**ДОКУМЕНТАЦИЯ**

**На программу**

**«ИТР Конфигуратор»**

**06.09.23**

Оглавление

[Вкладка Regul 3](#_Toc144910986)

[Преобразование xml файлов ПО 3](#_Toc144910987)

[Работа с файлом ЭЛСИ-ТМК.xml (крейт) 4](#_Toc144910988)

[Работа с файлом \_Application.xml (приложение) 7](#_Toc144910989)

[Этап 1. Текстовые замены 8](#_Toc144910990)

[Этап 2. Модификация dInitFlag 9](#_Toc144910991)

[Этап 3. Создание триггеров 9](#_Toc144910992)

[Этап 4. Преобразование глобальных Retain переменных в Persistent 10](#_Toc144910993)

[Этап 5. Модификация секции FB 11](#_Toc144910994)

[Этап 6. Резервирование 11](#_Toc144910995)

[Этап 7. Создание MBS\_GVL для modbus каналов 16](#_Toc144910996)

[Генерация Modbus устройств из csv 19](#_Toc144910997)

[Генерация Modbus устройств по .xlsx таблице 25](#_Toc144910998)

[Вкладка MKLogic 27](#_Toc144910999)

[Создание .csv файла привязки переменных 27](#_Toc144911000)

[Модификация POU 27](#_Toc144911001)

[Модификация индексов IEC 28](#_Toc144911002)

[Замена циклов FOR на WHILE 29](#_Toc144911003)

[Модификация ФБ 30](#_Toc144911004)

[Вкладка Alpha 32](#_Toc144911005)

[Генерация карт адресов для DevStudio из csv 32](#_Toc144911006)

[Генерация Modbus карты из карт уставок 32](#_Toc144911007)

[Alpha.HMI 33](#_Toc144911008)

[Пропатчить на водяной знак 33](#_Toc144911009)

[Сгенерить функцию updateAlarm 35](#_Toc144911010)

[Получить таблицу функций 36](#_Toc144911011)

[Генератор объектов 37](#_Toc144911012)

[Вкладка Инф. обесп. 39](#_Toc144911013)

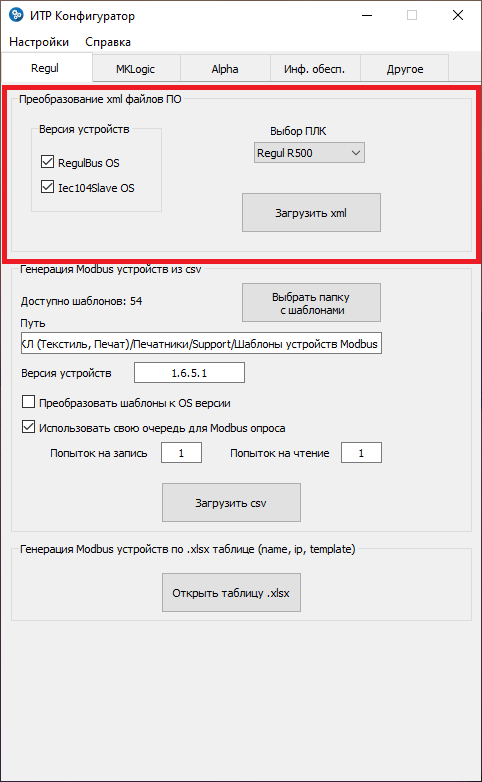
[Вкладка Другое 43](#_Toc144911014)

[ЭЛСИ-ТМК 43](#_Toc144911015)

[Проставить Deny Request флаги 43](#_Toc144911016)

# Вкладка Regul

## Преобразование xml файлов ПО



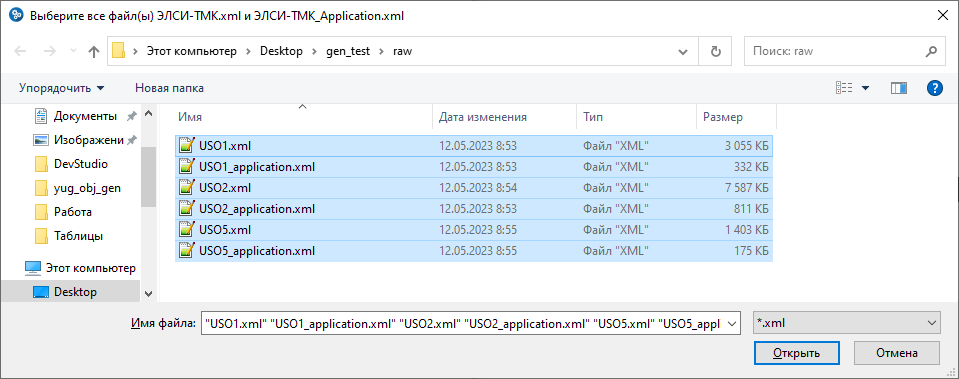
Рисунок

Нажатие кнопки «Загрузить XML» открывает диалоговое окно для выбора файлов ЭЛСИ-ТМК.xml и ЭЛСИ-ТМК\_Application.xml, которые предварительно были созданы программой ACS Configurator.

Одновременно можно выбрать несколько файлов с разных УСО. Программа автоматически попытается загрузить рядом лежащий файл ЭЛСИ-ТМК.xml, если был выбран только одноименный \_Application.xml и наоборот.

При выборе файла \_Application.xml и отсутствии рядом файла ЭЛСИ-ТМК.xml, программа будет модифицировать только приложение контроллера.

Если же выбрать только ЭЛСИ-ТМК.xml, без файла \_Application рядом, то программа сделает ровным счетом ничего.



Рисунок

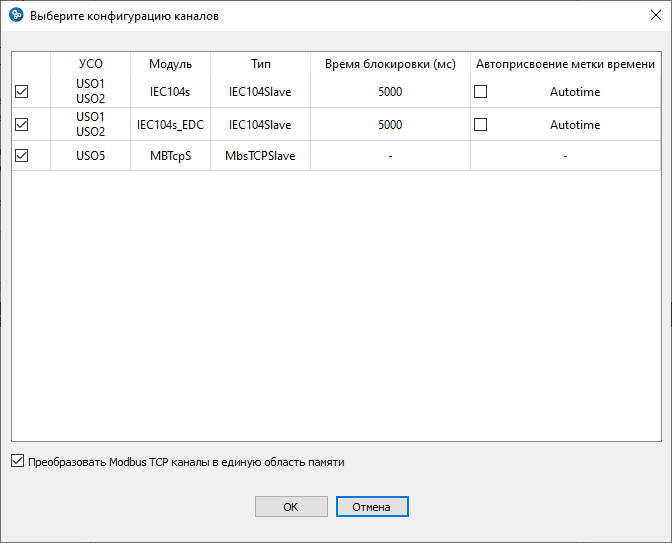
Дальнейшее преобразование этих двух типов файлов делится на два этапа:

* Работа с файлом ЭЛСИ-ТМК.xml (крейт)
* Работа с файлом \_Application.xml (приложение)

### Работа с файлом ЭЛСИ-ТМК.xml (крейт)

Конечная цель работы с крейтом – создание .xml файлов с Модбас или МЭК каналами (адресами и привязанными переменными), которые затем можно вручную или в автоматическом режиме импортировать в Epsilon LD/Astra IDE.

Просматривая каждый крейт, программа попытается найти все slave модули типа IEC104s или MBTcpS внутри него. В случае успеха откроется следующее окно со списком найденных модулей:

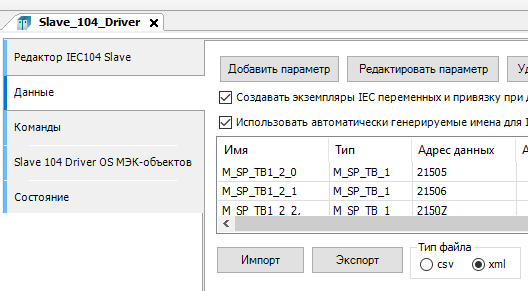


Рисунок

Для каждого выбранного МЭК модуля в УСО создается два файла:

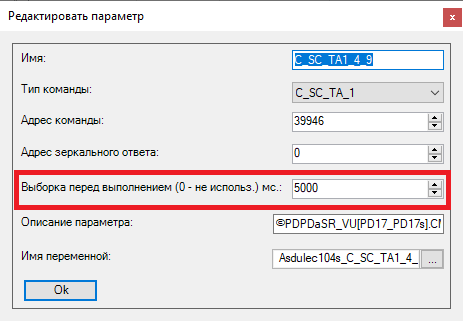
* REGUL\_IEC104\_ST\_CMD.iec104cmd.xml – для каналов команд
* REGUL\_IEC104\_ST\_DATA.iec104data.xml – для каналов данных

Которые можно загрузить с помощью кнопки Импорт в соответствующую вкладку Данные или Команды среды Epsilon LD/Astra IDE в модуль Slave\_104\_Driver:



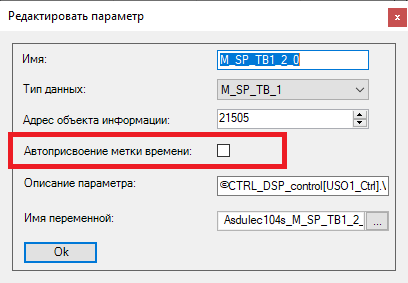
Рисунок

Настройка «Время блокировки (мс)» применима к каналам команд:



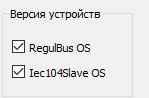
Рисунок

А опция «Автоприсвоение метки времени» к каналам данных:



Рисунок

Опция Iec104Slave OS, доступная в разделе Версия устройств:



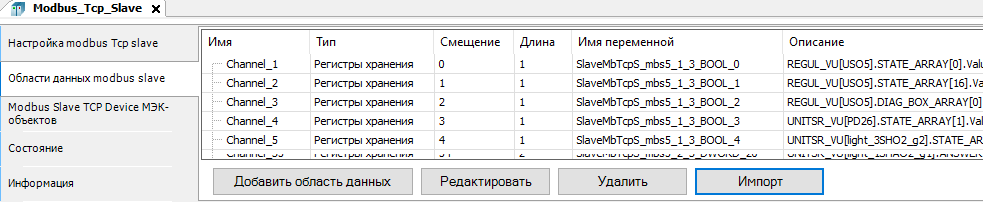
Рисунок

Влияет только на формат времени блокировки в каналах команд. Iec104Slave OS – отмечено, значит время попадет в мс (5000), иначе в сек (5).

