



# IntraSCADA

Программная платформа для систем диспетчеризации и мониторинга

Руководство пользователя

<b>Глава 1 Введение</b>	7
1.1. О системе IntraSCADA	7
1.2. Общие концепции	8
1.2.1. Архитектура системы	8
1.2.2. Работа с оборудованием	10
1.2.3. Скрипты	11
1.2.4. Диагностика	13
1.3. Версии	18
1.3.1. Регламент выхода новых версий	18
<b>Глава 2 Установка и запуск</b>	19
2.1. Технические требования	19
2.2. Установка системы	20
2.2.1. Все варианты	20
2.2.2. ALT Linux	22
2.2.3. Astra Linux	24
2.2.4. Linux (Deb)	27
2.2.5. Linux (RPM)	30
2.2.6. Windows	33
2.2.7. Wiren Board	34
2.2.8. JetHome	36
2.2.9. macOS	38
2.2.10. РЕД ОС	39
2.3. Первый запуск	42
2.4. Лицензирование	44
<b>Глава 3 Быстрый старт</b>	47
3.1. Введение	47
3.2. Установка системы и первый запуск	48
3.3. Получение демо-лицензии	50
3.4. Настройка устройств	54
3.5. Визуализация	57
<b>Глава 4 Интерфейс разработчика</b>	66
4.1. Общее описание	66
4.2. Инфопанель	68
4.2.1. О системе	68
4.2.2. Обновление	68
4.2.3. Плагины	70
4.2.4. Лицензии	71
4.2.5. Удаленный доступ	72
4.2.6. Консоль	75
4.2.7. Процессы	76
4.3. Структура проекта	77
4.3.1. Общее описание	77
4.3.2. Устройства	78
4.3.2.1. Общее описание	78
4.3.2.2. Экземпляры устройств	81
4.3.2.3. Механизм каналов	84
4.3.2.3.1. Привязка каналов	84
4.3.2.3.2. Состояние канала	87
4.3.2.3.3. Блокировка каналов	91
4.3.2.3.4. Примеры	93
4.3.2.4. Отладчик	96



4.3.3. Системные индикаторы	101
4.3.3.1. Общее описание	101
4.3.3.2. Индикаторы процессов	102
4.3.3.2.1. Индикаторы плагинов	102
4.3.3.2.2. Индикаторы агентов БД	103
4.3.4. Типы устройств	104
4.3.4.1. Общее описание	104
4.3.4.2. Команды	109
4.3.4.3. Обработчики	112
4.3.4.4. Тревоги	114
4.3.5. Обработчики	121
4.3.5.1. Доступ к свойствам устройства	121
4.4. Визуализация	125
4.4.1. Введение	125
4.4.2. Общая информация	127
4.4.3. Экраны	130
4.4.4. Контейнеры	141
4.4.5. Диалоги	147
4.4.6. Шаблоны визуализации	159
4.4.7. Элементы	166
4.4.7.1. Общее описание	166
4.4.7.2. Общие свойства	169
4.4.7.3. Привязки	171
4.4.7.4. Basic	175
4.4.7.4.1. - Rectangle	175
4.4.7.4.2. - Circle	176
4.4.7.4.3. - Text	177
4.4.7.4.4. - Image	179
4.4.7.4.5. - Button	181
4.4.7.5. Interactive	186
4.4.7.5.1. - Input	186
4.4.7.5.2. - Slider	192
4.4.7.5.3. - Checkbox	199
4.4.7.5.4. - Date Range	202
4.4.7.5.5. - Calendar	203
4.4.7.5.6. - Sector	205
4.4.7.6. View	208
4.4.7.6.1. - List	208
4.4.7.6.2. - Autocomplete	214
4.4.7.6.3. - Table	218
4.4.7.6.4. - Tree	225
4.4.7.6.5. - Grid	231
4.4.7.7. Charts	237
4.4.7.7.1. Виды графиков	237
4.4.7.7.2. - Chart Line	238
4.4.7.7.3. - Chart Multiline	244
4.4.7.7.4. - Chart Timeline	250
4.4.7.7.5. - Chart Columns	254
4.4.7.7.6. - Chart Pie	259
4.4.7.8. Widgets	262
4.4.7.8.1. - Iframe	262

