Транспорты AgavaSCADA/AgavaPLC

Редакция от 19.10.2026 · docs.kb-agava.ru

Содержание

1 Транспорты Modbus-RTU, Modbus-TCP 2
1.1 Узлы 2
1.1.1 Транспорт Modbus-RTU 2
1.1.2 Транспорт Modbus-TCP 2
1.1.3 Modbus регистр 2
1.2 Работа транспорта 3
1.2.1 Режим Master (ведущий) 3
1.2.2 Режим Slave (ведомый) 5
1.3 Оптимизации операций записи в Modbus 6
1.3.1 Модель команды и терминология 7
1.3.2 Этап 1. Дедупликация при постановке в очередь 7
1.3.3 Этап 2. Объединение команд записи (функции 15 и 16) 7
1.3.4 Ограничения 8
1.3.5 Примеры 8
2 Транспорт OPC-UA 8
2.1 Узлы 8
2.1.1 Транспорт OPC-UA 9
2.1.2 Группа тегов 10
2.1.3 Тег OPC-UA 11
2.2 Создание и настройка транспорта OPC-UA 12
2.3 Добавление параметров в Транспорт OPC-UA, работающий в режиме "Клиент" 12
3 Транспорт SNMP 13
3.1 Создание и настройка транспорта SNMP 13
3.1.1 Как определить версию SNMP для устройства 13
3.1.2 Авторизация в версиях SNMP v1 и SNMP v2c 14
3.1.3 Авторизация в версиях SNMP v3 14
3.2 Добавление параметров в Транспорт SNMP 14
4 Транспорты МЭК-60870 15
5 Транспорт MQTT 15
6 Транспорт OPC DA 16
7 Транспорт AgavaCloud 16

1 Транспорты Modbus-RTU, Modbus-TCP

Данные виды транспортов предназначены для приема и передачи данных по протоколу Modbus. Поддерживаются последовательные и Ethernet подключения.

1.1 Узлы

- `Транспорт Modbus-RTU` - транспорт, работающий по последовательным линиям связи.
- `Транспорт Modbus-TCP` - транспорт, работающий по ethernet.
- `Modbus регистр` - источник, хранящий значение регистра.

1.1.1 Транспорт Modbus-RTU

Узел, представляющий транспорт Modbus, работающий по последовательным линиям связи RS-485, RS-422, RS-232.

1.1.1.1 Доступные дочерние узлы

- Группа узлов.
- Modbus регистр.

1.1.1.2 Свойства

1.1.2 Транспорт Modbus-TCP

Узел, представляющий транспорт Modbus, работающий по линиям связи Ethernet.

1.1.2.1 Доступные дочерние узлы

- Группа узлов.
- Modbus регистр.

1.1.2.2 Свойства

1.1.3 Modbus регистр

Узел-источник, хранящий значение регистра.

1.1.3.1 Доступные дочерние узлы

Нет

1.1.3.2 Свойства

Наименование свойства	Идентификатор	Тип	Доступ	Описание
Устройство	DeviceAddr	INT	Чтение / запись	
Функция чтения	ReadFunction	INT	Чтение / запись	
Регистр чтения	ReadRegister	INT	Чтение / запись	
Размер регистра	ParamSize	INT	Чтение / запись	
Приоритет	Priority	INT	Чтение / запись	

Функция записи	WriteFunction	INT	Чтение / запись	
Регистр записи	WriteRegister	INT	Чтение / запись	
Тип значения	ValueType	INT	Чтение / запись	
Порядок байт	ByteOrder	INT	Чтение / запись	
Операции после чтения	OperationsOnRead	NODESARRAY	Чтение / запись	
Операции перед записью	OperationsOnWrite	NODESARRAY	Чтение / запись	
Узел для чтения/записи	SourceId	NODE	Чтение / запись	

Устройство

Функция чтения

Регистр чтения

Размер регистра

Приоритет

Функция записи

Регистр записи

Тип значения

Порядок байт

Операции после чтения

Операции перед записью

Узел для чтения/записи

1.2 Работа транспорта

Транспорты могут работать в двух режимах:

- Ведущий (Master).
- Ведомый (Slave).

1.2.1 Режим Master (ведущий)

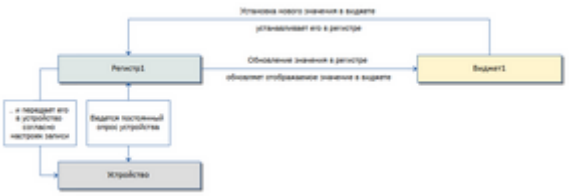


Схема работы Modbus мастера

В режиме Master транспорт ведет обмен со связанными устройствами - запрашивает значения определенных регистров, записывает в регистры устройств значения, получаемые из виджетов или других узлов проекта. В этом режиме применяется оптимизация команд записи.