Для каждого выбранного Модбас модуля создается один файл:

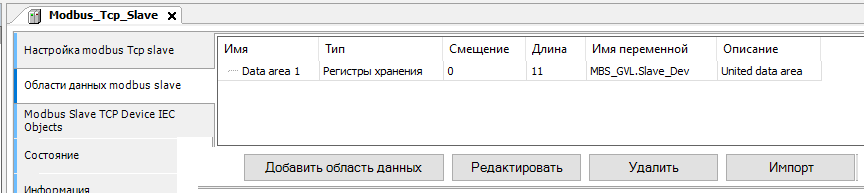
* REGUL\_XML.mb\_direct\_channels.xml

Если опция «Преобразовать Modbus TCP каналы в единую область памяти» не была отмечена, файл REGUL\_XML.mb\_direct\_channels.xml будет содержать все имеющиеся каналы, вместе с привязанной переменной, описанием, адресом и др. При импорте в модуль Modbus\_Tcp\_Slave, каналы будут выглядеть следующим образом:



Рисунок

Если же опция «Преобразовать Modbus TCP каналы в единую область памяти» была отмечена, то создается файл REGUL\_XML.mb\_direct\_channels.xml всего с одним каналом «Data area 1», который является массивом достаточной длины для размещения всех каналов в памяти друг за другом без пробелов. Этот канал имеет маппинг на переменную MBS\_GVL.Slave\_Dev.



Рисунок

Так же программа создаст в \_Application.xml файл MBS\_GVL, где сопоставит каждой modbus переменной ее адрес в массиве Slave\_Dev с помощью операции ADR.

Было выяснено, что при большом количестве каналов (сотни, тысячи), создание единой области памяти существенно влияет на производительность ПЛК, уменьшая загрузку ЦП.



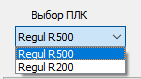
Рисунок

### Работа с файлом \_Application.xml (приложение)

Конечная цель работы с файлом \_Application.xml – создание файла REGUL\_Application.xml, который просто является модифицированной версией приложения для работы с ПЛК Регул.

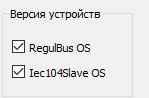
Эту версию приложения, конечно, можно импортировать и вручную в среду Epsilon LD/Astra IDE, но настоятельно рекомендуется производить импорт с помощью python скриптов CodesysLoader.py/CodesysLoader\_Astra.py, так как скрипты вносят финальные правки, необходимые для правильной работы программы (да и просто для компиляции без ошибок).

На модификацию приложения влияет как выбранный ПЛК:



Рисунок

Так и отмеченный чекбокс RegulBus OS:



Рисунок

Модификацию приложения можно разбить на следующие этапы:

* Текстовые замены
* Модификация dInitFlag
* Создание триггеров
* Преобразование глобальных Retain переменных в Persistent
* Модификация секции FB
* Резервирование
* Создание MBS\_GVL для modbus каналов

#### Этап 1. Текстовые замены

Построчно читаем файл приложения и модифицируем строчку при необходимости.

1. Удаление строчек .control

Например, строки:

|  |
| --- |
| A13\_R500\_DO\_32.control := 2 |

Если строка содержит

|  |
| --- |
| .control := |

то она удаляется.

1. Замены

|  |
| --- |
| .Value\_\_  На  . |
| := \_imit\_;  На  ; |

1. В зависимости от выбранных настроек, меняем обращения к модулям DI/DO

Если выбран **RegulBus OS** И ПЛК = **Regul R500**

Замены

|  |
| --- |
| \_R500\_DI\_32.value  На  \_Discrets |
| \_R500\_DO\_32.value  На  \_Discrets |

Иначе, если ПЛК = **Regul R200**

Замены

|  |
| --- |
| \_R500\_DI\_32.value  На  .Inputs\_v2^.Discrets |
| \_R500\_DO\_32.value  На  .Outputs\_v2^.Discrets |

Иначе, если ПКЛ = **Regul R500**

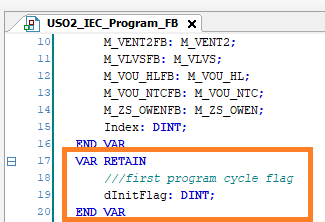
Замены

|  |
| --- |
| \_R500\_DI\_32.value  На  .Inputs\_v1^.Discrets |
| \_R500\_DO\_32.value  На  \_Discrets |

#### Этап 2. Модификация dInitFlag

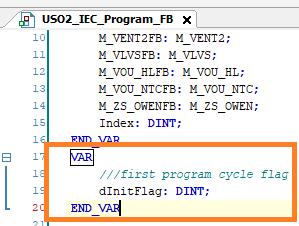
В каждом POU происходит поиск переменной dInitFlag и снимается флаг retain с true на false. Что приводит к следующим изменениям:

Было:



Рисунок

Стало:

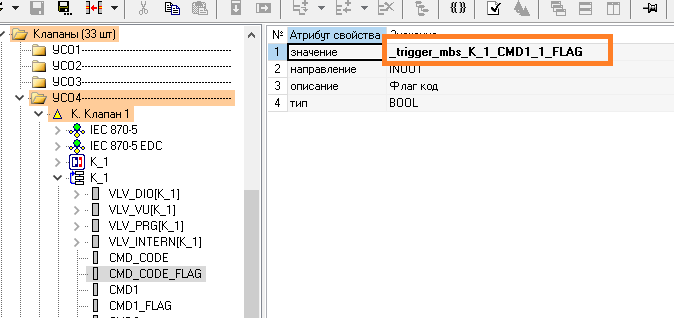


Рисунок

#### Этап 3. Создание триггеров

Триггеры необходимы для правильной работы команд в Modbus Serial Outer Slave устройствах. Эти устройства конфигурируются так, что каналы команд имеют тип канала – Триггер. И имеют соответствующий маппинг на глобальные переменные \_trigger\_mbs\_XX\_CMD\_FLAG. Когда ФБ устройства посылает команду на нижний уровень, он дергает этот флаг, тем самым синхронизируя посылку команды в Modbus Serial Outer Slave.

Переменные не создаются автоматически, они просто начинают использоваться при работе в программе ACS Configurator:



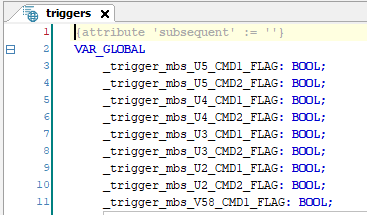
Рисунок

Соответственно конвертер проходится по каждой строчке в \_Application.xml, и, если видит строку содержащую

|  |
| --- |
| \_trigger\_ |

То он складывает полное имя переменной триггера в массив.

Затем создает глобальный объект triggers, содержащий все уникальные переменные \_trigger типа BOOL:

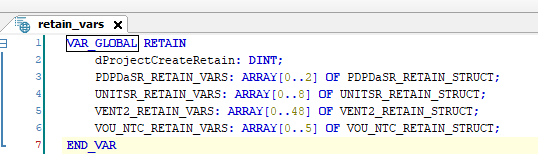


Рисунок

#### Этап 4. Преобразование глобальных Retain переменных в Persistent

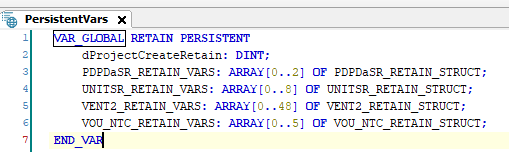
Программа проходится по всем глобальным переменным и если находит глобальные переменные с именем retain\_vars, то добавляет им атрибут persistent со значением true и меняет имя с retain\_vars на PersistentVars.

Было:



Рисунок

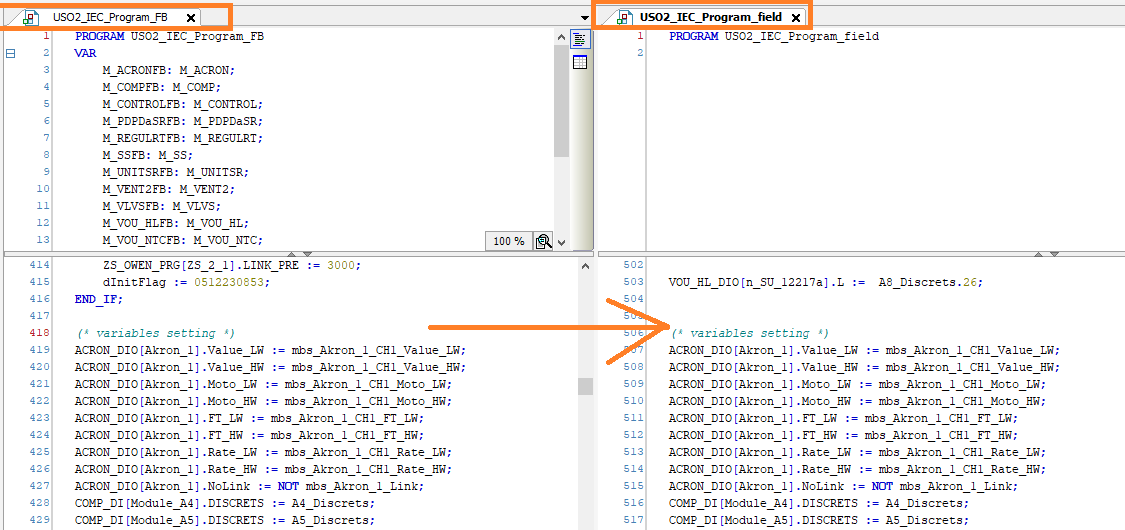
Стало:



Рисунок

#### Этап 5. Модификация секции FB

Вырезаем из секций FB все строчки под заголовком (\* variables setting \*) и вставляем их в конец field секции. Не помню, зачем точно это нужно, вроде для правильной работы блоков имитации.

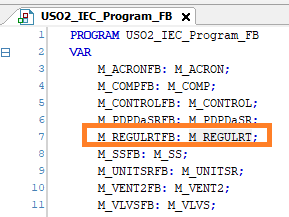


Рисунок

#### Этап 6. Резервирование

Для начала необходимо понять тип резервирования. Программа проходится по всем POU и ищет переменную, которая содержит слово

|  |
| --- |
| M\_REGUL |



Рисунок

Затем она смотрит на название типа этой переменной, если тип называется **M\_REGUL**, то тип резервирования = 0 (обычное).