4.4.7.8.3. - CCTV	271
4.4.7.8.4. - Map	272
4.4.7.8.5. - Device log	274
4.4.7.8.6. - Journal	277
4.4.7.8.7. - Alert journal	284
4.4.7.8.8. - Report	289
4.4.8. Скрипты визуализации	291
4.4.8.1. Для чего используются	291
4.4.8.2. Отличие от сценариев	293
4.4.8.3. Структура скрипта	295
4.4.8.4. Асинхронные функции	300
4.4.8.5. API скрипта визуализации	306
4.4.8.5.1. Работа с клиентом	306
4.4.8.5.2. Завершение сеанса	307
4.4.8.5.3. Доступ к устройствам и глобальным переменным	308
4.4.8.5.4. Запись в главный журнал	311
4.4.8.5.5. Вызов диалога, звуковое оповещение	312
4.4.8.5.6. Информирование	314
4.4.8.5.7. Действия со сценариями	315
4.4.8.5.8. Временная задержка	317
4.4.8.5.9. Отправка команды плагину	318
4.4.8.5.10. Снимок с камеры	320
4.4.8.5.11. Генерация отчёта	321
4.4.8.5.12. Работа с тревогами	324
4.4.8.5.13. Работа с файлами	327
4.4.8.5.14. Работа с клиентскими данными	330
4.4.8.5.15. Работа с деревьями проекта	332
4.4.8.5.16. Работа с пользовательскими таблицами	335
4.4.8.5.17. Чтение исторических данных	336
4.4.8.5.18. Системные функции	343
4.4.8.5.19. Операции CRUD	345
4.4.8.5.19.1. Операции CRUD в проекте	345
4.4.8.5.19.2. Обработка ошибок	346
4.4.8.5.19.3. Работа с устройствами	348
4.4.8.5.19.4. Работа с плагинами	356
4.4.8.5.19.5. Работа с каналами	365
4.4.8.5.19.6. Работа с папками/узлами каналов	371
4.4.8.6. Стартовый скрипт сеанса	378
4.4.8.7. Скрипты заполнения	381
4.4.8.8. Примеры	388
4.4.8.8.1. Список снимков с ip камеры по датам	388
4.4.9. Переменные клиента	393
4.4.10. Динамическая привязка устройств	396
4.5. Автоматизация	399
4.5.1. Сценарии	399
4.5.1.1. Общее описание	399
4.5.1.1.1. Общее описание	399
4.5.1.1.2. Структура сценария	401
4.5.1.1.3. Цикл работы сценария	412
4.5.1.1.4. Мульти-сценарии	416
4.5.1.2. API	418