Схема работы транспорта в режиме Master приведена на схеме.

Работа транспорта Modbus в этом режиме строится на следующих принципах:

- Обмен данными ведется циклами «Запрос — Ответ».
- Транспорт в этом режиме - Ведущий, то есть единственное устройство в сети, которое инициирует передачу данных.
- Ведомые (Slave) устройства никогда не начинают обмен самостоятельно, они только отвечают на запросы.

1.2.1.1 Цикл «Запрос — Ответ»

Работа транспорта в этом режиме организована как последовательные циклы:

1. Формирование запроса: Master создает кадр, куда входит: адрес ведомого, код функции и данные (с какого адреса, сколько регистров).
2. Отправка: Master передает кадр в шину (RS-485 или Ethernet).
3. Ожидание: Master включает таймер ожидания ответа.
4. Прием ответа: Если Slave с указанным адресом получил команду без ошибок, он отправляет ответный кадр.
5. Тайм-аут: Если за отведенное время ответа нет, Master фиксирует ошибку (связь потеряна) и переходит к опросу следующего устройства или повторяет запрос.

1.2.1.2 Управление таймингами

Master жестко контролирует паузы в сети:

- Между отправкой запроса и приемом ответа: Master выдерживает минимальную паузу на смену направления передачи (для RS-485).
- Таймаут ожидания ответа: Задается в настройках Master. Если Slave медленный, Master не будет ждать вечно — он прерывает ожидание ответа и переходит к следующей транзакции.
- Межкадровая задержка: Master выдерживает паузу настраиваемой длительности между отправкой разных запросов, чтобы Slave успел обработать команду.

1.2.1.3 Адресация

- Master отправляет запрос строго конкретному Slave (адрес от 1 до 247).
- Широковещательный адрес (0): Master может отправить команду всем Slave сразу (например, «запустить моторы»). В этом режиме ведомые не отвечают, чтобы не создать коллизию в шине. В данный момент не поддерживается.

1.2.1.4 Обработка ошибок

Master проверяет:

- Контрольную сумму ответа (CRC) (только `Modbus-RTU`).
- Код исключения: Не вернул ли Slave код ошибки (например, 0x02 — неверный адрес регистра).

1.2.2 Режим Slave (ведомый)

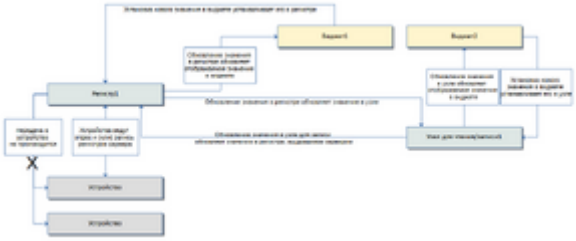


Схема работы Modbus slave

В режиме Slave транспорт ведет обмен со связанными устройствами в пассивном режиме - принимает запросы от ведущих и выдает ответы.

Схема работы транспорта в режиме Slave приведена на рисунке.

В режиме Slave транспорт Modbus работает как **сервер**, ожидающий входящие запросы от внешнего ведущего устройства (Master). Транспорт не инициирует обмен самостоятельно, а только обрабатывает команды, полученные от Master: отдаёт значения указанных регистров или записывает в свои регистры принятые данные. Полученные значения могут передаваться в виджеты или другие узлы проекта.

Работа транспорта Modbus в этом режиме строится на следующих принципах:

- Транспорт в этом режиме — **Ведомый (Slave)**, то есть отвечает только на запросы внешнего ведущего устройства.
- Обмен данными инициируется исключительно Master.
- Транспорт не опрашивает другие устройства и не формирует запросы самостоятельно.

1.2.2.1 Цикл «Запрос — Ответ»

Работа транспорта в этом режиме организована как обработка входящих запросов:

1. **Ожидание запроса:** Slave находится в режиме приёма и прослушивает шину (RS-485 или Ethernet).
2. **Приём запроса:** При получении кадра Slave проверяет, предназначен ли запрос ему (по адресу ведомого).
3. **Проверка целостности:** Slave вычисляет контрольную сумму (CRC для Modbus-RTU) и проверяет корректность структуры кадра.
4. **Обработка команды:** Если запрос адресован данному Slave и не содержит ошибок, транспорт выполняет требуемую операцию:
    - **Чтение:** извлекает значения из внутренних регистров (например, полученных от виджетов).
    - **Запись:** записывает принятые данные в указанные регистры.
5. **Формирование ответа:** Slave формирует ответный кадр, куда входит: адрес ведомого, код функции и запрошенные данные (для чтения) или подтверждение (для записи).
6. **Отправка ответа:** Slave передаёт ответ в шину.
7. **Обработка ошибок:** При обнаружении любой ошибки (неверная контрольная сумма, неверный адрес регистра, неверный код функции) Slave формирует **кадр исключения** с соответствующим кодом ошибки или игнорирует запрос (в зависимости от настроек).

