> кликнуть на кнопку «Подписаться на обновления» и оставить свои контактные данные.

В руководстве присутствуют знаки с предупреждающей и поясняющей информацией. Каждый знак обозначает следующее:

### ПРЕДУПРЕЖДАЮЩИЕ ЗНАКИ

	<p><b>ВНИМАНИЕ!</b></p> <p>Здесь следует обратить внимание на способы и приемы, которые необходимо в точности выполнять во избежание ошибок при эксплуатации или настройке.</p>
---	---

### ИНФОРМАЦИОННЫЕ ЗНАКИ

	<p><b>ИНФОРМАЦИЯ</b></p> <p>Здесь следует обратить внимание на <u>важную</u> информацию</p>
---	---

## СОДЕРЖАНИЕ

<b>ТЕРМИНЫ И СОКРАЩЕНИЯ</b> .....	<b>5</b>
<b>ВВЕДЕНИЕ</b> .....	<b>6</b>
Общие сведения.....	6
Перечень рекомендуемых документов .....	6
<b>СЕРВЕР REGUL IEC 61850</b> .....	<b>7</b>
<b>НАЧАЛО РАБОТЫ</b> .....	<b>8</b>
Перечень действий при настройке .....	8
Генерация объектов IEC 61850 .....	8
<b>НАСТРОЙКА ПАРАМЕТРОВ</b> .....	<b>12</b>
Настройки подключения .....	12
Настройка уровня доступа .....	13
Пользовательские функции.....	14
Дальнейшая логика работы приложения.....	14
<b>ПОДДЕРЖИВАЕМЫЕ СЕРВИСЫ</b> .....	<b>17</b>
Настройка обработки запросов телеуправления (Control).....	17
Описание выполнения команд управления.....	18
Настройка проверки запроса на валидность .....	20
Настройка управления группами уставок (Setting Group) .....	22
Работа с наборами данных (Data Set).....	25
Настройки отчётов (Reporting).....	26
Настройки логирования (Logging).....	27
Функционал поддержки источника времени (SyncTime) .....	28
Нормализация значений (Deadband) .....	29
Подмена значений (Substitution).....	30
<b>ПРИЛОЖЕНИЕ А Перечень библиотечных функций для работы с маской качества</b> .....	<b>31</b>
<b>ПРИЛОЖЕНИЕ Б Кодировка битов маски качества q</b> .....	<b>33</b>

**ТЕРМИНЫ И СОКРАЩЕНИЯ**

<b>Сокращение/Обозначение</b>	<b>Описание</b>
<b>ГФИ</b>	Сгенерированные функции инициализации
<b>ГФБЛУ</b>	Сгенерированные ФБЛУ, входящие в состав приложения пользователя, наследники библиотечных ФБЛУ
<b>ФБ</b>	Функциональный блок
<b>ФБЛУ</b>	Функциональный блок, содержащийся в библиотеке PS_61850_Lib.library и включающий в себя атрибуты и функциональность логических узлов стандарта IEC 61850
<b>CDC</b>	Common data classes – классы общих данных
<b>DA</b>	Data Attribute – атрибут данных стандарта IEC 61850
<b>DO</b>	Data Object – объект данных стандарта IEC 61850
<b>IED / ИЕУ</b>	Intelligent Electronic Device / интеллектуальное электронное устройство
<b>LD</b>	Logical Device – логическое устройство
<b>LN</b>	Logical Node – логический узел
<b>Plugin generator</b>	Генератор объектов IEC 61850
<b>POU</b>	Program Organization Unit – компонент организации программы. В большинстве случаев под этим термином понимается пользовательская программа или функциональный блок

## ВВЕДЕНИЕ

### Общие сведения

IEC 61850 (МЭК 61850) — стандарт «Коммуникационные сети и системы подстанций», представляющий собой свод правил для организации событийного протокола передачи данных по сети Ethernet. IEC 61850 описывает преобразование абстрактных объектов и сервисов в протоколы сети.

Основная концепция архитектуры, принятая в стандарте IEC 61850, состоит в абстрагированном описании элементов данных и обслуживания, т.е. создание элементов, объектов данных и сервисных функций не зависит от протокола нижнего уровня. Абстрактные определения позволяют распределить объекты данных и сервисные функции по любому другому протоколу, если он соответствует требованиям данных и обслуживания.

На контроллерах Regul RX00 используется редакция 2.1 стандарта IEC 61850. Для обмена данными по протоколу IEC 61850 на контроллерах Regul используется компонент **Сервер Regul IEC 61850**. Он позволяет отразить входные данные и управляющие воздействия контроллера на протокол IEC 61850 (подробное описание см. в разделе «Сервер Regul IEC 61850»).

### Перечень рекомендуемых документов

Для получения информации по настройке других параметров контроллеров серии Regul RX00 в среде разработки Astra.IDE рекомендуется ознакомиться со следующими документами (доступны на сайте <https://reglab.ru/>):

- Программное обеспечение Astra.IDE. Руководство пользователя;
- Regul R500. Системное руководство;
- Regul R400. Системное руководство.

## СЕРВЕР REGUL IEC 61850

Для поддержки стандарта IEC 61850 на контроллерах Regul используется **сервер Regul IEC 61850**, реализованный в виде исполняемого модуля на уровне ОС. Он позволяет взаимодействовать с ПЛК, используя протокол MMS.

В состав сервера Regul IEC 61850 также входят:

- Библиотека **PS\_61850\_Lib.library**, которая подключается к пользовательскому приложению. В ней описаны все поддерживаемые типы объектов данных, в каждом из которых присутствуют в виде полей все возможные для данного типа CDC атрибуты данных.
- **Генератор объектов IEC 61850 (plugin generator)**, который является дополнением к программному обеспечению Astra.IDE. Он генерирует заготовки объектов, в которые встраивается логика приложений пользователя.

**Plugin generator** формирует в приложении структуры данных, функциональные блоки, объявления методов и переменных, которые являются интерфейсом для взаимодействия пользовательского приложения и сервера IEC 61850.

Для работы генератору объектов IEC 61850 требуется конфигурация объекта управления в формате SCL (Substation Configuration Language).

SCL файлы могут иметь расширения:

- **\*.cid** (Configured IED Description) – файл описания базовой конфигурации устройства;
- **\*.icd** (IED Capability Description) – файл описания предварительно сконфигурированного устройства;
- **\*.iid** (Instantiated IED Description) – файл описания конфигурации устройства.

Файлы базируются на расширяемом языке разметки XML. Подробно элементы синтаксиса языка SCL описаны в документе «ГОСТ МЭК-61850-6-2009».

Общее дерево объектов, которое генерируется в проекте, соответствует тому, что описано в исходном конфигурационном файле.

## НАЧАЛО РАБОТЫ

Установите на компьютер программное обеспечение Astra.IDE. Описание процесса установки программы, а также инструкции по работе с программой приведены в документе «Программное обеспечение Astra.IDE. Руководство пользователя. DPA-302». Программа установки и документация доступны на сайте [www.prosoftsystems.ru](http://www.prosoftsystems.ru).

Запустите программу **Astra.IDE**. Откройте проект, в котором требуется настроить контроллер для обмена данными по протоколу IEC 61850. Если такого проекта нет, создайте его с помощью **Мастера конфигурации Regul** (описание приведено в разделе «Основные понятия среды разработки. Проект» документа «Программное обеспечение Astra.IDE. Руководство пользователя. DPA-302»).

### Перечень действий при настройке

В новом проекте необходимо создать приложение и сформировать в нём структуры данных, функциональные блоки, объявления методов и переменных, которые будут использоваться для взаимодействия с сервером IEC 61850. Для этого нужно загрузить через интерфейс **plugin generator** исходный файл, содержащий описание проекта или устройства в формате SCL.

После этого **plugin generator**, встроенный в пакет Astra.IDE, проверяет исходный файл на валидность и генерирует из содержимого файла шаблон проекта: набор типов, функциональных блоков и функций на языке ST.

Далее вам необходимо самостоятельно реализовать в программе логику заполнения атрибутов в дереве данных, которое драйвер будет отображать для клиента, логику обработки запросов на телеуправление и т.д.

### Генерация объектов IEC 61850

В верхнем меню откройте вкладку **Инструменты** и нажмите пункт **Сгенерировать объекты IEC 61850** (Рисунок 1), запускающий действие генератора объектов IEC 61850 (**plugin generator**).