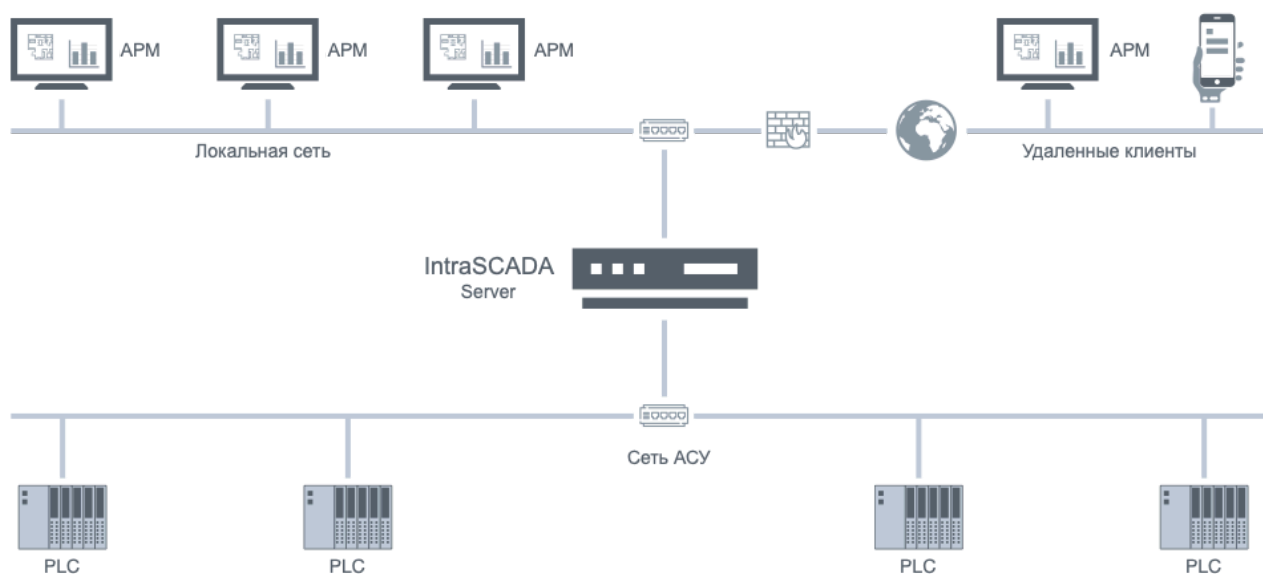
4.5.1.2.2. Взаимодействие с активными клиентами	424
4.5.1.2.3. Встроенные методы	425
4.5.2. Шаговые сценарии	434
4.5.2.1. Общее описание	434
4.5.2.2. Свойства шаговых сценариев	439
4.6. Источники данных	440
4.6.1. Плагины	440
4.6.1.1. О плагинах	440
4.6.1.2. Установка плагина	441
4.6.1.3. Список плагинов	449
4.6.1.3.1. CCTV	449
4.6.1.3.2. Emulator	453
4.6.1.3.3. Kerberos	456
4.6.1.3.4. LDAP client	463
4.6.1.3.5. MODBUS master (client)	473
4.6.1.3.6. MODBUS slave (server)	480
4.6.1.3.7. MQTT	482
4.6.1.3.8. MQTT server	488
4.6.1.3.9. OMRON FINS	489
4.6.1.3.10. OPC UA	498
4.6.1.3.11. OPC UA server	506
4.6.1.3.12. МЭК 60870-5-104	508
4.6.1.3.13. МЭК 60870-5-104multi	511
4.6.1.3.14. МЭК 60870-5-104server	518
4.6.1.3.15. Stimulsoft report	524
4.6.1.3.16. SNMP	525
4.6.1.4. Разработка плагинов	528
4.6.1.4.1. API плагина	528
4.6.1.4.2. Логирование	537
4.6.1.4.3. Свойства индикатора плагина	539
4.6.2. REST API	541
4.6.2.1. Общее описание	541
4.6.2.2. Доступ	542
4.6.2.3. Маршрут	545
4.6.2.4. Обработчик	547
4.7. Аналитика	559
4.7.1. Общее описание	559
4.7.2. Графики	560
4.7.3. Обработчик для графика	564
4.7.4. Таймлайны	574
4.7.5. Журналы	576
4.7.6. Журналы тревог	581
4.7.7. Отчеты	582
4.7.7.1. Формирование и показ	582
4.7.7.2. Переменные отчета	584
4.7.7.3. Таблица отчета	586
4.7.7.4. Обработчик для отчета	587
4.8. База данных	592
4.8.1. SQLite	592
4.8.2. Пользовательские таблицы	593
4.8.2.1. Общее описание	593