1.2.2.2 Управление таймингами

В режиме Slave транспорт соблюдает временные параметры, заданные ведущим устройством:

- **Таймаут ожидания ответа:** Настраивается на стороне Master. Slave не контролирует время между запросами — он отвечает в рамках разумного времени, определяемого его производительностью.
- **Межкадровая задержка:** Slave использует этот интервал (3.5 символа для RTU) для определения границ кадров. Если пауза между байтами превышает 3.5 символа, Slave считает кадр завершённым и начинает его обработку.
- **Время обработки запроса:** Slave обрабатывает запрос и отправляет ответ. Задержка между приёмом запроса и отправкой ответа должна быть минимальной (обычно несколько миллисекунд), но допустима в пределах, установленных Master в настройках таймаута.
- **Пауза перед ответом:** Для полудуплексных линий (RS-485) Slave перед отправкой ответа выдерживает минимальную технологическую паузу для смены направления передачи.

1.2.2.3 Адресация

- Каждый Slave в сети должен иметь **уникальный адрес** в диапазоне от 1 до 247. Адрес настраивается в конфигурации транспорта.
- Если пришедший запрос содержит **широковещательный адрес (0)**, Slave обрабатывает команду (например, запись в регистры), но **не отправляет ответ**. В данный момент не поддерживается.
- Запросы, адресованные другому Slave, игнорируются.

1.2.2.4 Обработка ошибок

Slave проверяет каждый входящий запрос по следующим критериям:

Проверка	Действие при ошибке
Контрольная сумма (CRC для RTU)	Кадр отбрасывается, ответ не отправляется (Master получит таймаут)
Адрес ведомого	Если адрес не совпадает — кадр игнорируется
Код функции	Если функция не поддерживается — отправляется исключение с кодом `0x01` (Illegal Function)
Адрес регистра	Если адрес выходит за допустимые пределы — отправляется исключение `0x02` (Illegal Data Address)
Количество регистров	Если запрашиваемое количество превышает допустимое — отправляется исключение `0x03` (Illegal Data Value)
Значение данных	Если записываемое значение недопустимо — отправляется исключение `0x03`

При успешной обработке запроса Slave отправляет **положительный ответ** — кадр, содержащий подтверждение выполнения операции или запрошенные данные.

1.3 Оптимизации операций записи в Modbus

- **Цель**: уменьшить трафик и задержки и ускорить достижение согласованного состояния на устройстве.

- **Область применения**: операции записи функций 5/6/15/16; чтение не оптимизируется.

1.3.1 Модель команды и терминология

- **Что содержит команда**: идентификатор устройства (`Slave ID`), номер функции (`func`), адрес или диапазон адресов, и полезную нагрузку (значения для записи).
  - Для функций 5 и 6 записывается один адрес.
  - Для функций 15 и 16 может записываться непрерывный диапазон адресов.
- При сравнении двух команд **одинаковыми командами** считаются те, у которых полностью совпадают `Slave ID`, функция и адрес регистра.
- **Смежными диапазонами** являются диапазоны, между которыми нет разрыва — один начинается сразу после другого. Примеры: смежные — `1..4` и `5..8`; `10..10` и `11..15`. Не смежные — `1..4` и `6..8` (разрыв), `3..7` и `5..9` (перекрытие).
- **Неподтверждённой записью**, является та, которая находится в очереди и ещё не была отправлена драйвером на устройство.

1.3.2 Этап 1. Дедупликация при постановке в очередь

- **Назначение**: убрать лишние повторные записи одного и того же адреса, если между ними не было попытки отправки на устройство.
- **Принцип дедупликации**: Если в очереди уже есть неподтверждённая запись для того же `Slave ID`, той же функции и того же адреса, то новая запись заменяет значение в существующей, а сама в очередь не добавляется.
- **Что не дедуплицируется**:
  - Разные адреса, разные функции или разные устройства.
  - Операции чтения.
  - Объединение диапазонов — отдельный механизм (см. Этап 2).
- **Эффект**:
  - Снижается нагрузка на канал и устройство при частых обновлениях одного тега.
  - В устройство уходит самое свежее значение на момент отправки.