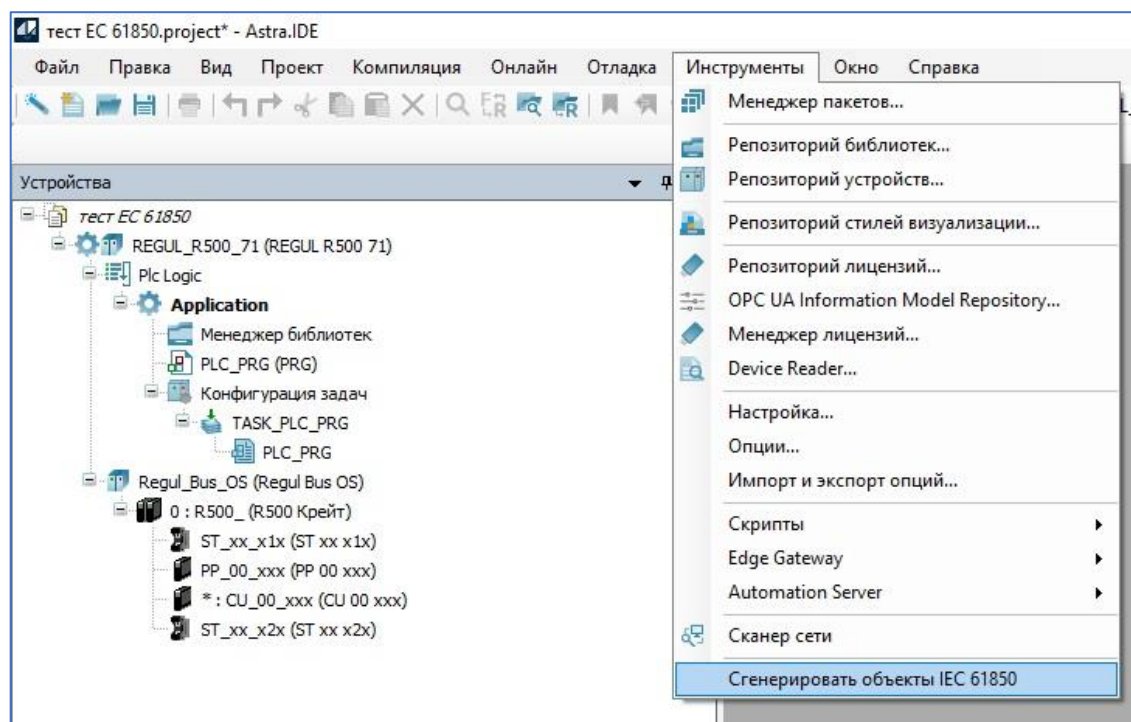


Рисунок 1 – Запуск генератора объектов IEC 61850

Откроется всплывающее окно с полем ввода (Рисунок 2). Вы можете указать путь к исходному ICD-файлу вручную, либо нажать кнопку [...] справа от поля ввода и выбрать нужную папку и файл в проводнике. Затем нажмите кнопку **Ok**.

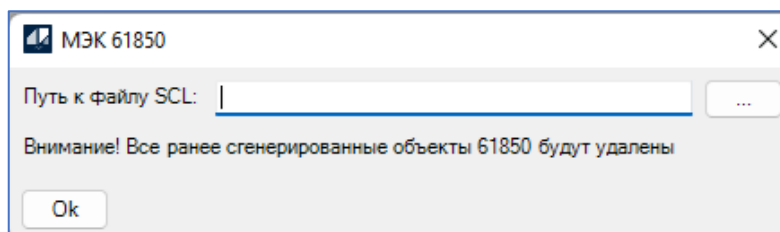


Рисунок 2 – Выбор пути к исходному файлу в формате SCL

После нажатия кнопки синтаксис файла будет автоматически проверен на соответствие формату SCL. В случае, если синтаксис не валиден, вы получите предупреждение об этом с указанием ошибки (Рисунок 3).

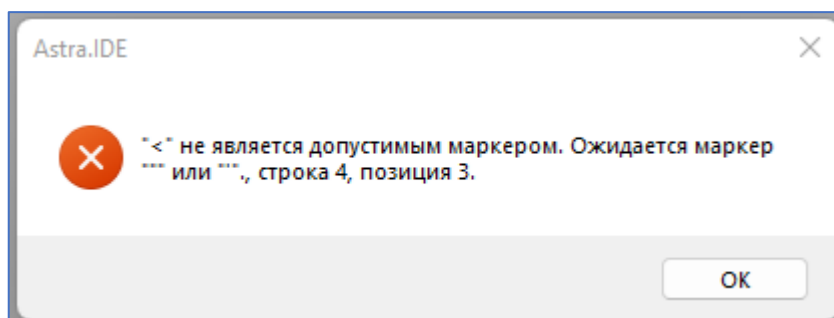


Рисунок 3 – Пример сообщения о наличии ошибок в синтаксисе

Если проверка пройдена успешно, **plugin generator** сгенерирует на базе исходного файла шаблон проекта с необходимыми наборами типов, функциональных блоков и функций. Одновременно генерируются пустые заготовки функций обработчиков (например, для обработки запросов телеуправления).

Сгенерированные типы и функции-обработчики сгруппированы в основном дереве проекта, которое отражается во вкладке **POU** (Рисунок 4).

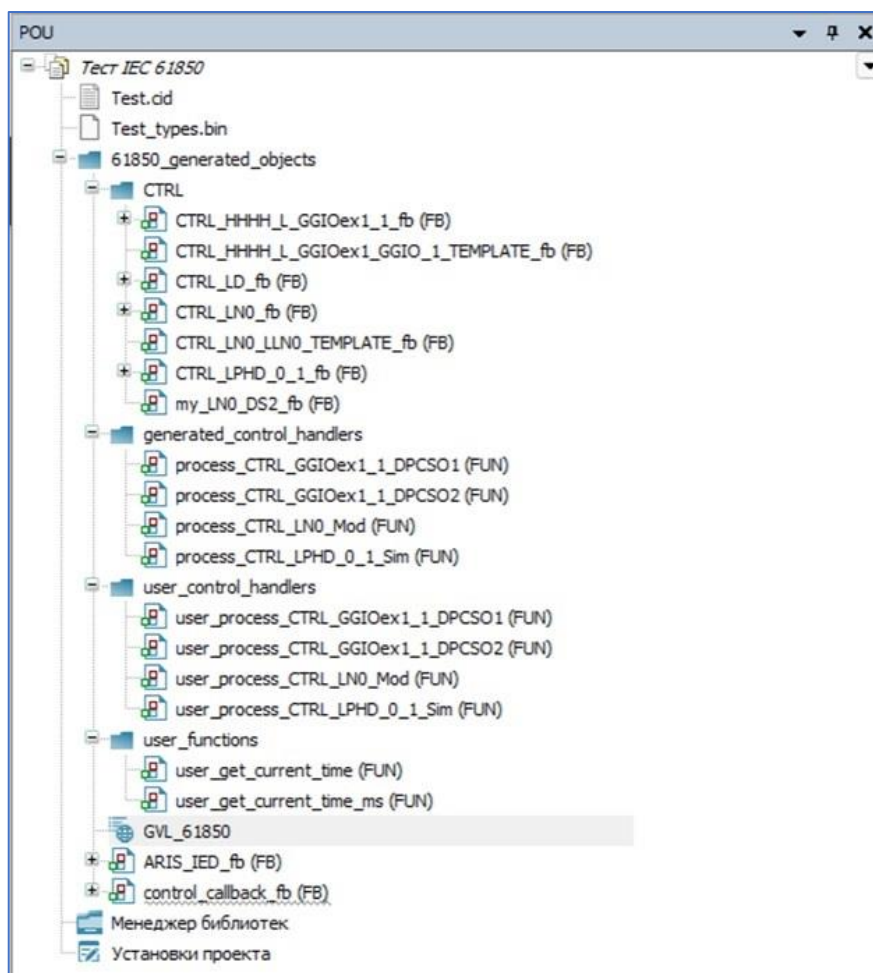


Рисунок 4 – Дерево проекта во вкладке POU

Все сгенерированные типы и функции помещаются в общий каталог **61850\_generated\_objects** (Рисунок 5, 1). В этом каталоге есть подкаталоги, имена которых соответствуют именам LD (Рисунок 5, 2-3). В них помещаются сгенерированные типы данных и объекты типов, которые содержит этот LD.

Каждый сгенерированный тип является наследником от библиотечного типа из **Ps61850Lib.library**, например, тип **CTRL\_DNVGL\_GGIOex3\_GGIO\_1\_TEMPLATE\_fb** является наследником от библиотечного типа **Ps61850Lib.GGIO\_TEMPLATE\_fb**.



## НАСТРОЙКА ПАРАМЕТРОВ

### Настройки подключения

Все параметры подключения (номер используемого tcp-порта, ip-адаптеры, селекторы для идентификации клиентов, настройки блока управления отчетами и блока управления логами) находятся в методе **FB\_Init** корневого функционального блока **IED\_fb** (Рисунок 7).

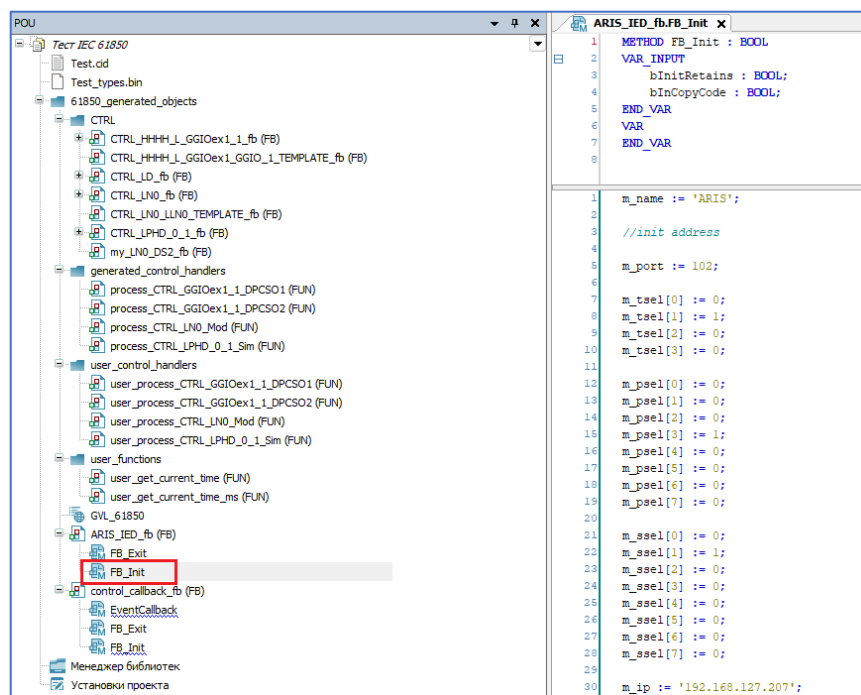


Рисунок 7 – Настройки подключения в методе FB\_Init корневого ФБ IED

Все эти параметры **plugin generator** берёт из исходного ICD-файла и конвертирует в код на языке ST. Так, в настройках параметров ассоциаций значение **m\_port** берется из атрибута `<P type="MMS-Port">102</P>`, **m\_ip** – из атрибута `<P type="IP">192.168.127.207</P>` и т.д. (Рисунок 8). Как правило, эти значения не нуждаются в корректировке, но при необходимости вы всегда можете доработать исходный код.