Если **M\_REGULOS,** то тип резервирования = 1 (ОС).

Если **M\_REGULRT,** то тип резервирования = 2 (РТ).

Иначе тип резервирования = 3 (без резервирования – нет модификаций).

Далее исходя из типа резервирования происходит модификация.

1. Тип резервирования 0 (обычное). Тип блока диагностики УСО - **M\_REGUL**

На данный момент данный тип резервирования можно увидеть на станции Терехово ЭМС.

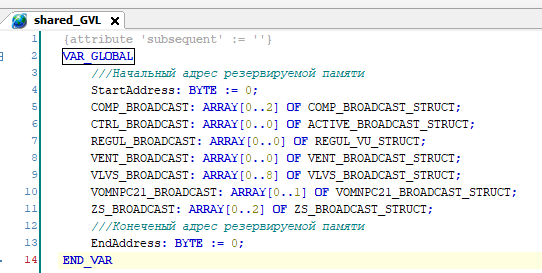
Создается файл shared\_GVL с атрибутом [**subsequent**](https://help.codesys.com/api-content/2/codesys/3.5.13.0/en/_cds_pragma_attribute_subsequent/)**,** который позволяет расположить все объявленные внутри переменные в памяти друг за другом без [выравнивания](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%80%D0%B0%D0%B2%D0%BD%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Так же создаются переменные StartAddress и EndAddress, задающие границы области резервирования между контроллерами.

Затем проходимся по всем глобальным переменным и если их имя оканчивается на

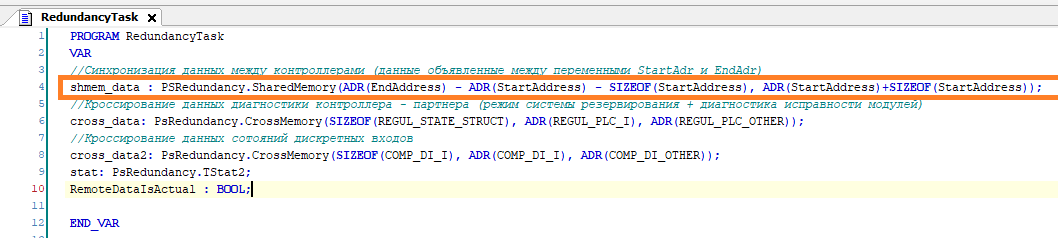
|  |
| --- |
| \_BROADCAST  Или  \_SHARED |

То они вырезаются из своего текущего месторасположения и вставляются внутрь shared\_GVL:



Рисунок

Затем область памяти между StartAddress и EndAddress уже используется разработчиком, например в RedundancyTask:



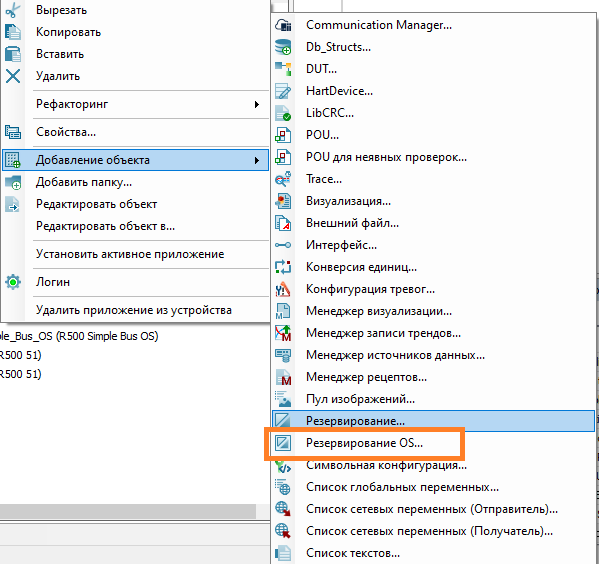
Рисунок

**SharedMemory** - При синхронизации данные копируются от ведущего модуля ЦП к ведомому, заменяя предыдущие данные в ведомом ЦП.

1. Тип резервирования 1 (OS). Тип блока диагностики УСО - **M\_ REGULOS**

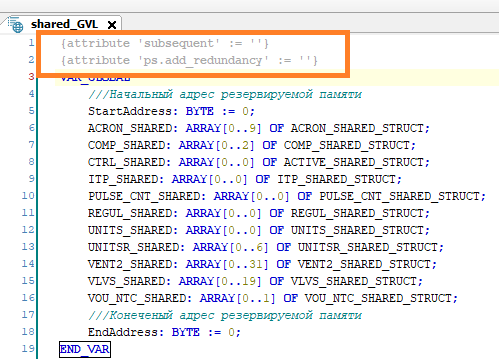
Вместо обычного объекта Redundancy в дереве проекта для этого типа резервирования используется RedundancyOS.

**Пока данный тип резервирования не используется в проектах из-за некорректной работы ПЛК**.



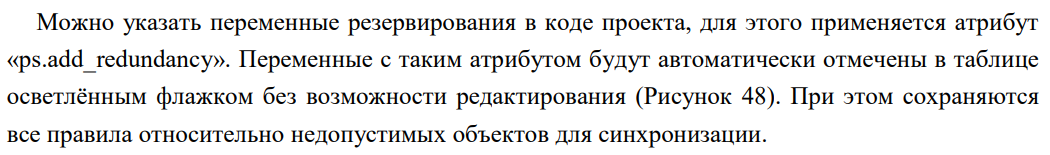
Рисунок

Так же, как и с типом резервирования 0, создается файл shared\_GVL, но помимо атрибута **subsequent**, добавляется так же атрибут **ps.add\_redundancy**.



Рисунок

Из документации [Redundancy\_User\_Guide\_DPA\_302\_4\_v2\_6\_rus.pdf](https://reglab.ru/uploads/Material/AstraIDE/Documents/Redundancy_User_Guide_DPA_302_4_v2_6_rus.pdf):



Рисунок

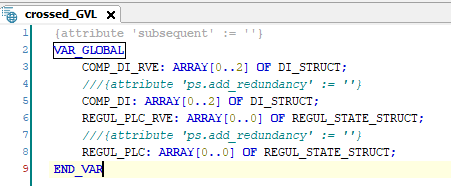
В shared\_GVL попадают переменные, которые оканчиваются на

|  |
| --- |
| \_SHARED |

В отличии от предыдущего типа резервирования, переменные оканчивающиеся на \_BROADCAST не трогаем.

Так же создается документ crossed\_gvl с атрибутом **subsequent**.

**CrossMemory** - при синхронизации осуществляет обмен данными между модулями ЦП.



Рисунок

В crossed\_gvl попадают переменные, которые оканчиваются на

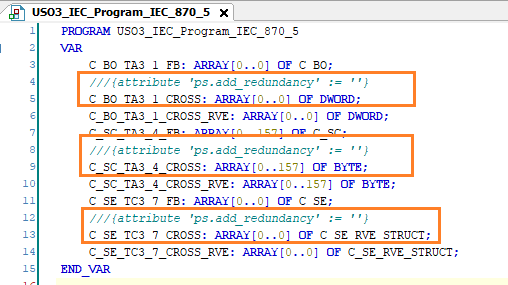
|  |
| --- |
| \_RVE |

А так же одноименные переменные, без этой приставки. Например name123 и name123\_RVE.

Одноименным переменным добавляется атрибут **ps.add\_redundancy.**

Последним этапом будет добавление атрибута **ps.add\_redundancy** переменным внутри любых POU, которые имеют окончание

|  |
| --- |
| \_CROSS |



Рисунок

Сейчас программа добавляет закоменченные атрибуты, их нужно вручную раскомментить (баг Astra IDE).

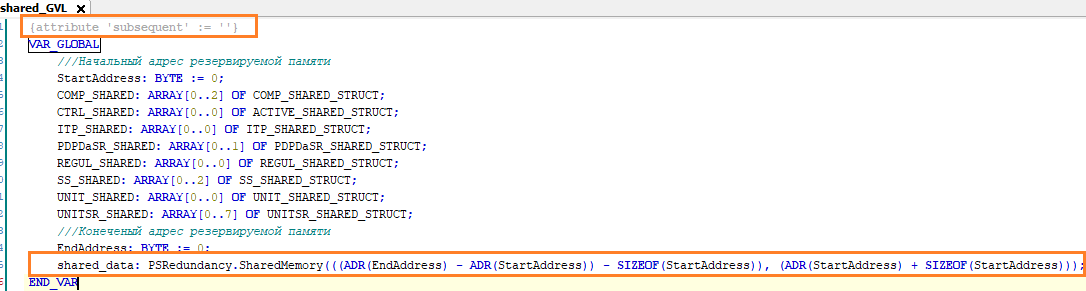
1. Тип резервирования 3 (RT). Тип блока диагностики УСО - **M\_ REGULRT**

Так же, как и в типе резервирования OS мы добавляем файлы shared\_GVL и crossed\_GVL. Но есть отличия.

shared\_GVL и crossed\_GVL создаются только с атрибутом **subsequent**. Атрибут **ps.add\_redundancy** больше не используется.

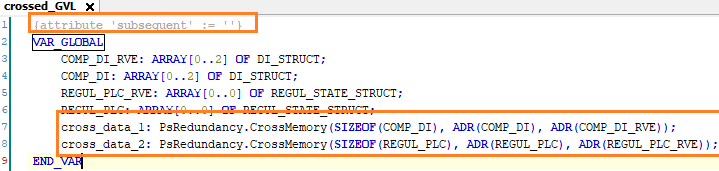
В конце shared\_GVL добавляется строчка с объявлением переменной shared\_data.

|  |
| --- |
| shared\_data: PSRedundancy.SharedMemory(((ADR(EndAddress) - ADR(StartAddress)) - SIZEOF(StartAddress)), (ADR(StartAddress) + SIZEOF(StartAddress))); |



Рисунок

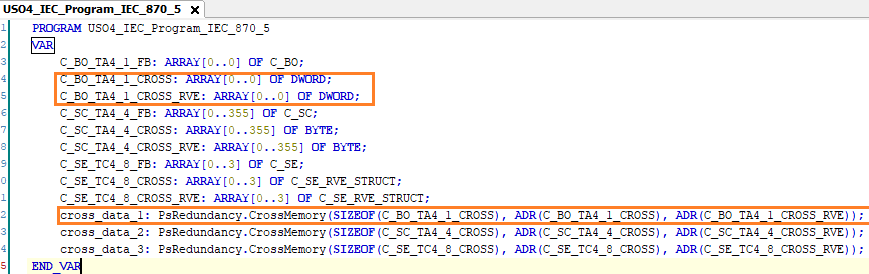
В конце crossed\_GVL добавляются строчки с объявлением переменных cross\_data\_x. Где x – порядковый номер пары переменных name\_RVE и name.