1.3.3 Этап 2. Объединение команд записи (функции 15 и 16)

- **Назначение**: сократить число запросов, объединив записи в соседние адреса в один непрерывный блок.
- **Что объединяется**:
  - Функция 15 (Write Multiple Coils): катушки с непрерывными адресами.
  - Функция 16 (Write Multiple Registers): регистры хранения с непрерывными адресами.
  - Только записи с одинаковыми `Slave ID` и одинаковой функцией (15 или 16).
- **Условия объединения**: Несколько записей с одинаковыми `Slave ID` и функцией 15 или 16, чьи адресные диапазоны идут подряд без разрывов, объединяются в одну запись с общим диапазоном от минимального до максимального адреса.
- **Формирование буфера значений**:
  - Для 15: формируется массив булевых значений по адресам в объединённом диапазоне.
  - Для 16: формируется массив 16-битных слов по адресам в объединённом диапазоне.
    - 32-битные типы (Int32/UInt32/Float32) занимают два 16-битных слова.
    - Соблюдается заданный порядок слов (Normal/Inverse).

    - Если для одного адреса есть несколько кандидатов, берётся самое свежее значение на момент отправки.
- **Порядок и атомарность**:
    - Адреса в блоке упорядочиваются по возрастанию.
    - Отправка выполняется одной операцией: `writeCoils` (15) или `writeRegisters` (16).
    - Логирование отражает одну агрегированную запись.
- **Что не объединяется**:
    - Разные `Slave ID` или разные функции (15 и 16 не смешиваются).
    - Диапазоны с разрывами.
    - Операции чтения и иные типы команд.

1.3.4 Ограничения

1. Ограничение длины кадра устройства/драйвера может приводить к разбиению больших блоков.
2. Объединение выполняется на этапе подготовки к отправке; до него действует дедупликация.
3. При ошибке статусы возвращаются для всей агрегированной операции, повторы — по общей логике.

1.3.5 Примеры

- **Дедупликация**:
    - Три записи в один регистр: 10 → 20 → 30. В очереди остаётся одна запись; отправляется значение 30.
    - Две записи в разные регистры: обе будут отправлены независимо.
    - Записи в один адрес, но с разными функциями (например, 5 и 15): считаются разными и отправляются раздельно.
- **Объединение 15/16**:
    - Функция 16: записи 40010..40011 и 40012 объединяются в 40010..40012 и уходят одной операцией.
    - Функция 15: записи катушек 1..4 и 5..8 объединяются в 1..8 и отправляются одной операцией.

2 Транспорт OPC-UA

Протокол **OPC-UA (OPC Unified Architecture)** представляет собой современный стандарт обмена данными в промышленной автоматизации, обеспечивая высокий уровень безопасности, масштабируемость и платформенную независимость. Он позволяет надёжно интегрировать устройства, системы управления и SCADA-приложения, гарантируя единый подход к взаимодействию между различными компонентами инфраструктуры.

В данном разделе описывается **транспорт OPC-UA**, предназначенный для настройки связи по протоколу OPC-UA. Далее будут изложены основные свойства, принципы конфигурации и особенности работы транспортного узла, что поможет пользователям быстро и эффективно интегрировать OPC-UA в свои проекты.

2.1 Узлы

- **Транспорт OPC-UA** – обеспечивает связь и обмен данными через протокол OPC-UA.
- **Группа тегов** – узел, предназначенный для логического объединения OPC-UA тегов в единое пространство имен.

- **Тег OPC-UA** – узел, служащий источником данных, который хранит и передает значения.

2.1.1 Транспорт OPC-UA

2.1.1.1 Свойства

2.1.1.1.1 Группа свойств "Транспорт OPC-UA"

Эта группа свойств определяет глобальное поведение плагина OPC-UA, задавая параметры для подключения и работы в выбранном режиме.

Группа свойств "Транспорт OPC-UA"

Наименование свойства	Идентификатор	Тип	Доступ	Значение по умолчанию
Режим работы	TransportMode	ENUM	Чтение / запись	`Сервер`
Адрес	IPAddr	STRING	Чтение / запись	`opc.tcp://localhost:4840`
Разделитель	TagSeparator	STRING	Чтение / запись	`.` (Точка)