Рисунок 8 – Настройки параметров ассоциации в исходном ICD-файле и в виде сгенерированного кода

## Настройка уровня доступа

Анонимное соединение разрешено в том случае, если корневой функциональный блок **IED\_fb** не содержит массив **m\_users**. Если определение такого массива задано, анонимное соединение становится невозможным, и необходимо установить для клиентов один из трёх видов доступа: **access\_read\_only**, **access\_read\_write** либо **access\_write\_only**.

Определение массива **m\_users** задаётся в корневом функциональном блоке **IED\_fb** (Рисунок 9).

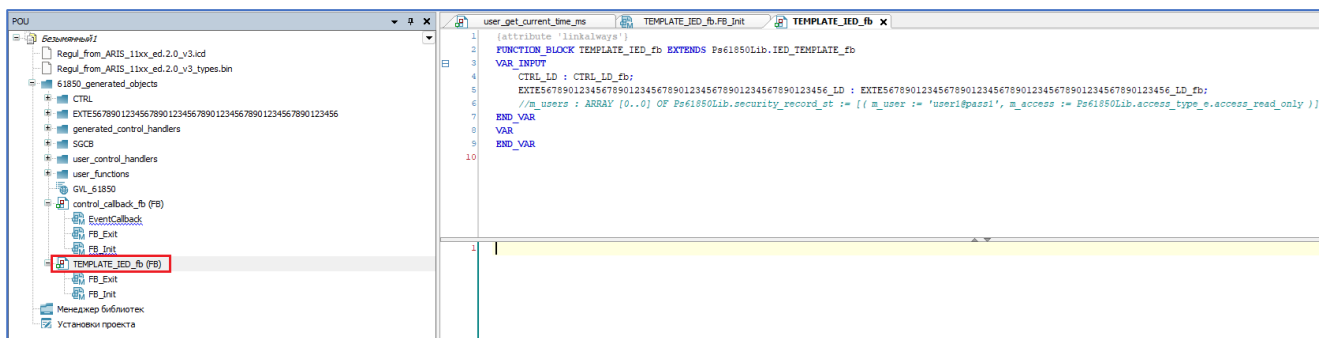


Рисунок 9 – Задание уровня доступа для пользователей

Там же в виде комментария содержится образец кода, позволяющего установить для клиента уровень доступа «только чтение» (**access\_read\_only**):

```
m_users : ARRAY [0..0] OF Ps61850Lib.security_record_st := [( m_user := 'user1@pass1', m_access := Ps61850Lib.access_type_e.access_read_only )];
```

Аналогичным образом устанавливаются виды доступа `access_read_write` и `access_write_only`.

## Пользовательские функции

Каталог `user_functions` содержит реализацию двух часто используемых вспомогательных функций:

- `user_get_current_time` – получить метку системного времени в секундах;
- `user_get_current_time_ms` – получить сетку системного времени в миллисекундах (Рисунок 10).

```

1  {attribute 'linkalways'}
2  FUNCTION user_get_current_time_ms : BOOL
3  VAR_INPUT
4      p : REFERENCE TO time_param_ms_st;
5  END_VAR
6  VAR
7      res : RTS_IEC_RESULT;
8  END_VAR
9
10 res := SysTimeRtcHighResGet(p.time_to_set);
11 IF ( res <> CmpErrors.Errors.ERR_OK )
12 THEN
13     user_get_current_time_ms := FALSE;
14     RETURN;
15 END_IF
16 p.clock_not_sync := FALSE;
17 p.leap_seconds_know := FALSE;
18 p.clock_failure := FALSE;
19 user_get_current_time_ms := TRUE;
    
```

Рисунок 10 – Пример пользовательской функции

В примере (Рисунок 10) мы видим функцию получения текущего системного времени для записи в атрибуты данных `t`.

Как правило, пользовательские функции не нуждаются в редактировании.

## Дальнейшая логика работы приложения

Всю остальную логику работы приложения необходимо самостоятельно прописать в ПЛК-программе (Рисунок 11).

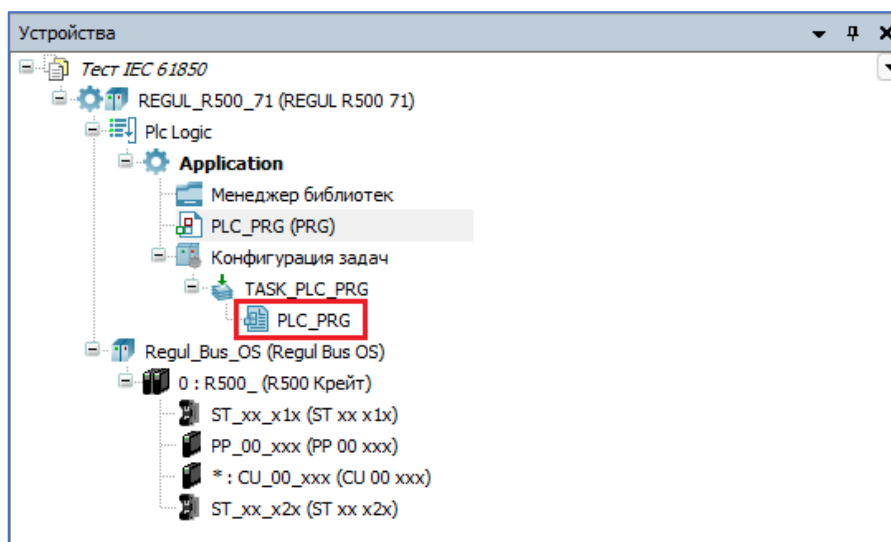


Рисунок 11 – Программа PLC\_PRG в дереве устройств

При этом необходимо соблюдать следующие требования стандарта IEC 61850:

- при записи значения в любой атрибут данных необходимо установить для этого значения временную метку (t), которая показывает, когда именно это значение было измерено, а также состояние источника времени на ПЛК. Для записи метки времени следует использовать ранее описанные функции из параграфа «Пользовательские функции»;
- помимо значения и времени, необходимо также указать качество значения (q), которое показывает клиенту источник данных для значения и их валидность (см. Приложение А).

Например, для значения:

```
GVL_61850.ARIS.ComplexArray_LD.m_GGIO1.m_AnIn2.m_mag.m_f := f_val;
```

нужно будет установить временную метку:

```
user_get_current_time(p);
Ps61850Lib.set_timestamp(p, GVL_61850.ARIS.ComplexArray_LD.m_GGIO1.m_AnIn2.m_t);
```

а также флаги в маске качества:

```
Ps61850Lib.set_good_quality(GVL_61850.ARIS.ComplexArray_LD.m_GGIO1.m_AnIn2.m_q);
```

и вызывать функцию обновления дерева данных (Рисунок 12):

```
GVL_61850.ARIS.update_server(ADR(GVL_61850.ARIS.ComplexArray_LD.m_GGIO1.m_AnIn2));
```

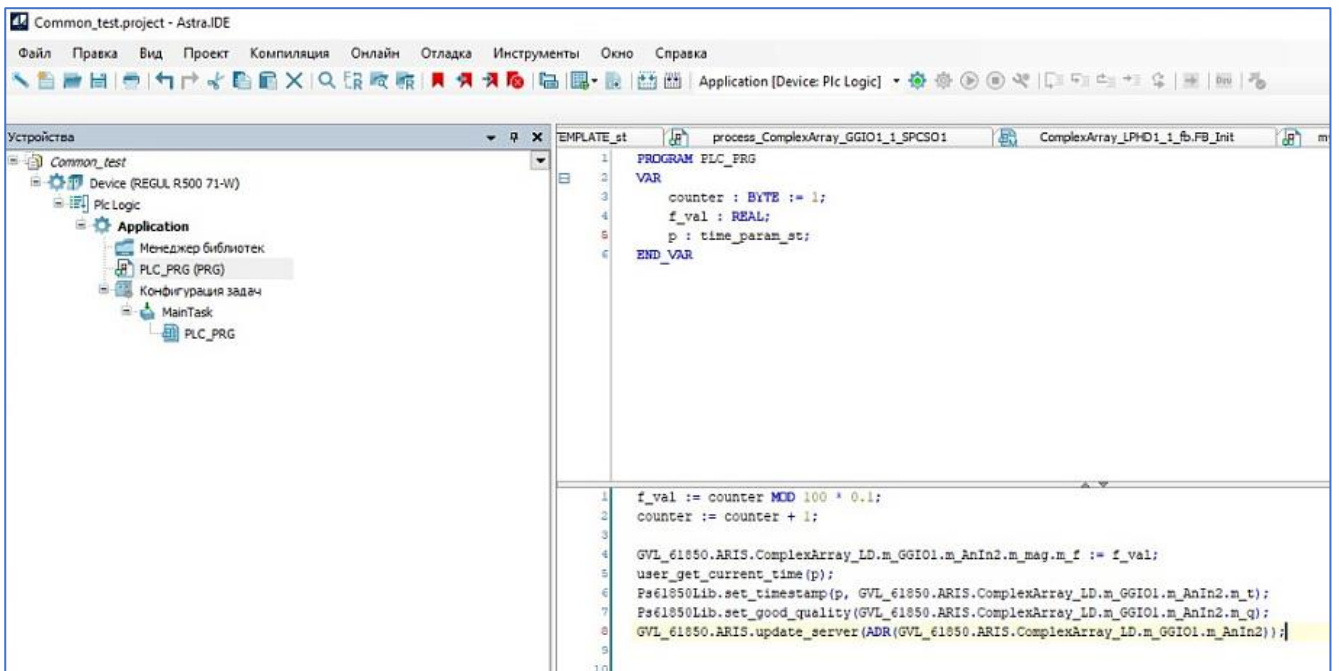


Рисунок 12 – Указание временной метки (t) и качества значения (q) при записи значения в data-атрибут



## ПОДДЕРЖИВАЕМЫЕ СЕРВИСЫ

### Настройка обработки запросов телеуправления (Control)

За обработку запросов телеуправления для тех или иных объектов данных отвечают функции, которые содержатся в подкаталогах (папках) **generated control handlers** и **user control handlers**. (Рисунок 13). **Generated control handlers** вызываются в теле общего обработчика **control\_callback\_fb.EventCallback()**. **User control handlers** вызываются в теле **Generated control handlers**.