4.8.2.3. API .....	598
4.9. Ресурсы .....	606
4.9.1. Изображения .....	606
4.9.2. Звуки .....	607
4.9.3. Списки .....	609
4.9.4. Сетки .....	611
4.9.5. Таблицы .....	615
4.9.6. Деревья .....	618
4.9.7. Файлы .....	620
4.9.8. Скриншоты .....	621
4.10. Рецепты .....	622
4.10.1. Общее описание .....	622
4.10.2. Хранение рецептов в БД .....	626
4.10.3. Редактирование формул рецептов .....	628
4.10.4. Создание рецепта .....	634
4.11. Доступ .....	643
4.11.1. Общее описание .....	643
4.11.2. Пользователи .....	644
4.11.2.1. Учетные записи .....	644
4.11.2.2. Информирование .....	647
4.11.3. Группы пользователей .....	649
4.11.4. Текущие подключения .....	655
4.11.5. Интеграция с LDAP .....	656
4.12. Настройка системы .....	658
4.12.1. Системные настройки .....	658
4.12.2. Работа с проектами .....	661
4.13. Резервирование серверов .....	665
4.13.1. Механизмы резервирования .....	665
4.13.2. Настройка резервирования .....	667
4.14. Информационная безопасность .....	669
4.14.1. Сводная информация по ИБ .....	669
4.14.2. Порты по умолчанию .....	672
4.14.3. Регистрация событий ИБ .....	673
4.14.4. Типовые инциденты .....	675
4.14.5. Резервное копирование проекта .....	676
4.14.6. Смена пароля .....	678
4.14.7. Аудит ИБ .....	680
<b>Глава 5 Интерфейс пользователя .....</b>	<b>683</b>
5.1. IntraSCADA Client .....	683

# IntraSCADA

Программная платформа для систем диспетчеризации и мониторинга в системах автоматизации и промышленного интернета вещей (IIoT), в системах управления технологическими процессами (АСУ ТП) и в системах учёта энергоресурсов (АСУЭ)

- Визуализация процессов в графическом режиме
- Возможность создавать алгоритмы управления
- Отслеживание трендов в реальном времени и доступ к архивным трендам
- Управление алармами
- Информирование о событиях через email и telegram
- Детализированные отчеты



# Архитектура системы

Система построена в архитектуре клиент-сервер.

## Сервер

Обеспечивает функционирование как исполнительной системы, так и системы разработки текущего проекта.

Главные задачи, решаемые серверной частью:

- связь с оборудованием
- функции web-сервера для пользовательского интерфейса и системы разработки проекта
- разработка и сохранение проекта, динамическое изменение рантайма при редактировании проекта
- выполнение сценариев и других скриптов
- сохранение исторических данных
- информирование клиентов о событиях системы
- формирование графиков и отчетов
- механизмы интеграции с внешними системами

Сервер запускается как служба (сервис) на почти любом компьютере, способном работать в режиме 24x7 под ОС Linux, Windows или MacOS.

Основной сервис:

- запускает множество плагинов (микросервисов) как дочерние процессы
- полностью контролирует запуск/останов и текущее состояние плагина
- работает на платформе Node.js

## Плагины

**Плагины** - это программные модули (драйверы) для работы с оборудованием и выполнения сервисных функций.

Работа с БД, подсистемы информирования, обмен данными со сторонними сервисами реализуются плагинами. Это позволяет масштабировать систему, сделать возможным простую замену и добавление новых протоколов, функций и сервисов.

Большинство плагинов созданы на JavaScript используют для коммуникации встроенный IPC канал.

Но есть возможность написать плагин практически на любом языке (Go, Python, C/C++).

Плагины системы IntraSCADA всегда запускаются как отдельный дочерний процесс. Это придает надежность и устойчивость системе.

Запущенный плагин имеет постоянную связь с основным процессом, передает и получает данные и диагностическую информацию.

При ошибке или остановке плагина он может быть просто перезапущен основным процессом.