Описание свойств группы "Транспорт OPC-UA" узла Транспорт OPC-UA

1. **Режим работы**. Определяет режим работы плагина OPC-UA. Допустимые значения:
   - `Не определен (ошибка)`: При попытке построить проект SCADA выдаст ошибку о некорректном параметре.
   - `Клиент`: В этом режиме плагин функционирует как клиент OPC-UA. Дочерние узлы транспортного узла представляют собой ссылки на удалённые OPC-UA-*узлы* и/или *пространства имен* (namespaces), откуда производится считывание данных. Для корректного получения данных с конкретного узла на сервере требуется указать адрес сервера, пространство имен (если задано) и название узла на сервере.
   - `Сервер`: В режиме сервера плагин предоставляет данные через протокол OPC-UA. Дочерние узлы транспортного узла (которые могут быть как отдельными *узлами*, так и *пространствами имен*) описывают данные и группы данных, доступные для чтения внешними клиентами. Клиентские приложения смогут обращаться к этим узлам, чтобы получать данные, предоставляемые сервером.
2. **Адрес.** Определяет адрес для работы OPC-UA транспорта – как для подключения клиента, так и для конфигурации сервера. Для клиента, адрес указывает на OPC-UA сервер, к которому клиент будет пытаться подключиться. При указании некорректного или недоступного адреса клиент будет автоматически пытаться установить соединение с интервалом в 1 секунду. Для сервера адрес используется для настройки конечных точек (endpoint) OPC-UA сервера. Он задаёт URL, по которому сервер будет принимать подключения. На адрес накладываются следующие ограничения:
   - Строка не может быть пустой.
   - Длина строки не должна превышать 512 символов.
3. **Разделитель**. Определяет символ, который разделяет *пространство имен* узлов и *названия* узлов. В качестве значения параметра допускается единичный символ; если символ отсутствует или длина строки превышает 1, SCADA выдаст ошибку сборки проекта.

2.1.1.1.2 Группа свойств "Клиент OPC-UA"

Эта группа свойств содержит настройки, определяющие поведение плагина в режиме работы `Клиент`.

Группа свойств "Клиент OPC-UA"

Наименование свойства	Идентификатор	Тип	Доступ	Значение по умолчанию
Способ опроса	DataRetrievalMethod	ENUM	Чтение / запись	Не определен (ошибка)
Пауза между пакетами	PacketDelay	INT	Чтение / запись	1 мс
Таймаут операций	Timeout	INT	Чтение / запись	5000 мс

Описание свойств группы "Клиент OPC-UA" узла Транспорт OPC-UA

1. **Способ опроса**. Возможные значения Циклический опрос и Подписка. Параметр определяет способ взаимодействия клиента с сервером для получения актуальной информации. При выборе режима Циклический опрос клиент последовательно запрашивает данные по списку узлов, используя при этом значения свойств *Таймаут операций* и *Пауза между пакетами*. Режим Подписка позволяет обозначить интерес клиента к изменениям значений узлов на сервере: при обнаружении изменений сервер отправляет уведомления, которые клиенту остаётся только обработать.
2. **Пауза между пакетами**. Используется только для режима Циклический опрос. Параметр определяет задержку клиента между каждым отдельным запросом, при запросе значений узлов у сервера. Измеряется в миллисекундах. Если значение ноль - задержки нет.
3. **Таймаут операций**. Используется только в режиме Циклический опрос. Параметр задаёт задержку между циклами опроса сервера. Это время в миллисекундах, которое клиент ожидает после завершения цикла опроса сервера, позволяя снизить нагрузку на сервер и обеспечить стабильную работу системы.

2.1.2 Группа тегов

2.1.2.1 Свойства

Узел Группа тегов реализует функциональность *пространств имен* (namespaces) протокола OPC-UA. Он используется для назначения OPC-UA тегов в определённое пространство имен. Чтобы узел Тег OPC-UA корректно ассоциировался с заданным пространством, его необходимо сделать дочерним узла, обозначающего это пространство имен.

Группа свойств "Группа OPC"

Наименование свойства	Идентификатор	Тип	Доступ	Значение по умолчанию
Пространство имен	NamespaceIndex	UCHAR	Чтение / запись	1

Описание свойства "Пространство имен" узла "Группа тегов"

Свойство "Пространство имен" определяет числовой индекс пространства имен, используемого для формирования полного имени OPC-UA тегов. Допустимый диапазон значений: в диапазоне от 1 до 255. Если задано значение 0, при сборке проекта появится сообщение об ошибке. Если введено число больше 255, лишние разряды будут обрезаны для попадания в допустимый диапазон (например, значение 256 преобразуется в 25, значение 4851 — в 48).

Если числовой индекс пространства имен, заданный на клиенте, не совпадает с индексом, зарегистрированным на сервере, то узлы, принадлежащие данной группе, будут полностью недоступны для обмена данными между клиентом и сервером. Для корректного взаимодействия клиента и сервера необходимо, чтобы совпадали не только числовые индексы пространств имен, но и имена узлов, обозначающих эти пространства. Например,

если у клиента пространство имен с индексом `3` именуется `Dynamic`, а у сервера — `Dynami` (из-за опечатки), получение данных из OPC-UA узлов в этом пространстве станет невозможным.

2.1.3 Тег OPC-UA

2.1.3.1 Свойства

Узел `Тег OPC-UA` содержит свойства, определяющие поведение OPC-UA тега, включая тип данных, операции обработки значений и привязку к узлу в древовидной структуре проекта.