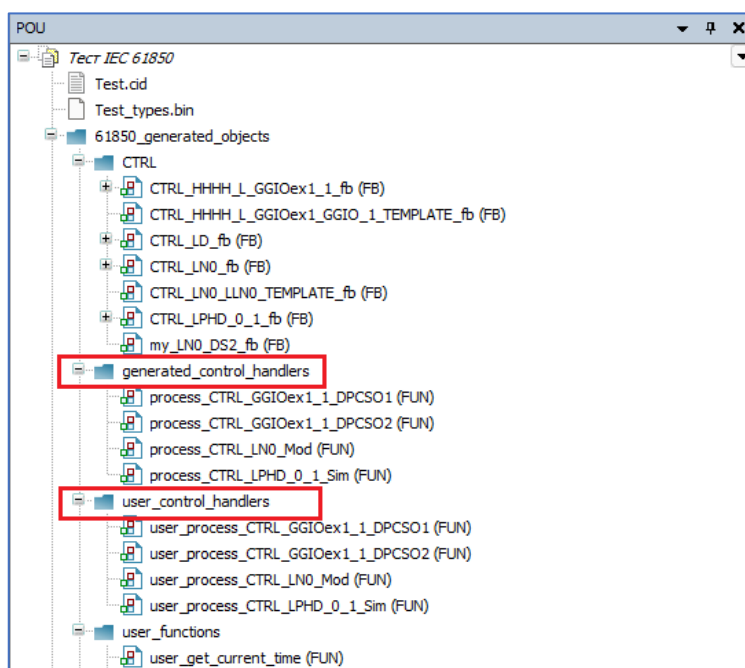


Рисунок 13 – Функции обработки запросов телеуправления

В папке **generated control handlers** помещаются сгенерированные функции обработки запросов телеуправления с уже прописанным кодом. Эти функции достаточно примитивны по своему содержанию, и их генерация избавляет пользователя от написания однотипных шаблонных функций.

Сгенерированные функции-обработчики уже содержат необходимый минимум функционала для обработки запросов телеуправления. Также в этих функциях в закомментированном виде присутствуют примеры кода для расширения логики обработки запросов на управление.

Пользовательские функции для обработки запросов телеуправления помещаются в папке **user control handlers** и содержат только заглушки вида:

```
{attribute 'linkalways'}
FUNCTION user_process_CTRL_CSWIb_DOes_1_CmdBlk : BOOL
VAR_INPUT
    v : BOOL;
END_VAR
VAR
```

```
END_VAR
=====
user_process_CTRL_CSWIb_DOes_1_CmdBlk := FALSE
```

В них пользователь должен самостоятельно реализовывать логику обработки каждого отдельного элемента управления (Рисунок 14).

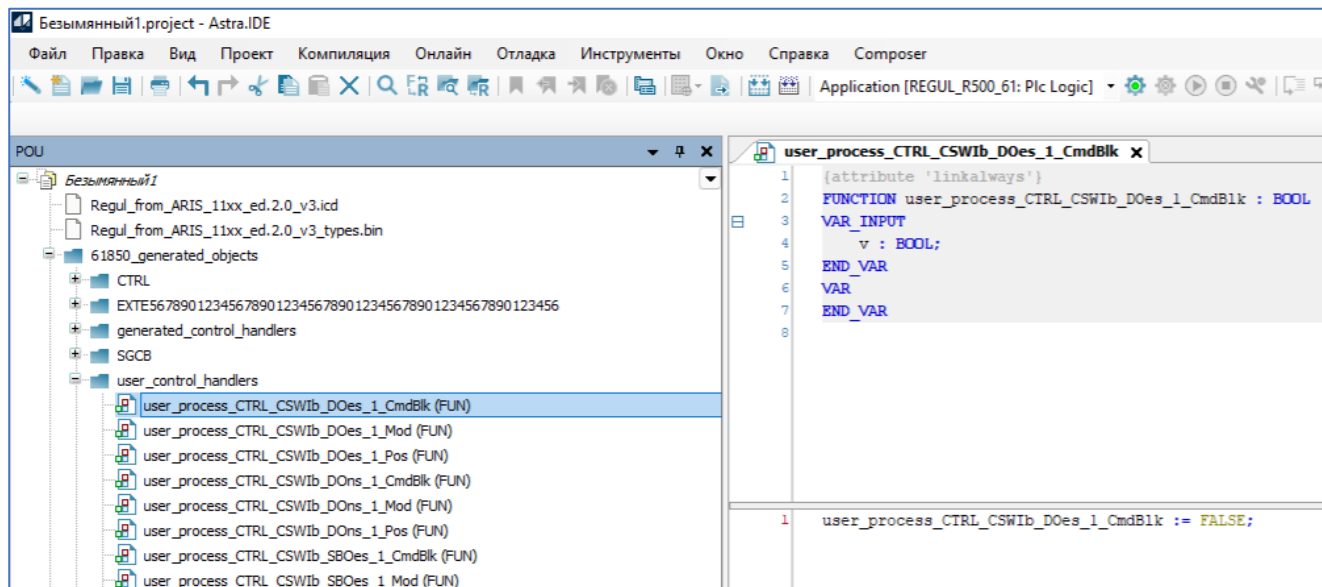


Рисунок 14 – Вид пользовательской функции обработки запросов телеуправления

## Описание выполнения команд управления

Сервер Regul IEC 61850 поддерживает следующие типы запросов на телеуправление:

- Direct control (Dons);
- Enhanced SBO control (SBOes).

Для выполнения команд управления сформируйте управляющий запрос в отношении управляемого объекта данных и отправьте его серверу.

Реакция сервера **Regul IEC 61850** на поступивший управляющий запрос определяется в пользовательской функции вида **user\_process\_CTRL\_GGIOex1\_1\_DPCSO4**. Имя функции уже содержит идентификатор управляемого объекта (в данном случае это будет **CTRL\_GGIOex1\_1\_DPCSO4**), единственный параметр функции такого вида содержит значение, которое клиент задал в управляющем запросе.

Тип значения соответствует типу управляемого объекта. Например, управляемый объект **CTRL\_GGIOex1\_1\_DPCSO4** имеет тип **SPC** (Controllable Single Point) и управляющее значение имеет тип **BOOL**. В зависимости от логики работы контроллера Regul, пользователь имеет возможность влиять на выполнение команды управления.

Например, если в данный момент не выполнены условия выполнения команды управления, то пользователь в функции **user\_process\_CTRL\_GGIOex1\_1\_DPCSO4** не выполняет действий,

связанных с выполнением команды управления, и возвращает значение **FALSE** из пользовательской функции, используя оператор:

```
user_process_CTRL_GGIOex1_1_DPCSO4 := FALSE;
```

В ответ на управляющий запрос клиент получит положительный ответ (success) только при выполнении следующих условий:

- объект данных задан в адресном пространстве сервера;
- пользователю разрешен доступ по записи к данному объекту данных;
- соблюдены условия выполнения команды управления (либо они не прописаны пользователем);
- пользовательская функция (**user\_process\_CTRL\_GGIOex1\_1\_DPCSO4** в примере выше) вернет **TRUE**.

Если хотя бы одно из этих условий не выполнено, ответ на MMS-операцию записи будет отрицательным (failed).

Для каждого управляемого объекта данных логика обработки запросов телеуправления прописывается отдельно, причём не только в пользовательской функции, но и в соответствующей сгенерированной функции из каталога **generated\_control\_handlers**.

Структурно код обработки запроса телеуправления разбит на две функции:

- **process\_CTRL\_GGIOex1\_1\_DPCSO2;**
- **user\_process\_CTRL\_GGIOex1\_1\_DPCSO2.**

В функциях вида **process\_\*\*\*** предполагается реализация пользователем проверок на возможность исполнения команды, а в функциях **user\_process\_\*\*\*** предполагается реализация непосредственно выполнения.

Например, пользовательская функция **user\_process\_CTRL\_GGIOex1\_1\_DPCSO2** вызывается в теле сгенерированной функции **process\_CTRL\_GGIOex1\_1\_DPCSO2**, и именно в функции **process\_CTRL\_GGIOex1\_1\_DPCSO2** производится проверка на условия выполнения команды управления (Рисунок 15).