## Клиентские приложения

В состав клиентских приложений входят:

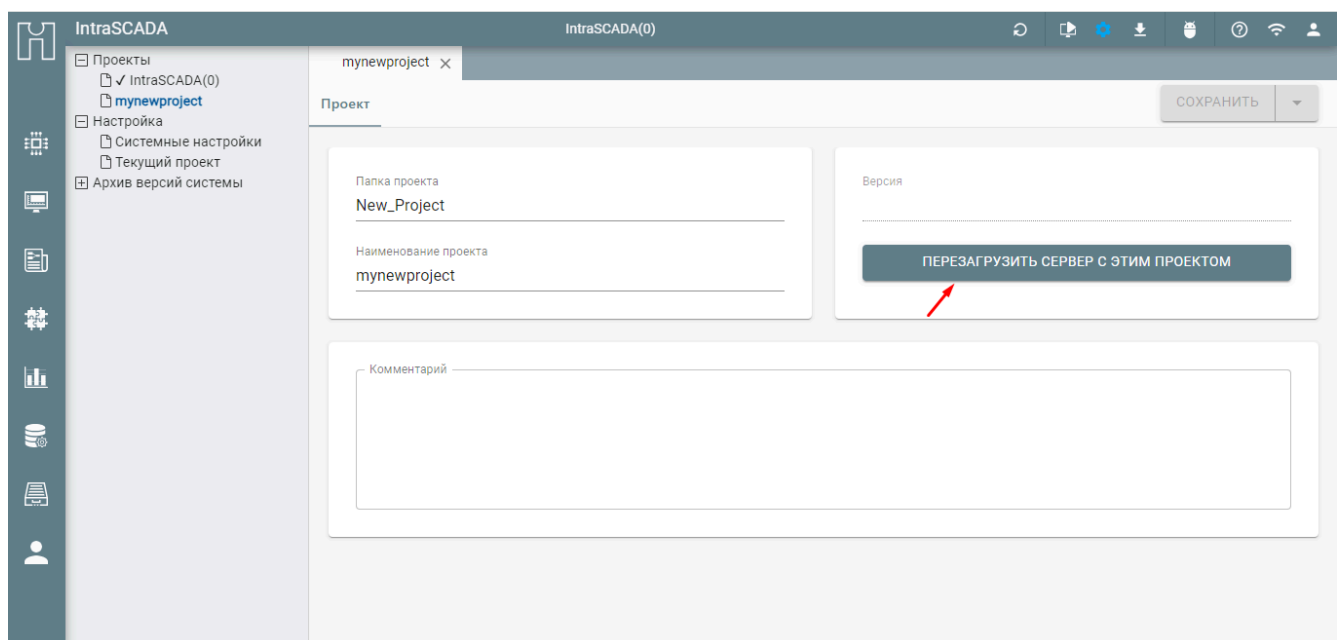
- Web-приложения для браузеров, которые подключаются к серверу по протоколам http(s) и WebSocket:
  - Менеджер проектов (Project Manager) для разработки и редактирования проекта
  - Пользовательский интерфейс (Среда Исполнения)
- Desktopное приложение под Linux, Windows или MacOS
  - Обеспечивает подключение к нескольким серверам в режиме разработки и/или исполнения.

Пользовательские интерфейсы, созданные для web-приложения, работают в десктопном приложении без изменений.

## Проекты и текущий проект

В каждый момент времени сервер работает с одним проектом. Текущий проект доступен для любых изменений (при наличии прав доступа). Изменение текущего проекта не требует остановки исполнительской системы, изменения сразу переходят в рантайм.