Свойства узла "Тег OPC-UA"

Наименование свойства	Идентификатор	Тип	Доступ	Значение по умолчанию
Тип значения	ValueType	ENUM	Чтение / запись	`UInt16`
Операции после чтения	OperationsOnRead		Чтение / запись	Отсутствует
Операции перед записью	OperationsOnWrite		Чтение / запись	Отсутствует
Узел для чтения/записи	SourceId		Чтение / запись	Отсутствует
Права доступа	AccessRights	ENUM	Чтение / запись	`Только чтение`

Описание свойств узла Тег OPC-UA

1. **Тип значения.** Определяет тип данных, с которыми работает тег. Допустимые значения:
    - `Не определен (ошибка)`: При сборке проекта SCADA выдаст диалоговое окно с ошибкой, информируя о невалидном значении параметра.
    - `BOOL`: Логический тип, принимающий значения `true` или `false`.
    - `Int16`: Целое число со знаком, диапазон от `-32768` до `32767`.
    - `Int32`: Целое число со знаком, диапазон от `-2147483648` до `2147483647`.
    - `UInt16`: Целое число без знака, диапазон от `0` до `65535`.
    - `UInt32`: Целое число без знака, диапазон от `0` до `4294967295`.
    - `Float32`: Число с плавающей запятой (32-битное, стандарт IEEE-754).
    - `String`: Текстовая строка.
2. **Операции после чтения.** Содержит список операций, которые применяются к значению тега сразу после его чтения с сервера и перед передачей клиенту. Операции выполняются последовательно, от первой к последней в списке. Результат каждой операции используется как вход для следующей. Если список операций оставлен пустым, считанное значение будет передано клиенту без изменений. Список возможных операций:
    - Умножение
    - Скрипт C++
    - Сложение
    - Битовое AND
    - Битовое OR
    - Битовое SHR
    - Битовое SHL
    - Логическое И
    - Логическое ИЛИ
    - Логическое НЕ

3. **Операции перед записью.** Содержит список операций, применяемых к значению тега перед тем, как сервер запишет его после получения от клиента. Список операций идентичен списку для *Операций после чтения*. Операции выполняются последовательно, как описано выше. Если список операций оставлен пустым, сервер запишет переданное от клиента значение без изменений.
4. **Узел для чтения/записи.** Определяет ссылку на узел в древовидной структуре проекта SCADA, который хранит значение тега. Используется только в режиме работы `Сервер`. Значение должно быть выбрано из существующих узлов дерева проекта. Если указанный узел не найден или его тип значения несовместим с заданным типом тега, в Журнал событий будет записано сообщение о недопустимом типе.
5. **Права доступа.** Определяют режим доступа к тегу. Допустимые значения:
    - `Не определен (ошибка)`: При наличии такого значения во время сборки проекта SCADA выдаст ошибку, информируя о невалидном параметре.
    - `Только чтение`: Тег доступен только для чтения; запись запрещена.
    - `Чтение и запись`: Тег доступен для операций чтения и записи.

2.2 Создание и настройка транспорта OPC-UA

Для использования транспорта OPC-UA в проекте должна присутствовать транспортная система. Добавьте в систему узел типа `Транспорт OPC UA`.

Выберите режим работы транспорта:

- Клиент - транспорт работает как ведущий и выполняет запрос значений параметров у сервера.
- Сервер - транспорт работает в режиме "ведомый" и отвечает на запросы клиентов.

Задайте свойствам транспорта необходимые значения.

2.3 Добавление параметров в Транспорт OPC-UA, работающий в режиме "Клиент"

Для добавления параметров (OPC-UA тегов) в транспорт необходимо знать их имена. Определение имен параметров, предоставляемых OPC UA сервером, можно произвести путем просмотра дерева параметров сервера с помощью доступных OPC-UA клиентов, например:

- Unified Automation UaExpert (бесплатный).

После определения имен интересующих параметров необходимо добавить в транспорт нужные параметры в виде древовидной структуры в соответствии со структурой параметров в сервере.

Для примера продемонстрируем порядок настройки транспорта на примере публичного OPC UA сервера по адресу `opc.tcp://milo.digitalpetri.com:62541/milo`

Данный сервер предоставляет большой набор параметров, среди которых есть динамически меняющиеся, которые можно найти с помощью упомянутых выше OPC-UA клиентов:

```
Root/Objects/Dynamic/RandomDouble
Root/Objects/Dynamic/RandomFloat
Root/Objects/Dynamic/RandomInt32
Root/Objects/Dynamic/RandomInt64
```

С помощью клиента определяем, что интересующие нас параметры имеют следующие идентификаторы:

```
ns=2;s=Dynamic/RandomDouble
ns=2;s=Dynamic/RandomFloat
ns=2;s=Dynamic/RandomInt32
ns=2;s=Dynamic/RandomInt64
```

Добавим в транспорт узел типа `Группа тегов`, установим его свойства следующим образом:

- Пространство имен = 2;
- Имя = `Dynamic`.

