

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

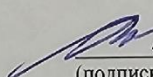
на тему

ОРС-СЕРВЕР ПРОТОКОЛОВ MODBUS
СПЕЦИФИКАЦИИ UNIFIED
ARCHITECTURE

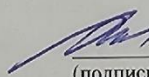
Студент

 06.06.2025 Вартанов Алексей Вартанович
(подпись, дата)

Руководитель

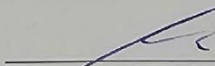
 10.06.2025 Бычков Андрей Станиславович
(подпись, дата)

Нормоконтролёр

 10.06.2025 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол заседания. кафедры от 28.05.2025г. № 16)

Заведующий кафедрой

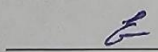

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

16 июня 2025г. № 13)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » 05 20 25 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Вартанов Алексей Вартанович гр.21BBB1 факультета вычислительной техники

2. Тема работы: OPC-сервер протоколов MODBUS спецификации Unified Architecture

Тема утверждена приказом ПГУ № 601/о от 6 мая 2025 по 30 мая 2025.

3. Руководитель бакалаврской работы к.т.н. Бычков А.С.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать OPC UA-сервер протоколов MODBUS, предназначенный для обеспечения взаимодействия между промышленными устройствами, использующими протокол MODBUS TCP, современными системами управления и мониторинга, поддерживающими спецификацию OPC Unified Architecture. Программа должна предоставить клиентам стандартизированный доступ для чтения данных из регистров MODBUS-устройств, а также возможность подписки на изменения этих данных. Сервер должен динамически формировать адресное пространство на основе загружаемой конфигурации, описывающее подключенные MODBUS-устройства и их точки данных(теги).

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы разработки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	32	26.04.2025– 29.05.2025	
2.	Анализ требований	32	29.05.2025– 31.05.2025	
3.	Выбор технологий, применяемых при проектировании и разработке	30	01.05.2025– 04.05.2025	
4.	Описание алгоритмов	60	04.05.2025– 09.05.2025	
5.	Разработка OPC-сервера протоколов MODBUS спецификации Unified Architecture	120	9.05.2025– 23.05.2025	
6.	Оформление пояснительной записки	54	24.05.2025– 31.05.2025	

Дата выдачи задания 6 мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Бычков А.С.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Вартанов А.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
11.06.2025

Реферат

Бакалаврская работа содержит 109 листов, 51 рисунок, 7 источников и 3 приложений.

OPC UA, VISUAL STUDIO 2022, UA EXPERT, MODBUS SLAVE, QT CREATOR

Цель работы – разработать OPC-сервер протоколов MODBUS спецификации Unified Architecture.

Разработанное приложение должно позволять пользователю создавать конфигуратор, в него входят каналы, устройства, группы и теги. В тегах задаётся тип регистра отображаемой информации и формат регистра. При запуске сервера, подключается устройство через протокол TCP/IP, которое в реальном времени обновляет получаемые данные и вывод на интерфейс сервера. Также подразумевается возможность загрузки конфигурации и подключение клиента к серверу. Взаимодействие сервера с устройством должно происходить через интерфейс программы.

Результатом проделанной работы является проект, который удовлетворяет поставленной задаче. А именно, разработано, реализовано и протестировано серверное приложение с подключенным устройством. При разработке было использовано большое количество технологий, к которым относятся: язык программирования C++, среда разработки Microsoft Visual Studio 2022, среда для разработки интерфейса Qt Creator, эмулятор ПЛК MODBUS-slave и эмулятор интерфейса OPC UA client - UA Expert.

					ПГУ 1.090301.04. 001 ПЗ		
	№ докум.	Подпись	Дата				
Разраб.	Вартанов А.В.		10.06.25	OPC-сервер протоколов MODBUS спецификации Unified Architecture Пояснительная записка	Лит.	Лист	Листов
Провер.	Бычков А.С.		10.06.25			2	109
Н. Контр.	Бычков А. С.		10.06.25		ФВТ, каф. ВТ гр. 21BBB1		
Утверд.	Митрохин М.А.		10.06.25				

Содержание

Перечень сокращений и обозначений.....	5
Введение	6
1 OPC-сервер протоколов MODBUS спецификации Unified Architecture	7
1.1 Анализ предметной области	7
1.1.1 Kerware KEPServerEX.....	7
1.1.2 Prosys OPC UA Modbus Server	8
1.1.3 Ignition OPC UA Gateway	10
1.2 Описание потребностей пользователя	12
1.3 Постановка задачи.....	13
1.4 Выбор инструментов разработки	14
1.4.1 Язык программирования C++	14
1.4.2 Microsoft Visual Studio 2022	14
1.4.3 QT creator	15
1.5 Используемые протоколы и библиотеки	15
1.5.1 Стандарт open62541	15
1.5.2 Библиотека парсинга SimpleIni.....	16
1.5.3 Библиотека логирования Plog.....	16
1.5.4 Библиотека преобразования кодировок	17
1.5.5 Modbus TCP	17
1.6 Архитектура приложения	17
1.7 Реализация приложения	20
1.8 Основные компоненты программы	23
1.8.1 Ядро OPC UA - сервера.....	23

1.8.2 Взаимодействие с Modbus.....	31
1.8.3 Интеграция Modbus и OPC UA.....	35
1.8.4 Система логирования	36
1.9 Тестирование и отладка	37
1.9.1 Тестирование серверной части	38
1.9.2 Тестирование клиентской части	52
2 Безопасность жизнедеятельности.....	59
2.1 Вред здоровью при работе с ПК	59
2.2 Снижение вреда для здоровья при работе с ПК	59
2.3 Техника безопасности при работе с компьютером	60
Заключение	61
Список использованных источников	62
Приложение А – UML-диаграммы.....	63
Приложение Б – Листинг программы	68
Приложение В – Презентация	104

Перечень сокращений и обозначений

ОЗУ - оперативное запоминающее устройство

ОС - операционные системы

ПЛК - программируемый логический контроллер

ПК – персональный компьютер

ПО - программное обеспечение

ЦП - центральный процессор

API (Application Programming Interface) - интерфейс программирования приложений

CSV (Comma-Separated Values) - текстовый формат, предназначенный для представления табличных данных

GUI (Graphical User Interface) - графический интерфейс пользователя

IP (Internet Protocol address) - уникальный сетевой адрес узла в компьютерной сети, построенной на основе стека протоколов TCP/IP

MES - система управления производственным процессам

MODBUS - открытый коммуникационный протокола MODBUS для передачи данных последовательным линиям связи

MODBUS TCP (Transmission Control Protocol) - вариант протокола MODBUS для передачи данных по сетям TCP/IP

OPC - набор программных технологий, который предоставляет единый интерфейс для управления и обмена данными среди разных устройств

OPC UA (Open Platform Communications Unified Architecture) - унифицированная архитектура открытых платформенных коммуникаций

SCADA (Supervisory Control and Data Acquisition) - система диспетчерского управления и сбора данных

Введение

Современные промышленные системы автоматизации сталкиваются с необходимостью интеграции одного интерфейса для разных устройств и протоколов, что требует не только гибкости, но и соответствия с актуальными требованиями безопасности и технологической независимости. Для взаимодействия с промышленным оборудованием использовались спецификации OPC Data Access (DA) и Historical Data Access (HDA), однако их реализация основана на устаревшей технологии DCOM (Distributed Component Object Model), которая имеет ряд критических ограничений. DCOM работает исключительно в среде операционных систем семейства Windows, что создаёт зависимость от зарубежного ПО и не поддерживает современные стандарты шифрования и аутентификации, что делает его уязвимым для кибератак.

В контексте государственной политики импортозамещения, закреплённой в Постановлении Правительства РФ №123 «О мерах реализации плана перехода на использование отечественного программного обеспечения», актуальным становится переход на отечественные ОС на базе Linux (например, Astra Linux «Альт»), где как раз DCOM и отсутствует, а единственной альтернативой остаётся OPC Unified Architecture.

Протокол Modbus, несмотря на свою распространённость в промышленности, не соответствует современным требованиям безопасности. Разработка OPC UA сервера для Modbus позволит модифицировать устаревшие системы и обеспечить их интеграцию в цифровые системы без зависимости от иностранных технологий.

Целью данной работы заключается в разработке OPC UA сервера для интеграции устройств Modbus, обеспечивающего, в будущем, кроссплатформенность, которая включает себя поддержку отечественных ОС на базе Linux, наличие безопасности данных и соответствие требованиям импортозамещения. Решение может быть использовано как основа для разработки отечественных аналогов зарубежных OPC-серверов.

					<i>ПГУ 1.090301.04.001 ПЗ</i>	Лист
						6
Изм	Лист	№ докум.	Подпись	Дата		

1 OPC-сервер протоколов MODBUS спецификации Unified Architecture

1.1 Анализ предметной области

Создание ПО для промышленной автоматизации требует ясного осознания потребностей пользователей, обследования имеющихся решений и постановки задачи. В этой главе мы сосредоточимся на необходимости обновления устаревших систем, которые используют протокол Modbus. Современные производственные системы нуждаются в обеспечении безопасности, совместимости разных устройств и удобстве использования продукта, что обосновывает наш переход на стандарт OPC UA. Текущие решения обычно страдают от недостаточной гибкости, сложности в настройке или привязанности к определенным ОС. Исходя из определенных проблем, ставится задача создания OPC UA-сервера, который может трансформировать данные Modbus в структурированную модель, обеспечивающую защиту между устройствами и совместимую с современными системами.

В настоящее время на рынке представлен ряд решений, которые обеспечивают интеграцией Modbus -устройств с системами OPC UA. Далее будет приведён анализ их функциональных возможностей, а также их преимущества и недостатки, см. таблицу 1.

Среди аналогов можно выделить:

- kepware KEPSeverEX;
- prosys OPC UA Modbus Server;
- ignition OPC UA Gateway.

1.1.1 Kerpware KEPSeverEX

Kerpware KEPSeverEX – промышленный американский сервер с поддержкой более 150 протоколов, включая Modbus RTU, TCP и Modbus Plus. Позиционируется как универсальное решение для интеграции legacy-оборудования в современные системы управления, см рисунок 1.

					ПГУ 1.090301.04.001 ПЗ	Лист
						7
Изм	Лист	№ докум.	Подпись	Дата		

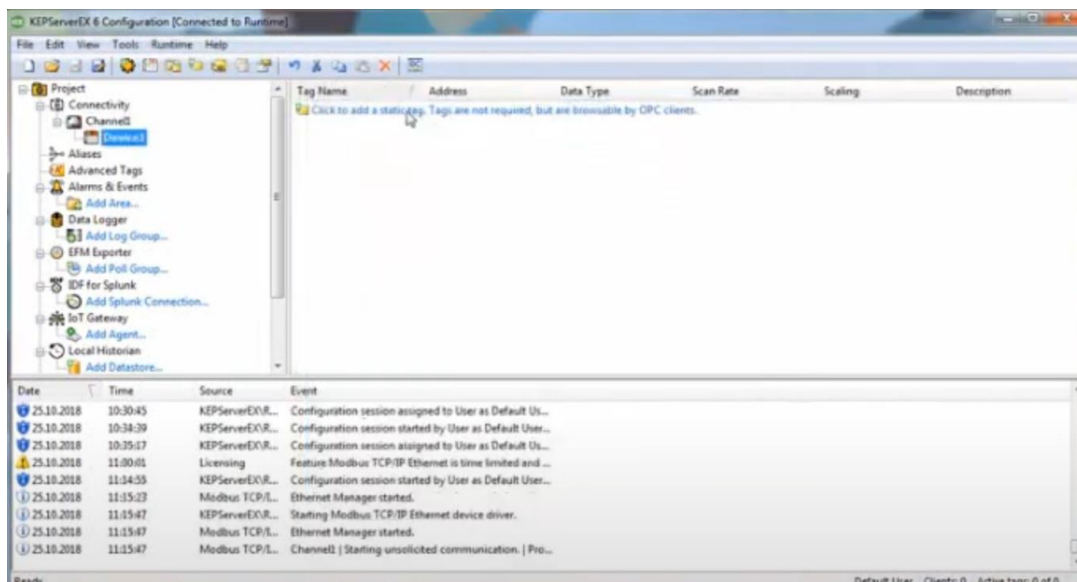


Рисунок 1 – Внешний вид конфигуратора KEPServerEX

Функционал:

- поддержка до 5000 тегов;
- встроенные шаблоны для различных устройств;
- интеграция с облачными платформами;
- преобразование регистров Modbus в орс ua-узлы с настраиваемыми атрибутами (единицы измерения, диапазоны).

Преимущества:

- высокая производительность и отказоустойчивость;
- готовые конфигурации для типовых устройств;
- поддержка орс ua pub/sub для работы в распределенных системах.

Недостатки:

- высокая стоимость лицензии (2000\$);
- сложная настройка нестандартных устройств;
- отсутствие встроенных инструментов семантического обогащения.

1.1.2 Prosys OPC UA Modbus Server

Prosys OPC UA Modbus Server – это облегчённое американское решение для базового преобразования данных Modbus TCP в OPC UA. Ориентирован на малые и средние предприятия, см. рисунок 2.

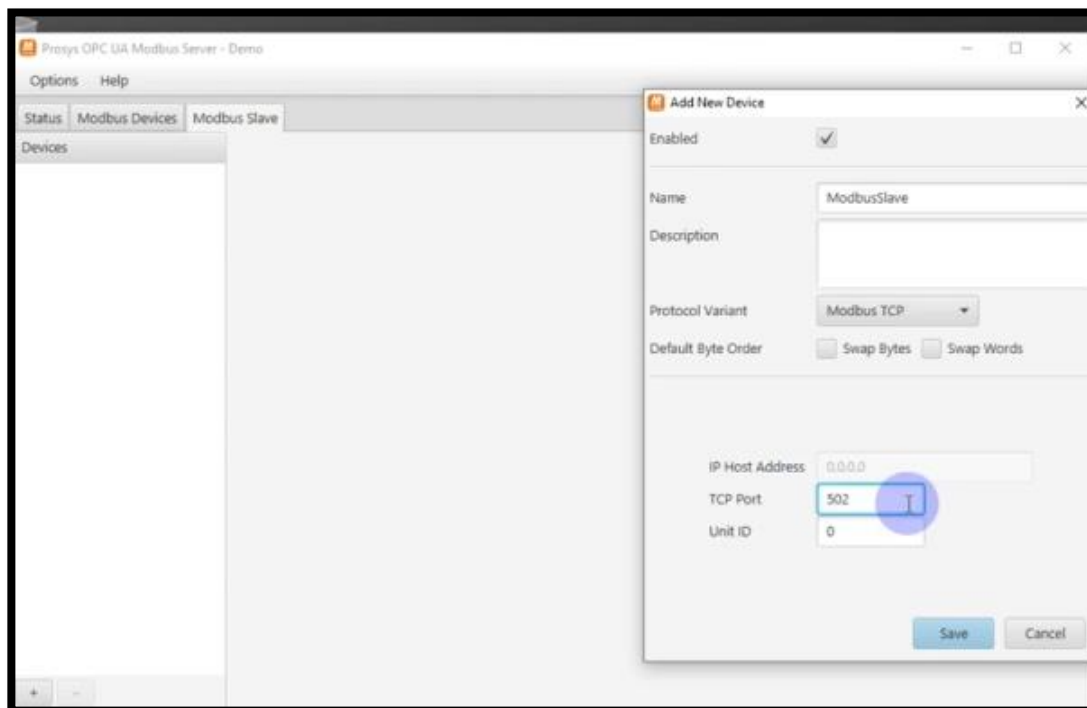


Рисунок 2 – Внешний вид конфигуратора Prosys OPC UA Modbus Server

Функционал:

- маппинг регистров modbus в узлы opc ua через графический интерфейс;
- поддержка до 1000 тегов;
- rest api для удаленного управления конфигурацией.

Преимущества:

- простой интерфейс;
- низкие требования к ресурсам;
- быстрое развертывание;
- присутствует пробная версия с ограничением на 30 тегов.

Недостатки:

- отсутствие поддержки modbus rtu;
- нет встроенной безопасности в бесплатной версии;
- ограниченная документация для нестандартных сценариев.

Изм	Лист	№ докум.	Подпись	Дата

1.1.3 Ignition OPC UA Gateway

Ignition OPC UA Gateway – европейская SCADA-платформа со встроенным OPC UA-сервером и Modbus-драйвером. Позиционируется как решение для цифровых двойников и Industry 4.0, см. рисунок 3.

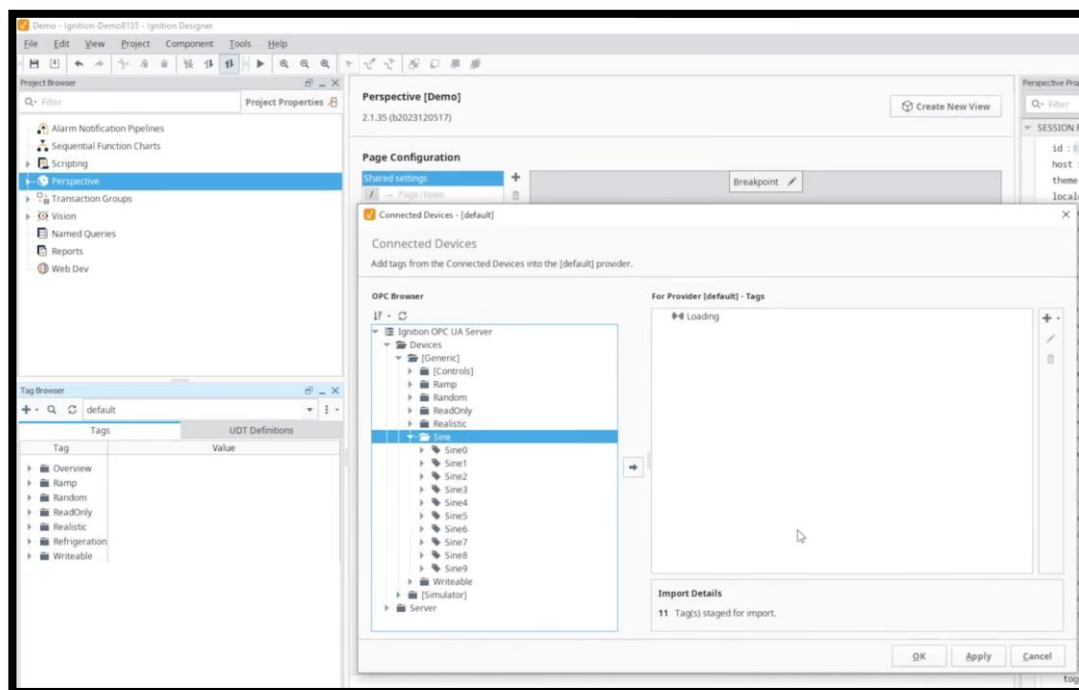


Рисунок 3 – Внешний вид конфигуратора Ignition OPC UA Gateway

Функционал:

- поддержка modbus rtu/tcp;
- визуальное конфигурирование связей между устройствами и тегами;
- встроенная база данных для хранения исторических данных.

Преимущества:

- гибкая модель подписок (по изменению значения, времени или событию);
- поддержка пользовательских скриптов на python;
- присутствует бесплатная пробная версия для ознакомления.

Недостатки:

- высокая цена (от 5000\$ за сервер);
- требуется установка полноценной scada-системы;

– сложность масштабирования для распределенных систем.

Таблица 1.1 – Сравнение функционала систем

Критерий	Аналоги серверов		
	Kepware KepserverEx	Prosys Modbus Server	Ignition Gateway
Поддержка Modbus	RTU, TCP	TCP	RTU, TCP
Кол-во Тегов	5000	1000	1000
Безопасность	TLS, X.509, RBAC	TLS (платная)	RBAC, Аудит-логи
Стоимость	От 2000 долларов	От 500 долларов	От 5000 долларов
Гибкость настройки	Высокая (API, Шаблоны)	Средняя (GUI)	Высокая (Python-скрипты)

Существующие OPC UA – серверы для Modbus решают базовые задачи интеграции, но не покрывают потребности в:

- двустороннем обмене информации (OPC UA в Modbus);
- гибкости обработки данных на уровне сервера;
- в доступности получения лицензии.

Это формирует потребность в разработке специализированного решения, сочетающего производительность промышленных продуктов с возможностями кастомизации открытых платформ.

1.2 Описание потребностей пользователя

Разработка OPC UA-сервера для протокола Modbus ориентирована на решение ключевых задач, возникающих у современных промышленных систем. Переход к цифровым технологиям по стандартам «Industry 4.0» требует от предприятий интеграции разнородных систем, включая устаревшее оборудование, при этом сохраняя высокий уровень безопасности, гибкости и масштабируемости.

Представим группы пользователей и их потребности. К целевым пользователям можно отнести промышленные предприятия, которые требуют:

- интеграцию оборудования с протоколом Modbus (ПЛК, датчики, приводы) в единую систему управления (MES, SCADA);
- минимизацию затрат на модернизацию инфраструктуры без замены работоспособного оборудования;
- обеспечение совместимости с современными стандартами для участия в цифровых цепочках поставок.

Ещё можно упомянуть потребности у разработчиков SCADA-систем и других аналитических платформ:

- стандартизированный интерфейс для работы с данными, вместо реализации поддержки Modbus для каждого устройства;
- возможность получать структурированные данные с метаданными (единица измерения, диапазоны значений, описания тегов);
- доступ к данным и событиям через OPC UA-механизмы (журнал аварийных остановок, логи).

Нужно упомянуть и про потребности в наличии безопасности:

- потребность в защите промышленных сетей от несанкционированного доступа к критически важным устройствам;
- шифрование данных при передаче между оборудованием и системами управления.

Таким образом, OPC UA-сервер для Modbus закрывает критически важный

					ПГУ 1.090301.04.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

разрыв между устаревшими промышленными протоколами и требованиями современной цифровой инфраструктуры, обеспечивая предприятиями плавный переход к технологиям будущего без потери инвестиций в действующее оборудование.

1.3 Постановка задачи

При постановке задачи перед проектом стоит учитывать ряд факторов и требований. В рамках данного проекта необходимо реализовать серверное приложение OPC UA-сервер протоколов MODBUS. Важно обеспечить не только функциональность, но и максимальную простоту использования, а также максимальную гибкость настройки, чтобы удовлетворить потребности различных категорий пользователей сервера. Для реализации задачи поставим следующие задачи:

- разработать поддержку протокола Modbus TCP через сети Ethernet;
- реализация преобразование данных, то есть создание специальных регистров с метаданными, в которых включены единицы измерения и его описание, а также последующий маппинг регистров в иерархические объекты;
- реализация динамического управление узлами, то есть возможность загрузки/сохранения конфигурации (в него включены каналы, группы и теги) из внешних файлов;
- обеспечение базовой безопасности для устройств, а для этого будет режим безопасности логина/пароля, на сервере будет храниться БД о логине и пароле пользователя, который сможет подключиться к серверу.

Для достижения поставленных целей и задач необходимо провести комплекс работ, включающий в себя анализ требований, проектирование программной части устройства, реализацию функционала, тестирование. Важно отметить, что успешная реализация поставленных задач требует не только высокой квалификации и опыта в области разработки, но и внимательного анализа потребностей пользователей и особенностей эксплуатационных условий. Таким образом можно создать приложение, которое будет надежно и

					ПГУ 1.090301.04.001 ПЗ	Лист
						13
Изм	Лист	№ докум.	Подпись	Дата		

эффективно выполнять свои функции в любых условиях использования. Но для начала нужно определиться с выбором инструментов разработки, которые помогут нам реализовать поставленные задачи.

1.4 Выбор инструментов разработки

Процесс выбора инструментов разработки для данного проекта является обязательным этапом при его создании. В данном случае были тщательно проанализированы доступные инструменты, учтены их функциональность, гибкость и соответствие требованиям проекта. После проведенного анализа были выбраны следующие инструменты, которые порекомендовали себя с лучшей стороны и подходят для разработки данного проекта:

1.4.1 Язык программирования C++

C++ - является кроссплатформенным, компилируемым, статически типизированным языком программирования общего назначения, который поддерживает объектно-ориентированное и процедурное программирование. Он предназначен для создания высокопроизводительных приложений, включая системное программирование, встраиваемые системы, драйверы устройств и промышленные решения. C++ сочетает в себе низкоуровневый контроль над аппаратными ресурсами и мощные абстракции высокого уровня, что делает его идеальным выбором для задач, требующих максимальной производительности и гибкости.