Рисунок

Последним этапом будет добавление переменных cross\_data\_X внутри любых POU, где объявлены пары переменных типа

|  |
| --- |
| \_CROSS  И  \_CROSS\_RVE |

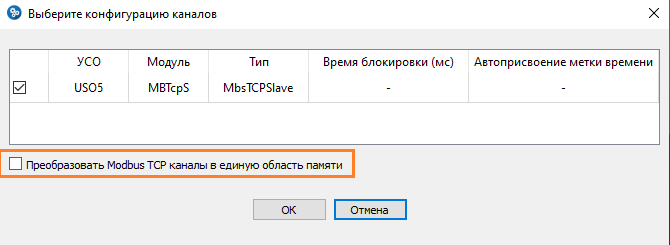


Рисунок

#### Этап 7. Создание MBS\_GVL для modbus каналов

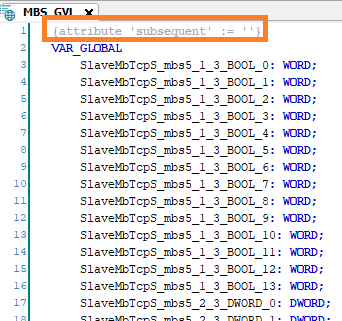
Это опциональный этап, делается лишь в случае наличия модуля MbsTcpSlave с каналами.

Если на этапе выбора конфигурации каналов не была отмечена опция «Преобразовать Modbus TCP каналы в единую область памяти»



Рисунок

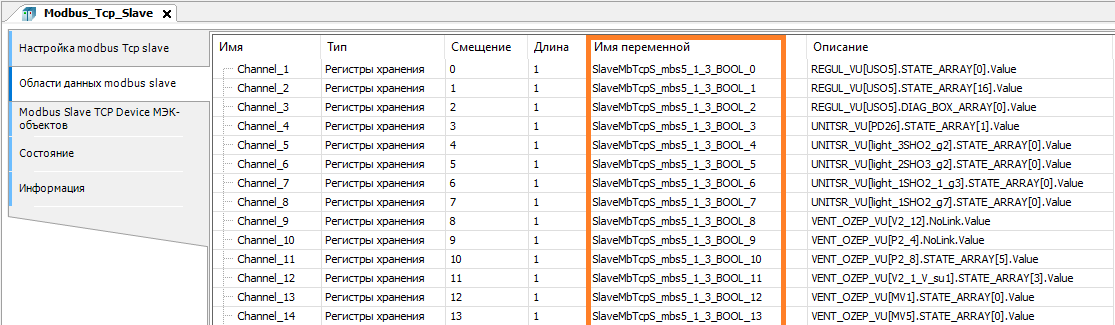
То создается файл глобальных переменных MBS\_GVL с атрибутом **subsequent** и просто со списком переменных SlaveMbTcpS\_mbs\_ххх:



Рисунок

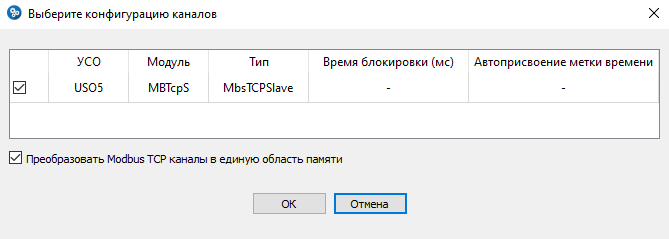
Названия переменных берутся из модуля MbsTcpSlave файла крейта ЭЛСИ-ТМК.xml, при парсинге каналов.

Их же можно найти и в модуле Modbus\_Tcp\_Slave, каждая переменная привязана к своему каналу:



Рисунок

Если на этапе выбора конфигурации каналов была отмечена опция «Преобразовать Modbus TCP каналы в единую область памяти»

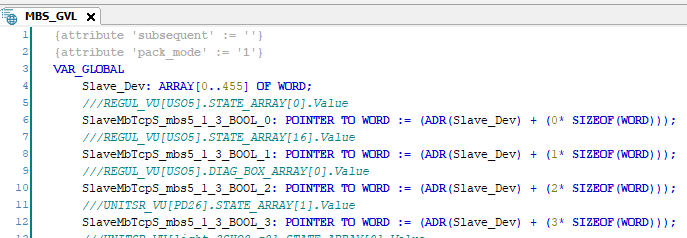


Рисунок

То созданный файл MBS\_GVL помимо атрибута **subsequent**, так же содержит атрибут [pack\_mode](https://help.codesys.com/api-content/2/codesys/3.5.13.0/en/_cds_pragma_attribute_pack_mode/) со значением 1.

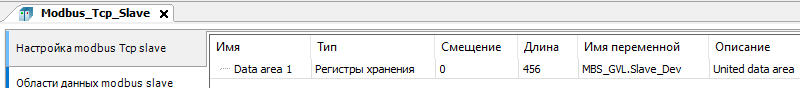
Так же, MBS\_GVL содержит в своем заголовке объявление массива Slave\_dev такой размерности, которая способна вместить все переменные SlaveMbTcpS\_mbs\_ххх, привязанные к каналам.

А переменные SlaveMbTcpS\_mbs\_ххх становятся указателями на массив Slave\_Dev.



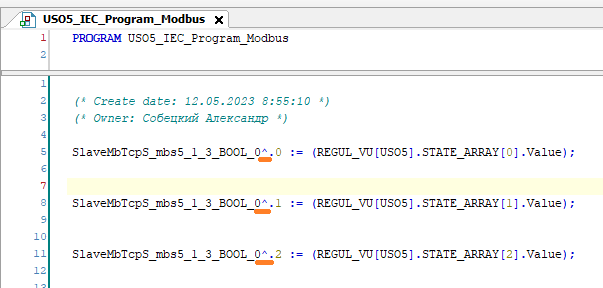
Рисунок

При импорте каналов в модуль Modbus\_Tcp\_Slave в списке будет всего один канал, связанный с переменной Slave\_Dev:



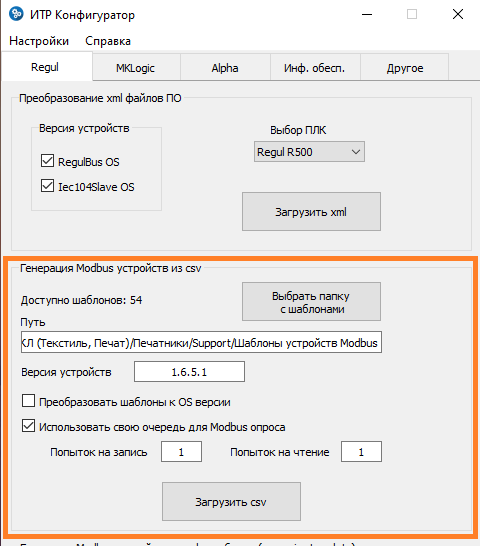
Рисунок

Так же текст внутри всех POU изменяется в связи с добавлением оператора разыменования ^ для указателей из MBS\_GVL



Рисунок

## Генерация Modbus устройств из csv

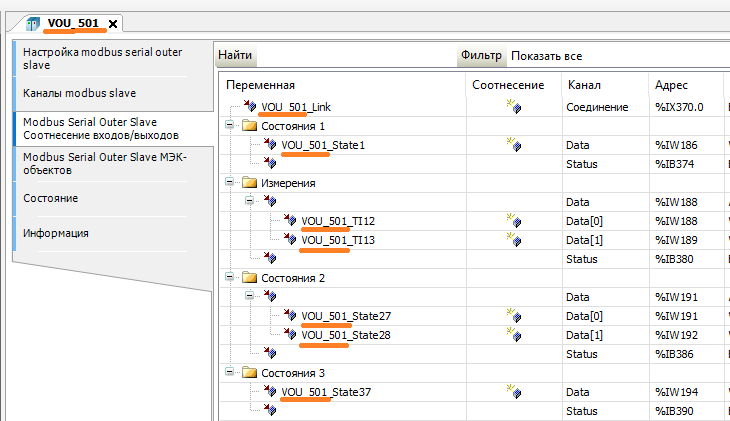


Рисунок

Сначала необходимо выбрать папку с шаблонами. Это директория, содержащая список шаблонов Modbus RTU/TCP устройств.

Новое устройство создается в среде Epsilon LD/Astra IDE, затем экспортируется в виде xml объекта.

Отличительной чертой шаблона является то, что все привязанные переменные начинаются на имя шаблона.



Рисунок

Затем при нажатии на кнопку Загрузить csv можно загрузить один или несколько csv файлов Infinity Server.csv, созданных в программе ACS Configurator.

Конвертер пройдется по всем объектам дерева Infinity Server в поисках следующих свойств:

* Тип Modbus (8814)
* Адрес КП (8888)
* Modbus модуль (8810)
* Modbus канал (8811)
* Маппинг (8812)
* IP адрес (8813)
* Шаблон устройства (8821)
* Modbus\_DelayBetweenPolls (89101)

Для каждого объекта проверяется валидность свойств. Если не было заполнено (найдено) хоть одно из необходимых свойств, то генерация modbus устройств по такому объекту не производится.

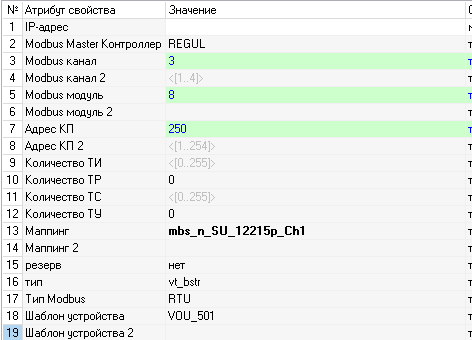
Для устройств с Тип Modbus равным RTU (0) необходимые поля:

* Тип Modbus
* Маппинг
* Шаблон устройства
* Адрес КП
* Modbus модуль
* Modbus канал

Для устройств с тип Modbus равным TCP (1) необходимые поля:

* Тип Modbus
* Маппинг
* Шаблон устройства
* Адрес КП
* IP адрес

Ниже приведен пример RTU объекта Infinity Server:



Рисунок

Для каждого RTU или TCP объекта Infinity Server будет произведена проверка наличия указанного .xml файла шаблона устройства в выбранной папке с шаблонами (в нашем примере наличие файла VOU\_501.xml).