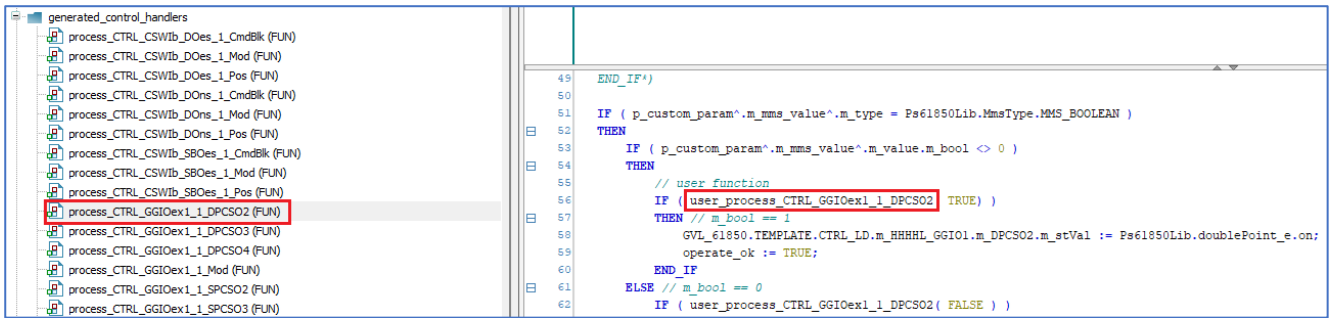


Рисунок 15 – пользовательская функция user\_process\_CTRL\_GGIOex1\_1\_DPCS02() в теле сгенерированной функции process\_CTRL\_GGIOex1\_1\_DPCS02()

## Настройка проверки запроса на валидность

Пример кода с реализацией проверки на валидность запроса на телеуправление содержится в закомментированном виде в сгенерированных функциях-обработчиках (Рисунок 16).

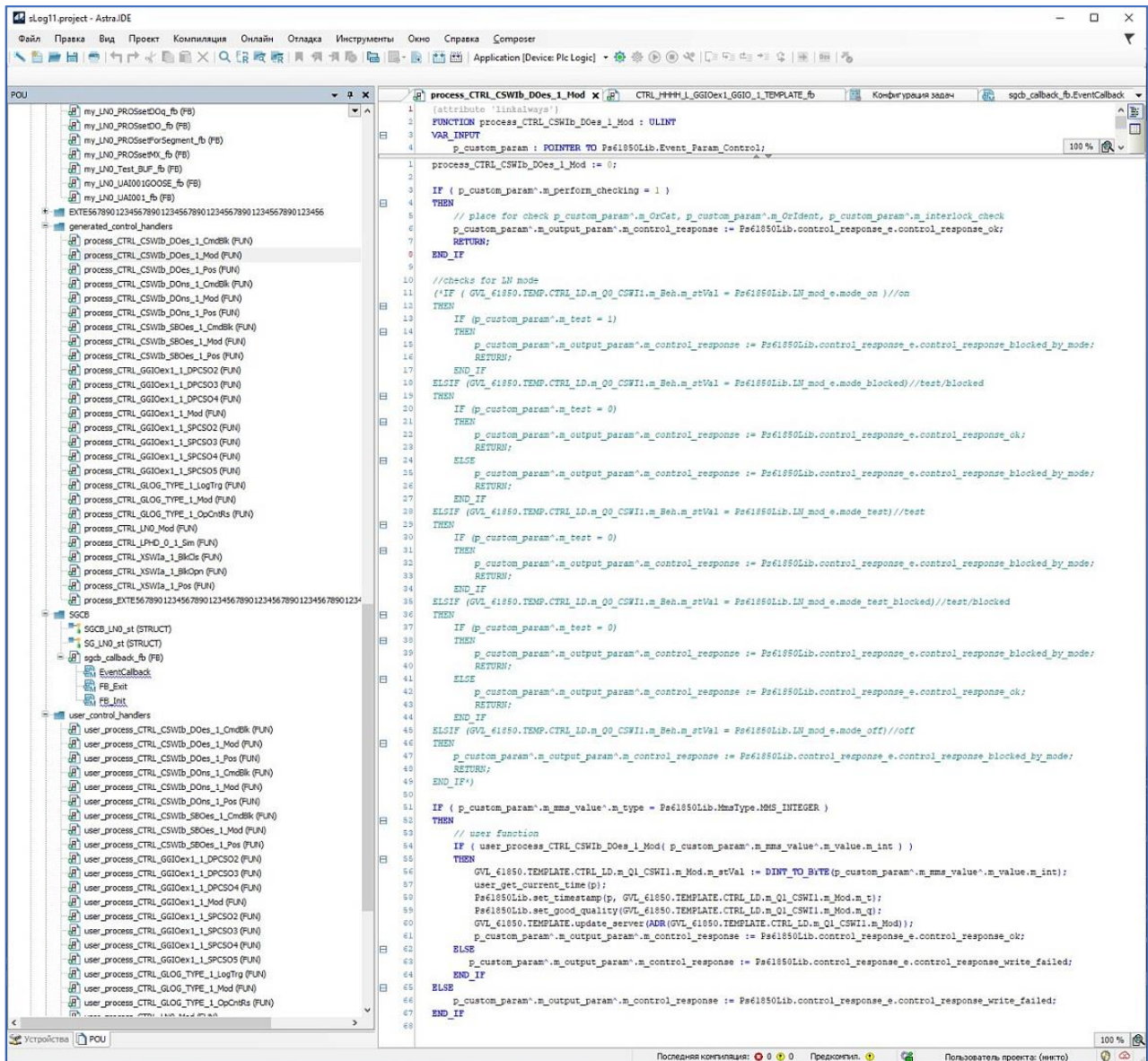


Рисунок 16 – Пример кода сгенерированной функции обработки запроса телеуправления

Так, проверку на валидность параметров запроса на телеуправление выполняет следующая секция (строки 3-8, Рисунок 16):

```
IF ( p_custom_param^.m_perform_checking = 1 )
THEN
    // place for check p_custom_param^.m_OrCat, p_custom_param^.m_OrIdent,
    p_custom_param^.m_interlock_check
    p_custom_param^.m_output_param^.m_control_response :=
Ps61850Lib.control_response_e.control_response_ok;
    RETURN;
END_IF
```

В таком виде (по умолчанию) секция пропускает абсолютно все запросы. Если вам необходимо выполнять проверки на валидность, то вместо:

```
p_custom_param^.m_output_param^.m_control_response :=
Ps61850Lib.control_response_e.control_response_ok;
```

допишите код для проверки соответствующих значений из параметров запроса:

```
check p_custom_param^.m_OrCat, p_custom_param^.m_OrIdent,
p_custom_param^.m_interlock_check
```



#### **ВНИМАНИЕ!**

Секция кода

```
IF ( p_custom_param^.m_perform_checking = 1 )
THEN
.....
    RETURN;
END_IF
```

нужна для корректной работы сервера, удалять ее или удалять команду RETURN из нее нельзя!

Если в ходе проверки на валидность вам нужно отменить выполнение команды, то вместо

```
p_custom_param^.m_output_param^.m_control_response :=
Ps61850Lib.control_response_e.control_response_ok
```

запишите в запросе возвращаемое значение с соответствующей ошибкой, например:

```
p_custom_param^.m_output_param^.m_control_response :=
Ps61850Lib.control_response_e.control_response_blocked_by_synchrocheck
```

В следующей секции (строки 10-49, Рисунок 16) приведен пример проверки запроса на соответствие состоянию логического устройства. Поскольку это именно пример, а не рабочий код, в случае использования его надо раскомментировать и заменить **CTRL\_LD** и **m\_Q0\_CSWI1** на соответствующие логический узел и логическое устройство из конфигурации пользователя.

Далее (строки 51-68, Рисунок 16) приведён вызов пользовательской функции (в примере это **user\_process\_CTRL\_CSWIb\_DOes\_1\_Mod**), в которой нужно самостоятельно прописать непосредственную логику выполнения запроса на управление.

## Настройка управления группами уставок (Setting Group)

Группа уставок представляет собой набор значений для нескольких атрибутов данных, который по команде пользователя может быть одновременно записан в эти атрибуты. Выбор атрибутов данных и их значений будет зависеть от конкретной задачи каждого устройства. Поэтому **plugin generator** генерирует простой шаблон, в котором есть заготовка для обработки запросов, но не указаны конкретные атрибуты и значения. Предполагается, что каждый пользователь самостоятельно объявляет массив структур с наборами значений для каждой группы уставок.



### ВНИМАНИЕ!

Массив структур должен быть объявлен в энергонезависимой памяти ПЛК, иначе после перезагрузки все группы уставок, настроенные пользователем, будут утеряны

Работа с группами уставок прописывается в методе **EventCallback** функционального блока **sgcb\_callback\_fb**, реализация которого размещена в каталоге **SGCB** (Рисунок 17).