1.4.2 Microsoft Visual Studio 2022

Microsoft Visual Studio – это мощная интегрированная среда разработки (IDE) от Microsoft, предназначенная для разработки приложений на различных языках программирования, включая C++. Благодаря этой среде разработки интегрируем библиотеку open 62541, настройки безопасности, работу с COM и TCP портами и реализация кроссплатформенных решений на базе Qt Framework.

1.4.3 QT creator

Qt Creator — это кроссплатформенная интегрированная среда разработки, созданная для работы с фреймворком Qt. Этот популярный фреймворк используется для разработки графических пользовательских интерфейсов (GUI) и кроссплатформенных приложений на C++.

1.5 Используемые протоколы и библиотеки

1.5.1 Стандарт open62541

Open62541 – это открытая реализация стандарта OPC UA, разработанная на языке C. Библиотека предоставляет полный набор инструментов для создания OPC UA-серверов и клиентов, включая безопасность, подписки, методы, исторические данные и другие механизмы стандарта.

Основные возможности стандарта open62541:

- библиотека поддерживает все ключевые аспекты стандарта OPC UA, такие как работа с узлами, подписки на данные, вызов удалённых методов, исторические данные;
- поддерживает кроссплатформенность, что соответствует требованиям проекта к работе в промышленных ПК (Windows) и встраиваемых системах (Linux);
- поддержка безопасности, что подразумевает реализацию сертификатов X.509 и шифрование TLS/SSL;
- наличие производительности, что подразумевает о наличии низкого уровня абстракции, которая позволяет оптимизировать производительность, что критично для работы с промышленными устройствами через Modbus TCP.

Основной функционал стандарта open62541:

- создание OPC UA-узлов, например, объекты (UA_Server_addObjectNode), переменных (UA_Server_addVariableNode) и методов (UA_Server_addMethodNode) для представления промышленных устройств (каналов, устройств, группу и тегов);

- специальная подписка на данные устройств. Использование UA_Client_MonitoredItems_createDataChanges для отслеживания изменений значений тегов в реальном времени;

- безопасность. Настройка и управления доверенными списками пользователей;

- логи. Интеграция с библиотекой plog для диагностики и отладки.

1.5.2 Библиотека парсинга SimpleIni

SimpleIni – это лёгкая кроссплатформенная библиотека для работы с конфигурационными файлами в формате “Ini”. В нашем проекте она используется для чтения и записи настроек OPC UA-сервера и клиента, включая параметры безопасности (анонимный режим или режим доверенных пользователей, а также другие параметры. Библиотека позволяет удобно работать с текстовыми конфигурационными файлами, разделяя данные на секции и ключи.

1.5.3 Библиотека логирования Plog

Plog (Portable Log) – это кроссплатформенная библиотека логирования на C++, которая используется для записи диагностических сообщений, ошибок и информации о работе OPC UA-сервера и клиента. Она обеспечивает гибкость, простоту интеграции и поддержку различных уровней детализации логов.

Основные возможности Plog в проекте:

- plog поддерживает Windows, Linux и macOS, что соответствует требованиям проекта к работе на промышленных ПК и встраиваемых системах;

- поддерживаются несколько уровней логирования, такие как none, fatal, error, warning, info, debug, verbose;

- запись логов в файлы и консоль. Логи сохраняются с возможностью ограничения размера файла и количества архивов;

- интеграция с OPC UA-библиотекой, в проекте реализован пользовательский логгер PlogLogger, совместимый с open62541;

– используется TxtFormatter для добавления временных меток, уровней логирования и других метаданных.

1.5.4 Библиотека преобразования кодировок

Iconv – это кроссплатформенная библиотека для преобразования строк между различными кодировками, включая UTF-16, UTF-8 и другие. В нашем случае используется для конвертации строк из формата UTF-16 в UTF-8, что критично для работы с логами, конфигурациями и данными OPC UA-сервера/клиента.

Основные возможности Iconv в проекте:

- utf-16 используется для строк на Windows, а UTF-8 на Linux (работает по умолчанию), Iconv обеспечивает совместимость между этими платформами;
- iconv поддерживает преобразование между десятками кодировок, что важно для работы с локализованными строками (например, на русском языке);
- позволяет обрабатывать данные на уровне байтов, что критично для работы с OPC UA-библиотекой(open62541) которая использует UTF-8.

1.5.5 Modbus TCP

Modbus – это открытый промышленный протокол связи, используемый для обмена данными между устройствами в системах автоматизации. В нашем проекте реализована версия Modbus TCP, а это подразумевается об интегрировании в OPC UA-сервер через класс CPortIO_TCP, который обеспечивает взаимодействие с промышленным оборудованием через OPC UA-интерфейс.

1.6 Архитектура приложения

Успешная разработка во многом определяется выбранной архитектурой. Архитектура приложения определяет его основные компоненты, их взаимодействие и принципы организации. В данной главе будет описан выбор архитектурного подхода для разрабатываемого сервера. При проектировании OPC UA-сервера для Modbus TCP были приняты во внимание следующие основополагающие архитектурные принципы

					<i>ПГУ 1.090301.04.001 ПЗ</i>	Лист
						17
Изм	Лист	№ докум.	Подпись	Дата		

для обеспечения надежности и удобства сопровождения системы:

- модульность. Система разбивается на ряд слабо связанных между собой модулей, каждый из которых отвечает за определенную часть функциональности. Это упрощает разработку, тестирование и модификацию отдельных компонентов без значительного влияния на остальную систему;

- слоистость. Функциональность приложения разделяется на несколько логических слоев. Каждый слой предоставляет определенный набор сервисов вышележащему слою и использует сервисы нижележащего. Таким образом можно разделить задачи для нашей программы;

- сервисная ориентированность. Сам OPC UA является сервисно-ориентированной архитектурой (SOA), принципы предоставления четко определённых сервисов (как внутренних, так и внешних были учтены при проектировании внутренних компонентов сервера;

- управляемость событиями. Для эффективной обработки асинхронных операций, таких как получение данных от Modbus TCP устройств или уведомление OPC UA клиентов по подписке, архитектура должна поддерживать событийные механизмы. На основе эти принципов для разрабатываемого OPC UA Modbus TCP сервера была выбрана многослойная модульная архитектура, см рисунок 4.

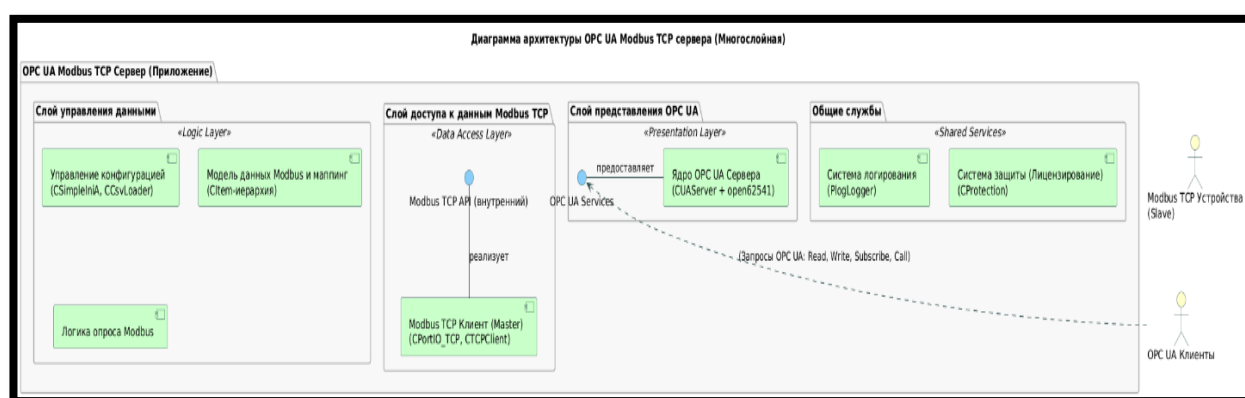


Рисунок 4 - UML - диаграмма многослойной архитектуры

Эта архитектура включает следующие основные логические слои:

– слой доступа к данным Modbus TCP. Этот слой отвечает за все низкоуровневое взаимодействие с Modbus TCP Slave устройствами. Он инкапсулирует детали протокола Modbus TCP. К основным задачам относятся установление и поддержка TCP-соединений, формирование Modbus-запросов, отправка запросов, прием ответов, разборов ответов и проверка ошибок на уровне протоколов Modbus. К компонентам, которые реализуют функциональность будем относить CPortIO_TCP и CTPClient, а также логика формирования/разбора Modbus-пакетов, которая в нашей реализации распределена по методам классов управления данными (CChannel, CDevice);

– слой управления данными. Этот слой является ядром, связывающий специфику Modbus с обобщенными представлениями OPC UA. Он управляет внутренней моделью данных, конфигурацией, логикой опроса и преобразованием данных. Основной её задачей является представлением Modbus-устройств, каналов, групп и тегов в виде иерархической структуры объектов. А также реализация логики опроса Modbus-устройств и отображение данных из регистров Modbus на типы данных OPC UA. К ключевым классам относится Citem и его наследники (CChannel, CDevice, CGroup, CDATag), CCsvLoader (для конфигурации Modbus) и CSimpleInuA (для параметров сервера);

– слой представления OPC UA. Этот слой реализует всю функциональность OPC UA сервера, предоставляя внешний интерфейс для OPC UA клиентов. Он использует библиотеку open62541, ключевым классом является CUAServer. К основной задаче относится инициализация и управление жизненными циклами OPC UA-сервера, создание и менеджмент адресного пространства OPC UA, а также и отражающие данные из слоя управления данными и обработка запросов от OPC UA клиентов (чтение, запись, подписка, вызов методов).

1.7 Реализация приложения

В этой части будет представлен процесс реализации нашего OPC UA-сервера, который обеспечивает интеграцию устройств Modbus в систему OPC UA. Решение позволяет промышленным предприятиям мониторить данные с датчиков в реальном времени, а также управлять оборудованием через запись значений в регистры. Программа при это может динамически адаптировать систему к изменениям инфраструктуры.

Эта программа представляет собой сервер, который предназначен для сбора данных с промышленных устройств, работающих в нашем случае по протоколу TCP, и предоставления этих данных OPC UA клиентам в соответствии со спецификацией OPC Unified Architecture.

Рассмотрим общие принципы функционирования программы:

– запуск и инициализация (см. рисунок 5). При запуске сервер считывает конфигурационные файлы. Один файл, в нашем случае формат `cfg`, определяет параметры OPC UA-сервера, такие как сетевой порт, настройки логов, а также данные для аутентификации пользователей. Другой конфигурационный файл, в нашем случае `.csv`, который загружается через `CCsvLoader`, описывает иерархию Modbus-устройств. К этой иерархии относят канал связи (TCP-соединение), сами устройства (их Modbus-адреса), группы тегов и отдельные теги (регистры Modbus, их адреса, типы данных и права доступа). Сервер строит адресное пространство OPC UA. Эта иерархическая структура отражает настроенные каналы, устройства, группы и теги Modbus. Каждый тег Modbus становится переменной (Node) в OPC сервере. После этого сервер запускает слушающий сокет OPC UA соединений и готов принимать подключение от клиентов;

– взаимодействие с Modbus-устройствами (см. рисунок 6). Сервер периодически опрашивает сконфигурированные Modbus -устройства. Логика опроса реализована внутри классов, представляющих Modbus - сущности (`CChannel`, `CDevice`, `CGroup`). Полученные от Modbus - устройств данные (значение регистров) сохраняются и обновляются во внутренних структурах

данных сервера, связанных с соответствующими тегами (CDATag). Сервер также обрабатывает ошибки связи с Modbus – устройствами и может отражать их в качестве соответствующих OPC UA переменных;

– взаимодействие с Modbus - устройствами (см. рисунок 7). OPC UA-клиенты могут обнаруживать сервер в сети и подключаться к нему, используя одну из предоставленных конечных точек, которые могут отличаться методами аутентификации (анонимный доступ, имя пользователя/пароль). После подключения клиент может просматривать адресное пространство сервера, чтобы обнаружить доступные теги Modbus. Клиент может читать текущие значения тегов. При этом OPC UA-сервер через механизм DataSource обращается к соответствующей функции readNodeValue, которая либо возвращает последнее прочитанное с Modbus-устройства значение, либо будет инициировать новый опрос Modbus-устройства. Клиенту даётся возможность вызывать методы OPC UA, предоставляемые сервером для удаленного управления работой сервера или конфигурацией;

– безопасность. Поддержка разных уровней доступа, таких как открытый доступ и доступ по авторизации. Режим открытого доступа обозначает, что каждый пользователь может подключиться к серверу без какой-либо авторизации. Режим доступа по авторизации обозначает, что на стороне сервера создаётся список пользователей, которые могут подключиться к серверу. В список входит логин и пароль пользователя. Пользователь при подключении к серверу должен ввести соответствующие данные, чтобы подключиться к серверу;

– управление работой сервера. Возможность запуска и остановки процесса опроса Modbus - устройств через OPC UA-методы (режим мониторинга), это относится к функциям startExchange, stopExchange. Присутствует введение журнала событий и ошибок работы сервера для диагностики и мониторинга, это модуль PlogLogger. Этот модуль отвечает за запись логов в файл/на консоль и поддержку различных уровней логирования (Debug, Info, Warning, Error).

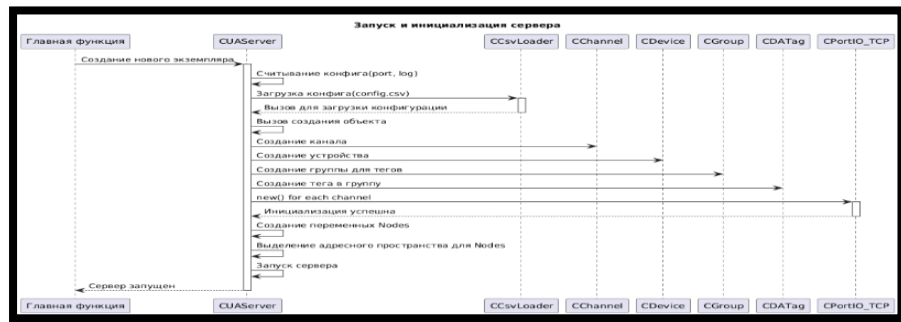


Рисунок 5 – UML - диаграмма последовательности запуска и инициализации сервера



Рисунок 6 – UML-диаграмма последовательности чтение тега клиентом OPC UA

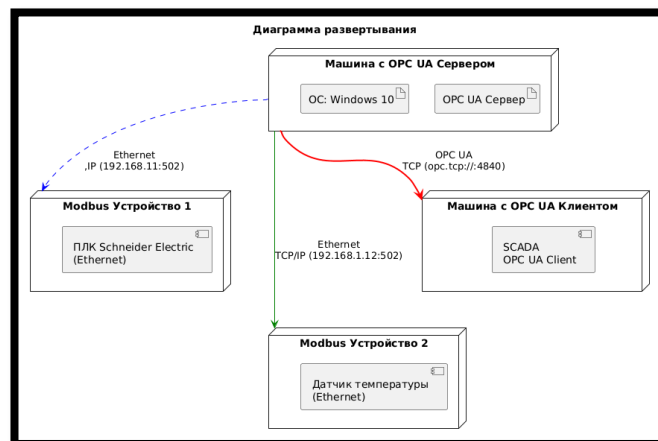


Рисунок 7 – Диаграмма развёртывания Modbus-устройств

1.8 Основные компоненты программы

Программное приложение состоит из следующих основных компонентов:

1.8.1 Ядро OPC UA - сервера

Это центральный компонент, который отвечает за всю функциональность OPC UA. Он управляет адресным пространством, обрабатывает клиентские запросы (чтение, запись, подписки, вызов методов), обеспечивает безопасность соединений и управляет сессиями клиентов.

Следующий блок кода демонстрирует первоначальную настройку сервера OPC UA, включая определение порта, подключение системы логирования и установку ограничений на количество подписок и отслеживаемых элементов. Важным шагом является создание пользовательского пространства имен, которое будет содержать все узлы, специфичные для нашего Modbus - сервера.

Для активации сервера используем функцию UA_Server_run, после чего сервер начинает принимать клиентские подключения и обрабатывать их в соответствии с настроенной логикой, см. листинг 1.

Листинг 1. Метод UA_StatusCode Create(). Инициализация и конфигурация сервера:

```
UA_StatusCode Create()
{
    UA_ServerConfig config = {0}; // инициализация структуры
    конфигурации сервера
    UA_StatusCode retval = UA_ServerConfig_setMinimal(&config,
m_port, NULL);
    if (retval != UA_STATUSCODE_GOOD)
        return retval; // проверка на ошибку
    config.context = this;
    config.logging = &PlogLogger;
    config.eventLoop->logger = &PlogLogger;
    config.securityPolicies->logger = &PlogLogger;
    config.monitoredItemRegisterCallback = mIRegisterCallback;
    config.maxMonitoredItems = 1000; // макс. кол-во
```

отслеживаемых элементов

```
config.maxSubscriptions = 1000; // макс. кол-во подписок
...
Server = UA_server_newWithConfig(&config); // Создание
экземпляра OPC UA-сервера с заданной конфигурацией
...
return UA_Server_run(server, &running); }
```

Следующий блок программы относится к адресному пространству OPC UA — это иерархическая структура узлов (объектов, переменных, методов), которую видит клиент. Этот компонент отвечает за создание и организацию этой структуры, отображая в ней сущности Modbus. Следующий фрагмент показывает, как создается иерархия в OPC UA. В отрывке всё начинается с корневой папки «Каналы», а затем рекурсивно вызывается CreateUAItems для построения остальной структуры (каналов, устройств, групп тегов) на основе нашей внутренней модели данных (CItem), см. листинг 2.

Листинг 2. Функция CreateNodes(). Создание корневого узла и иерархии:

```
void CreateNodes()
{
    CItem *pItem = &gGlobal.m_root;
    std::string rootnodename = "Каналы";
    UA_ObjectAttributes          objectAttr          =
UA_ObjectAttributes_default;
    // добавить верхний уровень "Каналы"
    UA_NodeId_init(&m_rootFolderNodeId);
    UA_NodeId      NodeId      =      UA_NODEID_STRING_ALLOC(m_ns,
rootnodename.c_str());
    UA_QualifiedName OpQName = UA_QUALIFIEDNAME_ALLOC(m_ns,
rootnodename.c_str());
    UA_Server_addObjectNode(          server,          NodeId,
UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER), UA_NODEID_NUMERIC(0,
UA_NS0ID_ORGANIZES), OpQName,          UA_NODEID_NUMERIC(0,
UA_NS0ID_FOLDERTYPE), objectAttr, NULL, &m_rootFolderNodeId);
    pItem->m_NodeId = m_rootFolderNodeId;
```

Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист
24

```

        m_ItemsMap[rootnodename] = pItem;
        UA_QualifiedName_clear(&OpQName);
        UA_NodeId_clear(&NodeId);
        MaxTagNodes = protector.Init();
        if (MaxTagNodes > 30)
        { protector.StartChecking();
          std::string str = "MaxTagNodes - " +
std::to_string(MaxTagNodes);
          printf(""); // типа flush
          UA_LOG_INFO(&PlogLogger, UA_LOGCATEGORY_USERLAND,
str.c_str());
        } TagNodesCount = 0;
        for (imItem it = pItem->begin(), it_end = pItem->end();
it != it_end; ++it)
        {CreateUAItems(it->second, &m_rootFolderNodeId);
        }
}

```

Третий отрывок кода отвечает за создание узлов - переменных для тегов Modbus. Для каждого тега Modbus (CDATag) создаётся узел - переменная в OPC UA. Устанавливается dataType (тип данных OPC UA), accessLevel (права доступа), а также привязывается UA_DataSource. Это позволяет серверу динамически получать значения от Modbus (через readNodeValue) и записывать их (через WriteNodeValue), а не хранить статичные значения, см. листинг 3.

Листинг 3. Отрывок булевой функции CreateUAItems. Создание узлов-переменных для тегов Modbus:

```

bool CreateUAItems(CItem *pItem, UA_NodeId *pParentNode)
{ ...
if(pItem->m_ItemType == IT_TAG)
{
    TagNodesCount++;
    if (TagNodesCount <= MaxTagNodes)
    {
        std::string strFullItemName;

```

```

        br = pItem->GetFullName(strFullItemName);
        // Нода
        UA_NodeId attributeNode;
        UA_NodeId_init(&attributeNode);
        CBaseTag*          DAtag          =
reinterpret_cast<CBaseTag*>(pItem);
        // Атрибуты ноды
        UA_VariableAttributes attributeAttr =
UA_VariableAttributes_default;
        attributeAttr.displayName =
UA_LOCALIZEDTEXT_ALLOC("ru-RU", strFullItemName.c_str());
        attributeAttr.description =
UA_LOCALIZEDTEXT_ALLOC("ru-RU", strFullItemName.c_str());
        // Доступ к чтению и записи
        UA_Byte accessLevel = DAtag->m_accessRights;
        attributeAttr.accessLevel = accessLevel;
        attributeAttr.userAccessLevel = accessLevel;
        // Запись значения атрибута
        attributeAttr.dataType = DAtag->m_nativeType >typeId;
        UA_StatusCode quality;
        DAtag->GetValue(attributeAttr.value, quality);
        UA_NodeId NodeId = UA_NODEID_STRING_ALLOC(m_ns,
strFullItemName.c_str());
        UA_QualifiedName      QName      =
UA_QUALIFIEDNAME_ALLOC(m_ns, strFullItemName.c_str());
        UA_DataSource nodeDataSource = { 0 };
        nodeDataSource.read = readNodeValue;
        nodeDataSource.write = writeNodeValue;
        //auto          start          =
std::chrono::system_clock::now();
        UA_StatusCode      res      =
UA_Server_addDataSourceVariableNode(server, NodeId, *pParentNode,
        UA_NODEID_NUMERIC(0,
UA_NS0ID_HASCOMPONENT), QName,
        UA_NODEID_NUMERIC(0,

```

```

UA_NS0ID_BASEDATAVARIABLETYPE),    attributeAttr,    nodeDataSource,
DTag, &attributeNode);
... }

```

Следующий отрывок определяет как сервер реагирует на запросы клиентов на запись/чтение данных. Механизм DataSource является ключевым для связи OPC UA переменных с реальными данными из Modbus, см. листинг 4.

Листинг 4. Обработка клиентских запросов (Data Access, DataSource):

```

static    UA_StatusCode    writeNodeValueService(UA_Server*
server, const UA_NodeId* sessionId, void* sessionContext,
        const UA_NodeId* nodeId, void* nodeContext,
        const UA_NumericRange* range, const UA_DataValue* data)
{CDATag* pTag = reinterpret_cast<CDATag*>(nodeContext);
    UA_Variant val;
    UA_Variant_init(&val);
    UA_Variant_copy(&data->value, &val);
    int Error = 0;
    unsigned int Count = 1;
    pTag->m_pParent->Write(nodeContext, &val, &Error);
    UA_Variant_clear(&val);
    return UA_STATUSCODE_GOOD;
}

```

При записи значения на стороне клиента, эта функция получает новое значение (data → value), преобразует его (если необходимо) и инициирует операцию записи в соответствующий регистр Modbus-устройства через иерархию объектов (CDATag → ... → CPortIO_TCP).

Следующим идёт реализация OPC UA методов, что позволяет клиентам вызывать predetermined функции на сервере, то есть запускать выбранные конфигурации для работы с сервером. В следующем отрывке объявляется метод, в котором определяются его атрибуты (имя, описание), входные и выходные аргументы, а затем он добавляется в адресное пространство и связывается с конкретной функцией C++, как пример addNode, которая будет выполнена при вызове метода, см. листинг 5.