Если шаблона нет в папке, объект пропускается. В конце программа выведет список отсутствующих шаблонов, если такие имеются.

После того, как программа пройдет по всем объектам Infinity Server и составит список объектов, по которым нужно сгенерить Modbus устройства, будет произведено удаление найденных дубликатов среди списка объектов.

Например, несколько объектов Infinity Server могут забирать информацию с одного и того же устройства modbus. Например, в случае некоторых клапанов, в ACS Configurator’e каждый клапан представляет собой отдельный объект Infinity Server, но modbus устройство на все клапана – одно.

Дубликатом будут считаться объекты в следующих случаях:

а). Если тег объекта начинается на USOx., то дубликатами считаются объекты с одного УСО и одинаковыми значениями поля Маппинг

б). Или если тег объекта не начинается на USOx, то дубликатом считаются объекты из одного и того же .csv файла с одинаковым значением поля Маппинг.

Затем программа проходится по каждому объекту, берет соответствующий шаблон, модифицирует его и сохраняет в папку modules для RTU шаблонов, и в папку modules\_tcp для TCP шаблонов.

В папке modules\_tcp содержатся все TCP шаблоны по одиночке для каждого найденного устройства, а также файл DEV\_ALL.xml, который содержит внутри себя все шаблоны.

В папке modules все RTU шаблоны располагаются внутри вложенных папок согласно номеру модуля и номеру канала.

Автоматические python скрипты импортируют файл DEV\_ALL.xml.

Модификация Modbus шаблона включает этапы:

* Замена имени шаблона – в нашем примере имя шаблона VOU\_501 поменяется на mbs\_n\_SU\_12215p\_Ch1 как в имени modbus устройства, так и во всех его внутренних переменных. Например, вместо VOU\_501\_Link привязка будет mbs\_n\_SU\_12215p\_Ch1\_Link.
* Замена адреса КП – число из свойства «Адрес КП» попадает в графу «Адрес ведомого устройства».
* Замена IP адреса – текст из свойства «IP адрес» попадает в графу «IP адрес»
* Замена версии устройства – текст из поля «Версия устройств» конвертера заменяется внутри узла Version шаблона
* Замена типа устройства – в зависимости от отмеченной опции «Преобразовать шаблоны к OS версии», результирующее устройство будет либо OS версии, либо нет. Сам шаблон при этом может быть как OS-ный, так и обычный.

Если Infinity Server объект содержал так же свойство «Modbus\_DelayBetweenPolls» со значением 1, то в файле DEV\_ALL.xml после этого объекта вставится пустышка oven\_dummy.xml с адресом КП = 247 и одним каналом по нулевому адресу. Это своего рода костыль для ОВНов, которые то ли неправильно отвечают на запросы, то ли что-то с ними еще не так и они пропадают со связи. С пустышками становится вроде лучше.

Вообще, каждый Infinity Server объект может содержать до 5 разных шаблонов устройств. Например, если необходимо несколько каналов для одних и тех же данных (шахты, ВОУ). Для каждого нового шаблона имеется свой набор свойств:

Шаблон 1:

* Адрес КП (свойство 8888)
* Modbus модуль (свойство 8810)
* Modbus канал (свойство 8811)
* Маппинг (свойство 8812)
* IP адрес (свойство 8813)
* Шаблон устройства (свойство 8821)

Шаблон 2:

* Адрес КП 2 (свойство 8816)
* Modbus модуль 2 (свойство 8817)
* Modbus канал 2 (свойство 8818)
* Маппинг 2 (свойство 8819)
* IP адрес 2 (свойство 8820)
* Шаблон устройства 2 (свойство 8822)

Шаблон 3

* Адрес КП 3 (свойство 8827)
* Modbus модуль 3 (свойство 8828)
* Modbus канал 3 (свойство 8829)
* Маппинг 3 (свойство 8830)
* IP адрес 3 (свойство 8831)
* Шаблон устройства 3 (свойство 8832)

Шаблон 4:

* Адрес КП 4 (свойство 8833)
* Modbus модуль 4 (свойство 8834)
* Modbus канал 4 (свойство 8835)
* Маппинг 4 (свойство 8836)
* IP адрес 4 (свойство 8837)
* Шаблон устройства 4 (свойство 8838)

Шаблон 5:

* Адрес КП 5 (свойство 8911)
* Modbus модуль 5 (свойство 8912)
* Modbus канал 5 (свойство 8913)
* Маппинг 5 (свойство 8914)
* IP адрес 5 (свойство 8915)
* Шаблон устройства 5 (свойство 8916)

Следующие свойства могут присутствовать в ACS Конфигураторе, но не берутся во внимание конвертером:

* Modbus Master Контроллер
* Количество ТИ
* Количество ТР
* Количество ТС
* Количество ТР

Отдельно про недавно добавленный функционал с использованием очереди для Modbus опроса:



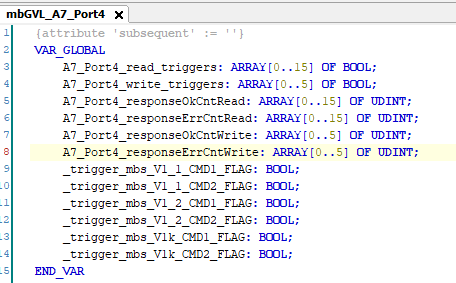
Рисунок

При отмеченном чекбоксе «Использовать свою очередь для Modbus опроса», программа перед сохранением всех модбас устройств на диск, сортирует устройства по портам УСО, открывает каждое и считает количество каналов на запись и на чтение.

Если встречает любой (input/output) канал типа «таймер», то меняет его на тип «триггер». Каждый триггер канал привязывается к элементу массива read\_triggers (канал на чтение) или write\_triggers (канал на запись).

Автоматически создает в выходной папке modules/Queue файлы mbGVL\_Ax\_Portx для импорта. Каждый GVL файл характеризует один порт и содержит:

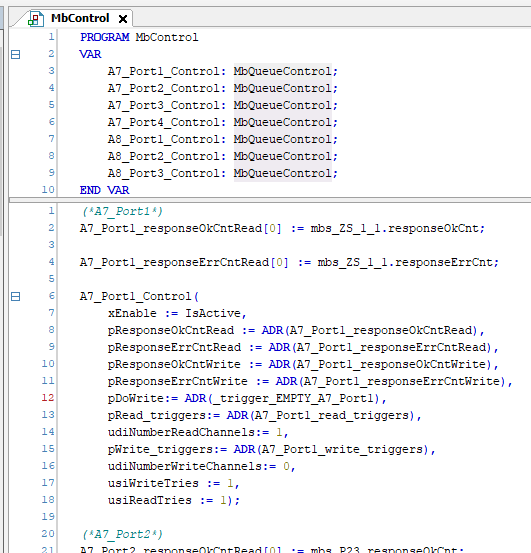
* read\_triggers – массив триггеров на чтение каналов на порту
* write\_triggers – массив триггеров на запись каналов на порту
* responseOkCntRead – массив успешного счета операций на чтение
* responseErrCntRead - массив счета неудачных операций на чтение
* responseOkCntWrite - массив успешного счета операций на запись
* responseErrCntWrite - массив счета неудачных операций на запись
* \_trigger\_ - триггеры, на которые уже были привязаны output каналы модбас устройств



Рисунок

Размерность массивов определяется количеством каналов на запись/чтение. Если количество каналов было 0, то массив размерностью [0..0] все равно создастся. Если на порту у устройств не было найдено триггеров, то создастся переменная \_trigger\_EMPTY.

Помимо GVL файлов, программа так же создаст PRG с названием MbControl, где будет произведен вызов библиотечного ФБ MbQueueControl для каждого из портов.

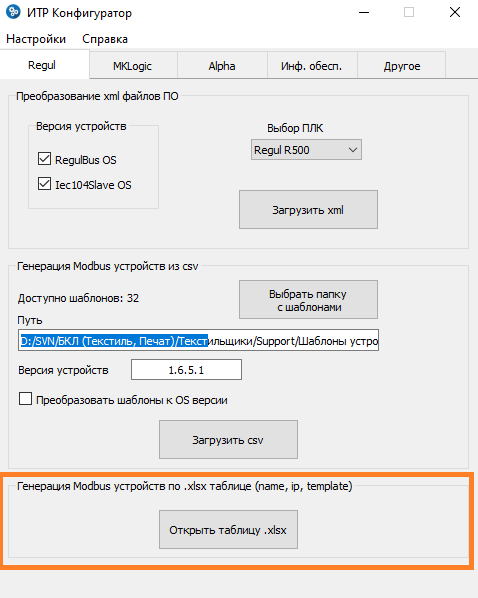


Рисунок

Числа попыток на запись/чтение из программы, попадают на входы usiWriteTries/usiReadTries функционального блока MbQueueControl.

Python скрипт загрузки при обнаружении папки modules/Queue, затянет все лежащие внутри файлы, затем при наличии GVL с названием triggers, закомментирует там все совпадающие по именам переменные \_trigger\_ с аналогичными переменными из глобальных списков mbGVL\_Ax\_Portx. Если абсолютно все переменные в GVL triggers будут закомментированы, то скрипт удалит triggers GVL.

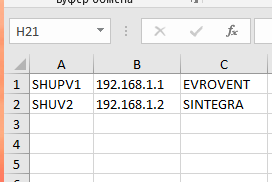
## Генерация Modbus устройств по .xlsx таблице



Рисунок

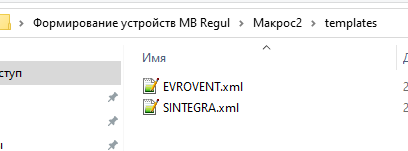
Этот вариант используется только на депо Сокол.

Необходимо выбрать таблицу .xlsx такого формата:

  
Рисунок

Где в первой колонке пишем имя устройства, во второй колонке его IP адрес, а в третьей колонке его имя шаблона.