Добавим в данную группу узлы типа `Тег OPC-UA` с именами `RandomDouble`, `RandomFloat`, `RandomInt32`, `RandomInt64`и установим в их свойствах соответствующий тип значения.

После сохранения, компиляции и запуска проекта транспорт будет запрашивать у сервера добавленные теги.

3 Транспорт SNMP

Протокол **SNMP (Simple Network Management Protocol)** широко используется для мониторинга и управления сетевыми устройствами, такими как маршрутизаторы, коммутаторы, серверы и принтеры. Он позволяет собирать информацию о состоянии устройств и выполнять удалённые настройки. Для эффективного использования SNMP в AgavaSCADA необходимо правильно настроить **транспорт SNMP** и его параметры. В следующих разделах описывается процесс создания и настройки транспорта SNMP, а также добавления необходимых параметров для обеспечения надежного взаимодействия с управляемыми устройствами.

3.1 Создание и настройка транспорта SNMP

1. В дереве проекта, правой кнопкой мыши выделить группу узлов, представляющие транспорты проекта, выбрать `Добавить узел` -> `Транспорт SNMP`, в результате чего в дереве проекта будет создан узел транспорта SNMP.
2. В свойствах созданного транспорта необходимо указать IP-адрес устройства, версию SNMP протокола, используемого устройством, и необходимую информацию для получения доступа к устройству (community-строку для `SNMP v2c` или имя пользователя и пароль для `SNMP v3`).

3.1.1 Как определить версию SNMP для устройства

Версию SNMP можно определить, обратившись к документации или настройкам устройства. Если доступ к этой информации ограничен, можно попробовать последовательно опросить устройство с использованием разных версий SNMP и определить, на какую версию оно откликается.

3.1.2 Авторизация в версиях SNMP v1 и SNMP v2c

Авторизация осуществляется с помощью **community-строк**, которые действуют как простые пароли. Community-строка передается в открытом виде и может быть установлена на значения по умолчанию, такие как `public` для операций чтения или `private` для операций записи. Администратор системы может установить собственные значения для community-строк, используемых устройством. Безопасность в этих версиях протокола минимальна.

3.1.3 Авторизация в версиях SNMP v3

Предоставляет усовершенствованные механизмы безопасности, включая аутентификацию и шифрование. Авторизация требует имя пользователя, а также может потребовать пароль аутентификации и пароль шифрования. Данная версия протокола поддерживает различные уровни безопасности:

- **noAuthNoPriv**: Только имя пользователя без аутентификации и шифрования.
- **authNoPriv**: Аутентификация без шифрования.
- **authPriv**: Полная аутентификация и шифрование данных.

Пример корректно заданного Транспорта SNMP

Свойство	**Значение**
Версия SNMP	SNMP v2c
IP адрес	172.16.3.22
Имя пользователя	public
Пароль	

3.2 Добавление параметров в Транспорт SNMP

1. В дереве проекта найти и правой кнопкой мыши выделить узел транспорт SNMP и выбрать `Добавить узел` -> `Параметр SNMP`, в результате чего в дереве проекта будет создан узел параметра SNMP.
2. В свойствах узла, представляющего параметр SNMP, помимо группы свойств `Основные`, доступна также группа свойств `Параметр SNMP`, содержащая свойства, специфичные для параметра транспорта: `OID`, `Тип значения` и `Права доступа`.
    - `OID` (**O**bject **Id**entifier) — это уникальный идентификатор объекта управления в MIB (Management Information Base). Он представляет собой последовательность чисел, разделенных точками, например: `1.3.6.1.2.1.1.1.0`. OID можно получить несколькими способами: обратиться к документации производителя, которая часто содержит список доступных OID и их описание; использовать MIB-файлы — специальные файлы, предоставляемые производителем устройства и содержащие все OID и их свойства; или воспользоваться сторонними утилитами (например SNMP Walk, позволяющая просмотреть все доступные OID на устройстве).
    - `Тип значения` определяет тип данных, которые будет получен и/или отправлен при взаимодействии с устройством по определенному `OID`.
    - `Права доступа` — свойство определяющее, какие операции могут быть выполнены с данным OID — только чтение или чтение и запись.

Пример корректно заданного Параметра SNMP

Свойство	**Значение**
Тип значения	String
Права доступа	Чтение и запись

OID	1.3.6.1.2.1.1.1.0

После корректного задания свойств **Транспорта SNMP** и добавления **параметров SNMP** для данного транспорта, пользователь AgavaSCADA сможет эффективно использовать передаваемую информацию узлами-параметрами SNMP. Это позволит осуществлять мониторинг и анализ данных устройств, интегрированных в сеть.