					ПГУ 1.090301.04.001 ПЗ	Лист
						27
Изм	Лист	№ докум.	Подпись	Дата		

Листинг 5. Добавление узла метода:

```
bool CreateMethods()
{
    UA_StatusCode res = UA_STATUSCODE_BAD;
    UA_Argument inputArgument;
    UA_Argument_init(&inputArgument);
    inputArgument.description = UA_LOCALIZEDTEXT("en-US",
"String");
    inputArgument.name = UA_STRING("MyInput");
    inputArgument.dataType = UA_TYPES[UA_TYPES_STRING].typeId;
    inputArgument.valueRank = UA_VALUERANK_SCALAR;

    UA_Argument outputArgument;
    UA_Argument_init(&outputArgument);
    outputArgument.description = UA_LOCALIZEDTEXT("en-US",
"String");
    outputArgument.name = UA_STRING("MyOutput");
    outputArgument.dataType = UA_TYPES[UA_TYPES_STRING].typeId;
    outputArgument.valueRank = UA_VALUERANK_SCALAR;

    UA_MethodAttributes attrs = UA_MethodAttributes_default;
    attrs.description = UA_LOCALIZEDTEXT("ru-RU", "Начать
мониторинг");
    attrs.displayName = UA_LOCALIZEDTEXT("en-US",
"startExchange");
    attrs.executable = true;
    attrs.userExecutable = true;
    UA_NodeId parentNode =
UA_NODEID_STRING_ALLOC(uaserver.m_ns, "Каналы");
    UA_NodeId OpNodeIdStart =
UA_NODEID_STRING_ALLOC(uaserver.m_ns, "Каналы.startExchange");
    UA_QualifiedName OpQNameStart =
UA_QUALIFIEDNAME_ALLOC(uaserver.m_ns, "Каналы.startExchange");
    res = UA_Server_addMethodNode(uaserver.server,
OpNodeIdStart,
```

Изм	Лист	№ докум.	Подпись	Дата

```

        parentNode,                                     UA_NODEID_NUMERIC(0,
UA_NS0ID_HASCOMPONENT),
        OpQNameStart, attrs, &startExchange,
        1, &inputArgument, 1, &outputArgument, NULL, NULL);

if (res != UA_STATUSCODE_GOOD)
{
    UA_LOG_WARNING(&PlogLogger,    UA_LOGCATEGORY_USERLAND,
"Failed to add Каналы.startExchange");
}

```

Листинг 6. Реализация функции-обработчика метода addNode:

```

static UA_StatusCode addNode(UA_Server* server,
    const UA_NodeId* sessionId, void* sessionHandle,
    const UA_NodeId* methodId, void* methodContext,
    const UA_NodeId* objectId, void* objectContext,
    size_t inputSize, const UA_Variant* input,
    size_t outputSize, UA_Variant* output)
{
    UA_String* inputStr = (UA_String*)input->data;
    UA_String outputStr;
    if (inputStr->length > 0)
    {
        // Конверт из UA_String в std::string
        std::string nodeIdString = std::string((const
char*)inputStr->data);
        nodeIdString = nodeIdString.substr(0, inputStr-
>length);

        std::string str;
        std::vector<std::string> tokens;
        std::string token;
        std::istringstream istr(nodeIdString);
        while (std::getline(istr, token, ';'))
        {

```

```

        tokens.push_back(token);
    }
    // Последним параметром может быть \0, строку обрезаем
    if (tokens.size() == 8)
    {
        tokens.push_back("");
    }
    UA_NodeId parentNode =
UA_NODEID_STRING_ALLOC(uaserver.m_ns, tokens[0].c_str());
    if (uaserver.CreateCItem(&parentNode, tokens))
    {
        outputStr = UA_STRING_ALLOC("Success");
    }
    else
    {
        outputStr = UA_STRING_ALLOC("Failed");
    }
    UA_NodeId_clear(&parentNode);
    UA_Variant_setScalarCopy(output, &outputStr,
&UA_TYPES[UA_TYPES_STRING]);
    //UA_String_clear((UA_String*)outputStr);
    UA_LOG_INFO(&PlogLogger, UA_LOGCATEGORY_SERVER,
"addNode was called");
    return UA_STATUSCODE_GOOD;
}

```

Функция-обработчик метода получает входные данные от клиента (строку с параметрами для создания нового узла Modbus). Она парсит эти параметры и вызывает внутренние функции сервера (uaserver. CreateCItem) для выполнения запрошенной операции (создания элемента в модели данных и соответствующего узла в OPC UA). Затем она формирует ответное сообщение и возвращает его клиенту. Аналогично реализованы deleteNode, editNode, startExchange, stopExchange.

1.8.2 Взаимодействие с Modbus

Этот компонент отвечает за всю низкоуровневую работу по обмену данными с Modbus - устройствами. Он абстрагирует детали транспортного TCP/IP уровня от остальной логики нашей программы. Следующий блок мы рассмотрим, который отвечает за взаимодействие через TCP/IP (CportIO_TCP).

Перед каждой операцией записи (а иногда и чтения, если соединение не поддерживается постоянно) проверяется состояние TCP-соединения. Если оно не установлено, предпринимается попытка подключения к Modbus TCP устройству по указанному IP-адресу и порту, см. листинг 7.

Листинг 7. Взаимодействие через TCP/IP, класс CportIO_TCP:

```
bool CPortIO_TCP::CheckConnect()
{
    bool bConnected = false;
    if (!(bConnected = m_TCPClient.IsConnected()))
    {
        // Установка TCP-соединения
        m_TCPClient.SetTimeOut(m_TimeoutConnect);
        m_TCPClient.SetIPAddress(m_IP, m_TCP_Port);
        m_TCPClient.ReAttach();
        UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d:
CChannel::CheckConnect()      TRY      Connect      %d.%d.%d.%d:%d\n",
std::this_thread::get_id(),      (((unsigned char*) (&m_IP))[0]),
(((unsigned char*) (&m_IP))[1]), (((unsigned char*) (&m_IP))[2]),
(((unsigned char*) (&m_IP))[3]), m_TCP_Port);
        int hrConnect = m_TCPClient.Connect(true, true);
        UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d:
Connect %d\n", std::this_thread::get_id(), (hrConnect == S_OK));

        if (hrConnect != S_OK)
        {
            // Не законектились => закрываем сокет
            m_TCPClient.Detach();
        }
        bConnected = m_TCPClient.IsConnected();
    }
}
```

```

    }
    else
    {
//UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"Connected\n")
;

    }
    return bConnected;
}

```

Если TCP-соединение активно, данные читаются из сокета с помощью метода `m_TCPClient.Read()`. Устанавливается общий таймаут операции, см. листинг 8.

Листинг 8. Чтение данных через TCP/IP:

```

int CPortIO_TCP::Read (unsigned char* pRcvBuf, unsigned int*
pNmByteToRead)
{
    int hr = S_FALSE;
    if (m_TCPClient.IsConnected())
    {
        m_TCPClient.SetTimeOut(m_Timeout);
        if (!pRcvBuf || !pNmByteToRead)
        {
//UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d: Read:
Clear buffer\n", GetCurrentThreadId());
            // Чистим входящий буфер
            unsigned char ReadBuf[100] = {0};
            unsigned int NmByteToRead = sizeof(ReadBuf);
            while (m_TCPClient.Read(ReadBuf, &NmByteToRead,
0)==S_OK)
            {
                NmByteToRead = sizeof(ReadBuf);
            }
            hr = S_OK;
        }
    }
    else

```

```

        {
            //UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d: Start
read\n", GetCurrentThreadId());

            hr = m_TCPCClient.Read(pRcvBuf, pNmByteToRead); //
Чтение

            if (hr==S_OK)
            {
                #ifdef _DEBUG
                UA_LOG_TRACE(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d:
CChannel::Read(%d): ", GetCurrentThreadId(), *pNmByteToRead);
                for(unsigned int ui=0; ui<*pNmByteToRead; ui++)
                {
                    UA_LOG_TRACE(&PlogLogger,UA_LOGCATEGORY_USERLAND,"
                                0x%02X",
pRcvBuf[ui]);
                }
                //UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"\n");
                #endif
                //UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d: Stop
read\n", GetCurrentThreadId());
            }
        }
    else
    {
        if (pNmByteToRead)
        {
            *pNmByteToRead = 0;
        }
    }
    return hr;}

```

Дальше идёт функция отправки данных в сокет с помощью m_TCPCClient.Write(), см. листинг 9.

Листинг 9. Запись данных через TCP/IP:

```
int CPortIO_TCP::Write(unsigned char* pSndBuf, unsigned int*
pNmByteToWrite)
{
    int hr = S_FALSE;
    bool bMakeWrite = false;
    if (m_TCPClient.IsConnected())
    {
        bMakeWrite = true;
    }
    else
    {
        if (CheckConnect())
        {
            bMakeWrite = true;
        }
    }
    if (bMakeWrite)
    {
        if (!pSndBuf || !pNmByteToWrite)
        {
            hr = S_OK;
        }
        else
        {
            hr = m_TCPClient.Write(pSndBuf, pNmByteToWrite,
m_Timeout); // Запись в канал
            if (hr==S_OK)
            {
#ifdef _DEBUG
                UA_LOG_TRACE(&PlogLogger,UA_LOGCATEGORY_USERLAND,"%d:
CChannel::Write(%d): ", GetCurrentThreadId(), *pNmByteToWrite);
                for(unsigned int ui=0; ui<*pNmByteToWrite;
ui++)
                {
```

```

        UA_LOG_TRACE(&PlogLogger,UA_LOGCATEGORY_USERLAND,"
0x%02X", pSndBuf[ui]);
    }
    UA_LOG_TRACE(&PlogLogger,UA_LOGCATEGORY_USERLAND,"\n");
    #endif
    }
    }
    }
    return hr;}

```

1.8.3 Интеграция Modbus и OPC UA

Это одна из ключевых задач: реализовать способ отображение Modbus данных в OPC UA. Классы CChannel, CDevice, CGroup, CDATag формируют иерархическую модель:

- crootItem (gGlobal.m_root) является корневым элементом;
- ccChannel представляет канал связи, то есть содержит ссылку на CPortIO_TCP. В параметрах находятся тип протокола, тип канала, таймауты и кол-во попыток. В TCP указывается IP-адрес и порт;
- cDevice представляет из себя Modbus-устройство. К параметрам относятся адрес Modbus (m_Addr), флаги обмен/эмуляции и ограничение на кол-во регистров в запросе;
- cGroup это группа тегов. Позволяет объединить теги с общим периодом опроса(m_UpdateRate);
- cDATag представляет из себя отдельный тег Modbus. К параметрам относятся имя, тип регистра Modbus (m_RegType), адрес регистра (m_Addr), тип значения (m_Valtype), порядок байта (m_ByteOrder), право доступа(чтение/запись), параметры преобразования в эмуляции;
- m_nativeType является типом данных OPC UA, соответствующий этому тегу;
- getValue()/setValue() являются методом для получения/установки значения тега, включая преобразование и обработку качества.

Перейдём теперь к отображению в адресное пространство OPC UA. Как уже

упоминалось, функция `CUAServer::CreateUAItems` рекурсивно проходит по этой иерархии и создает соответствующие узлы OPC UA:

- `cRootItem` → папка “Каналы” (объекты OPC UA);
- `cChannel` → подобъект (папка) внутри родительского элемента;
- `cDevice` → подобъект (папка);
- `cGroup` → подобъект (папка);
- `cDATag` → переменная OPC UA внутри родительской группы/устройства.

Запрос на чтение OPC UA клиент запрашивает значение тега, которое вызывается функцией `CUAServer::readNodeValue`. Эта функция, в свою очередь, может инициировать чтение из Modbus устройства через цепочку вызовов объектов (`CDATag` → `CGroup` → `CDevice` → `CChannel` → `CPortIO_TCP`);

С функцией записи всё аналогично, функция `CUAServer::writeNodeValue` инициирует запись в Modbus устройство через ту же цепочку вызовов.

1.8.4 Система логирования

Обеспечивает запись информации о ходе работы приложения, ошибках и важных событиях, что критически важно для отладки, мониторинга и анализа работы сервера. Начнём с интеграции (`PlogLogger`), в котором используется внешняя библиотека логирования (`plog`) для вывода сообщений в консоль или файл. Рассмотрим инициализацию логгера (см. листинг 10). В следующем отрывке задаётся уровень детализации, путь к файлу лога (с параметром ротации) и подключается вывод в консоль.

Листинг 10. Инициализация логгера в main:

```
fs::create_directories(path); // путь к каталогу логов
path += "OPCUAServer.log"; //полный путь к каталогу логов
plog::init(severity, path.c_str(), 10000000,
10); //Инициализация plog: уровень, файл, макс. размер, кол-во файлов
ротации

plog::ColorConsoleAppender<plog::TxtFormatter>
consoleAppender; //аппендер для вывода

plog::get()->addAppender(&consoleAppender);
```

Дальше идёт подключение логгера к open62541. В этом блоке экземпляр кастомного логгера PlogLogger (который должен реализовывать интерфейс UA_Logger из open62541) передается в конфигурацию сервера OPC UA. Это позволяет библиотеке open62541 использовать систему логов для своих внутренних сообщений (см. листинг 11).

Листинг 11. Подключение логгера к open62541:

```
//server.cpp - CUAServer::Create() {
...
    config.context = this;
    config.logging = &PlogLogger; // PlogLogger - глобальный
экземпляр или статический объект, который реализует интерфейс
UA_Logger
    config.eventLoop->logger = &PlogLogger;
...
}
```

Напоследок нужно сказать, что макросы UA_LOG_DEBUG, UA_LOG_INFO и UA_LOG_ERROR используются по всему коду для записи сообщений с различными уровнями важности и категориями. Они обеспечивают единообразный подход к логам во всем приложении.

1.9 Тестирование и отладка

Для проверки работоспособности клиент-серверного приложения необходимо провести его тестирование. Тестирование будет состоять из двух групп: тестирование серверной части программы, а также тестирование её клиентской части. Тестирование серверной части включает в себя проверку работы программы. Тестирование состоит из следующих действий:

- тестирование настройки конфигулятора сервера;
- тестирование работы нескольких режимов сервера;
- тестирование работы сервера при подключенном ПЛК.

Тестирование клиентской части включает в себя проверку интерфейса

приложения и отображения работы сервера, к которому подключается пользователь. Для проведения тестирования будем использовать клиентскую часть программы. Тестирование состоит из следующих действий:

- подключение пользователя к серверу;
- просмотр информации на сервере в реальном времени.

1.9.1 Тестирование серверной части

Запустим собранный проект, как показано на рисунке 8, приступим к тестированию серверной части. Нам нужно проверить работу записи/считывания конфигуратора и затем запустить сервер с подготовленным конфигуратором. Запускается GUI сервера, см. рисунок 9. Нам нужно создать конфигуратор, а это значит собрать канал, устройство, группу и затем занести туда теги. Теперь начнём создавать объекты, кликнем ПКМ на категорию “Каналы” и добавляем TCP канал, как показано на рисунке 10.

	орсua-server-modbus	12.02.2025 14:38	Приложение	847 КБ
	open62541.dll	06.12.2024 16:05	Расширение при...	1 724 КБ
	Qt6Core.dll	02.10.2024 7:43	Расширение при...	5 946 КБ
	Qt6Gui.dll	02.10.2024 7:43	Расширение при...	9 017 КБ
	Qt6Widgets.dll	02.10.2024 7:44	Расширение при...	6 350 КБ
	unins000.dat	24.04.2025 16:47	Файл "DAT"	10 КБ
	unins000	24.04.2025 16:47	Приложение	3 314 КБ

Рисунок 8 - Окно с собранным проектом

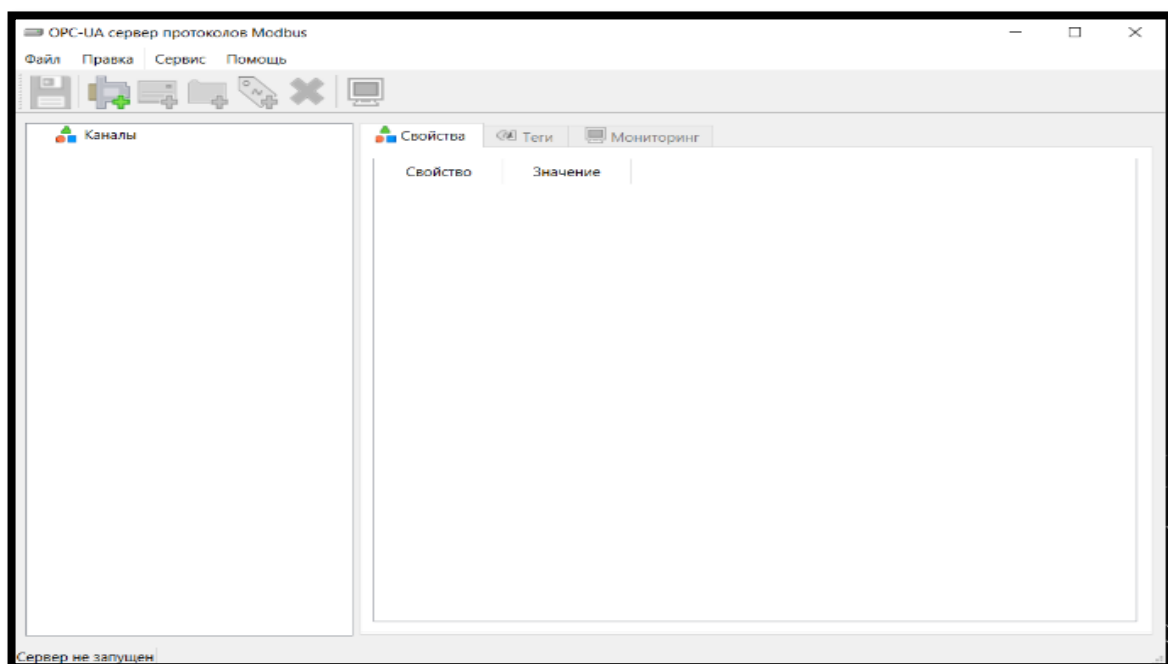


Рисунок 9 - Главное меню сервера

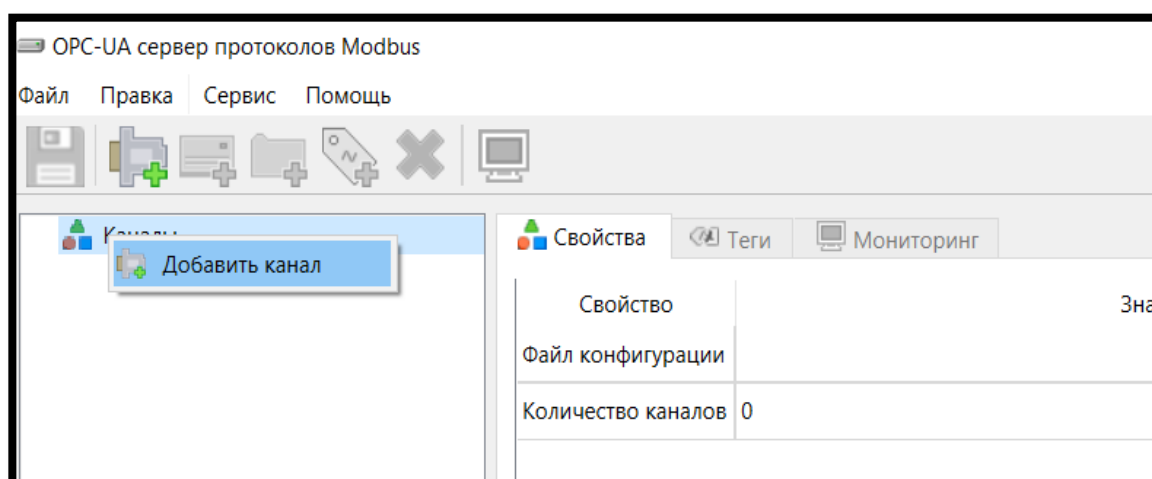


Рисунок 10 - Добавление канала к объекту

Нажав “добавить канал, выскакивает окно настройки канала, как показано на рисунке 11, вводим параметры и сохраняем настройку. Дальше также создадим устройство и группу в нынешний конфигуратор, см. рисунок 12 и 13.

Настройка канала

Имя канала	Channel_1
Тип канала	TCP-соединение
Тип протокола	MODBUS TCP
Таймаут ожидания ответа (мс)	500
Кол-во попыток	3
Пауза перед запросом (мс)	0
IP адрес	127.0.0.1
TCP порт	502
Таймаут установки соединения (мс)	5000

Рисунок 11 - Окно настройки канала

Настройка устройства

Имя устройства	Device_1
Адрес	1
Обмен с устройством	Да
Эмуляция	Нет
Максимальное кол-во регистров в запросе	125
Максимальное кол-во состояний в запросе	2000
Максимально допустимая фрагментация	0

OK Отмена

Рисунок 12 - Окно настройки устройства

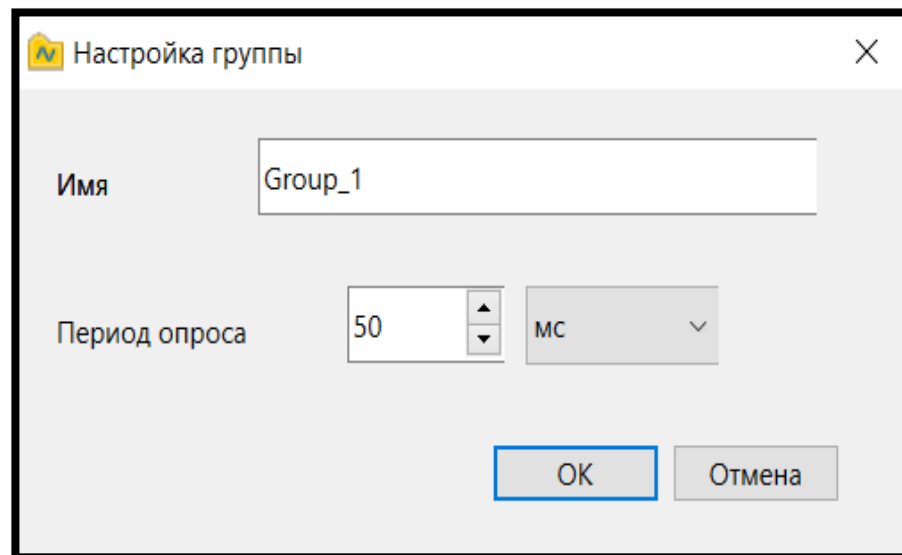


Рисунок 13 - Окно настройки группы

Осталось добавить для группы несколько тегов и дальше сохраним конфигуратор и именуем его, чтобы потом можно к нему вернуться для дальнейшей работы, см. рисунок 14 и 15, нажимаем на панель справа ПКМ и кликаем на “добавить”. Как мы видим у нас есть несколько типов регистра, создадим несколько тегов разных типов и сохраним в дальнейшем конфигуратор.

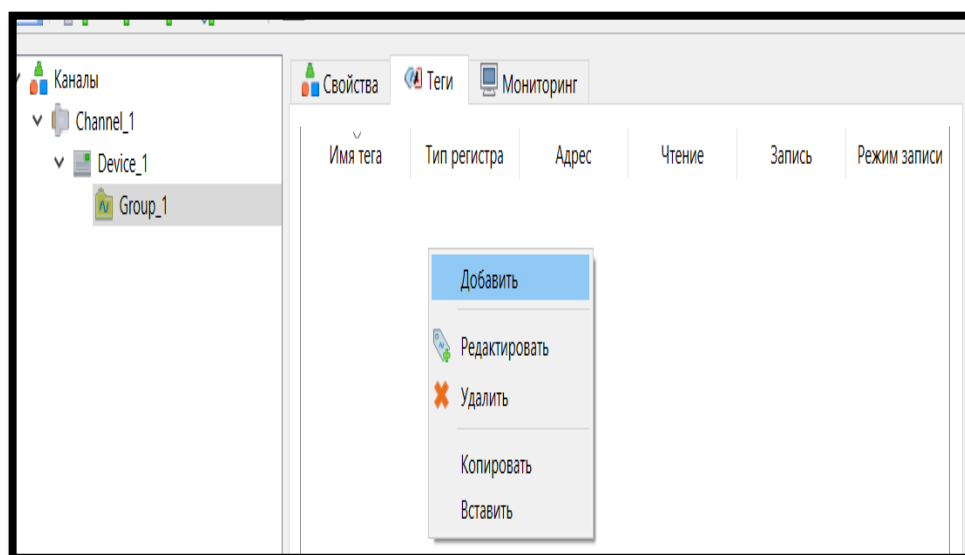


Рисунок 14 - Добавление тегов в группу

Настройка тега

Имя: Tag_1

Чтение (функция 4): Да

Тип регистра: Входящий регистр

Запись (-): Да

Адрес регистра: 1

Тип команды записи: Групповая

Тип значения: Вещ32

Кол-во регистров в сегменте: 0

Порядок байт: AB CD

Смещение параметра в сегменте: 0

Преобразование:

Тип: Нет

Эмуляция значений:

Тип: Константа

Константа: 0

Рисунок 15 - Панель настройки тега

По итогу у нас имеется скромный набор тегов, см. рисунок 16, который в дальнейшем можно будет использовать для тестирования, но для начала сохраним конфигуратор, как показано на рисунке 17.