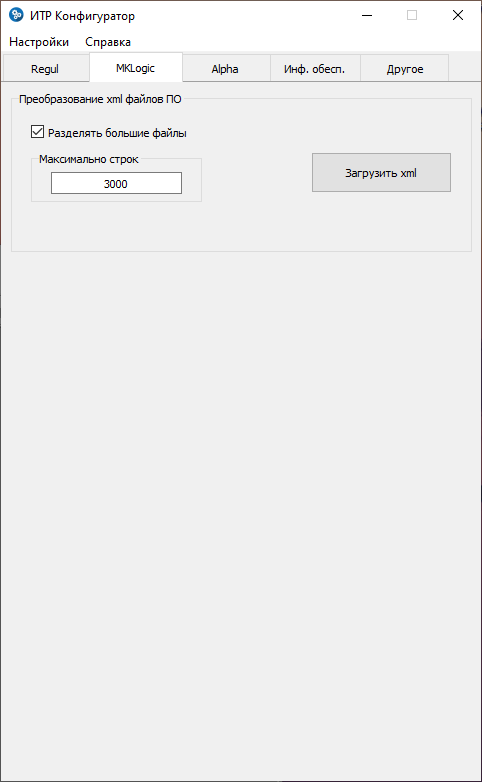
На одном уровне с таблицей должна быть папка templates с шаблонами:



Рисунок

Конвертер просто пойдет по списку, откроет нужный шаблон, поменяет там IP адрес и имя и сохранит в предложенном месте все табличные устройства.

# Вкладка MKLogic



Рисунок

Нажатие кнопки «Загрузить XML» открывает диалоговое окно для выбора файлов ЭЛСИ-ТМК\_Application.xml, которые предварительно были созданы программой ACS Configurator.

Работа с крейтом не производиться.

Цель – модификация приложения контроллера.

Для каждого выбранного Application.xml файла создается одноименная папка, куда попадают выходные модифицированные конвертером файлы.

Этапы модификации приложения:

1. Создание .csv файла привязки переменных
2. Модификация POU

## Создание .csv файла привязки переменных

Программа находит все списки глобальных переменных (global\_vars, const\_vars и др.) и переносит все переменные согласно их типу в .csv файл такого формата, который может быть импортирован в среду isagraf.

Каждой переменной создается строка типа:

|  |
| --- |
| "%1,%2,%3,,%4,Var,ReadWrite,,,,,False,," |

Где в плейсхолдеры %1-%4 – подставляются значения:

%1 – имя переменной

%2 – тип переменной

%3 – для массивов размерность, для остальных типов – ничего

%4 – начальное значение (если имеется)

## Модификация POU

Программа проходит по всем объектам POU, которые имеют тип программа (program), т.е. имеют какой-то ST текст внутри.

Забирает внутренний ST текст каждого POU, модифицирует его и сохраняет отдельными .txt файлами каждую секцию.

Секция USO1\_sFB становится USO1\_FB.txt.

Секция USO1\_sFBModbus становится USO1\_FBModbus.txt.

Секция USO1\_IEC\_870\_5 становится USO1\_IecProgram.txt.

Секция USO1\_IEC\_870\_5\_ED становится USO1\_IecProgram\_EDC.txt.

В зависимости от настройки «Разделять большие файлы» в совокупности с максимальным числом строк, каждая из секций может быть сохранена в несколько файлов, каждый из которых будет содержать количество строк, не превышающих указанное число.

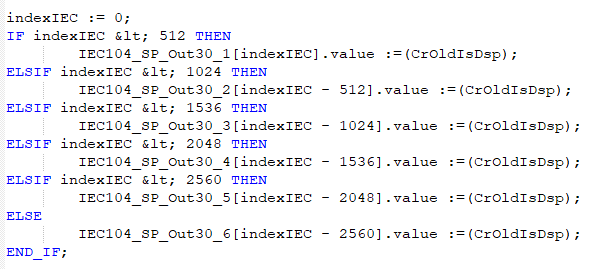
Этапы модификации ST текста каждого POU:

1. Модификация индексов IEC
2. Замена циклов FOR на WHILE
3. Модификация ФБ

Так же, любая пустая строчка, не содержащая текста, пропускается, в результате чего выходные .txt файлы не содержат пробелов между строчками.

### Модификация индексов IEC

В секциях IEC\_870\_5 преобразуем следующий блок кода:



Рисунок

В одну строку в зависимости от значения переменной indexIEC.

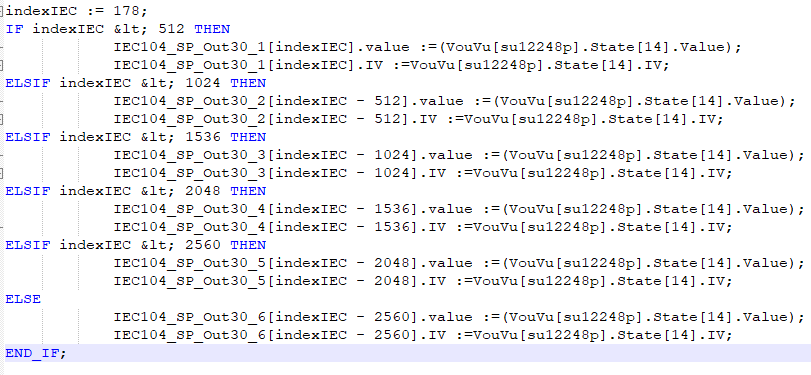
Например, для indexIEC := 511 строка будет такой:

|  |
| --- |
| IEC104\_SP\_Out30\_**1**[**511**].value :=(CrOldIsDsp); |

А для индекса 512 уже будет так:

|  |
| --- |
| IEC104\_SP\_Out30\_**2**[**0**].value :=( CrOldIsDsp); |

Помимо строчек с .value в условиях так же может быть указан флаг качества .IV:



Рисунок

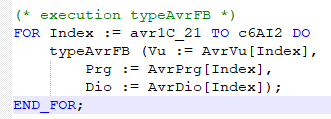
Тогда на выходе будет две строчки:

|  |
| --- |
| IEC104\_SP\_Out30\_**1**[**178**].value :=(VouVu[su12248p].State[14].Value);  IEC104\_SP\_Out30\_**1**[**178**].IV :=VouVu[su12248p].State[14].IV; |

### Замена циклов FOR на WHILE

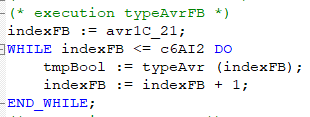
Все циклы FOR заменяется на аналогичные WHILE. Переменная Index меняется на indexFB везде, кроме секции sFBModbus (там она меняется на indexFBModbus).

Было:



Рисунок

Стало:



Рисунок

Обратите внимание, что вызов ФБ в цикле тоже изменился, но это уже следующий этап.

### Модификация ФБ

Когда программа находит ФБ, она его видит в формате

|  |
| --- |
| nameFB (arg1 := arg2[arg3], arg1 := arg2[arg3]...) |

Все преобразование сводится к тому, что нам нужно вызвать этот же ФБ и сохранить результат в переменную tmpBool.

Мы всегда из имени FB удаляем в конце символы FB

Варианты модификаций:

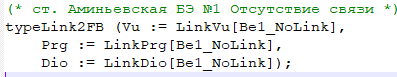
1. Если ФБ вызывается без аргументов:

|  |
| --- |
| testFB(); |

То он будет заменен на такую строчку:

|  |
| --- |
| tmpBool := test (); |

1. Если имя ФБ НЕ содержит typeModbus, например:



Рисунок

То мы вызываем ФБ и единственным его аргументом становится индекс:



Рисунок

То есть:

|  |
| --- |
| Было  typeLink2FB (arg1 := arg2[Index], arg1 := arg2[Index]...)  Стало  tmpBool := typeLink2 (Index); |

1. Если имя ФБ содержит typeModbus, например



Рисунок

То мы преобразовываем это к вызову ФБ без имен аргументов, только сами аргументы.



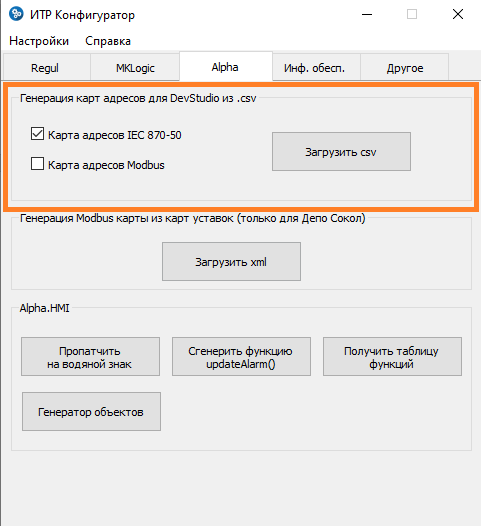
Рисунок

То есть:

|  |
| --- |
| Было  typeModbusTestFB (arg1 := arg2[arg3], arg1 := arg2[arg3]...)  Стало  tmpBool := typeModbusTest (arg2[arg3], arg2[arg3] …); |

# Вкладка Alpha

## Генерация карт адресов для DevStudio из csv



Рисунок

Рассматриваемое действие позволяет из .csv файлов дерева сигналов Infinity Server’a, экспортируемого программой ACS Configurator, создать карты привязки сигналов к адресам для программы DevStudio.

В зависимости от выбранной настройки IEC 870-5 или Modbus программа будет создавать привязку адресов либо на МЭК протокол, либо на Modbus.

Для правильной работы программы, в ACS Configurator должны быть проставлены все связи, что бы теги Infinity Server были адресные.

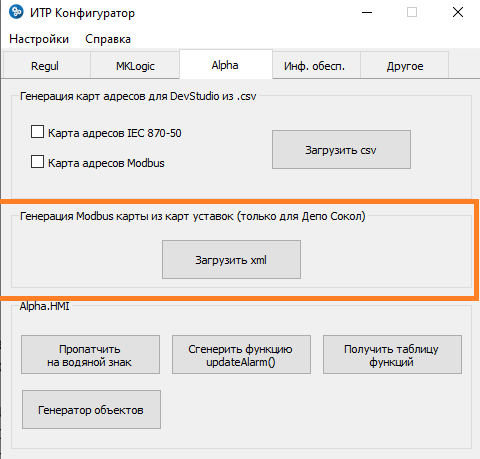
Можно загружать несколько .csv файлов, чтобы на выходе получить несколько карт адресов. Причем не обязательно иметь несколько .csv файлов по одному на каждую КП (modbus) или станцию (МЭК). Можно иметь один общий .csv файл с различными номерами КП и номерами станций внутри.

Карты адресов Modbus сохраняются в файл Modbus\_TCP\_USO?.xml, где ? – номер КП.