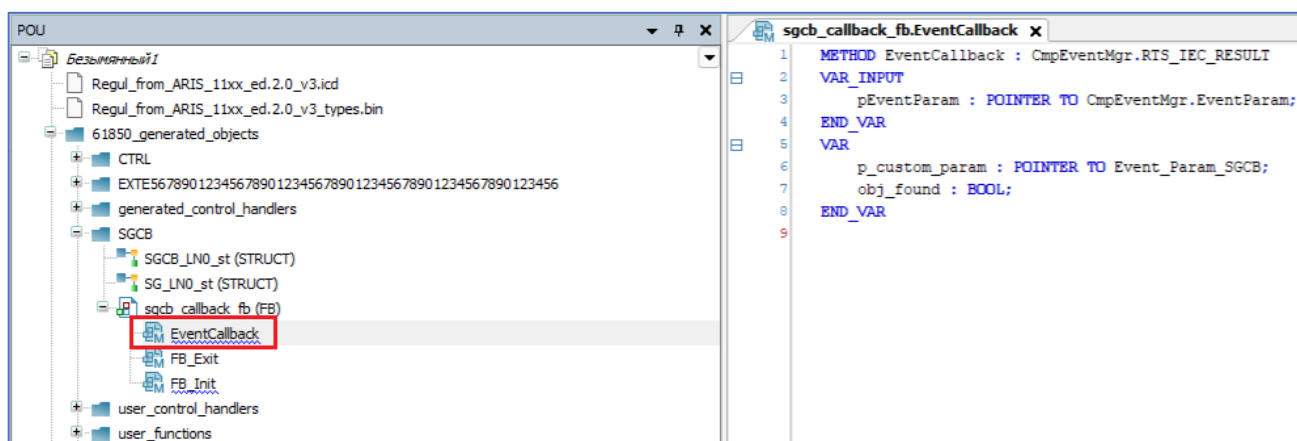


Рисунок 17 – Метод EventCallback для работы с группами уставок

Работа с группами уставок предполагает 3 вида запросов, примеры которых прописаны в редакторе в виде комментариев:

**1. Выбор активной группы уставок.** В закомментированном коде (Рисунок 18) приведён пример, как может быть реализована логика обработки запроса на выбор группы уставок **SelActSg**, когда необходимо из элемента массива с уставками скопировать значения в соответствующие атрибуты данных. После этого нужно обязательно обновить эти значения в дереве данных сервера (вызов функции **update\_server**).

```

4   p_custom_param := pEventParam^.pParameter;
5
6   IF ( p_custom_param^.m_var_address = ADR(GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB )
7   THEN
8       // проверка на допустимое значение номера группы уставок
9       IF ( p_custom_param^.ActSG <= GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_NumOfSG )
10      THEN
11          // обработка запроса на выбор группы уставок
12          IF ( p_custom_param^.rq_type = SGCB_request_e.rq_type_selActSG )
13          THEN
14              // при обработке запроса на выбор активной группы уставок, необходимо обновить значение текущей группы уставок
15              GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG := UDINT_TO_UINT(p_custom_param^.ActSG);
16              // пример копирования значений из массива уставок в атрибуты данных с fc=SG
17              //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_StrVal.m_setMag_sg.m_f :=
18              //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG - 1].strVal;
19
20              //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_TmMult.m_setMag_sg.m_f :=
21              //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG - 1].TmMult;
22
23              //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MinOpTms.m_setVal_sg :=
24              //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG - 1].MinOpTms;
25
26              //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MaxOpTms.m_setVal_sg :=
27              //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG - 1].MaxOpTms;
28
29              //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_DirMod.m_setVal_sg :=
30              //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG - 1].DirMod;
31          END_IF
32
33          // обработка запроса на выбор группы уставок для редактирования
34          IF ( p_custom_param^.rq_type = SGCB_request_e.rq_type_selEditSG )
35          THEN

```

Рисунок 18 – Пример кода для выбора активной группы уставок

**2. Выбор группы уставок для редактирования** (обработка запроса **SelectEditSg**). Пример выбора группы уставок для редактирования приведён в закомментированном коде в методе **EventCallback** (Рисунок 19). В нём показано, как взять значения уставок из элемента массива и записать их в соответствующие атрибуты данных с **fc=SE** в дереве данных.

```

28
29         //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_DirMod.m_setVal_sg :=
30         //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG - 1].DirMod;
31     END_IF
32
33     // обработка запроса на выбор группы уставок для редактирования
34     IF ( p_custom_param^.rq_type = SGCB_request_e.rq_type_selEditSG )
35     THEN
36         // пример копирования значений из массива уставок в атрибуты данных с fc=SE
37         //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_StrVal.m_setMag_se.m_f :=
38         //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].strVal;
39
40         //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_TmMult.m_setMag_se.m_f :=
41         //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].TmMult;
42
43         //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MinOpTms.m_setVal_se :=
44         //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].MinOpTms;
45
46         //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MaxOpTms.m_setVal_se :=
47         //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].MaxOpTms;
48
49         //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_DirMod.m_setVal_se :=
50         //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].DirMod;
51     END_IF
52
53     // обработка запроса на подтверждение изменения группы уставок
54     IF ( p_custom_param^.rq_type = SGCB_request_e.rq_type_confirmEditSG )
55     THEN

```

Рисунок 19 – Пример кода для выбора группы уставок для редактирования

**3. Подтверждение редактирования группы уставок** (обработка запроса **confirmEditSg**). Подтверждение редактирования группы уставок предполагает, что необходимо сохранить в энергонезависимой памяти новые значения атрибутов данных, которые клиент отредактировал в

дереве данных на этапе 2. На рисунке 20 приведен пример кода, в котором значения из атрибутов данных с **fc=SE** записываются в соответствующий элемент массива уставок.

Например, если редактировалась группа третья из пяти групп уставок, то значения нужно записать в третий элемент массива. Так, значение

`GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_StrVal.m_setMag_se.m_f`

должно быть записано в массив

`GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].strVal`

```

51      END_IF
52
53      // обработка запроса на подтверждение изменения группы уставок
54      IF ( p_custom_param^.rq_type = SGCB_request_e.rq_type_confirmEditSG )
55      THEN
56          // пример копирования значений из атрибутов данных с fc=SE в массив уставок
57          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].strVal :=
58          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_StrVal.m_setMag_se.m_f;
59
60          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].TmMult :=
61          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_TmMult.m_setMag_se.m_f;
62
63          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].MinOpTms :=
64          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MinOpTms.m_setVal_se;
65
66          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].MaxOpTms :=
67          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MaxOpTms.m_setVal_se;
68
69          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].DirMod :=
70          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_DirMod.m_setVal_se;
71          // В случае, если подтверждается редактирование активной группы уставок, то дополнительно необходимо обновить значения в атрибутах данных с fc=SG
72          //IF (GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_ActSG = p_custom_param^.ActSG)
73          //THEN
74          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_StrVal.m_setMag_sg.m_f :=
75          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].strVal;
76
77          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_TmMult.m_setMag_sg.m_f :=
78          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].TmMult;
79
80          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MinOpTms.m_setVal_sg :=
81          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].MinOpTms;
82
83          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_MaxOpTms.m_setVal_sg :=
84          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].MaxOpTms;
85
86          //GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1.m_DirMod.m_setVal_sg :=
87          //GVL_61850.TEMPLATE.CTRL_LD.m_LLNO.m_SGCB.m_settings[UDINT_TO_UINT(p_custom_param^.ActSG) - 1].DirMod;
88          //END_IF
89      END_IF
90
91      // обновление значение атрибутов в дереве данных

```

Рисунок 20 – Пример кода для подтверждения редактирования группы уставок

**Обновление значений атрибутов в дереве данных.** Запись значения в атрибут в дереве данных проекта не обеспечивает автоматическое обновление этого значения в дереве данных, которое отображается в программе-клиенте. Для синхронизации этих двух структур используется функция **update\_server** (Рисунок 21). Эта функция рекурсивно обновит значения всех атрибутов в соответствующем узле дерева данных клиента. Здесь вместо комментария необходимо прописать обновление тех узлов дерева данных, в которых были обновлены атрибуты данных после работы с группами уставок.



```

END_IF
// обновление значение атрибутов в дереве данных
//GVL_61850.TEMPLATE.update_server(ADR(GVL_61850.TEMPLATE.CTRL_LD.m_PTOC1));

p_custom_param^.m_output_param^.m_response := Ps61850Lib.SGCB_response_e.SGCB_response_ok;
ELSE
p_custom_param^.m_output_param^.m_response := Ps61850Lib.SGCB_response_e.SGCB_response_out_of_range;
END_IF
obj_found := TRUE;

```

Рисунок 21 – Вызов функции update\_server

## Работа с наборами данных (Data Set)

Согласно стандарту IEC 61850 наборы данных (Data Set) используются для хранения наборов значений из объектов дерева данных. Наборы данных входят в состав логического узла LLN0, который всегда содержится в единичном экземпляре в каждом логическом устройстве. Например, если в исходном файле логическое устройство CTRL описано следующим образом:

```

<LDevice inst="CTRL">
  <LN0 lnClass="LLN0" lnType="LN0" inst="">
    <LN lnType="LPHD_0" inst="1" lnClass="LPHD">
      <LN lnType="CILOa" prefix="Q0" lnClass="CILO" inst="1" desc=""/>
    </LN>
  </LN0>
</LDevice>

```

то в сгенерированном шаблоне приложения это будет выглядеть следующим образом:

```

{attribute 'linkalways'}
{attribute 'IEC61850.DEVICE_INST' := 'CTRL'}
FUNCTION_BLOCK CTRL_LD_fb EXTENDS Ps61850Lib.LD_TEMPLATE_fb
VAR_INPUT
  m_LLN0 : CTRL_LN0_fb;
  m_LPHD1 : CTRL_LPHD_0_1_fb;
  m_Q0_CILO1 : CTRL_Q0_CILOa_1_fb;
END_VAR
VAR
END_VAR