Имя тега	Тип регистра	Адрес	Чтение	Запись	Режим записи	Тип значения	Порядок байт	Размер сегмента	Смещение в сегменте	Преобразование	Тип эмуляции
Tag_1	Выходной регистр	1	Да	Да	Групповая	Вещ32	AB CD	0	0	Нет	Константа
Tag_2	Выходной регистр	1	Да	Да	Групповая	Вещ32	AB CD	0	0	Нет	Константа
Tag_4	Выходное состояние	1	Да	Да	Групповая	-	-	0	0	-	Константа
Tag_6	Выходное состояние	1	Да	Да	Групповая	-	-	0	0	-	Константа
Tag_3	Входное состояние	1	Да	-	-	-	-	0	0	-	Константа
Tag_5	Входное состояние	1	Да	-	-	-	-	0	0	-	Константа

Рисунок 16 - Набор тегов для конфигуратора

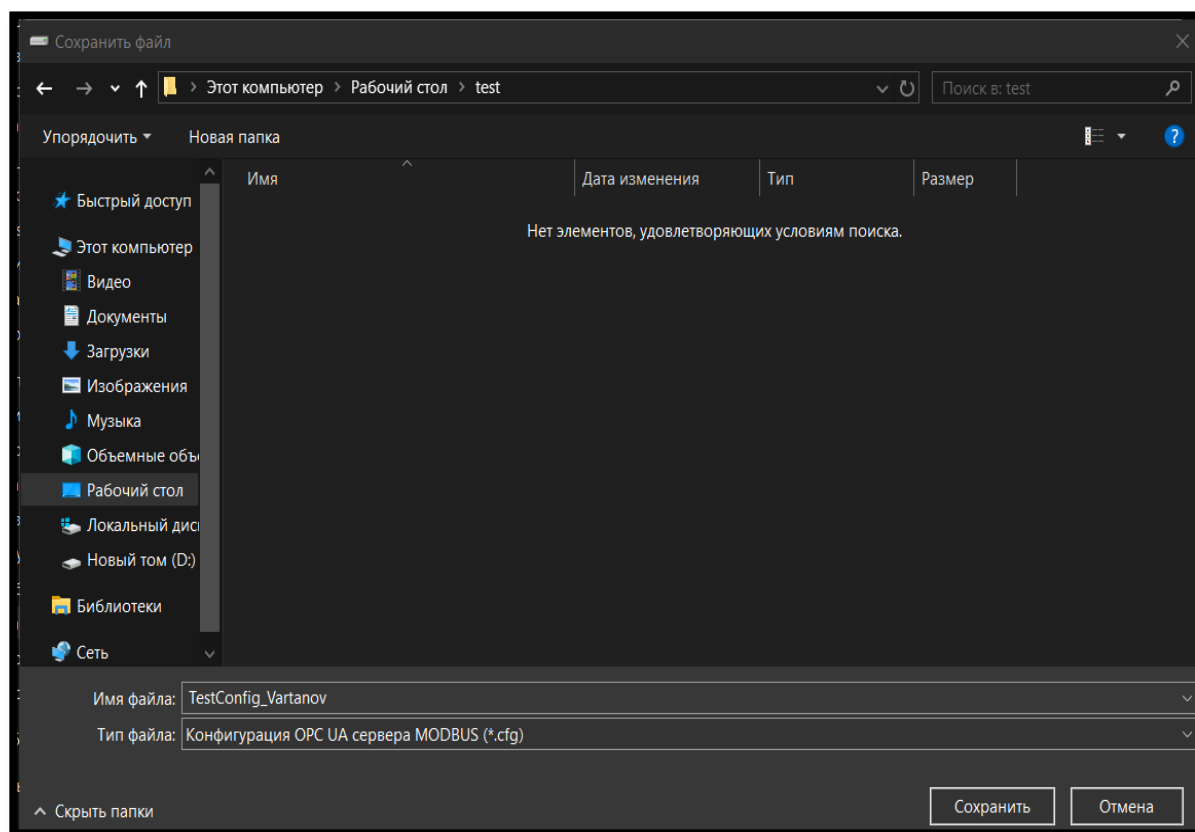


Рисунок 17 - Сохранение в папку конфигурактор

После сохранения проверяем, работает ли созданный нами конфигурактор, закрыв приложение и снова запустив его, выберем в панели “Файл”, затем нажимаем “Выбор конфигурации” и выбираем последний созданный конфигурактор из папки “test”, см. рисунок 18. После этого убедимся, что созданный нами конфигурактор полностью передаёт все параметры, которые мы задали на начальном этапе создания конфигулятора. Теперь дальше мы можем тестировать работу мониторинга тегов, а для этого нам нужен эмулятор контроллера с протоколом MODBUS. В рамках нашего тестирования мы воспользуемся эмулятором Modbus Slave с графическим интерфейсом электрической сети от «ООО ИТЦ АСУ комплект сервис», см. рисунок 19. Этот эмулятор предоставляет работать с 20 тегам в реальном времени, чего нам хватает для тестирования серверной части. Этот эмулятор находится в открытом доступе и можно установить по ссылке <http://opc-servers.ru/>.

Изм	Лист	№ докум.	Подпись	Дата

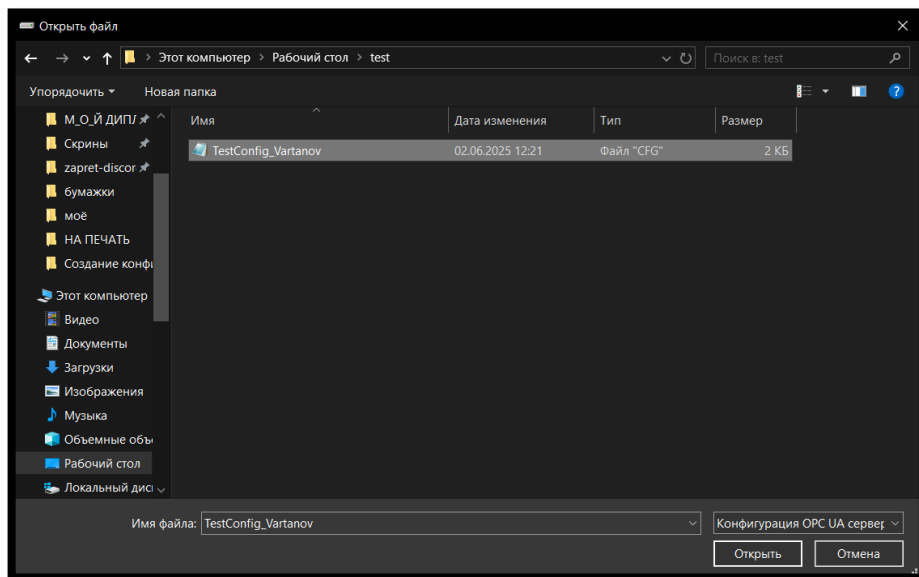


Рисунок 18 - Загрузка конфигулятора из папки



Рисунок 19 - Окно ModbusSlave

Открываем параметр запуска сервера, выбрав открытый режим вещания, выбираем наш ранее созданный конфигурактор и запускаем, см рисунок 20. Далее в эмуляторе находим панель сетевой настройки, см рисунок 21, и вводим адрес 127.0.0.1, стартовый порт 502, ставим галочку на “Использовать Modbus TCP”, см. рисунок 22. Теперь основываясь на доступных регистрах мы воспользуемся двумя дискретными входа, двумя выходными регистрами(статичный и динамический), а также выходные регистры(флаги), см. рисунок 23. После этого запускаем эмулятор, см. рисунок 24, а далее на серверной части запускаем режим мониторинга, кликнув на значок монитора чуть выше над тегами, см. рисунок 25.

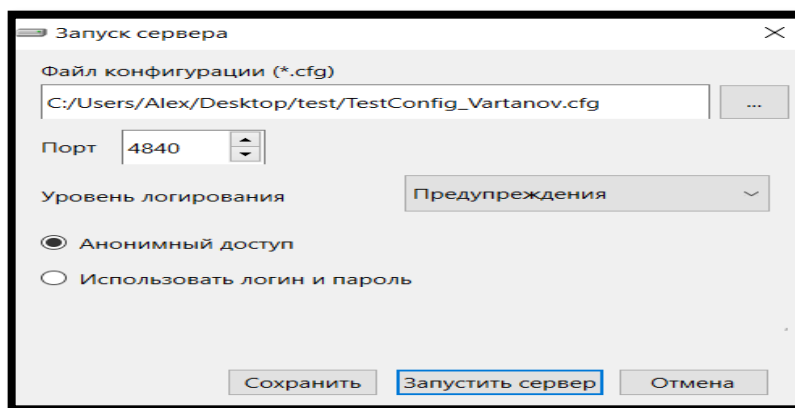


Рисунок 20 - Запуск сервера

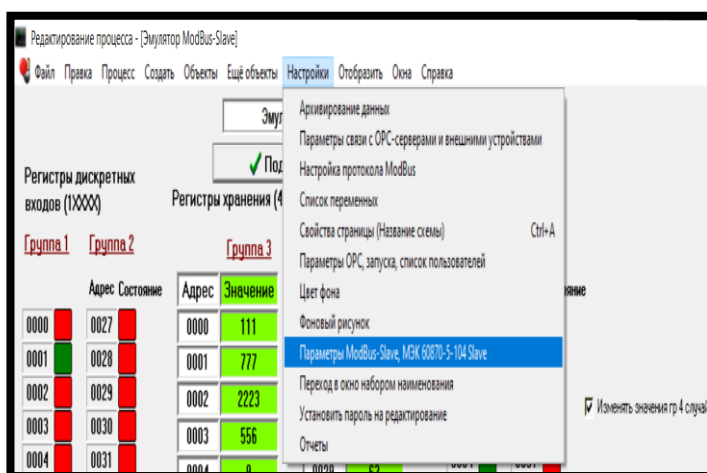


Рисунок 21 - Выбор сетевой настройки для эмулятора

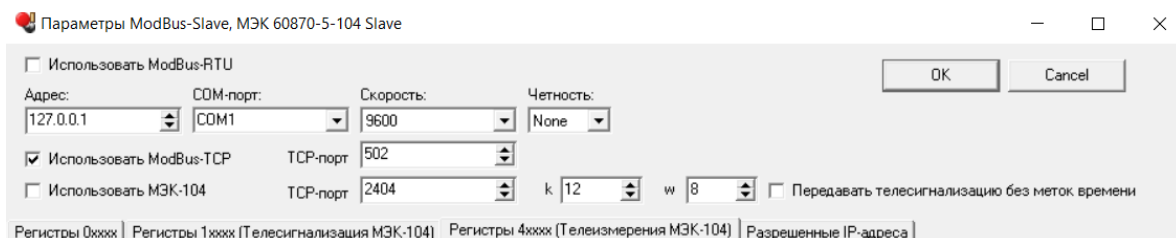


Рисунок 22 - Сетевая настройка эмулятора

Имя тега	Тип регистра	Адрес	Чтение	Запись	Тип значения
Статичный регистр хранения	Выходной регистр	24	Да	Да	Цел16
Динамический регистр хранения	Выходной регистр	31	Да	Да	Цел16
Вход 1	Входное состояние	1	Да	-	-
Вход 2	Входное состояние	2	Да	-	-
Выход 1	Выходное состояние	3	Да	Да	-
Выход 2	Выходное состояние	4	Да	Да	-

Рисунок 23 - Набор тегов для дальнейшего тестирования

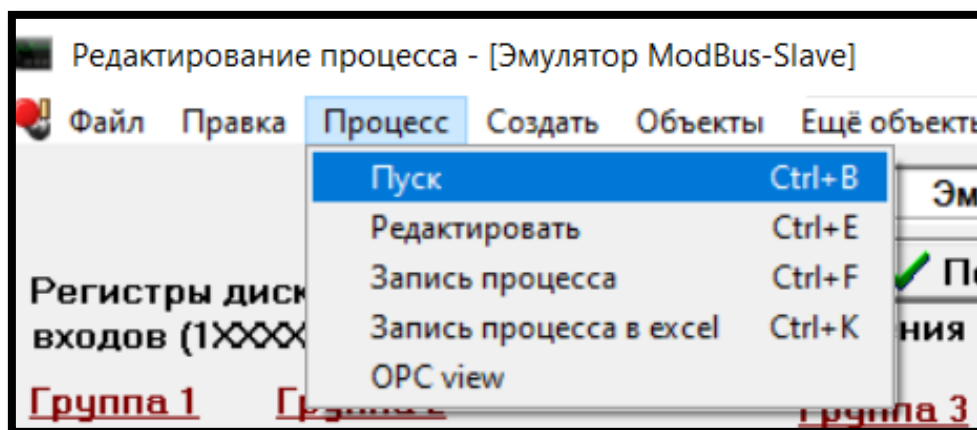


Рисунок 24 - Кнопка запуска эмулятора

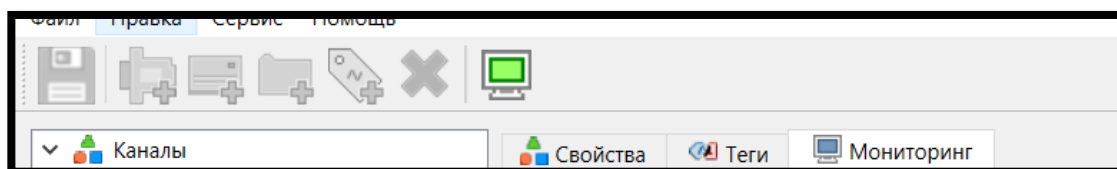


Рисунок 25 - Активированная кнопка активации мониторинга

Теперь каждый интервал времени обновляется подписка на данные и на серверной части будут обновляться данные с эмулятора Modbus Slave через Modbus, которые мы можем отслеживать, см. рисунок 26. Теперь начнём проверять точность работ тегов, сначала отследим работу статического регистра по адресу “24”, который можно редактировать вручную через эмулятор Modbus slave, как показано на рисунках 27, 28 и 29.

Свойства	Теги	Мониторинг		
Имя тега	Значение	Качество	Время	
Статичный регистр	32	Good	2025-06-06 08:00:04.863	

Рисунок 26 - Мониторинг статичного регистра

Регистры хранения (4XXXX)	
Группа 3	
Адрес	Значение
0000	111
0001	777
0002	2223
0003	556
0004	9
0005	0
0006	0
0007	74
0008	0
0009	0
0010	3
0011	0
0012	0
0013	0
0014	0
0015	0
0016	0
0017	0
0018	55
0019	0
0020	0
0021	0
0022	1
0023	23
0024	32

Рисунок 27 - Эмуляция регистров со статичными значениями

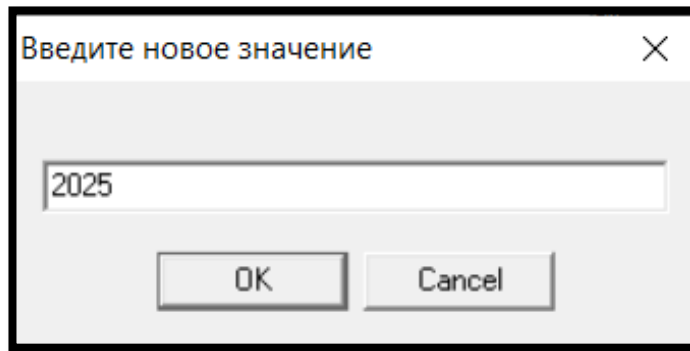


Рисунок 28 - Ввод новых значений для статичного регистра в 24 ячейку

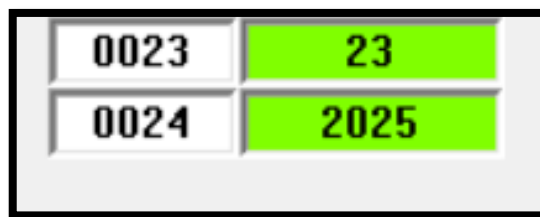


Рисунок 29 - Обновленный статичный регистр

Как мы дальше видим, в мониторинге сервера обновятся данные, проверяем логи времени обновления данных для проверки актуальности данных, см. рисунок 30.

Свойства Теги Мониторинг			
Имя тега	Значение	Качество	Время
Статичный регистр	2025	Good	2025-06-06 08:01:04.068

Рисунок 30 - Обновленные данные регистров

Дальше наблюдаем за обновляющимся регистром хранения, который находится по адресу “31”, см. рисунки 31, 32, 33 и 34. Как мы видим, регистр обновляется с каждой итерацией на стороне эмулятора и очень быстро обновляет подписку данных на серверной части.

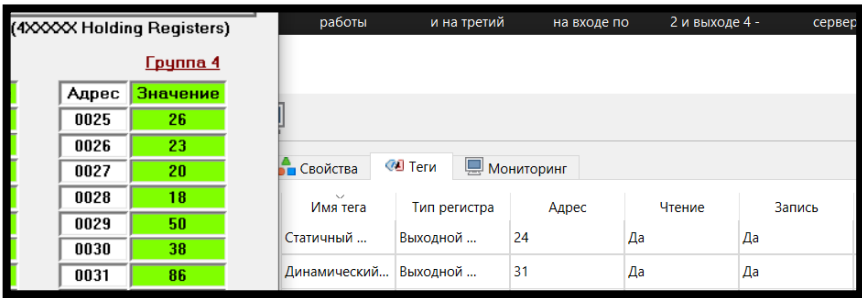


Рисунок 31 - Проверка выбранных регистров

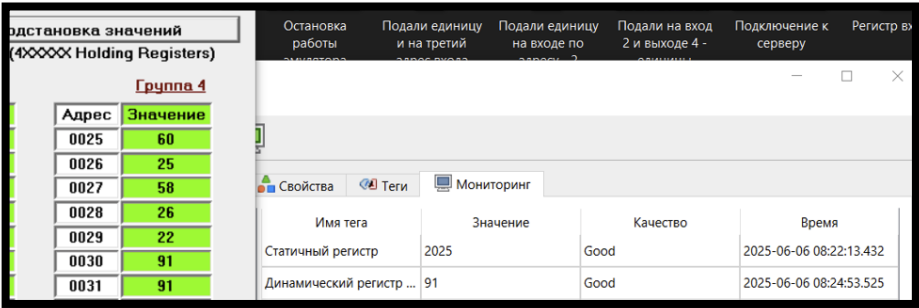


Рисунок 32 - Первая итерация

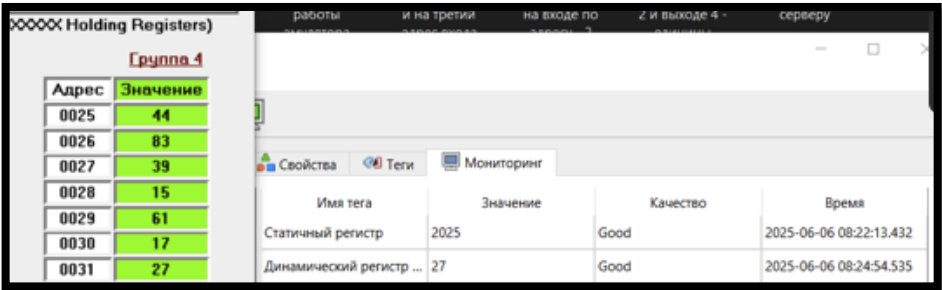


Рисунок 33 - Вторая итерация

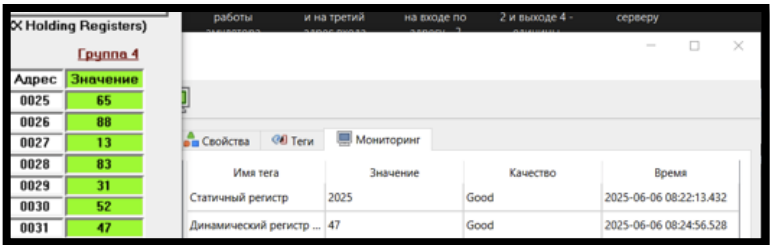


Рисунок 34 - Третья итерация

Дальше идёт проверка работы входов и выходов, см. рисунок 35. В режиме мониторинга мы начинаем в запущенный эмулятор нажимать на красные кнопки, которые отвечают за состояние входа и выхода и наблюдаем за состоянием работы регистров. Красный - обозначает false (в нашем случае 1), зелёный - true (в нашем случае 0). Подадим на вход по адресу 1 - единицу, а на адрес 2 - ноль. На мониторинге должно вывести 1 и 0, как показано на рисунке 36. Проверим также регистры выходов, только подадим на 3 адрес - ноль, а на 4 адрес - единицу, см рисунок 37.

Регистры дискретных входов (1XXXX)				Регистры флагов (0XXXX Coil Status)			
Группа 1		Группа 2		Группа 5		Группа 6	
		Адрес	Состояние			Адрес	Состояние
0000	☐	0027	☐	0000	☐	0027	☐
0001	☐	0028	☐	0001	☐	0028	☐
0002	☐	0029	☐	0002	☐	0029	☐
0003	☐	0030	☐	0003	☐	0030	☐
0004	☐	0031	☐	0004	☐	0031	☐
0005	☐	0032	☐	0005	☐	0032	☐
0006	☐	0033	☐	0006	☐	0033	☐
0007	☐	0034	☐	0007	☐	0034	☐
0008	☐	0035	☐	0008	☐	0035	☐
0009	☐	0036	☐	0009	☐	0036	☐
0010	☐	0037	☐	0010	☐	0037	☐
0011	☐	0038	☐	0011	☐	0038	☐
0012	☐	0039	☐	0012	☐	0039	☐
0013	☐	0040	☐	0013	☐	0040	☐
0014	☐	0041	☐	0014	☐	0041	☐
0015	☐	0042	☐	0015	☐	0042	☐
0016	☐	0043	☐	0016	☐	0043	☐
0017	☐	0044	☐	0017	☐	0044	☐
0018	☐	0045	☐	0018	☐	0045	☐
0019	☐	0046	☐	0019	☐	0046	☐
0020	☐	0047	☐	0020	☐	0047	☐
0021	☐	0048	☐	0021	☐	0048	☐
0022	☐	0049	☐	0022	☐	0049	☐
0023	☐	0050	☐	0023	☐	0050	☐
0024	☐	0051	☐	0024	☐	0051	☐
0025	☐	0052	☐	0025	☐	0052	☐
0026	☐	0053	☐	0026	☐	0053	☐

Рисунок 35 - Регистры входов и выходов на эмуляторе

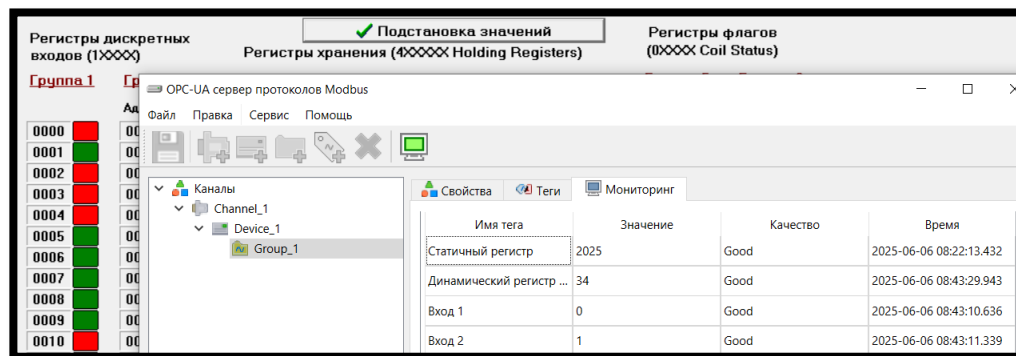


Рисунок 36 - Мониторинг регистров входов

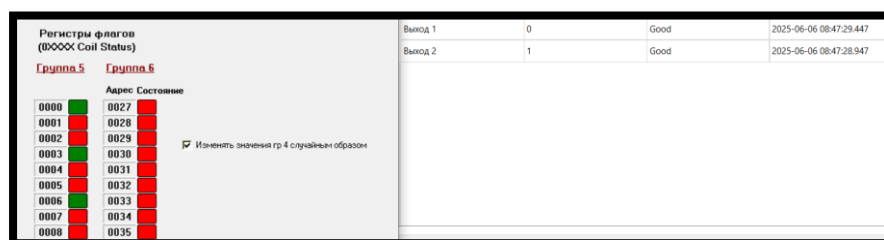


Рисунок 37 - Мониторинг регистров выходов

Как мы видим, наша серверная часть полностью исправно и стабильно выводит информацию без каких-либо критических ошибок. Чтобы отключить сервер в панели «Сервис» находите кнопку «Остановить сервер OPC UA» и нажимаете, выведется сообщение об остановке сервера, см рисунок 38 и 39.