Карты адресов МЭК сохраняются в файл IEC104\_USO?.xml, где ? – номер станции.

## Генерация Modbus карты из карт уставок

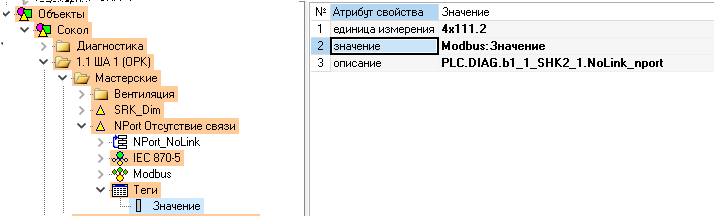
Данный функционал используется только для депо Сокол, для остальных проектов и Депо – используется генерация карт адресов из .csv.



Рисунок

Кнопка Загрузить xml позволяет загрузить одну или несколько карт уставок из программы ACS Configurator проекта Депо Сокол. И на выходе получить файлы для DevStudio с заполненными адресами Modbus.

В программе ACS конфигуратор каждый тег в карте уставок объекта содержит поле описание, где храниться тег и поле единица измерения где хранится адрес у этого тега:



Рисунок

Результирующий .xml файл карт Modbus адресов будет называться так же, как и карта уставок, только с заменой слова «Теги» на «MAP».

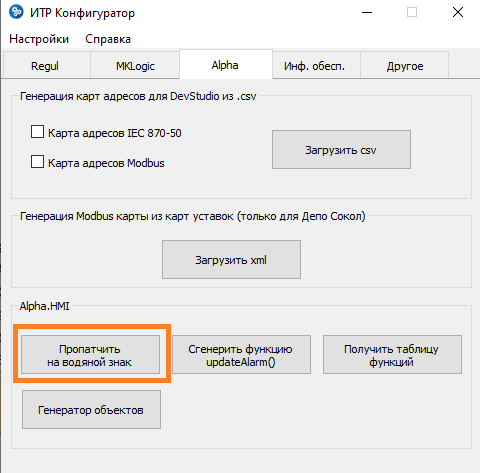
## Alpha.HMI

### Пропатчить на водяной знак

Программа предлагает выбрать файл библиотеки **alpha.hmi.ui.dll** для Windows или **libalpha.hmi.ui.so** для Linux.

Данный файл лежит папке вместе с установленными бинарниками Alpha.HMI.

Пропатчив его, исчезнет водяной знак на HMI формах, появляющийся после 15-ти совокупных минут работы формы в режиме исполнения.



Рисунок

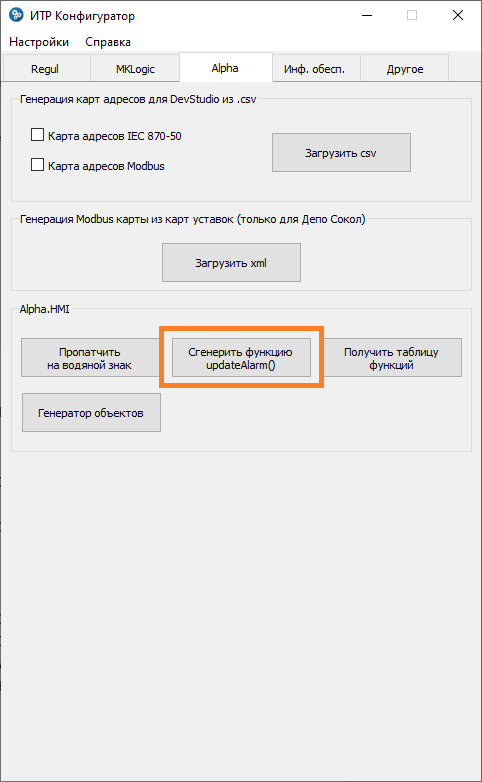
Конвертер должен пропатчить загруженную библиотеку на предмет замены четырех строчек на пробелы:

* Лицензия не обнаружена или ограничена.
* Использование %1 без лицензии допустимо только для разработки и испытания проектов автоматизации инжиниринговыми компаниями.
* Не допускается использование нелицензированных компонентов конечными пользователями на объектах.
* Все права защищены © %1

Первая строка - 71 пробел, вторая - 232, третья - 183, четвертая - 40. На все русские символы - по два пробела, на другие - один.

Т.е. мы просто скрываем водяной знак, сама его отрисовка воспроизводится и в больших проектах может все так же чуть снижать производительность.

### Сгенерить функцию updateAlarm



Рисунок

Данная опция предлагает выбрать файл SETTINGS.xml, который лежит в папке support внутри любого Alpha.HMI проекта ЭМС, Депо.

Данный файл содержит внутри себя список экранов проекта и список типов элементов. У каждого экрана есть свой код. У каждого типа элемента есть свой аварийный тег.

Конвертер проходится по списку экранов, открывает в режиме чтения каждый .omobj экран из папки objects HMI проекта, смотрит имеющиеся внутри объекты и их типы. Если найденный объект по типу совпадает с описанным типом внутри файла SETTINGS.xml, то с объекта забирается алиас, а с типа – аварийный тег, таким образом формируя полный аварийный тег до конкретного объекта.

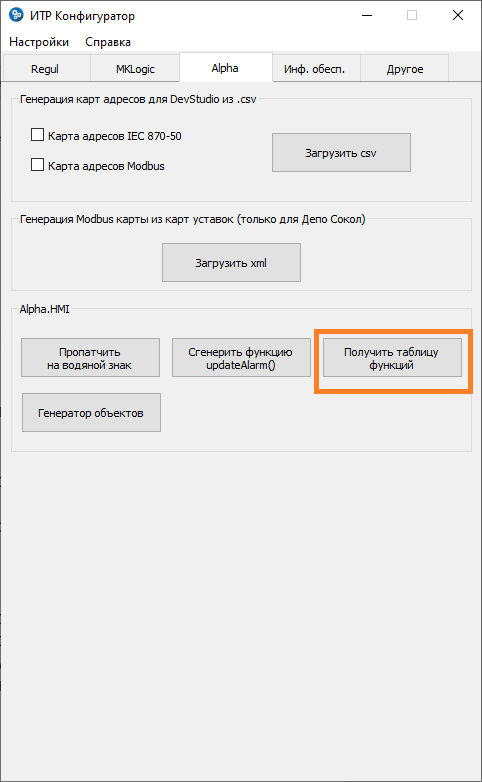
В итоге получаем актуальные списки аварийный тегов объектов по каждому экрану. И javascript функцию, которая возвращает нужный список по коду экрана.

Дальше уже внутри HMI проекта эту функцию вызывают верхние кнопки навигации, передавая свой код, чтобы по аварийным тегам загораться красным.

Более подробно про формат файла SETTINGS.xml можно прочитать внутри самого файла SETTINGS.xml.

Программа сохраняет содержимое функции внутри файла func.txt и предлагает сразу же обновить содержимое функции в самом HMI проекте. Если отказаться от автоматической замены, то все содержимое файла func.txt нужно будет потом вручную вставить по пути unit.Global.updateAlarm().

### Получить таблицу функций

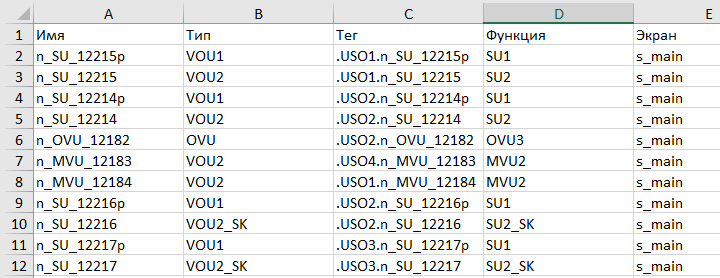


Рисунок

Данная кнопка позволяет загрузить все тот же файл SETTINGS.xml, но важен будет лишь список экранов.

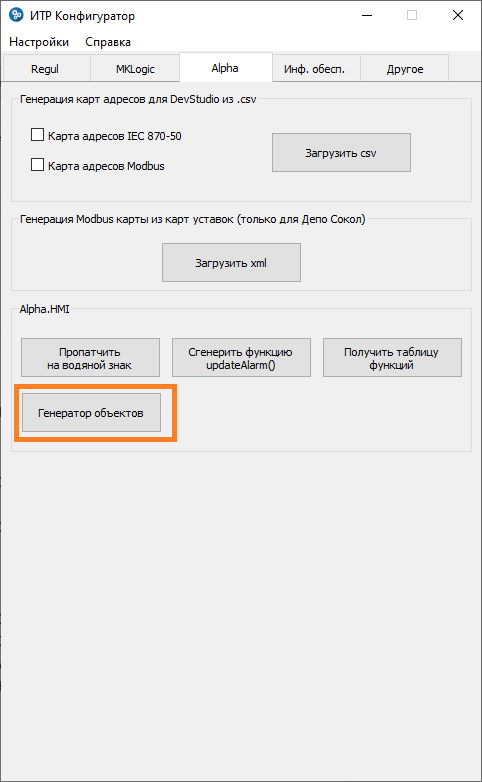
Программа откроет каждый экран проекта, указанный в файле SETTINGS.xml и создаст файл func\_table.xlsx - таблицу объектов и их функций (поле in\_FUNCTION) для построения попапов.

Таким образом можно быстро будет понять, какой объект использует какую функцию для построения своего попапа.



Рисунок

### Генератор объектов



Рисунок

Данная опция позволяет создавать типовые объекты на HMI форме и расключать их на нужные теги.

Для этого нам нужно выбрать сначала .xml файл с типами, затем .csv файл с объектами.

XML файл с типами имеет формат:

|  |
| --- |
| <types>  <object ...  <object ...  ...  </types> |

Для его создания для начала создайте в Alpha.HMI любую форму, затем перетащите нужные мнемознаки на форму и расключите их свойство in\_Alias на какой-то алиас, например на:

|  |
| --- |
| unit.Global.RootTag + “.USO1.Test” |

После этого найдите созданную сохраненную форму внутри папки objects HMI проекта (она будет иметь расширение .omobj), откройте ее текстовым редактором и найдите узлы <object … соответствующие вашим добавленным объектам.

Скопируйте все нужные узлы <object … внутрь узлов <types> документа types.xml.

На этом создание XML файла с типами завершено.