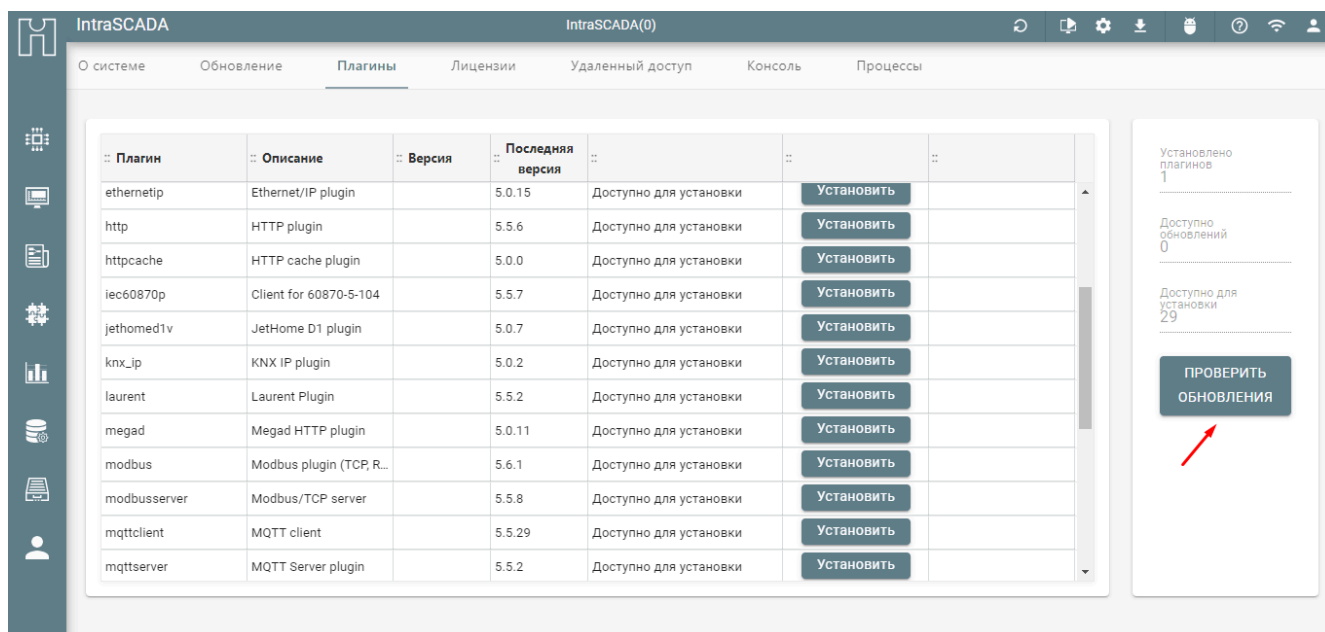
Проект можно выгрузить средствами системы и загрузить на другой сервер IntraSCADA. В среде разработки(Project Manager) в разделе **Проекты** можно увидеть список всех проектов на сервере. Переключение на другой проект происходит с перезагрузкой сервера.



## С каким оборудованием работает система

Возможность работы с оборудованием обеспечивают специальные модули - **плагины**. Плагины устанавливаются из дашборда системы. Список плагинов постоянно расширяется.

В составе системы есть плагины, реализующие протоколы: **Modbus, OPCUA, SNMP, MQTT, Http, Zigbee и другие**. Полный список общедоступных плагинов на текущий момент можно получить на вкладке **Плагины** дашборда по кнопке **Проверить обновления**.



Иногда плагины пишутся для конкретного оборудования или даже для проекта.

Открытое API плагинов позволяет **написать плагин** самостоятельно.

Также возможна разработка плагинов под заказ.

## Как настраивается связь с оборудованием

Этапы создания связи:

1. Создаем устройство, имеющее нужные свойства и внутреннюю логику. Устройство сразу можно использовать для построения пользовательского интерфейса и разработки сценариев.
2. Устанавливаем плагин, обеспечивающий нужный протокол.  
В плагине настраиваем каналы. Используем сканирование или загрузку из Excel. В крайнем случае, создаем каналы вручную
3. Привязываем свойства устройства к каналам

- первые два пункта можно выполнять в произвольном порядке. Это даже могут делать разные люди
- если нужно будет изменить привязку к каналу, это никак не повлияет на визуализацию и на сценарии
- при разработке визуализации информация о привязке к каналам не нужна



# Скрипты

Возможность создавать скрипты при разработке проекта придает системе гибкость.