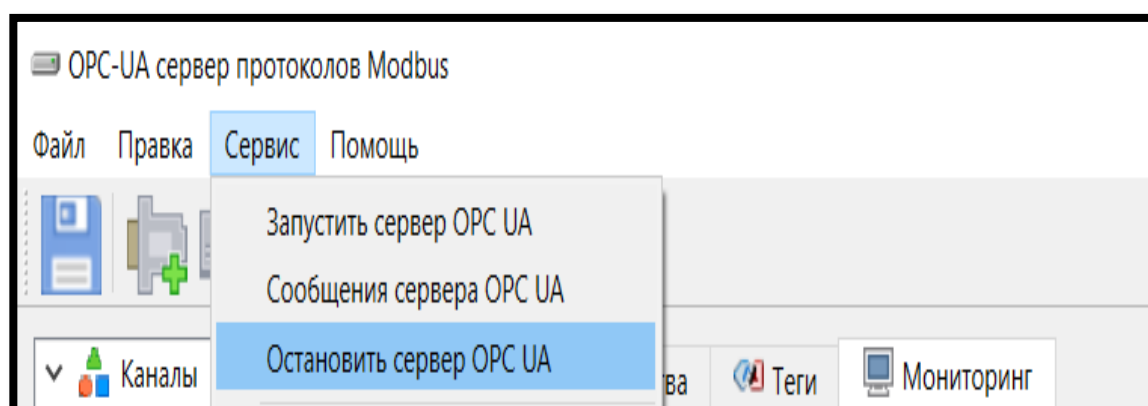


Рисунок 38 - Панель отключения сервера

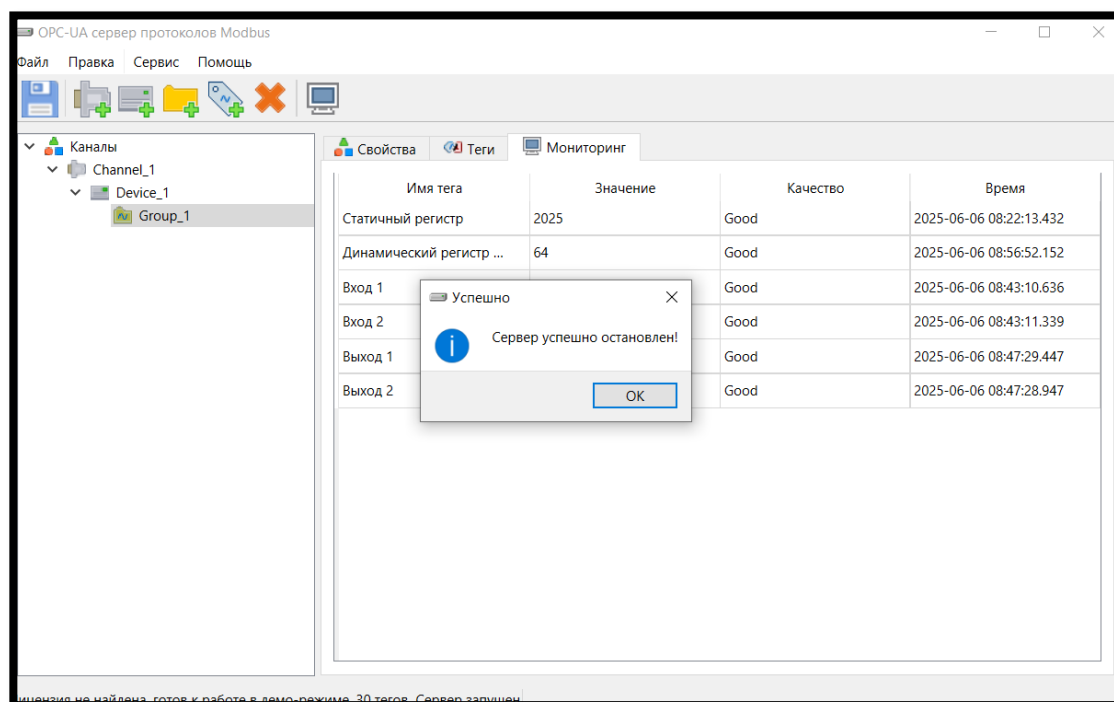


Рисунок 39 - Уведомление об остановке сервера

Дальше можем тестировать клиентскую часть чтобы удостовериться, что наша программа соответствует поставленным нашим задачам.

1.9.2 Тестирование клиентской части

Перед тем началом тестирования запускается программа, которая будет отвечать за клиентский интерфейс OPC UA-client. Программа называется UaExpert, разработана Unified Automation и является бесплатной, её установить можно после регистрации по ссылке <https://www.unified-automation.com/products/development-tools/uaexpert.html>.

Для начала запустим снова сервер с той же конфигурацией, но теперь активируем режим аутентификации под логином и паролем, а дальше запустим сервер, см. рисунок 40. После этого создадим для запущенного сервера список пользователей, которые будут должны авторизоваться, а для этого в панели «Сервис» запустим «Добавить пользователей», см. рисунок 41. Потом как показано на рисунках 42 и 43 отобразится окно ввода пользователей, где в специальные строки для ввода записываем логин и пароль пользователя.

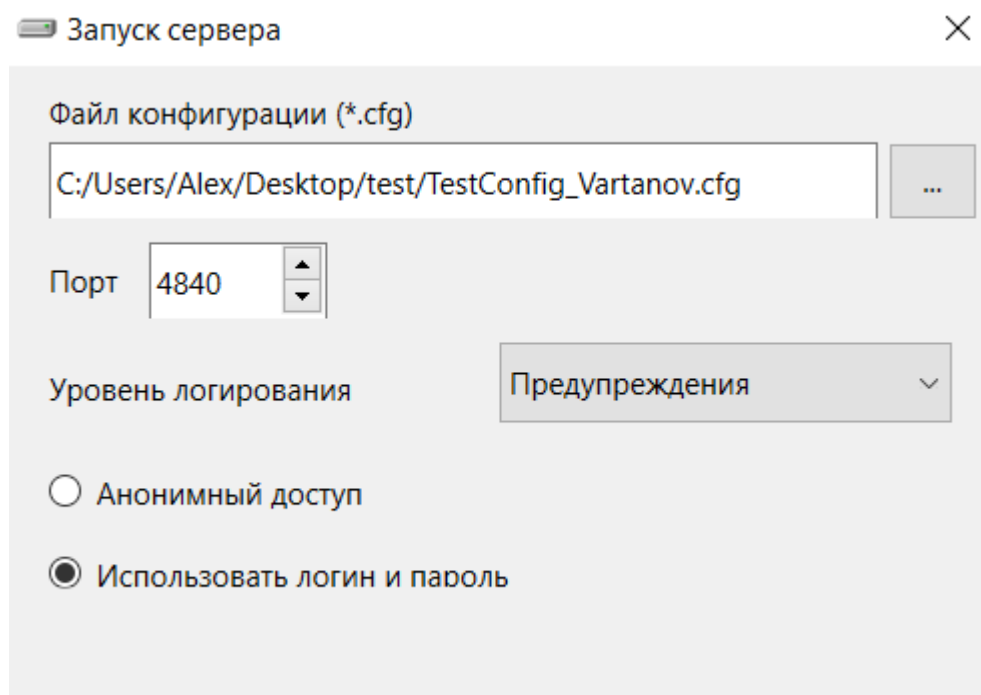


Рисунок 40 – Выбор режима сервера

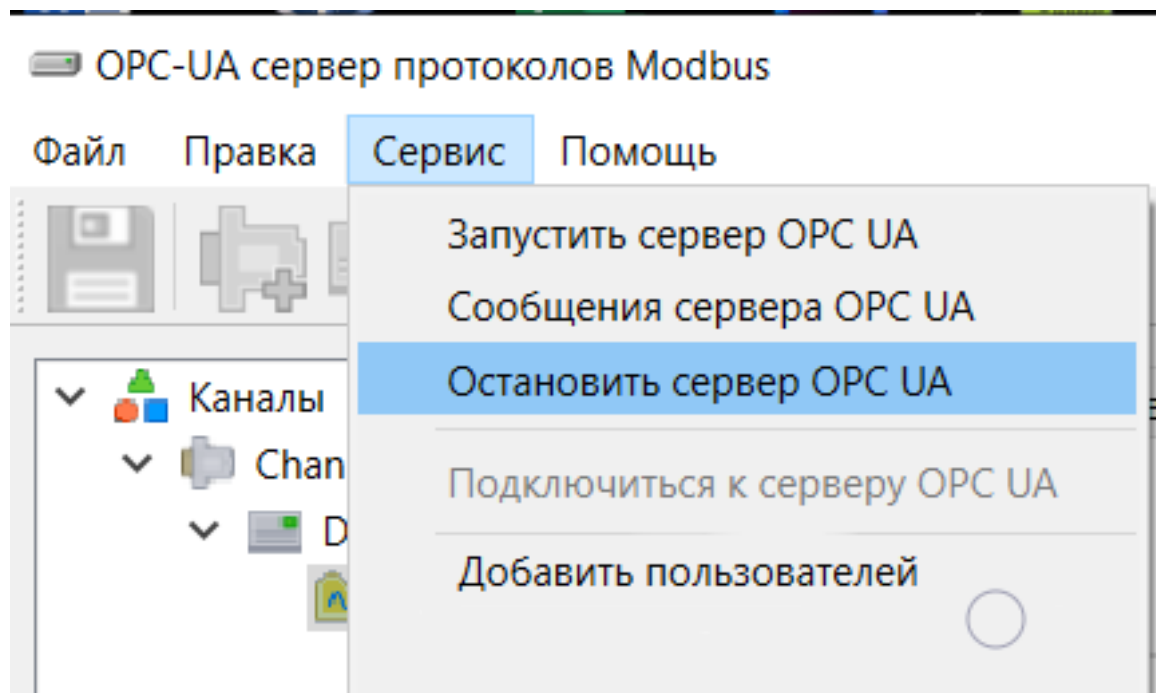


Рисунок 41 – Панель, где открывается список пользователей

Добавить пользователей

Логин

Пароль

Логин	Пароль

Добавить

Удалить

OK

Отмена

Рисунок 42 – Окно для ввода списка пользователей

Логин

Пароль

Логин	Пароль
alex	*****
vartanov	*****
testusser1	*****

Рисунок 43 – Ввод списка пользователей

После создания списка пользователей запускаем UaExpert, откроется главное окно программы, см. рисунок 44. Нам нужно подключиться к запущенному серверу, кликнем на панели сверху кнопку добавления сервера, см. рисунок 45, и выберем наш нужный сервер, см. рисунок 46. Как мы видим, наш

сервер пока что не имеет никакого шифрования, но у нас стоит наша защита в виде авторизации для подключения к серверу. В области «Project» будет отображаться сервер, нажимаем ПКМ на проект и кликаем по кнопке «connect», см. рисунок 47.

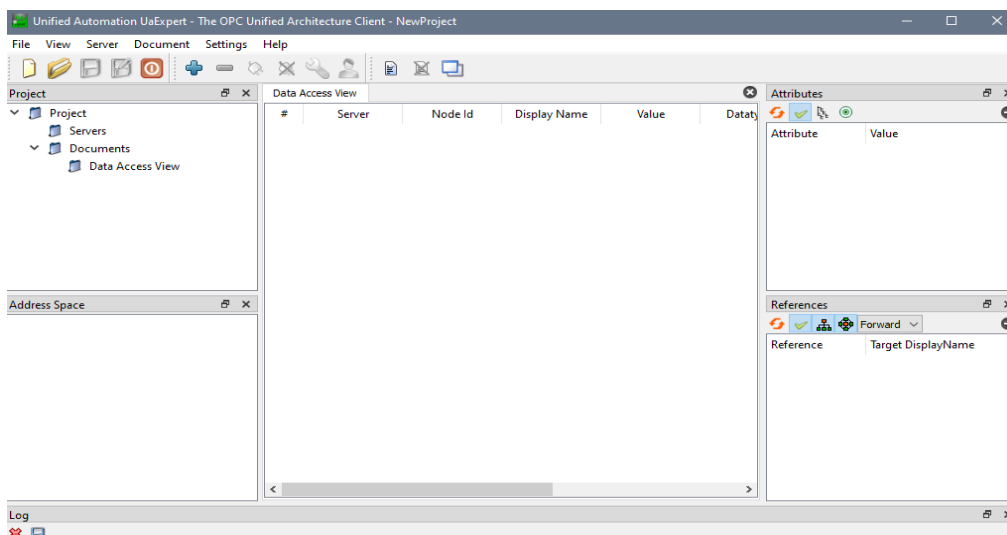


Рисунок 44 – Главное окно UaExpert

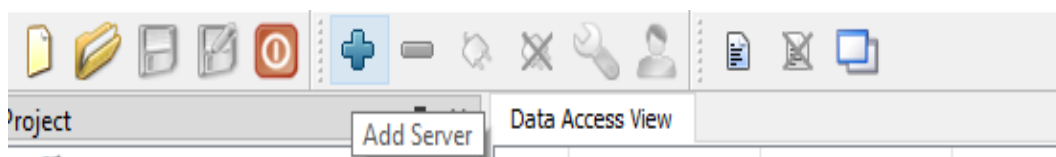


Рисунок 45 – Кнопка добавления сервера

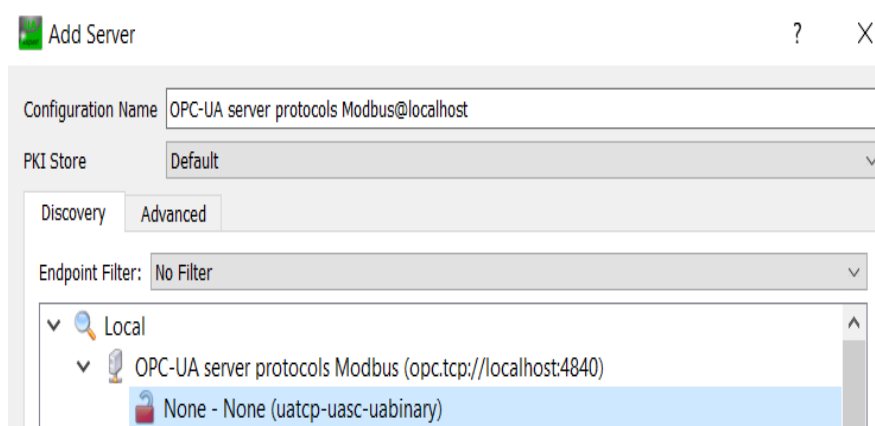


Рисунок 46 – Добавление сервера

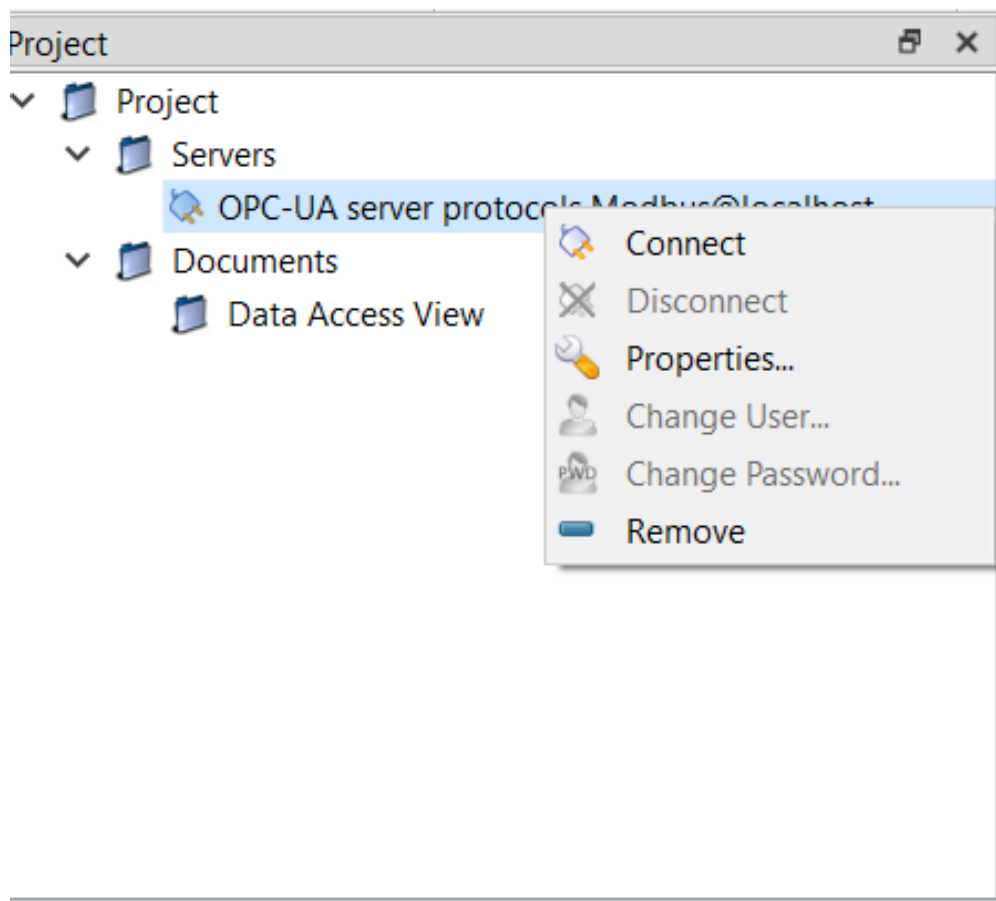


Рисунок 47 – Подключение к добавленному серверу

После попытки подключения через клиентский интерфейс выведется окно, которое требует ввести логин и пароль, см. рисунок 48. Если ввод данных будет верным, то выведется на панели «Adress Space» узлы устройства OPC UA, в которых включены и узлы нашего устройства, который мы конфигурировали на серверной части, см. рисунок 49. Перенесём узлы на рабочую панель справа, см. рисунок 50, и проверим работу на активном мониторинге данных, то есть запускаем эмулятор, запускаем режим мониторинга на серверной части и отслеживаем работу подписок на клиентской части, см. рисунок 51. Как мы видим, данные обновляются и корректно выводятся. Регистры входов и выходов на клиентской части отображаются как true (состояние 0 на сервере) и false (состояние 1 на сервере). Тестирование можно считать завершённой успешно, критических ошибок не было выведено, сервер работает стабильно и бесперебойно.

Изм	Лист	№ докум.	Подпись	Дата

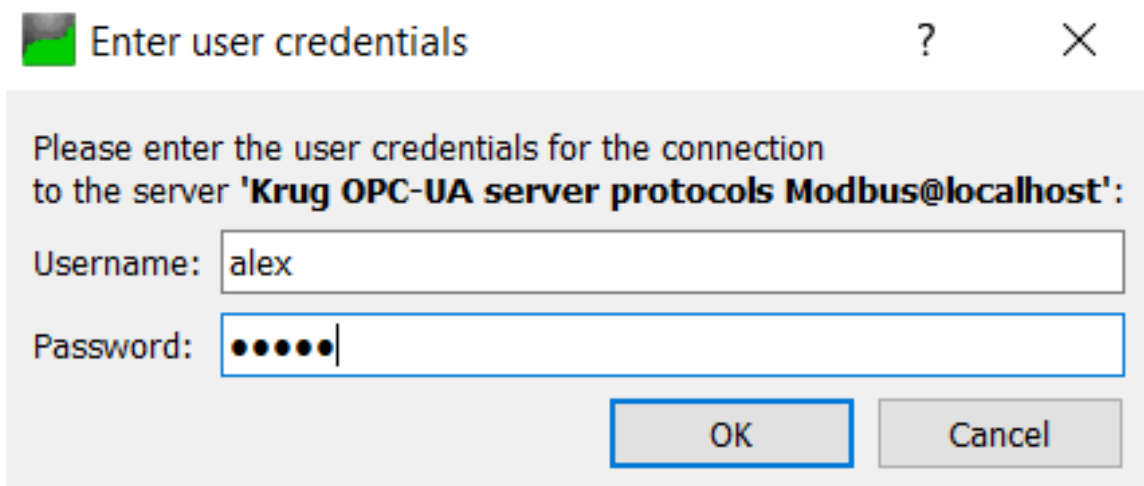


Рисунок 48 – Авторизация на клиентской части

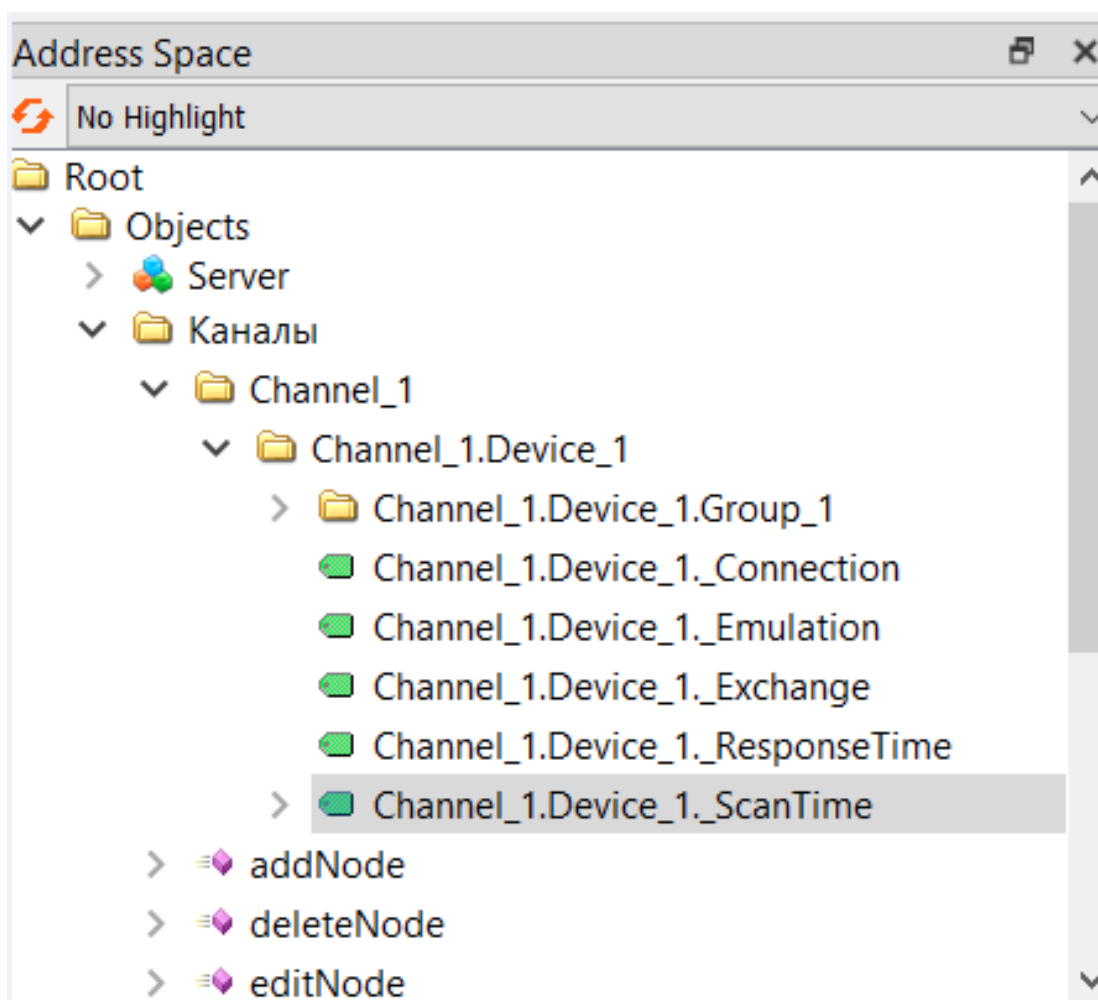


Рисунок 49 – Список узлов нашей группы

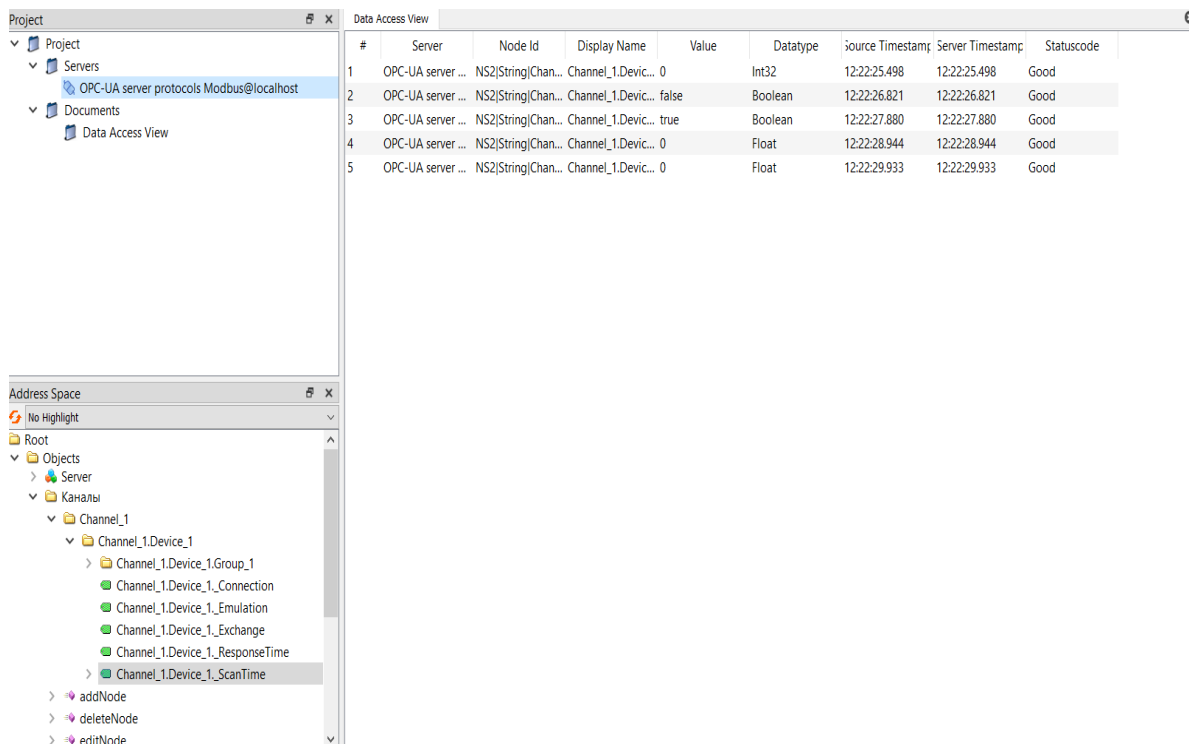


Рисунок 50 – Рабочая панель выводов узлов с регистрами

Data Access View								
#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	OPC-UA server ...	NS2[String](Channel_1.Device_1.Group_1.Tag_1	Channel_1.Device_1.Group_1.Tag_1	23	UInt16	12:31:49.631	12:31:49.631	Good
2	OPC-UA server ...	NS2[String](Channel_1.Device_1.Group_1.Tag_2	Channel_1.Device_1.Group_1.Tag_2	74	UInt16	12:34:09.350	12:34:09.350	Good
3	OPC-UA server ...	NS2[String](Channel_1.Device_1.Group_1.Tag_3	Channel_1.Device_1.Group_1.Tag_3	true	Boolean	12:31:54.904	12:31:54.904	Good
4	OPC-UA server ...	NS2[String](Channel_1.Device_1.Group_1.Tag_4	Channel_1.Device_1.Group_1.Tag_4	false	Boolean	12:31:56.388	12:31:56.388	Good
5	OPC-UA server ...	NS2[String](Channel_1.Device_1.Group_1.Tag_5	Channel_1.Device_1.Group_1.Tag_5	true	Boolean	12:31:57.079	12:31:57.079	Good
6	OPC-UA server ...	NS2[String](Channel_1.Device_1.Group_1.Tag_6	Channel_1.Device_1.Group_1.Tag_6	true	Boolean	12:31:57.905	12:31:57.905	Good