```

Допустим в исходном файле логический узел LLN0 содержит следующий набор данных (Рисунок 22).

```

<LDevice inst="CTRL">
  <LN0 lnClass="LLN0" lnType="LN0" inst="">
    <DataSet name="DS1">
      <FCDA ldInst="CTRL" prefix="Q0" lnClass="CILO" lnInst="1" fc="ST" doName="EnaOpn"/>
      <FCDA ldInst="CTRL" prefix="Q0" lnClass="CILO" lnInst="1" fc="ST" doName="EnaCls"/>
      <FCDA ldInst="CTRL" prefix="Q0" lnClass="MMXU" lnInst="1" fc="MX" doName="TotW"/>
      <FCDA ldInst="CTRL" prefix="HHHH_L" lnClass="GGIO" lnInst="1" fc="ST" doName="SPCSO2"/>
      <FCDA ldInst="CTRL" prefix="HHHH_L" lnClass="GGIO" lnInst="1" fc="ST" doName="DPCSO2"/>
      <FCDA ldInst="CTRL" prefix="F1" lnClass="GGIO" lnInst="1" fc="ST" doName="Ind1"/>
    </DataSet>
  </LN0>
</LDevice>

```

Рисунок 22 – пример исходного ICD-файла с описанием набора данных DS1 в составе логического узла LLN0

В этом случае **plugin generator** поместит в функциональный блок CTRL\_LN0\_fb следующую структуру с описанием этого набора данных:

```
{attribute 'linkalways'}
```

```
{attribute 'IEC61850.TEMPLATE' := 'LN0;;'}
FUNCTION_BLOCK CTRL_LN0_fb EXTENDS CTRL_LN0_LLNO_TEMPLATE_fb
VAR_INPUT
    ds_DS1 : my_LN0_DS1_fb := (name := 'DS1', FCDAs := [
        (ldInst := 'CTRL', lnClass := 'CILO', lnInst := '1', fc := 'ST', doName :=
        'EnaOpn', daName := '', prefix := 'Q0'),
        (ldInst := 'CTRL', lnClass := 'CILO', lnInst := '1', fc := 'ST', doName :=
        'EnaCls', daName := '', prefix := 'Q0'),
        (ldInst := 'CTRL', lnClass := 'MMXU', lnInst := '1', fc := 'MX', doName := 'TotW',
        daName := '', prefix := 'Q0'),
        (ldInst := 'CTRL', lnClass := 'GGIO', lnInst := '1', fc := 'ST', doName :=
        'SPCSO2', daName := '', prefix := 'HHHH_L'),
        (ldInst := 'CTRL', lnClass := 'GGIO', lnInst := '1', fc := 'ST', doName :=
        'DPCSO2', daName := '', prefix := 'HHHH_L'),
        (ldInst := 'CTRL', lnClass := 'GGIO', lnInst := '1', fc := 'ST', doName := 'Ind1',
        daName := '', prefix := 'F1')
    ]);
```



### ВНИМАНИЕ!

Менять содержимое кода, который описывает наборы данных, не рекомендуется. Изменения в состав Data Set целесообразно вносить путем правки исходного конфигурационного файла и регенерации проекта

## Настройки отчётов (Reporting)

На рисунке 23 показано, как конвертируются настройки отчётов из исходного конфигурационного файла.

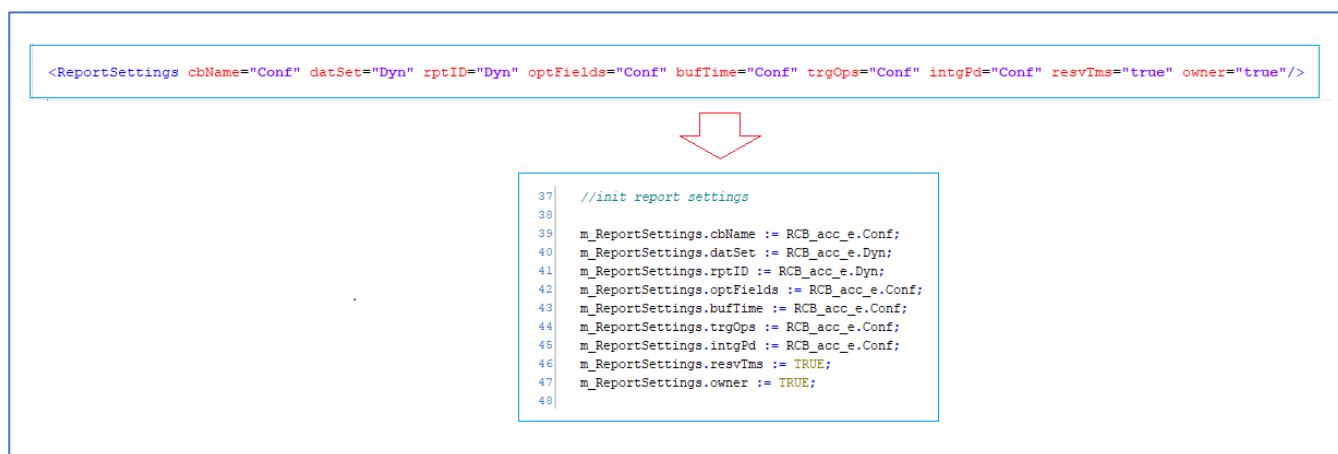


Рисунок 23 – Настройки отчётов в исходном ICD-файле и в виде сгенерированного кода

Как и в случае с наборами данных, структуры, описывающие отчеты (report), входят в состав FB\_IED->FB\_LD->m\_LLNO.

Например, если в исходном файле описан следующим образом отчет **brcb**:

```
<ReportControl confRev="1" name="brcb1" intgPd="60000" datSet="DS1"
buffered="true" bufTime="200" rptID="RegulCTRL/LLN0$BR$brcb1" indexed="true">
    <TrgOps gi="true" dchg="true" qchg="true" dupd="false"
period="true"/>
```

```
<OptFields seqNum="true" timeStamp="true" reasonCode="true"
dataSet="true" configRef="true" entryID="true" bufOvfl="true"/>
<RptEnabled max="10">
</RptEnabled>
</ReportControl>
```

то в сгенерированном коде этот отчет будет выглядеть так:

```
rcb_brcb101 : Ps61850Lib.BRCB_st:= (m_Name := 'brcb101', m_RptID :=
'RegulCTRL/LLN0$$BR$$brcb1', m_DataSet := 'DS1', m_OptFlds := 239, m_IntgPd :=
60000, m_BufTm := 200, m_ConfRev := 1, m_TrgOp := (TriggerConditions_e.data_change
OR TriggerConditions_e.quality_change OR TriggerConditions_e.general_interrogation
OR TriggerConditions_e.integrity));
...
rcb_brcb110 : Ps61850Lib.BRCB_st:= (m_Name := 'brcb110', m_RptID :=
'RegulCTRL/LLN0$$BR$$brcb1', m_DataSet := 'DS1', m_OptFlds := 239, m_IntgPd :=
60000, m_BufTm := 200, m_ConfRev := 1, m_TrgOp := (TriggerConditions_e.data_change
OR TriggerConditions_e.quality_change OR TriggerConditions_e.general_interrogation
OR TriggerConditions_e.integrity));
```

**Сервером Regul IEC 61850** поддерживаются как простые, так и индексированные отчеты, а также буферизированные и небуферизированные отчеты.



### ВНИМАНИЕ!

Не рекомендуется менять содержимое кода, который описывает отчеты. Вносить изменения в состав отчетов целесообразно путём правки исходного конфигурационного файла и регенерации проекта

## Настройки логирования (Logging)

Настройки логирования конвертируются в код из исходного файла по тому же принципу, что и остальные сервисы (Рисунок 24).

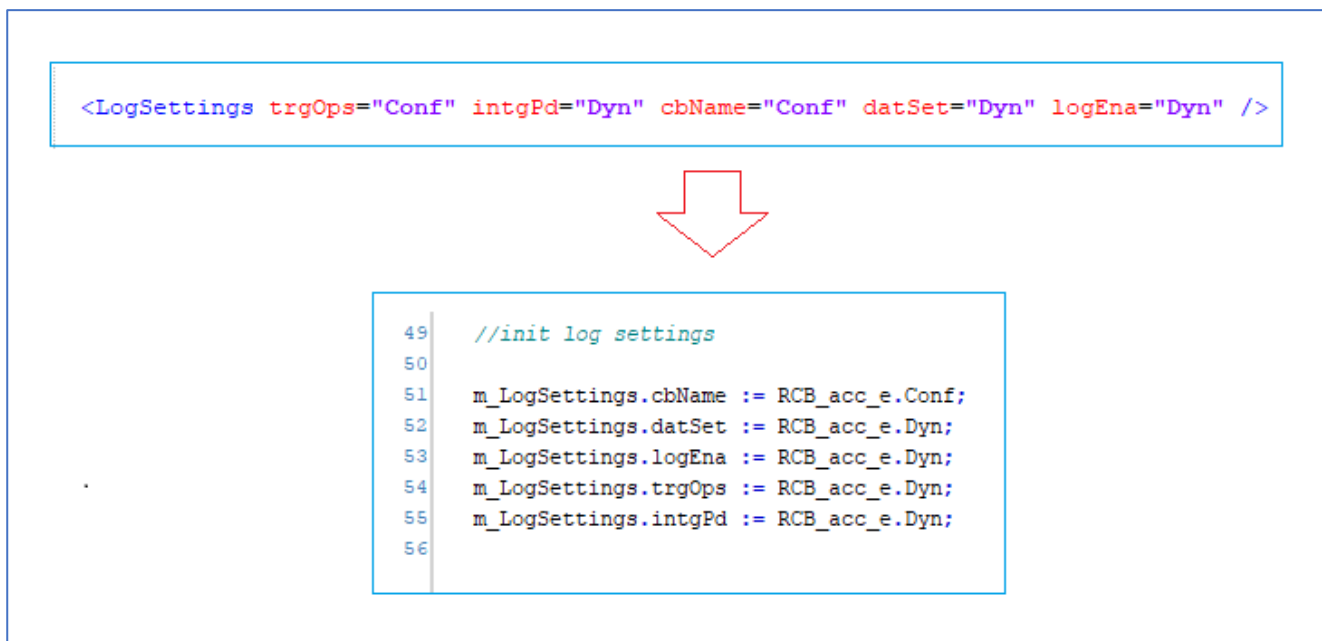


Рисунок 24 – Настройки параметров логирования в исходном ICD-файле и в виде сгенерированного кода