Пример узла object мнемознака SHU\_V:

|  |
| --- |
| <object access-modifier="private" name="SHU\_V" display-name="SHU\_V" uuid="68157a60-fd21-4f20-9957-6393dc4dcc8f" base-type="SHU\_V" ...  ...  </object> |

Следом нам нужно создать .csv файл формата:

<Тип объекта, Имя, Тег, Расположение (помещение)>

Например:

|  |
| --- |
| SHU\_V,ША-П4; В4.1,.USO2.Vent.P4\_V4\_1,Венткамера (7019)  SHU\_V,ША-П2; В2.1; В2.2; В4.3,.USO2.Vent.P2\_V2\_1\_V2\_2\_V4\_3,Венткамера (7019)  SHU\_V,ША-В1.1,.USO2.Vent.V1\_1,Венткамера (7019)  ... |

Обратите внимание, что csv формат (csv – comma separated values), отделяет атрибуты строки запятыми. Поэтому у вас не должно быть никаких запятых внутри названий помещений или имени установки.

Конвертер создаст на выходе файл FORM\_BIG.omobj, являющийся формой HMI, на которую поместит все объекты из csv файла.

Первый аргумент «Тип объекта» соотносится с названием нужного типа из документа types.xml. Соответственно из types.xml просто скопируется нужный узел object и вставится на форму FORM\_BIG.omobj.

Затем внутри объекта произойдет замена свойства in\_Alias на третий аргумент csv строки – «Тег». Замена происходит по следующему алгоритму:

Если найден знак + в строке in\_Alias объекта, то заменяется весь текст справа после + (потому что обычно алиас выгялдит как unit.Global.RootTag + ".USOX.Obj.Tag"). Если знака + нет, то меняется полностью все поле на «Тег».

Программа постарается размещать все объекты на форме внутри прямоугольников-комнат, сортируя все объекты по расположению.

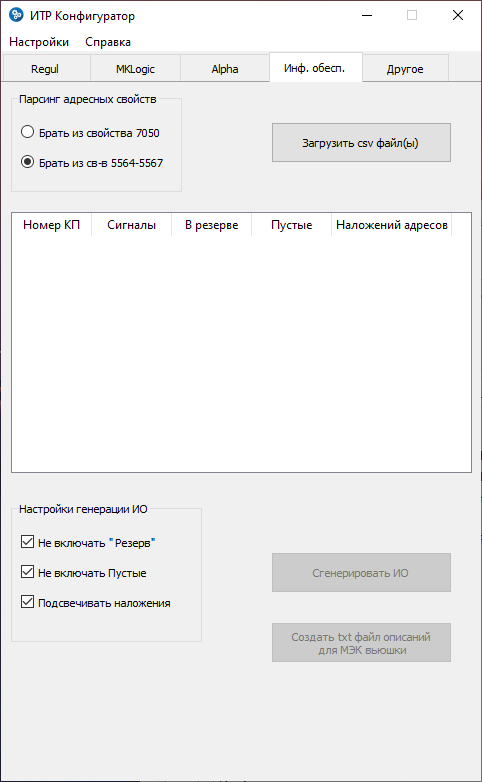
Если среди всех csv строк будет хоть одна строка со скобочками (), то программа будет сортировать объекты строго по номеру помещений. Если же не указывать помещения через скобочки, то сортировка будет просто по текстовому расположению.

Если у объекта будет внутри вложенный текстовый объект "Name", то в его значение подставится «Имя» из второго аргумента csv файла. Обычно для Депо используется.

И последнее.

Если внутри файла с типами types.xml будет хотя бы один тип с названием NoLink, то все объекты из csv файла независимо от их поля «Тип объекта» выгрузятся на форму с типом NoLink.

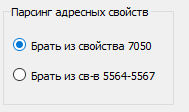
# Вкладка Инф. обесп.



Рисунок

Данная вкладка позволяет по .csv файлам Infinity Server создать файл информационного обеспечения (ИО) для протокола МЭК-104.

Можно получить файл ИО двумя способами, исходя из выбора настройки Парсинг адресных свойств:

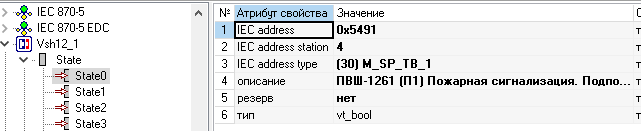


Рисунок

Если выбрана опция «Брать из свойства 7050», то конвертер возьмет атрибуты ProtocolType, Address, Station из свойства 7050. Данное свойство (7050) появляется, если в программе ACS Configurator создать связи между отображениями Infinity Server и IEC 870-5.

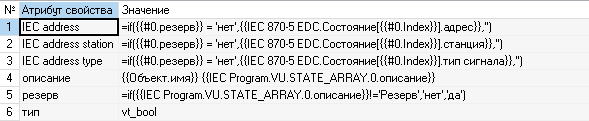
Если же выбрана опция «Брать из св-в 5564-5567», то необходимо везде, где необходимо, добавить свойства для тегов отображения Infinity Server:

* IEC address (5564)
* IEC address type (5565)
* IEC address station (5567)



Рисунок

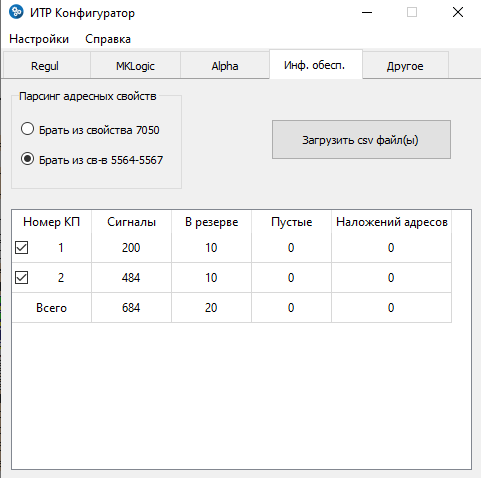
И собрать формулами нужные параметры с других отображений:



Рисунок

Описание сигнала берется со свойства «описание» не зависимо от выбранной настройки «Парсинг адресных свойств».

После загрузки .csv файлов в программу, отобразится таблица найденных сигналов:



Рисунок

Чекбокс напротив номера КП позволяет убрать или добавить станцию в результирующий файл ИО.

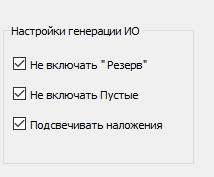
Колонка «Сигналы» показывает общее количество найденных адресных сигналов.

Колонка «В резерве» показывает сколько сигналов имеют описание либо равным слову «Резерв» либо оканчивающиеся на « Резерв».

Колонка «Пустые» показывает сколько сигналов имеют незаполненное поле «описание».

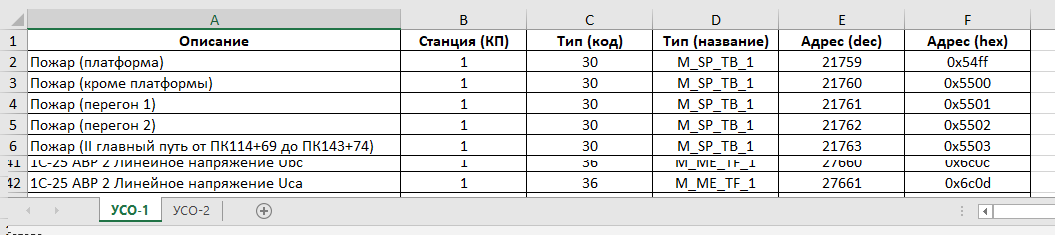
Поле «Наложений адресов» показывает, сколько сигналов внутри данного номера КП (станции) имеют одинаковый адрес и тип протокола.

После этого можно отметить необходимые «Настройки генерации ИО»:



Рисунок

И нажать на кнопку «Сгенерировать ИО». Программа предложит сохранить файл МЭК-104 Инф. обеспечение.xlsx. Который выглядит следующим образом:



Рисунок

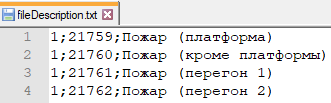
По каждому КП создается своя вкладка УСО-<номер КП>, куда попадают адресные сигналы.

Если была выбрана настройка «Не включать Резерв», в файле ИО будут отсутствовать строчки, которые по таблице – В резерве. То же самое с настройкой «Не включать Пустые».

Если было выбрано «Подсвечивать наложения», но строчки внутри одного листа (номера КП), имеющие одинаковые адрес и тип протокола, будут подсвечены желтым цветом.

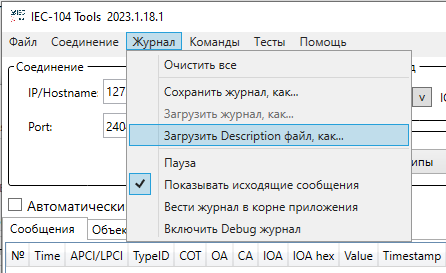
Нажав на кнопку «Создать txt файл описаний для МЭК вьюшки», программа создаст файл fileDescription.txt формата

|  |
| --- |
| номер станции; адрес (десятич); описание |



Рисунок

Данный файл можно затем импортировать в программу IEC-104 Tools.exe, которую создал нам Баублис Дмитрий, пока еще у нас работал (потом ушел зашибать бабки работая программистом у Бикмуллина).



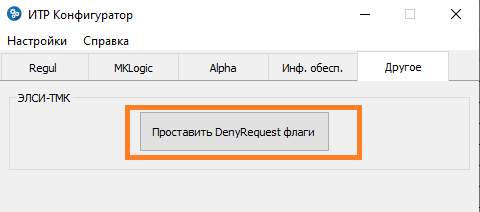
Рисунок

Импортировав файл описаний в программу IEC-104 Tools.exe появляется возможность отслеживать МЭК сигналы более удобно, благодаря подгрузившимся описаниям сигналов.

# Вкладка Другое

## ЭЛСИ-ТМК

### Проставить Deny Request флаги



Рисунок

Данная опция позволяет загрузить один или несколько файлов крейта ЭЛСИ-ТМК.xml, экспортируемых программой ACS Конфигуратор.

И исправить флаги DenyRequest во всех найденных модулях TN713 (то есть взвести их в true).

Потому что несмотря на то, что на всех modbus slave устройствах в канале проставлено одно и то же значение «Запрет выдачи опроса», программа ACS Configurator может рандомно игнорировать эту настройку.

Модифицированный файл крейта сохраняется рядом с загруженным файлом и имеет постфикс \_mod.