4 Транспорты МЭК-60870

Узлы **«Транспорт МЭК-60870-101»** и **«Транспорт МЭК-60870-104»** предназначены для обмена телемеханической информацией. Для МЭК-60870-101 используется последовательное соединение, для МЭК-60870-104 — TCP-соединение.

В обоих случаях выбирается режим **Master** или **Slave**. Дополнительно настраиваются интервал опроса и параметры адресации: размеры полей COT, CA, IOA и ASDU. Эти параметры должны соответствовать настройкам другой стороны соединения.

Порядок настройки:

1. Добавить транспорт требуемого типа и выбрать режим работы.
2. Для МЭК-60870-101 задать последовательный порт, скорость, чётность, стоповые биты, балансный или небалансный режим. Для МЭК-60870-104 указать IP-адрес, порт и параметры контроля соединения.
3. Добавить в транспорт узел **«Станция МЭК»**.
4. Задать свойство станции **«Сетевой адрес(ASDU)»**. Для МЭК-60870-101 также задать **«Канальный адрес»**.
5. Добавить в станцию узлы **«Переменная МЭК»**. Для каждой переменной указать **«Адрес(IOA)»**, тип значения и права доступа.
6. В режиме Slave связать переменные с узлами проекта через свойство **«Узел для чтения/записи»**. При необходимости включить **«Периодическая отправка»** и **«Спорадическая отправка»**.
7. Если предусмотрены команды управления, настроить **«Адрес(IOA) записи»** и использование метки времени.

Свойство **«Метка времени»** переменной определяет использование полученных меток времени. Для контроля обмена предусмотрено свойство транспорта **«Отладочный режим»**, включающее регистрацию принимаемых и передаваемых пакетов в журнале.

Подробно использование транспортов МЭК-60870-101 и МЭК-60870-104 рассмотрено в документе Транспорт МЭК-60870.

5 Транспорт MQTT

Узел **«Транспорт MQTT»** предназначен для получения и публикации значений через MQTT-брокер. Данные представлены узлами **«Топик MQTT»**.

В свойствах транспорта предусмотрены поля **«Адрес брокера»**, **«Порт брокера»**, **«Имя пользователя»**, **«Пароль»** и **«ID пользователя»**. Последнее поле задаёт идентификатор MQTT-клиента.

В текущей версии транспорта подключение выполняется к порту **1883** без передачи имени пользователя и пароля. Следует использовать брокер, допускающий такое подключение. Значения полей **«Порт брокера»**, **«Имя пользователя»** и **«Пароль»** не изменяют эти условия подключения.

Порядок настройки:

1. Добавить транспорт MQTT и указать адрес брокера, порт 1883 и идентификатор клиента.
2. Добавить узлы **«Топик MQTT»**. Имя узла определяет имя топика. Вложенные топики образуют путь с разделителем `/`.
3. Указать **«Тип значения»**: String, Int32, UInt32 или Float32.
4. Выбрать **«Права доступа»**: **«Только чтение»** для получения сообщений либо **«Чтение и запись»** для получения и публикации значений.
5. Добавить топики в источники сигналов проекта.

Содержимое сообщений должно соответствовать выбранному типу значения. Например, для числового топика требуется числовое текстовое представление значения.

6 Транспорт OPC DA

Узел **«Транспорт OPC DA»** предназначен для подключения к OPC DA-серверу. Этот транспорт используется в среде исполнения для Windows.

В свойствах транспорта задаются **«Имя хоста»**, **«Имя сервера»**, **«Версия протокола»** и **«Режим опроса»**. Предусмотрены синхронный и асинхронный режимы; возможность асинхронного обмена зависит от сервера.

Для настройки необходимо добавить транспорт, указать сервер, затем создать группы и теги OPC DA. Для тегов задаются адресующие их имена, типы данных и права доступа. Полученные значения связываются с сигналами проекта. Доступ к удалённому серверу должен быть настроен средствами операционной системы.

7 Транспорт AgavaCloud

Узел **«Транспорт AgavaCloud»** предназначен для передачи выбранных данных проекта на сервер AgavaCloud.

Для настройки необходимо:

1. Добавить транспорт в транспортную систему.
2. Указать **«Адрес»** сервера: `cloud.kb-agava.ru`
3. Указать **«Идентификатор»** для регистрации - УИН, получаемый в проекте AgavaCloud при добавлении устройства типа "ПЛК АГАВА".
4. В свойстве **«Адресация параметра»** выбрать используемое обозначение параметра: **«Имя узла»**, **«Описание узла»** или **«Шифр узла»**.
5. В свойстве **«Узлы»** выбрать существующие источники, сигналы или операции, значения которых необходимо передавать.
6. Проверить соответствие обозначений параметров настройкам сервера.