Например, если в исходном файле в логическом узле **GGIO1** содержится блок управления логированием такого вида:

```
<LogControl name="AnLog" datSet="TestGGIO" logName="AnLog" intgPd="5000"
logEna="true">
  <TrgOps dchg="true" qchg="true" />
</LogControl>
<Log name="AnLog" />
```

то в сгенерированном коде эти настройки конвертируются в код на языке ST следующим образом:

```
lcb_AnLog : Ps61850Lib.LogControl_st:= (m_name := 'AnLog', m_DataSet :=
'TestGGIO', m_logName := 'AnLog', m_logEna := True, m_reasonCode := True,
m_IntgPd := 5000, m_TrgOps := (TriggerConditions_e.data_change OR
TriggerConditions_e.quality_change OR TriggerConditions_e.general_interrogation));
log_AnLog : Ps61850Lib.Log_st := (m_name := 'AnLog');
```

Данные структуры будут включены в состав **ФБ\_LN**, описывающего узел **GIO1**. Файлы логов драйвер создает и хранит в каталоге */mnt/user/codesys/logs/*. Общий максимальный объем памяти, которую могут занимать файл логов – 50 Мб, максимальное количество записей в одном логе рассчитывается, исходя из количества сконфигурированных логов по формуле:

**(50Мб / 256) / кол-во логов**

Для каждого настроенного блока управления логами **LogControl** создается свой отдельный файл с именем в формате *имя IED\_имя LN\_имя LCB.db*.

Например, если в исходном файле описан IED с именем **CTRL**, содержащий логический узел **GGIO1**, а в узле описан блок управления логом **AnLog**, то полное имя файла будет соответственно */mnt/user/codesys/logs/CTRL\_GGIO1\_AnLog.db*.



#### **ВНИМАНИЕ!**

Не рекомендуется менять содержимое кода, который описывает настройки логирования, изменения в состав логов целесообразно вносить путем правки исходного конфигурационного файла и регенерации проекта

## **Функционал поддержки источника времени (SyncTime)**

Поддержка источника времени выполнена в виде отдельной библиотеки **PsTime**, которая входит в состав стандартного пакета обновлений для Astra.IDE (см. «Программное обеспечение Astra.IDE. Руководство пользователя. DPA-302»). Библиотека **PsTime** содержит функции, которые предоставляют данные о состоянии источника времени на ПЛК. На основании этих данных вы можете устанавливать биты качества в метках времени наборов данных.

Например, если необходимо установить метку времени для атрибута данных:

```
GVL_61850.ARIS.ComplexArray_LD.m_PDIF1.m_Op.m_t
```

код будет выглядеть следующим образом:

```
user_get_current_time(p);
Ps61850Lib.set_timestamp(p, GVL_61850.ARIS.ComplexArray_LD.m_PDIF1.m_Op.m_t);
user_get_current_time_ms(p_ms);
Ps61850Lib.set_timestamp_ms(p_ms,
GVL_61850.ARIS.ComplexArray_LD.m_PDIF1.m_Op.m_t);
```

Для работы с меткой времени можно использовать набор пользовательских функций **user\_get\_current\_time** и **user\_get\_current\_time\_ms**, описанных в подразделе «Пользовательские функции».

Функция **user\_get\_current\_time(p)** берет значение текущего системного времени в секундах и записывает это значение в структуру **p**, которая имеет следующий библиотечный тип:

```
TYPE time_param_st :
STRUCT
    time_to_set: DWORD;
    clock_failure : BOOL;
    clock_not_sync : BOOL;
    leap_seconds_know : BOOL;
END_STRUCT
END_TYPE
```

Функция **user\_get\_current\_time\_ms (p\_ms)** берёт значение текущего системного времени в миллисекундах и записывает это значение в структуру **p\_ms**. Структура **p** имеет библиотечный тип:

```
TYPE time_param_ms_st :
STRUCT
    time_to_set: ULINT;
    clock_failure : BOOL;
    clock_not_sync : BOOL;
    leap_seconds_know : BOOL;
END_STRUCT
```

На основании данных о состоянии источника времени, полученных при помощи **PsTime**, можно изменять значения битовых полей **clock\_failure**, **clock\_not\_sync**, **leap\_seconds\_known** и таким образом изменять биты качества метки времени в атрибуте **m\_t** после записи.

Для записи значения метки времени в атрибут данных **m\_t** дерева данных используются библиотечные функции **Ps61850Lib.set\_timestamp** для секундной точности и **Ps61850Lib.set\_timestamp\_ms** для миллисекундной.

## Нормализация значений (Deadband)

Функционал нормализации значений реализован на уровне драйвера. Нормализация выполняется для выходных значений DA объектов данных следующих типов: **MV**, **CMV**, **APC**, **BAC**. Настройка производится, согласно стандарту, при помощи установки в исходном файле или в сгенерированном коде значений атрибутов данных **db**, **zeroDb**, **dbRef**, **zeroDbRef** и т.д.

## Подмена значений (Substitution)

Функционал Substitution (подмена значений) реализован на уровне драйвера. Выполняется автоматически для выходных значений DA объектов данных следующих типов: **INC, CMV, DPC, INS, MV, SPC, SPS, BSC, ISC, APC, ENC, ENS, BAC, DPS**. Настройка и управление подстановкой производится, согласно стандарту, при помощи подключения через программу-клиента и запись соответствующих значений в атрибуты данных **subEna, subVal, subQ, subID, blkEna**.

## ПРИЛОЖЕНИЕ А

### Перечень библиотечных функций для работы с маской качества

Таблица А.1. Функции для работы с битовой маской качества атрибута данных *m\_q* (см. IEC61850-8-1)

Функция	Описание
<code>set_good_quality</code>	устанавливает в перечисление <code>validity</code> (биты 0, 1) значение <code>good</code>
<code>set_quality_badReference</code>	устанавливает бит 4 в значение <code>true</code>
<code>set_quality_failure</code>	устанавливает бит 6 в значение <code>true</code>
<code>set_quality_inaccurate</code>	устанавливает бит 9 в значение <code>true</code>
<code>set_quality_inconsistent</code>	устанавливает бит 8 в значение <code>true</code>
<code>set_quality_invalid</code>	устанавливает в перечисление <code>validity</code> (биты 0, 1) значение <code>invalid</code>
<code>set_quality_oldData</code>	устанавливает бит 7 в значение <code>true</code>
<code>set_quality_operatorBlocked</code>	устанавливает бит 12 в значение <code>true</code>
<code>set_quality_oscillatory</code>	устанавливает бит 5 в значение <code>true</code>
<code>set_quality_outOfRange</code>	устанавливает бит 3 в значение <code>true</code>
<code>set_quality_overflow</code>	устанавливает бит 2 в значение <code>true</code>
<code>set_quality_process</code>	устанавливает в перечисление <code>source</code> (бит 10) значение <code>process</code>
<code>set_quality_questionable</code>	устанавливает в перечисление <code>validity</code> (биты 0, 1) значение <code>questionable</code>
<code>set_quality_reserved</code>	устанавливает в перечисление <code>validity</code> (биты 0, 1) значение <code>reserved</code>
<code>set_quality_substituted</code>	устанавливает в перечисление <code>source</code> (бит 10) значение <code>substituted</code>
<code>set_quality_test</code>	устанавливает бит 11 в значение <code>true</code>
<code>unset_quality_badReference</code>	устанавливает бит 4 в значение <code>false</code>
<code>unset_quality_failure</code>	устанавливает бит 6 в значение <code>false</code>
<code>unset_quality_inaccurate</code>	устанавливает бит 9 в значение <code>false</code>

<b>Функция</b>	<b>Описание</b>
unset_quality_inconsistent	устанавливает бит 8 в значение false
unset_quality_oldData	устанавливает бит 7 в значение false
unset_quality_operatorBlocked	устанавливает бит 12 в значение false
unset_quality_oscillatory	устанавливает бит 5 в значение false
unset_quality_outOfRange	устанавливает бит 3 в значение false
unset_quality_overflow	устанавливает бит 2 в значение false
unset_quality_test	устанавливает бит 11 в значение false



**ПРИЛОЖЕНИЕ Б****Кодировка битов маски качества q**

Таблица Б.1. Кодировка битов маски качества q (см. IEC 61850-7-2)

Номер бита	IEC 61850-7-2		Битовая строка	
	Имя атрибута	Значение атрибута	Значение	По умолчанию
0-1	validity	good	0 0	0 0
		invalid	0 1	
		reserved	1 0	
		questionable	1 1	
2	overflow		TRUE	FALSE
3	outofRange		TRUE	FALSE
4	badReference		TRUE	FALSE
5	oscillatory		TRUE	FALSE
6	failure		TRUE	FALSE
7	oldData		TRUE	FALSE
8	inconsistent		TRUE	FALSE
9	inaccurate		TRUE	FALSE
10	source	process	0	0
		substituted	1	
11	test		TRUE	FALSE
12	operatorBlocked		TRUE	FALSE