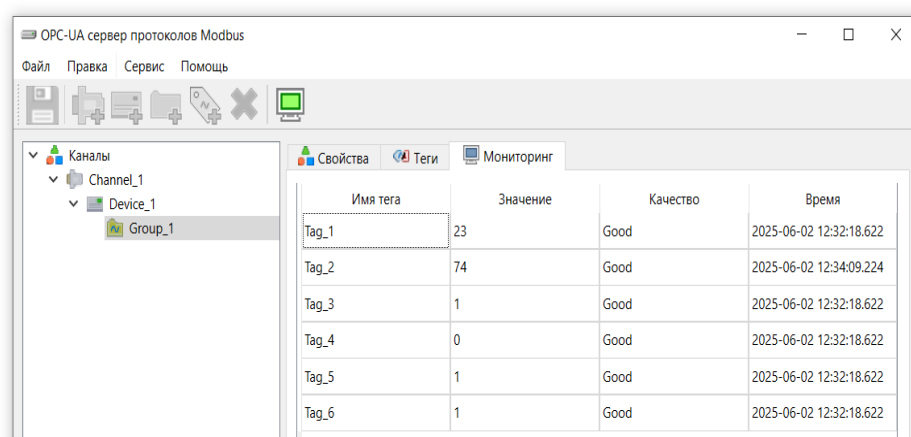


Рисунок 51 – Проверка работы клиентской части

2 Безопасность жизнедеятельности

При взаимодействии с компьютером, необходимо внимательно соблюдать меры безопасности, чтобы предотвратить опасность для здоровья после использования. Если пользователь ПК будет пренебрегать техникой безопасности, то проблемы со здоровьем будут неизбежны.

2.1 Вред здоровью при работе с ПК

В процессе взаимодействия с компьютером могут появиться некоторые проблемы со здоровьем, которые имеют свойство накапливаться. Это значит, что здоровье ухудшается не сразу, а постепенно. Рассмотрим наиболее опасные факторы:

- нагрузка на суставы. Если работать за компьютером продолжительное время в неправильной посадке, на суставы и особенно кисти рук будет идти большая нагрузка, что может привести к онемению рук;
- нагрузка на спину. При том же неправильном положении осанки во время работы за компьютером, есть шанс на то, что появится искривление позвоночника или, например, грыжа межпозвоночного диска;
- нагрузка на органы зрения. Монитор оказывает нагрузку на глаза, и если долго смотреть в монитор, то будет ухудшаться здоровье глаз, что может привести к синдрому «красного глаза» и «сухого глаза»;
- нагрузка на психику и нервную систему. При долгом взаимодействии с ПК, у пользователя могут возникать проблемы с памятью, а также может появиться депрессия и даже бессонница.

2.2 Снижение вреда для здоровья при работе с ПК

Далее приведены простые правила, соблюдение которых может помочь избежать серьезных проблем со здоровьем:

- соблюдение необходимого расстояния до монитора. Правильным расстоянием до дисплея является расстояние, при котором пользователь может коснуться указательным пальцем до верхнего края монитора;

- соблюдение оптимального угла экрана. Оптимальным углом является наклон монитора в 30 градусов;
- правильное расположение клавиатуры. Клавиатура должна находиться на расстоянии 20–30 сантиметров от края стола;
- организация перерывов. Примерно каждый час необходимо делать разминку для органов зрения, а именно отвлечься от монитора и делать гимнастику для глаз: фокусировать внимание на дальние объекты и переводить фокус с одних предметов на другие, рисовать различные фигуры глазами. Также примерно каждые полтора–два часа необходимо выполнять зарядку;
- подбор правильного с точки зрения поддержки осанки кресла. При выборе данной мебели, необходимо приобрести такую, которая будет обеспечивать прямую осанку, при которой спина будет опираться в спинку.

2.3 Техника безопасности при работе с компьютером

При работе с компьютером необходимо соблюдать правила, связанные с техникой безопасности:

- не оставлять компьютер без присмотра на длительное время;
- не использовать компьютер, который работает от питания без заземления;
- не работать за ПК, у которого есть повреждение в системном блоке или в его периферийных устройствах;
- не включать компьютер, если в помещении слишком высокая влажность;
- не класть на корпус компьютера или монитора посторонние предметы.

Заключение

В ходе выполнения данной выпускной квалификационной работы были получены навыки написания логики и программ для работы с OPC-сервером протоколов MODBUS спецификации UA. Разработанный сервер успешно решает задачу интеграции оборудования в современные информационные системы, предоставляя унифицированный, безопасный и стандартизированный доступ к производственным данным.

Была спроектирована и реализована многоуровневая архитектура приложения, которая обеспечивает модульностью, гибкостью и возможностью дальнейшей модификации. Ключевой особенностью является динамическое построение адресного пространства OPC UA на основе внешних конфигурационных файлов. Сервер поддерживает основные операции OPC UA, включая чтение, запись и подписки на изменения данных, а также предоставляет OPC UA-методы для клиентской части. После проведения тестирования было выявлено, что наша разработка соответствует всем поставленным в начале работы требованиям и работает корректно и не вызывает критические ошибки в программе.

В дальнейшем можно добавить усовершенствования, по типу улучшения уровня безопасности путём добавления сертификатов с шифрованием. Ещё можно усовершенствовать систему логов, чтобы в более понятном виде отображало информацию о событиях и ошибках для удобства пользователя. Методы OPC UA позволяют реализовать расширенные сервисы, которые отвечают за исторические данные и аварийные сигнала для отображения полной картины о состоянии производственного процесса.

Список использованных источников

1. OPC Unified Architecture Specification. Version 1.05. [Электронный ресурс]/Режим доступа: <https://reference.opcfoundation.org/>, свободный (дата обращения: 10.05.2025).

2. Modbus Application Protocol Specification v1.1b3. [Электронный ресурс]/Режим доступа: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf, свободный. (дата обращения 10.05.2025).

3. Просто о стандартах OPC. [Электронный ресурс]/Режим доступа: <https://ipc2u.ru/articles/prostye-resheniya/prosto-o-standartakh-opc-da-i-opc-ua/#std>, свободный (дата обращения 10.05.2025).

4. Modbus Messaging on TCP/IP Implement Guide v1.0b. [Электронный ресурс]/Режим доступа: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf, свободный (дата обращения: 10.05.2025).

5. Документация библиотеки open62541. [Электронный ресурс]/Режим доступа: <https://www.open62541.org/doc/open62541-v1.4.11.1.pdf>, свободный (дата обращения 16.05.2025).

6. Скотт Мейерс. “Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ”. 2014 - 300с.

7. Бьерн Страуструп. “Язык программирования C++”. 2019 - 322 с.

Приложение А

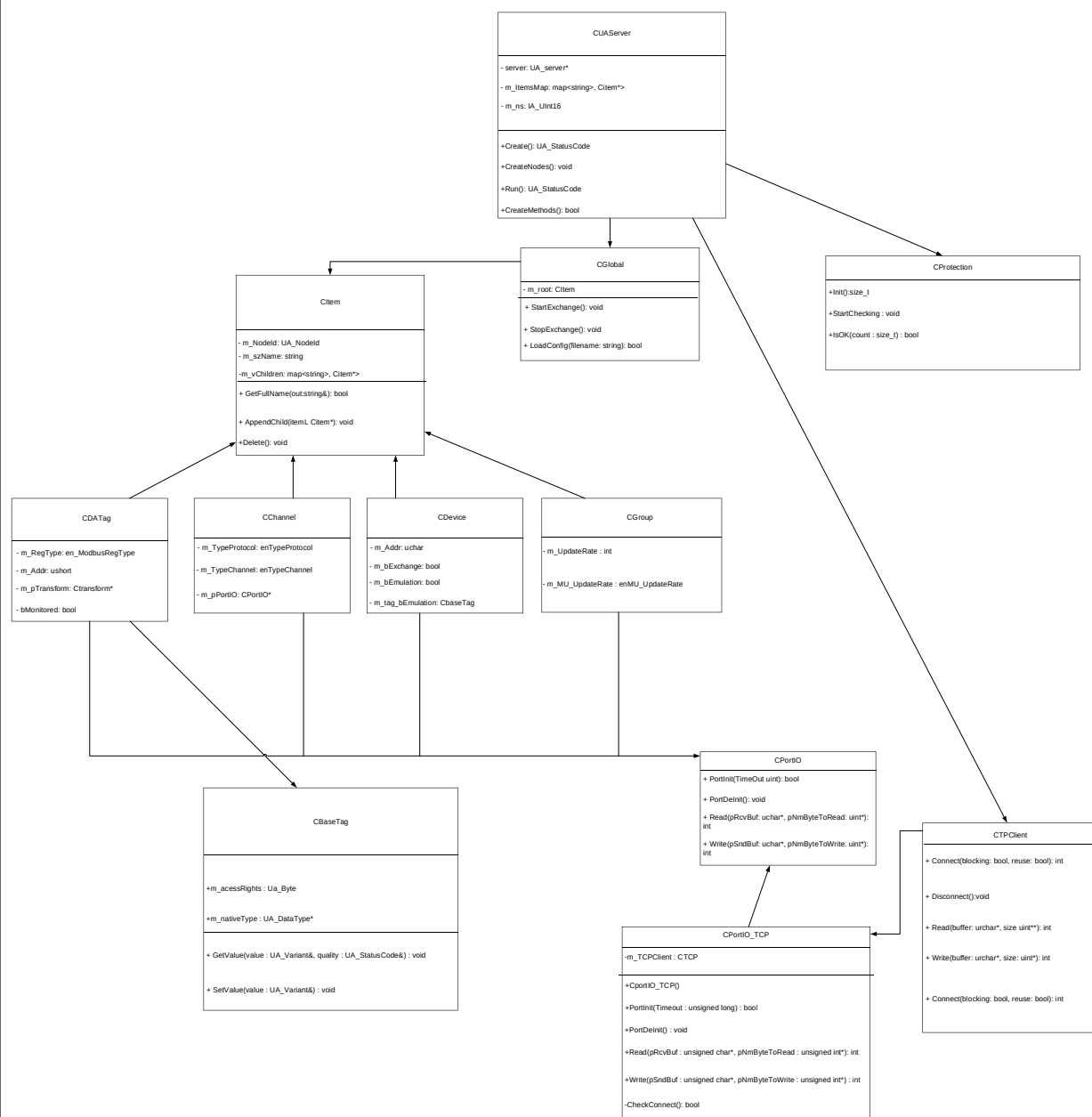
(обязательное)

OPC-сервер протоколов MODBUS спецификации Unified Architecture

UML-диаграммы

					<i>ПГУ 1.090301.04.001 ПЗ</i>	Лист
						63
Изм	Лист	№ докум.	Подпись	Дата		

Приложение А.1— Диаграмма классов

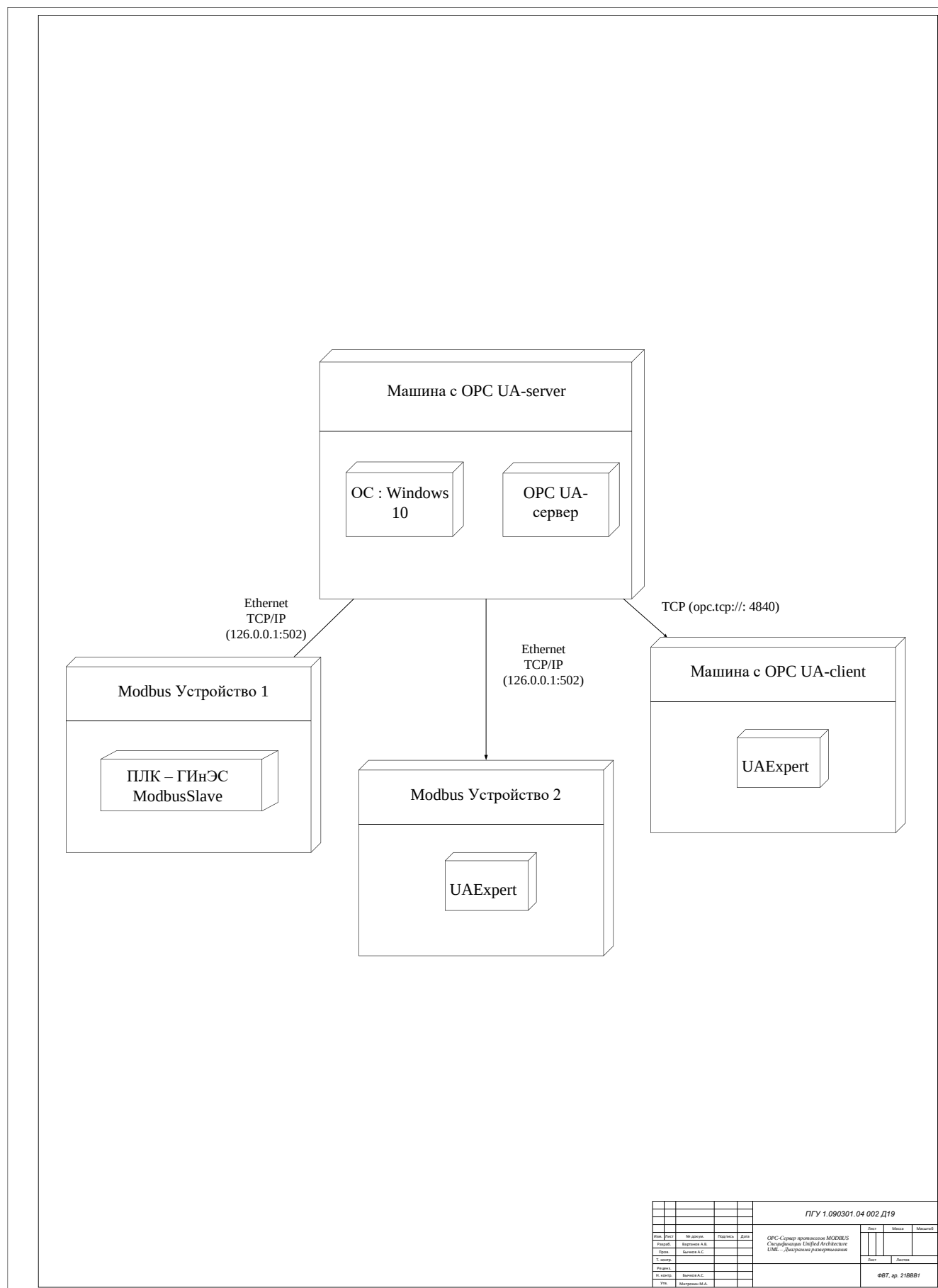


					ПГУ 1.090301.04.002 Д11				
Лист	№ докум.			Подпись	Дата	ОПС Сервис проектирования MODBUS Спецификация Unified Architecture UML – Диаграмма классов			
Разработ	Борисов А.В.								
Проект	Борисов А.С.								
Т. экз.									
Листов									
И. экз.	Борисов А.С.								
Уч.	Михайлов А.А.								

Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Приложение А.2– Диаграмма развертывания



						ПГУ 1.090301.04.002 Д19		
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	OPC-Сервер промышленного MODBUS Configuration Unified Architecture UME – Описание развертывания		
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия			
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия			
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия			
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	ФБТ, стр. 218881		

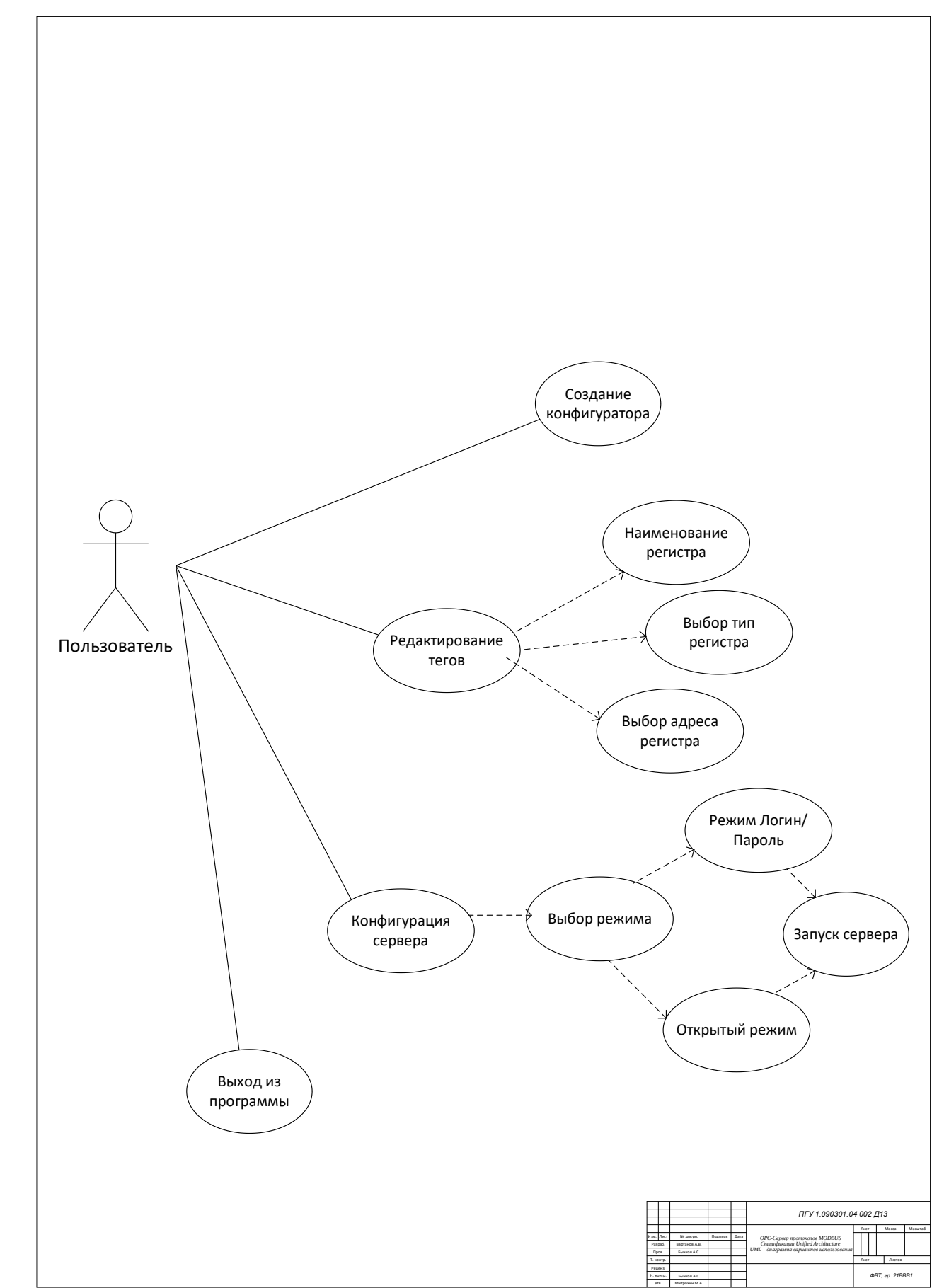
Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист

65

Приложение А.3– Диаграмма вариантов использования



ПГУ 1.090301.04 002 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

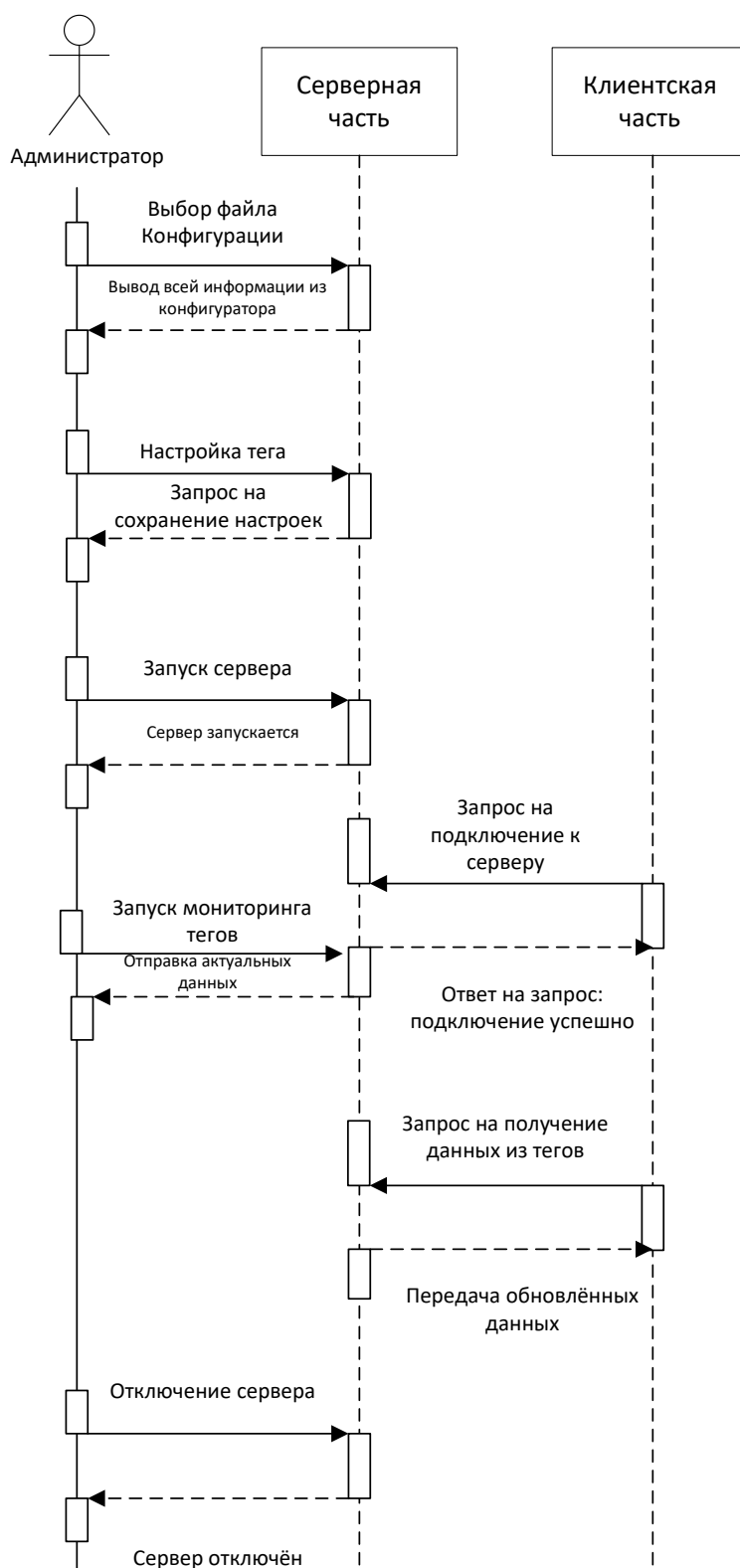
Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист

66

Приложение А.4— Диаграмма последовательностей



					ПГУ 1.090301.04.002 Д13		
Имя	Дата	№ докум.	Подпись	Дата	ОПС-Сервер промышленной МОДУЛЬНОЙ Структурированной Unified Architecture UME – диаграмма последовательностей		
Разработчик		Версия 1.0					
Проверен		Выполнен А.С.			Имя	Место	Масштаб
Утвержден					Дата	Листов	
И. исполнитель		Выполнен А.С.			Ф8Т, стр. 218881		
Исполн.		Михайлов М.А.					

Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист

67

Приложение Б

(обязательное)

OPC-сервер протоколов MODBUS спецификации Unified Architecture

Листинг программы

					<i>ПГУ 1.090301.04.001 ПЗ</i>	Лист
						68
Изм	Лист	№ докум.	Подпись	Дата		

Приложение Б.1– Листинг программы

```
#include "StdAfx.h"
#include <signal.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>
#include <vector>

#include <open62541/server.h>
#include <open62541/server_config_default.h>
#include <open62541/plugin/accesscontrol_default.h>
#include <open62541/plugin/log_stdout.h>

#include "global.h"
#include "datag.h"
#include "group.h"
#include "ploglogger.h"

#include "device.h"

#include "../plog/Initializers/RollingFileInitializer.h"
#include "../plog/Formatters/TxtFormatter.h"
#include "../plog/Appendors/ColorConsoleAppender.h"

#include "csvimporter.h"
#include "protection.h"
#include "portio_tcp.h"

#include "../simpleINI/SimpleIni.h"
#include "iconv.h"
#include <map>
```

```

#ifdef WIN32

#include "minidumpapiset.h"
#pragma comment(lib, "Dbghelp.lib")

void CreateDump(_EXCEPTION_POINTERS* ExceptionInfo)
{

    char FileName[32];
    SYSTEMTIME LocalTime;    GetLocalTime(&LocalTime);
    sprintf_s(FileName, _countof(FileName),
        " ",
        LocalTime.wYear, LocalTime.wMonth, LocalTime.wDay,
        LocalTime.wHour, LocalTime.wMinute, LocalTime.wSecond);

    std::string path = std::getenv("USERPROFILE");
    path += " ";
    fs::create_directories(path);
    path += FileName;

    HANDLE    hFile    =    CreateFile(path.c_str(),    GENERIC_WRITE,
FILE_SHARE_READ, 0, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
    if (INVALID_HANDLE_VALUE != hFile)
    {
        MINIDUMP_EXCEPTION_INFORMATION ExInfo;
        ExInfo.ThreadId = GetCurrentThreadId();
        ExInfo.ExceptionPointers = ExceptionInfo;
        ExInfo.ClientPointers = 0;
        MiniDumpWriteDump(GetCurrentProcess(),
GetCurrentProcessId(), hFile, MiniDumpWithFullMemory, &ExInfo, 0,
0);

        CloseHandle(hFile);
    }
}

```

```

LONG      WINAPI      MyUnhandledExceptionFilter(_EXCEPTION_POINTERS*
    ExceptionInfo)
{
    CreateDump(ExceptionInfo);
    return EXCEPTION_CONTINUE_SEARCH;
}

class INITILIZE
{
public:
    INITILIZE()
    {
        SetUnhandledExceptionFilter(MyUnhandledExceptionFilter);
    }
} Initilize;
#endif

using namespace std;

UA_Boolean running = true;

CProtection protector;

static UA_StatusCode readNodeValue(UA_Server* server, const
    UA_NodeId* sessionId, void* sessionContext,
        const UA_NodeId* nodeId, void* nodeContext,
        UA_Boolean sourceTimeStamp, const UA_NumericRange* range,
        UA_DataValue* dataValue);

static UA_StatusCode writeNodeValue(UA_Server* server, const
    UA_NodeId* sessionId, void* sessionContext,
        const UA_NodeId* nodeId, void* nodeContext,
        const UA_NumericRange* range, const UA_DataValue* data);

#ifdef WIN32

```

					ИГУ 1.090301.04.001 ПЗ	Лист
						71
Изм	Лист	№ докум.	Подпись	Дата		

```

static BOOL WINAPI stopHandler(DWORD dwCtrlType)
{
    UA_LOG_INFO(&PlogLogger,    UA_LOGCATEGORY_SERVER,    "received
ctrl-c");
    running = false;
    return TRUE;
}
#else
static void stopHandler(int sign)
{
    UA_LOG_INFO(&PlogLogger,    UA_LOGCATEGORY_SERVER,    "received
ctrl-c");
    running = false;
}
#endif

static void SetStopHandler()
{
#ifdef WIN32
    SetConsoleCtrlHandler(stopHandler, TRUE);
#else
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);
#endif

}

enum class FileType { csv, dat };

class CUAServer
{
public:
    CUAServer()
    {

```

					ИГУ 1.090301.04.001 ПЗ	Лист
						72
Изм	Лист	№ докум.	Подпись	Дата		

```

        unsigned int m_port = 4840;
        bool m_AllowAnonimus = true;
    }

    ~CUAServer()
    {
        std::for_each(m_Logins.begin(), m_Logins.end(), [](auto
&item)
        {
            UA_String_delete(&(item.username));
            UA_String_delete(&(item.password));
        });
        UA_NodeId_clear(&m_rootFolderNodeId);
        UA_Server_delete(server);
    }

    void CreateNodes()
    {
        CItem *pItem = &gGlobal.m_root;

        std::string rootnodename = "Каналы";
        UA_ObjectAttributes          objectAttr
UA_ObjectAttributes_default;

        // добавить верхний уровень "Каналы"
        UA_NodeId_init(&m_rootFolderNodeId);
        UA_NodeId      NodeId      =      UA_NODEID_STRING_ALLOC(m_ns,
rootnodename.c_str());
        UA_QualifiedName OpQName = UA_QUALIFIEDNAME_ALLOC(m_ns,
rootnodename.c_str());
        UA_Server_addObjectNode(          server,          NodeId,
UA_NODEID_NUMERIC(0,          UA_NS0ID_OBJECTSFOLDER),
UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),

```

```

                                OpQName,  UA_NODEID_NUMERIC(0,
UA_NS0ID_FOLDERTYPE), objectAttr, NULL, &m_rootFolderNodeId);

    pItem->m_NodeId = m_rootFolderNodeId;
    m_ItemsMap[rootnodename] = pItem;

    UA_QualifiedName_clear(&OpQName);
    UA_NodeId_clear(&NodeId);

    MaxTagNodes = protector.Init();
    if (MaxTagNodes > 30)
    {
        protector.StartChecking();
        std::string str = "MaxTagNodes - " +
std::to_string(MaxTagNodes);
        printf(""); // типа flush
        UA_LOG_INFO(&PlogLogger, UA_LOGCATEGORY_USERLAND,
str.c_str());
    }
    TagNodesCount = 0;

    for (imItem it = pItem->begin(), it_end = pItem->end(); it
!= it_end; ++it)
    {
        CreateUAItems(it->second, &m_rootFolderNodeId);
    }
}

bool CreateCItem(UA_NodeId* pParentNode,
std::vector<std::string> param)
{
    std::string parentName = param[0];
    std::string type = param[1];

    CItem* item = m_ItemsMap[parentName];

```

```

        if (!item)
        {
            UA_LOG_ERROR(&PlogLogger,          UA_LOGCATEGORY_SERVER,
"Failed get CItem, CreateCItem error");
            return false;
        }

        if (atoi(type.c_str()) == IT_CHAN)
        {
            CChannel* channel = new CChannel();

            std::string name = param[2];
            std::string typeProtocol = param[3];
            std::string typeChannel = param[4];
            std::string timeout = param[5];
            std::string quanRetry = param[6];
            std::string timeBusy = param[7];

            if (atoi(typeChannel.c_str()) == TYPE_CHAN_TCP)
            {
                std::string ip = param[8];
                std::string port = param[9];
                std::string timeoutConnect = param[10];

                channel->m_szName = name;
                channel->m_TypeProtocol =
(enTypeProtocol)atoi(typeProtocol.c_str());
                channel->m_TypeChannel =
(enTypeChannel)atoi(typeChannel.c_str());
                channel->m_Timeout = atoi(timeout.c_str());
                channel->m_QuanRetry = atoi(quanRetry.c_str());
                channel->m_TimeBusy = atoi(timeBusy.c_str());
                CPortIO_TCP* pPortIO_TCP = new CPortIO_TCP();
                channel->m_pPortIO = pPortIO_TCP;
                pPortIO_TCP->m_IP = atoi(ip.c_str());
            }
        }
    }
}

```

```

        pPortIO_TCP->m_TCP_Port = atoi(port.c_str());
        pPortIO_TCP->m_TimeoutConnect
atoi(timeoutConnect.c_str());
    }

    item->AppendChild(channel);
    if (!CreateUAItems(channel, pParentNode))
    {
        return false;
    }
}

if (atoi(type.c_str()) == IT_GROUP)
{
    std::string name = param[2];
    std::string updateRate = param[3];
    std::string MU_updateRate = param[4];

    CGroup* group = new CGroup();
    group->m_szName = name;
    group->m_UpdateRate = atoi(updateRate.c_str());
    group->m_MU_UpdateRate
(enMU_UpdateRate)atoi(MU_updateRate.c_str());

    item->AppendChild(group);
    if (!CreateUAItems(group, pParentNode))
    {
        return false;
    }
}

if (atoi(type.c_str()) == IT_TAG)
{
    std::string name = param[2];
    std::string regType = param[3];

```

```

std::string addr = param[4];
std::string valType = param[5];
std::string byteOrder = param[6];
std::string bRead = param[7];
std::string bWrite = param[8];
std::string bWriteGroup = param[9];
std::string regsInSegment = param[10];
std::string segmentPos = param[11];
std::string pTransform = param[12];
std::string transformType = param[13];
std::string pEmul = param[14];
std::string emulType = param[15];

CDataTag* tag = new CDataTag();
tag->Init();
tag->m_szName = name;
tag->m_RegType =
en_ModbusRegType(atoi(regType.c_str()));
tag->m_Addr = (unsigned short)atoi(addr.c_str());
tag->m_ValType =
en_ModbusValType(atoi(valType.c_str()));
tag->m_ByteOrder = (unsigned
short)atoi(byteOrder.c_str());
tag->m_bRead = atoi(bRead.c_str());
tag->m_bWrite = atoi(bWrite.c_str());
tag->m_bWriteGroup = atoi(bWriteGroup.c_str());
tag->m_RegsInSegment = (unsigned
short)atoi(regsInSegment.c_str());
tag->m_SegmentPos = (unsigned
short)atoi(segmentPos.c_str());
tag->m_pTransform =
CDataTag::New_Transform(en_TransformType(atoi(pTransform.c_str()))
);
tag->m_TransformType =
en_TransformType(atoi(transformType.c_str()));

```

```

tag->m_pEmul                                     =
CDATag::New_Emul(en_EmulType(atoi(pEmul.c_str())));
tag->m_EmulType = en_EmulType(atoi(emulType.c_str()));

item->AppendChild(tag);
if (!CreateUAItems(tag, pParentNode))
{
    return false;
}

return true;
}

bool EditCItem(std::vector<std::string> param)
{
    std::string parentName = param[0];
    std::string type = param[1];
    std::string name = param[2];

    CItem* item = m_ItemsMap[parentName];
    if (!item)
    {
        UA_LOG_ERROR(&PlogLogger,                UA_LOGCATEGORY_SERVER,
"Failed get CItem, EditCItem error");
        return false;
    }

    if (atoi(typeChannel.c_str()) == TYPE_CHAN_TCP)
    {
        if (param.size() != 11)
        {
            UA_LOG_ERROR(&PlogLogger,
UA_LOGCATEGORY_SERVER, "Failed add Channel-TCP, EditCItem error");

```

```

        return false;
    }

    std::string ip = param[8];
    std::string port = param[9];
    std::string timeoutConnect = param[10];

    channel->m_TypeProtocol =
(enTypeProtocol)atoi(typeProtocol.c_str());
    channel->m_TypeChannel =
(enTypeChannel)atoi(typeChannel.c_str());
    channel->m_Timeout = atoi(timeout.c_str());
    channel->m_QuanRetry = atoi(quanRetry.c_str());
    channel->m_TimeBusy = atoi(timeBusy.c_str());
    CPortIO_TCP* pPortIO_TCP = new CPortIO_TCP();
    channel->m_pPortIO = pPortIO_TCP;
    pPortIO_TCP->m_IP = atoi(ip.c_str());
    pPortIO_TCP->m_TCP_Port = atoi(port.c_str());
    pPortIO_TCP->m_TimeoutConnect =
atoi(timeoutConnect.c_str());
    }
}

if (atoi(type.c_str()) == IT_DEV)
{
    if (param.size() != 9)
    {
        UA_LOG_ERROR(&PlogLogger,    UA_LOGCATEGORY_SERVER,
"Failed add Device, EditCItem error");
        return false;
    }

    std::string addr = param[3];
    std::string bExchange = param[4];
    std::string bEmulation = param[5];

```

```

        std::string maxRegsInReqv = param[6];
        std::string maxDiscretsInReqv = param[7];
        std::string maxFragmentation = param[8];

        CDevice* device = (CDevice*)item;
        device->m_szName = name;
        std::vector<unsigned char> addrVec(addr.data(),
addr.data() + addr.length() + 1);
        device->m_Addr = addrVec.front();
        device->m_bExchange = atoi(bExchange.c_str());
        device->m_bEmulation = atoi(bEmulation.c_str());
        std::vector<unsigned char>
maxRegsInReqvVec(maxRegsInReqv.data(), maxRegsInReqv.data() +
maxRegsInReqv.length() + 1);
        device->m_MaxRegsInReqv = maxRegsInReqvVec.front();
        std::vector<unsigned char>
maxDiscretsInReqvVec(maxDiscretsInReqv.data(),
maxDiscretsInReqv.data() + maxDiscretsInReqv.length() + 1);
        device->m_MaxDiscretsInReqv =
maxDiscretsInReqvVec.front();
        std::vector<unsigned char>
maxFragmentationVec(maxFragmentation.data(),
maxFragmentation.data() + maxFragmentation.length() + 1);
        device->m_MaxFragmentation =
maxFragmentationVec.front();
    }

    if (atoi(type.c_str()) == IT_GROUP)
    {
        if (param.size() != 5)
        {
            UA_LOG_ERROR(&PlogLogger, UA_LOGCATEGORY_SERVER,
"Failed add Group, EditCItem error");
            return false;
        }
    }

```

```

std::string updateRate = param[3];
std::string MU_updateRate = param[4];

CGroup* group = (CGroup*)item;
group->m_szName = name;
group->m_UpdateRate = atoi(updateRate.c_str());
group->m_MU_UpdateRate =
(enMU_UpdateRate)atoi(MU_updateRate.c_str());
}

if (atoi(type.c_str()) == IT_TAG)
{
    if (param.size() != 16)
    {
        UA_LOG_ERROR(&PlogLogger,    UA_LOGCATEGORY_SERVER,
"Failed add Tag, EditCItem error");
        return false;
    }

    std::string regType = param[3];
    std::string addr = param[4];
    std::string valType = param[5];
    std::string byteOrder = param[6];
    std::string bRead = param[7];
    std::string bWrite = param[8];
    std::string bWriteGroup = param[9];
    std::string regsInSegment = param[10];
    std::string segmentPos = param[11];
    std::string pTransform = param[12];
    std::string transformType = param[13];
    std::string pEmul = param[14];
    std::string emulType = param[15];

    CDataTag* tag = (CDataTag*)item;

```

```

        tag->Init();
        tag->m_szName = name;
        tag->m_RegType =
en_ModbusRegType(atoi(regType.c_str()));
        tag->m_Addr = (unsigned short)atoi(addr.c_str());
        tag->m_ValType =
en_ModbusValType(atoi(valType.c_str()));
        std::vector<unsigned char>
byteOrderVec(byteOrder.data(), byteOrder.data() +
byteOrder.length() + 1);
        tag->m_ByteOrder = byteOrderVec.front();
        tag->m_bRead = atoi(bRead.c_str());
        tag->m_bWrite = atoi(bWrite.c_str());
        tag->m_bWriteGroup = atoi(bWriteGroup.c_str());
        tag->m_RegsInSegment = (unsigned
short)atoi(regsInSegment.c_str());
        tag->m_SegmentPos = (unsigned
short)atoi(segmentPos.c_str());
        tag->m_pTransform =
CDATag::New_Transform(en_TransformType(atoi(pTransform.c_str())
));
        tag->m_TransformType =
en_TransformType(atoi(transformType.c_str()));
        tag->m_pEmul =
CDATag::New_Emul(en_EmulType(atoi(pEmul.c_str())));
        tag->m_EmulType = en_EmulType(atoi(emulType.c_str()));
    }

    return true;
}

bool CreateUATags(std::vector<UA_NodeId> vecNodeId,
std::vector<CItem*> vecItem)
{
    bool br = false;

```

```

for (int i = 0; i < vecNodeId.size(); i++)
{
    for (auto it = vecItem[i]->m_vChildren.begin(); it !=
vecItem[i]->m_vChildren.end(); it++)
    {
        TagNodesCount++;
        if (TagNodesCount <= MaxTagNodes)
        {
            std::string strFullItemName;
            br = it->second->GetFullName(strFullItemName);

            // Нода
            UA_NodeId attributeNode;
            UA_NodeId_init(&attributeNode);

            CBaseTag*          DAtag          =
reinterpret_cast<CBaseTag*>(it->second);

            // Атрибуты ноды
            UA_VariableAttributes      attributeAttr      =
UA_VariableAttributes_default;
            attributeAttr.displayName      =
UA_LOCALIZEDTEXT_ALLOC("ru-RU", strFullItemName.c_str());
            attributeAttr.description      =
UA_LOCALIZEDTEXT_ALLOC("ru-RU", strFullItemName.c_str());
            // Доступ к чтению и записи
            UA_Byte accessLevel = DAtag->m_accessRights;

            attributeAttr.accessLevel = accessLevel;
            attributeAttr.userAccessLevel = accessLevel;

            // Запись значения атрибута

            attributeAttr.dataType = DAtag->m_nativeType-

```

```

>typeId;

        UA_StatusCode quality;
        DATag->GetValue(attributeAttr.value, quality);

        UA_NodeId                NodeId                =
UA_NODEID_STRING_ALLOC(m_ns, strFullItemName.c_str());
        UA_QualifiedName          QName                =
UA_QUALIFIEDNAME_ALLOC(m_ns, strFullItemName.c_str());

        UA_DataSource nodeDataSource;
        nodeDataSource.read = readNodeValue;
        nodeDataSource.write = writeNodeValue;

        UA_StatusCode                res                =
UA_Server_addDataSourceVariableNode(server, NodeId, vecNodeId[i],
        UA_NODEID_NUMERIC(0,
UA_NS0ID_HASCOMPONENT), QName,
        UA_NODEID_NUMERIC(0,
UA_NS0ID_BASEDATAVARIABLETYPE), attributeAttr, nodeDataSource,
        DATag, &attributeNode);

        if (res != UA_STATUSCODE_GOOD)
        {
            UA_LOG_ERROR(&PlogLogger,
UA_LOGCATEGORY_USERLAND, "CDACallback::DefinedDAItem    <%s>
UA_Server_addDataSourceVariableNode                FAILED                %s",
strFullItemName.c_str(), UA_StatusCode_name(res));
            UA_NodeId_clear(&NodeId);
            UA_QualifiedName_clear(&QName);

            UA_NodeId_clear(&attributeNode);
            return false;
        }

```

```

        it->second->m_NodeId = attributeNode;
        m_ItemsMap[strFullItemName] = it->second;

        UA_NodeId_clear(&NodeId);
        UA_QualifiedName_clear(&QName);

        UA_NodeId_clear(&attributeNode);

        UA_LOG_DEBUG(&PlogLogger,
UA_LOGCATEGORY_USERLAND,      "CDACallback::DefinedDAItem      <%s>",
strFullItemName.c_str());
    }
    }
    }
    return br;
}

bool CreateUAItems(CItem *pItem, UA_NodeId *pParentNode)
{
    bool br = false;
    if(pItem->m_ItemType < IT_TAG)
    {
        std::string nodename = pItem->m_szName;
        std::string strFullItemName;
        br = pItem->GetFullName(strFullItemName);

        UA_ObjectAttributes      objectAttr      =
UA_ObjectAttributes_default;
        UA_NodeId NodeId;
        UA_NodeId_init(&NodeId);
        UA_NodeId  OpNodeId  =  UA_NODEID_STRING_ALLOC(m_ns,
strFullItemName.c_str());
        UA_QualifiedName      OpQName      =
UA_QUALIFIEDNAME_ALLOC(m_ns, strFullItemName.c_str());
        UA_StatusCode res = UA_Server_addObjectNode( server,

```

```

OpNodeId, *pParentNode,

UA_NODEID_NUMERIC(0,
UA_NS0ID_ORGANIZES), OpQName,

UA_NODEID_NUMERIC(0,
UA_NS0ID_FOLDERTYPE), objectAttr, NULL, &NodeId);

    if (res != UA_STATUSCODE_GOOD)
    {
        UA_LOG_ERROR(&PlogLogger, UA_LOGCATEGORY_USERLAND,
"Failed to add %s", strFullItemName.c_str());
        return false;
    }

    pItem->m_NodeId = NodeId;
    m_ItemsMap[strFullItemName] = pItem;

    UA_QualifiedName_clear(&OpQName);
    UA_NodeId_clear(&OpNodeId);

    for (imItem it = pItem->begin(), it_end = pItem->end();
it != it_end; it++)
    {
        br = CreateUAItems(it->second, &NodeId);
    }
    if(pItem->m_ItemType == IT_DEV)
    {
        CDevice* pDev = (CDevice*)pItem;
        UA_Variant bEmulation;
        UA_Variant_init(&bEmulation);
        UA_Variant_setScalar(&bEmulation, &pDev->m_bEmulation, &UA_TYPES[UA_TYPES_BOOLEAN]);
        pDev->m_tag_bEmulation.SetValue(bEmulation);

        UA_Variant bExchange;
        UA_Variant_init(&bExchange);
        UA_Variant_setScalar(&bExchange, &pDev->m_tag_bExchange, &UA_TYPES[UA_TYPES_BOOLEAN]);
    }
}

```

```

>m_bExchange, &UA_TYPES[UA_TYPES_BOOLEAN]);
    pDev->m_tag_bExchange.SetValue(bExchange);

    CreateUAItemsService(&pDev->m_tag_bEmulation,
&NodeId);

    CreateUAItemsService(&pDev->m_tag_bExchange,
&NodeId);

    CreateUAItemsService(&pDev->m_tag_bConnection,
&NodeId);

    CreateUAItemsService(&pDev->m_tag_ResponseTime,
&NodeId);

    CreateUAItemsService(&pDev->m_tag_ScanTime,
&NodeId);
    }
}
else if(pItem->m_ItemType == IT_TAG)
{
    TagNodesCount++;
    if (TagNodesCount <= MaxTagNodes)
    {
        std::string strFullItemName;
        br = pItem->GetFullName(strFullItemName);

        // Нода
        UA_NodeId attributeNode;
        UA_NodeId_init(&attributeNode);

        CBaseTag*          DAtag          =
reinterpret_cast<CBaseTag*>(pItem);

        // Атрибуты ноды
        UA_VariableAttributes attributeAttr =
UA_VariableAttributes_default;

        attributeAttr.displayName =
UA_LOCALIZEDTEXT_ALLOC("ru-RU", strFullItemName.c_str());

```

```

        attributeAttr.description                                     =
UA_LOCALIZEDTEXT_ALLOC("ru-RU", strFullItemName.c_str());
        // Доступ к чтению и записи
        UA_Byte accessLevel = DATag->m_accessRights;

        attributeAttr.accessLevel = accessLevel;
        attributeAttr.userAccessLevel = accessLevel;

        // Запись значения атрибута

        attributeAttr.dataType      =      DATag->m_nativeType-
>typeId;

        UA_StatusCode quality;
        DATag->GetValue(attributeAttr.value, quality);

        UA_NodeId  NodeId  =  UA_NODEID_STRING_ALLOC(m_ns,
strFullItemName.c_str());
        UA_QualifiedName      QName      =
UA_QUALIFIEDNAME_ALLOC(m_ns, strFullItemName.c_str());

        UA_DataSource nodeDataSource = { 0 };
        nodeDataSource.read = readNodeValue;
        nodeDataSource.write = writeNodeValue;

        UA_StatusCode      res      =
UA_Server_addDataSourceVariableNode(server, NodeId, *pParentNode,
        UA_NODEID_NUMERIC(0,      UA_NS0ID_HASCOMPONENT),
        QName,
        UA_NODEID_NUMERIC(0,
UA_NS0ID_BASEDATAVARIABLETYPE),  attributeAttr,  nodeDataSource,
        DATag, &attributeNode);

        if (res != UA_STATUSCODE_GOOD)
        {

```

```

UA_LOG_ERROR(&PlogLogger,UA_LOGCATEGORY_USERLAND,"CDACallback::D
efineDAItem <%s> UA_Server_addDataSourceVariableNode FAILED %s",
strFullItemName.c_str(),
UA_StatusCode_name(res));

```

```

    UA_NodeId_clear(&NodeId);
    UA_QualifiedName_clear(&QName);

```

```

    UA_NodeId_clear(&attributeNode);
    return false;

```

```

}

```

```

pItem->m_NodeId = attributeNode;
m_ItemsMap[strFullItemName] = pItem;

```

```

UA_NodeId_clear(&NodeId);
UA_QualifiedName_clear(&QName);

```

```

UA_NodeId_clear(&attributeNode);

```

```

UA_LOG_DEBUG(&PlogLogger,UA_LOGCATEGORY_USERLAND,"CDACallback::D
efineDAItem <%s>", strFullItemName.c_str());
    }
}

```

```

    return br;

```

```

}

```

```

UA_StatusCode Run()
{
    return UA_Server_run(server, &running);
}

```

```

static void mIRegisterCallback(UA_Server *server, const
UA_NodeId *sessionId, void *sessionContext, const UA_NodeId
*nodeId,

                                void *nodeContext,
UA_UInt32 attributeId, UA_Boolean removed)
{

UA_LOG_INFO(&PlogLogger,UA_LOGCATEGORY_USERLAND,"monitoredItemRe
gisterCallback");

    CDataTag* pTag = reinterpret_cast<CDataTag*>(nodeContext);
    if (pTag)
    {
        pTag->bMonitored = !removed;
    }

}

void SetUAPort(unsigned int Port)
{
    if (Port > 0 && Port <= 65535) m_port = Port;
}

void AllowAnonimus(bool Allow)
{
    m_AllowAnonimus = Allow;
}

void SetLoginsPath(const std::string &path)
{
    m_LoginsPath = path;
}

```

```

void SetTrustedListDir(const std::string &path)
{
    m_trustedList_dir = path;
}

UA_Server* server;
UA_UInt16 m_ns;
std::map<std::string, CItem*> m_ItemsMap;
size_t TagNodesCount;

private:
    std::vector<UA_NodeId> vecNodeId;
    std::vector<CItem*> vecItem;

void LoadLogins()
{
    if (m_LoginsPath.empty()) return;
    try
    {
        std::ifstream File(m_LoginsPath);
        std::wstringstream wss;

        std::string str;
        while (std::getline(File, str))
        {
            std::istringstream istr(str);
            std::string token;
            std::vector<std::string> tokens;
            while (std::getline(istr, token, ';'))
            {
                tokens.push_back(token);
            }
            if(tokens.size() > 1)
            {
                for (int i = 0; i < tokens.size(); i += 2)

```

```

        {
            UA_UsernamePasswordLogin login;
            login.username =
UA_STRING_ALLOC(tokens[i].c_str());
            login.password = UA_STRING_ALLOC(tokens[i
+ 1].c_str());
            m_Logins.push_back(login);
        }
    }
}

}
catch (std::system_error& e)
{
    UA_LOG_ERROR(&PlogLogger,    UA_LOGCATEGORY_USERLAND,
"LoadLogins::Load    File    %s    error    %s    [%s]",
m_LoginsPath.c_str(),e.code().message().c_str(),e.what());
}
}

return fileContents;
}

const UA_NodeId* sessionId, void* sessionHandle,
const UA_NodeId* methodId, void* methodContext,
const UA_NodeId* objectId, void* objectContext,
size_t inputSize, const UA_Variant* input,
size_t outputSize, UA_Variant* output);

bool CreateMethods();

if (!protector.IsOK(uaserver.TagNodesCount)) return

```

```

UA_STATUSCODE_BADLICENSENOTAVAILABLE;
    CDATag* pTag = reinterpret_cast<CDATag*>(nodeContext);

    UA_Variant val;
    UA_Variant_init(&val);

    UA_Variant_copy(&data->value, &val);

    int Error = 0;
    unsigned int Count = 1;
    pTag->m_pParent->Write(nodeContext, &val, &Error);

    UA_Variant_clear(&val);

    return UA_STATUSCODE_GOOD;
}

bool CreateMethods()
{
    UA_StatusCode res = UA_STATUSCODE_BAD;

    UA_Argument inputArgument;
    UA_Argument_init(&inputArgument);
    inputArgument.description = UA_LOCALIZEDTEXT("en-US",
"String");
    inputArgument.name = UA_STRING("MyInput");
    inputArgument.dataType = UA_TYPES[UA_TYPES_STRING].typeId;
    inputArgument.valueRank = UA_VALUERANK_SCALAR;

    UA_Argument outputArgument;
    UA_Argument_init(&outputArgument);
    outputArgument.description = UA_LOCALIZEDTEXT("en-US",
"String");
    outputArgument.name = UA_STRING("MyOutput");
    outputArgument.dataType = UA_TYPES[UA_TYPES_STRING].typeId;

```

```

outputArgument.valueRank = UA_VALUERANK_SCALAR;

UA_MethodAttributes attrs = UA_MethodAttributes_default;
attrs.description = UA_LOCALIZEDTEXT("ru-RU", "Начать
мониторинг");
attrs.displayName = UA_LOCALIZEDTEXT("en-US",
"startExchange");
attrs.executable = true;
attrs.userExecutable = true;
UA_NodeId parentNode = UA_NODEID_STRING_ALLOC(uaserver.m_ns,
"Каналы");
UA_NodeId OpNodeIdStart =
UA_NODEID_STRING_ALLOC(uaserver.m_ns, "Каналы.startExchange");
UA_QualifiedName OpQNameStart =
UA_QUALIFIEDNAME_ALLOC(uaserver.m_ns, "Каналы.startExchange");
res = UA_Server_addMethodNode(uaserver.server, OpNodeIdStart,
parentNode, UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
OpQNameStart, attrs, &startExchange,
1, &inputArgument, 1, &outputArgument, NULL, NULL);
if (res != UA_STATUSCODE_GOOD)
{
    UA_LOG_WARNING(&PlogLogger, UA_LOGCATEGORY_USERLAND,
"Failed to add Каналы.startExchange");
}

attrs.description = UA_LOCALIZEDTEXT("ru-RU", "Остановить
мониторинг");
attrs.displayName = UA_LOCALIZEDTEXT("en-US", "stopExchange");
UA_NodeId OpNodeIdStop = UA_NODEID_STRING_ALLOC(uaserver.m_ns,
"Каналы.stopExchange");
UA_QualifiedName OpQNameStop =
UA_QUALIFIEDNAME_ALLOC(uaserver.m_ns, "Каналы.stopExchange");
res = UA_Server_addMethodNode(uaserver.server, OpNodeIdStop,
parentNode, UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
OpQNameStop, attrs, &stopExchange,

```

Изм	Лист	№ докум.	Подпись	Дата

```

        1, &inputArgument, 1, &outputArgument, NULL, NULL);
if (res != UA_STATUSCODE_GOOD)
{
    UA_LOG_WARNING(&PlogLogger,          UA_LOGCATEGORY_USERLAND,
"Failed to add Каналы.stopExchange");
}

    attrs.description      =    UA_LOCALIZEDTEXT("ru-RU",    "Добавить
ноду");
    attrs.displayName = UA_LOCALIZEDTEXT("en-US", "addNode");
    UA_NodeId OpNodeIdAdd = UA_NODEID_STRING_ALLOC(uaserver.m_ns,
"Каналы.addNode");
    UA_QualifiedName          OpQNameAdd          =
UA_QUALIFIEDNAME_ALLOC(uaserver.m_ns, "Каналы.addNode");
    res = UA_Server_addMethodNode(uaserver.server, OpNodeIdAdd,
parentNode, UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
OpQNameAdd, attrs, &addNode,
    1, &inputArgument, 1, &outputArgument, NULL, NULL);
if (res != UA_STATUSCODE_GOOD)
{
    UA_LOG_WARNING(&PlogLogger,          UA_LOGCATEGORY_USERLAND,
"Failed to add Каналы.addNode");
}

    attrs.description = UA_LOCALIZEDTEXT("ru-RU", "Удалить ноду");
    attrs.displayName = UA_LOCALIZEDTEXT("en-US", "deleteNode");
    UA_NodeId          OpNodeIdDelete          =
UA_NODEID_STRING_ALLOC(uaserver.m_ns, "Каналы.deleteNode");
    UA_QualifiedName          OpQNameDelete          =
UA_QUALIFIEDNAME_ALLOC(uaserver.m_ns, "Каналы.deleteNode");
    res = UA_Server_addMethodNode(uaserver.server, OpNodeIdDelete,
parentNode, UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
OpQNameDelete, attrs, &deleteNode,
    1, &inputArgument, 1, &outputArgument, NULL, NULL);
if (res != UA_STATUSCODE_GOOD)

```

```

{
    UA_LOG_WARNING(&PlogLogger,          UA_LOGCATEGORY_USERLAND,
"Failed to add Каналы.deleteNode");
}

    attrs.description      =    UA_LOCALIZEDTEXT("ru-RU",    "Изменить
ноду");
    attrs.displayName = UA_LOCALIZEDTEXT("en-US", "editNode");
    UA_NodeId OpNodeIdEdit = UA_NODEID_STRING_ALLOC(uaserver.m_ns,
"Каналы.editNode");
    UA_QualifiedName          OpQNameEdit          =
UA_QUALIFIEDNAME_ALLOC(uaserver.m_ns, "Каналы.editNode");
    res = UA_Server_addMethodNode(uaserver.server, OpNodeIdEdit,
    parentNode, UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
    OpQNameEdit, attrs, &editNode,
    1, &inputArgument, 1, &outputArgument, NULL, NULL);
    if (res != UA_STATUSCODE_GOOD)
    {
        UA_LOG_WARNING(&PlogLogger,          UA_LOGCATEGORY_USERLAND,
"Failed to add Каналы.editNode");
    }

    UA_NodeId_clear(&parentNode);
    UA_QualifiedName_clear(&OpQNameStart);
    UA_NodeId_clear(&OpNodeIdStart);
    UA_QualifiedName_clear(&OpQNameStop);
    UA_NodeId_clear(&OpNodeIdStop);
    UA_NodeId_clear(&OpNodeIdAdd);
    UA_QualifiedName_clear(&OpQNameAdd);
    UA_NodeId_clear(&OpNodeIdDelete);
    UA_QualifiedName_clear(&OpQNameDelete);
    UA_NodeId_clear(&OpNodeIdEdit);
    UA_QualifiedName_clear(&OpQNameEdit);

    return res;

```

```

}

static UA_StatusCode startExchange(UA_Server* server,
    const UA_NodeId* sessionId, void* sessionHandle,
    const UA_NodeId* methodId, void* methodContext,
    const UA_NodeId* objectId, void* objectContext,
    size_t inputSize, const UA_Variant* input,
    size_t outputSize, UA_Variant* output)
{
    UA_String* inputStr = (UA_String*)input->data;
    UA_String outputStr = UA_STRING_ALLOC("Success");
    if (inputStr->length > 0)
    {
        gGlobal.StartExchange();
    }
    UA_Variant_setScalarCopy(output, &outputStr,
        &UA_TYPES[UA_TYPES_STRING]);

    UA_LOG_INFO(&PlogLogger, UA_LOGCATEGORY_SERVER, "startExchange
was called");
    return UA_STATUSCODE_GOOD;
}

static UA_StatusCode stopExchange(UA_Server* server,
    const UA_NodeId* sessionId, void* sessionHandle,
    const UA_NodeId* methodId, void* methodContext,
    const UA_NodeId* objectId, void* objectContext,
    size_t inputSize, const UA_Variant* input,
    size_t outputSize, UA_Variant* output)
{
    UA_String* inputStr = (UA_String*)input->data;
    UA_String outputStr = UA_STRING_ALLOC("Success");
    if (inputStr->length > 0)
    {
        gGlobal.StopExchange();
    }
}

```

```

    }

    UA_Variant_setScalarCopy(output,                                &outputStr,
    &UA_TYPES[UA_TYPES_STRING]);

    UA_LOG_INFO(&PlogLogger, UA_LOGCATEGORY_SERVER, "stopExchange
was called");
    return UA_STATUSCODE_GOOD;
}

static UA_StatusCode addNode(UA_Server* server,
    const UA_NodeId* sessionId, void* sessionHandle,
    const UA_NodeId* methodId, void* methodContext,
    const UA_NodeId* objectId, void* objectContext,
    size_t inputSize, const UA_Variant* input,
    size_t outputSize, UA_Variant* output)
{

}

static UA_StatusCode deleteNode(UA_Server* server,
    const UA_NodeId* sessionId, void* sessionHandle,
    const UA_NodeId* methodId, void* methodContext,
    const UA_NodeId* objectId, void* objectContext,
    size_t inputSize, const UA_Variant* input,
    size_t outputSize, UA_Variant* output)
{

}

static UA_StatusCode editNode(UA_Server* server,
    const UA_NodeId* sessionId, void* sessionHandle,
    const UA_NodeId* methodId, void* methodContext,
    const UA_NodeId* objectId, void* objectContext,
    size_t inputSize, const UA_Variant* input,
    size_t outputSize, UA_Variant* output)

```

```

{

}

int main(int argc, char *argv[])
{
    SetStopHandler();

    std::vector<std::string> raw_args(argv, argv + argc);
    plog::Severity severity = plog::info;

    FileType ft = FileType::csv;

    if (raw_args.size() > 1)
    {
        for (auto it = raw_args.begin() + 1; it != raw_args.end();
++it)
        {
            if (it->substr(0, 1) == "-")
            {

                if (*it == "-help")
                {
                    printHelp();
                    return UA_STATUSCODE_GOOD;
                }

            }

        }
    }
    else
    {
        printHelp();
        return UA_STATUSCODE_GOOD;
    }
}

```

```

if (ft == FileType::csv)
{
    CCsvLoader importer;

    if (gGlobal.m_CfgFileName.empty())
    {
        gGlobal.m_CfgFileName = raw_args[1];
    }

    bool importres = importer.Load(gGlobal.m_CfgFileName);
    if (!importres)
    {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
"Failed call Load, exit");
        return EXIT_FAILURE;
    }

    std::vector<std::string> vecParam;
    if (uaserver.readParametrStart(gGlobal.m_CfgFileName,
vecParam) != 0)
    {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
"Failed call readParametrStart, exit");
        return EXIT_FAILURE;
    }

    if (vecParam[0] == "")
    {
        uaserver.SetUAPort(4840);
    }
    else

```

```

{
    uaserver.SetUAPort(atoi(vecParam[0].c_str()));
}

if (vecParam[1] == "")
{
    severity = plog::info;
    std::string s = "LOG level - INFO";
    UA_LOG_INFO(UA_Log_Stdout,          UA_LOGCATEGORY_SERVER,
s.c_str());
}
else
{
    if (vecParam[1] == "none") severity = plog::none;
    if (vecParam[1] == "fatal") severity = plog::fatal;
    if (vecParam[1] == "error") severity = plog::error;
    if (vecParam[1] == "warning") severity = plog::warning;
    if (vecParam[1] == "info") severity = plog::info;
    if (vecParam[1] == "debug") severity = plog::debug;
    if (vecParam[1] == "verbose") severity = plog::verbose;
    std::string s = "LOG level - " + vecParam[1];
    UA_LOG_INFO(UA_Log_Stdout,          UA_LOGCATEGORY_SERVER,
s.c_str());
}

if (vecParam[2] == "")
{
    uaserver.AllowAnonimus(true);
}
else
{
    if (vecParam[2] == "true")
uaserver.AllowAnonimus(true);
    if (vecParam[2] == "false")
uaserver.AllowAnonimus(false);
}

```

```

    }
    if (vecParam[3] != "")
    {
        uaserver.SetTrustedListDir(vecParam[5]);
    }

}

fs::create_directories(path);
path += "OPCUAServer.log";
plog::init(severity, path.c_str(), 10000000, 10);

plog::ColorConsoleAppender<plog::TxtFormatter>
consoleAppender;
plog::get()->addAppender(&consoleAppender);

UA_StatusCode res = uaserver.Create();
if (res != UA_STATUSCODE_GOOD)
{
    UA_LOG_ERROR(&PlogLogger, UA_LOGCATEGORY_SERVER, "Failed
call Create, exit");
    return EXIT_FAILURE;
}

auto start = std::chrono::system_clock::now();

uaserver.CreateNodes();

auto end = std::chrono::system_clock::now();
std::chrono::duration<double> elapsed_seconds = end - start;
std::string info = "Add variable time (seconds): " +
std::to_string(elapsed_seconds.count());
UA_LOG_INFO(&PlogLogger, UA_LOGCATEGORY_SERVER, info.c_str());

```

```

    res = CreateMethods();

    if (res != UA_STATUSCODE_GOOD)
    {
        UA_LOG_ERROR(&PlogLogger, UA_LOGCATEGORY_SERVER, "Failed
call CreateMethods, exit");
        return EXIT_FAILURE;
    }

    UA_StatusCode retval = uaserver.Run();

    return  retval  ==  UA_STATUSCODE_GOOD  ?  EXIT_SUCCESS  :
EXIT_FAILURE;
}

```

Приложение В

(обязательное)

OPC-сервер протоколов MODBUS спецификации Unified Architecture

Презентация

					<i>ПГУ 1.090301.04.001 ПЗ</i>	Лист
						104
Изм	Лист	№ докум.	Подпись	Дата		

ТЕМА ВКР: «ОПС-СЕРВЕР ПРОТОКОЛОВ MODBUS СПЕЦИФИКАЦИИ UNIFIED ARCHITECTURE»

Автор работы: студент группы 21BBB1 Вартанов А.В.
Научный руководитель: к. т. н. Бычков А. С.

Пенза, 2025

Цели и задачи работы

Целью работы является разработка ОПС-сервера
протоколов MODBUS спецификации Unified
Architecture

Задачами работы являются:

1. Изучение аналогов;
2. Анализ предметной области;
3. Проектирование архитектуры приложения;
4. Программная реализация ОПС UA-сервера и реализация работы MODBUS TCP-клиента;
5. Тестирование программы.

1

Актуальность работы

- Требование к следованию стандарта "Industry 4.0"
- Проблема интеграции старых устройств
- Отсутствие безопасности в Modbus TCP
- Курс нашей страны на импортозамещение
- У аналоговых решений нет исходного кода



2

Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист

105

Аналоги



Kepware KEPServerEX



Prosys OPC UA MODBUS



Ignition OPC UA Gateway

3

Выбор среды разработки и инструментов

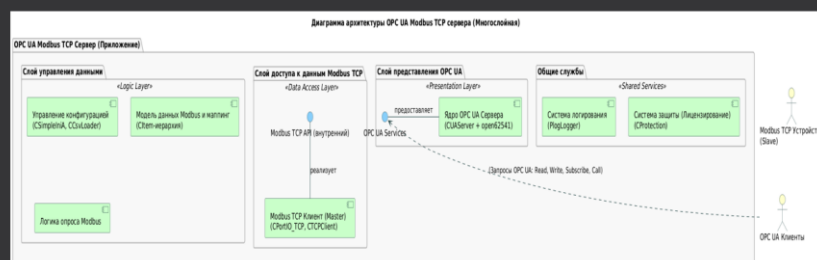


open62541



4

Диаграмма архитектуры сервера



5

Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист
106

6

Интерфейс для проверки работы
клиентской части OPC UA

8

Демонстрация работы OPC UA-Server

Набор тегов для тестирования

Конфигуратор эмулятора

Запуск эмулятора

статичный тег до редактирования

статичный тег после редактирования

[illegible]

The screenshot displays the 'Проверка входов' (Input Check) window, which is part of a larger application for verifying circuit inputs and outputs. The window is titled 'Проверка входов' and contains two main panes.

Left Pane: Результаты проверки входов (100000)

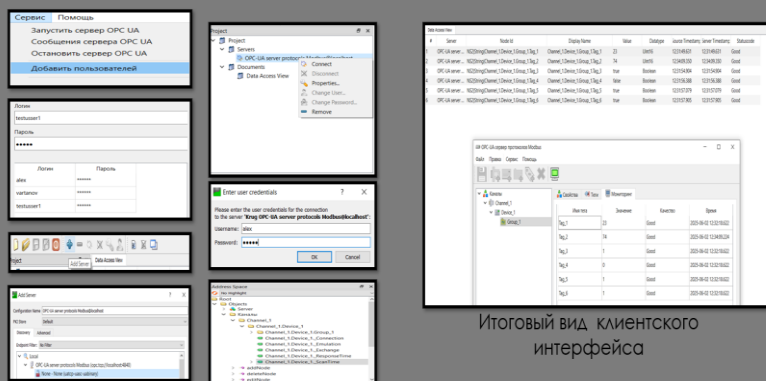
This pane shows a table of input data. The table has columns for 'Вход' (Input), 'Выход' (Output), and 'Состояние' (State). The data is organized into rows, with the first row labeled 'Вход' and the subsequent rows labeled 'Выход' and 'Состояние'. The table is currently empty, showing only the column headers.

Right Pane: Результаты проверки выходов (80000 Cell Values)

This pane shows a table of output data. The table has columns for 'Выход' (Output) and 'Состояние' (State). The data is organized into rows, with the first row labeled 'Выход' and the subsequent rows labeled 'Состояние'. The table is currently empty, showing only the column headers.

The application is running on a Windows 10 desktop environment, as evidenced by the taskbar and window title bar.

Демонстрация работы OPC UA-client



Итоговый вид клиентского
интерфейса

12

Заключение

В ВКР был успешно разработан OPC UA-сервер протокола TCP Modbus, а также:

- ✓ успешно реализован полный цикл шлюзования данных: чтение, запись и подписки;
- ✓ Обеспечен механизм безопасности (аутентификация, в будущем шифрование с использованием сертификатов);
- ✓ Подтверждена работоспособность и стабильность сервера в ходе комплексного тестирования;
- ✓ Продемонстрирована практическая значимость решения путем создания серверного GUI для тестирования и мониторинга работы решения;

Разработанное решение является гибким и экономически эффективным инструментом для модернизации промышленных систем.

13

Спасибо за внимание

Изм	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист

109