Большая часть настройки проекта выполняется интерактивно и написания скриптов не требует.

Скрипты применяются там, где требуется расширить функционал.

Все скрипты пишутся на стандартном JavaScript с использованием движка V8, который поддерживает современную спецификацию языка ES6. Ввод и редактирование скриптов выполняется во встроенном редакторе среды разработки. После сохранения скрипт сразу готов к работе.

## Виды скриптов

В системе есть несколько видов скриптов. Они отличаются способом и моментом запуска, местом выполнения (на сервере или на клиенте), применяемым API, областью видимости и реакцией на ошибки.

- **Обработчики устройства** - это функции, заложенные в типе устройства. Предназначены для обработки входных данных, расчета промежуточных значений, выполнения команд устройства. Запускаются для каждого экземпляра устройства. Работают в рамках одного устройства, не имеют доступ к другим устройствам.
- **Сниппеты** - это источники значений свойств одного устройства без привязки к каналам плагинов. Слипсет можно использовать для формирования значения глобальной переменной. Запускаются с заданным периодом для каждого экземпляра.
- **Сценарии** - это скрипты автоматизации. Запускаются по событиям устройств, по расписанию. Решают широкий спектр задач контроля и управления. Могут работать с несколькими (многими) устройствами и слушать их события.
- **Скрипты визуализации и визуальных элементов**. Обеспечивают интерактивность клиентских интерфейсов. Запуск скрипта обычно происходит при интерактивных действиях пользователя. Выполняется скрипт на сервере в контексте конкретного пользователя и экрана.
- Функции, выполняемые на клиенте. Решают задачи дополнительной обработки данных на уровне визуализации. Запускаются на клиенте. На сервере информации о запуске этих функций нет.
- **REST API скрипты** для обработки входящих запросов к серверу IntraSCADA. Служат для обмена данными с другими системами.

## Когда использовать

Ниже даны более подробные характеристики каждого вида и рекомендации по выбору скрипта в зависимости от задачи:

- **Обработчики устройства** работают в рамках отдельного устройства. Независимы от интерфейсов пользователей. Интерактивно не запускаются. Есть механизм защиты скриптов от зависания. В случае ошибки скрипта дальнейший запуск обработчика блокируется. Следует делать эти функции достаточно короткими, быстрыми и герметичными. Они должны инкапсулировать логику работы одного устройства. Высокий потенциал переиспользования, полностью переносятся при выгрузке типа.

- **Сниппеты** - это легковесная замена плагину для одного устройства или глобальной переменной. Независимы от интерфейсов пользователей. Интерактивно не запускаются. В случае ошибки скрипта дальнейший запуск сниппета блокируется. Не имеет доступа к другим устройствам или функциям системы. Есть потенциал переиспользования.
- **Сценарии** - это скрипты автоматизации. Могут работать с несколькими устройствами с учетом их взаимосвязи в динамике. Независимы от интерфейсов пользователей. Есть механизм защиты скриптов от зависания. Асинхронные операции и очистка ресурсов контролируются движком. Гарантируется, что в один момент будет запущен только один экземпляр сценария. Может находиться в активном состоянии длительное время. В случае ошибки скрипта дальнейший запуск сценария блокируется. Может быть запущен интерактивно или из скрипта визуализации, но работает без прямой связи с интерфейсом пользователя. Это лучший вариант для реализации сложной логики работы взаимосвязанных устройств, не требующей интерактива. Есть потенциал переиспользования, особенно для кода мультисценариев.
- **Скрипты визуализации и скрипты визуальных элементов** (списков, таблиц, графиков). Решают задачи создания интерактивных интерфейсов, включая навигацию, динамическое заполнение и обновление элементов экрана при действиях пользователя и валидацию вводимых данных. В один момент может быть запущено множество экземпляров скрипта для разных пользователей (сессий). Возможны асинхронные операции внутри скрипта, ограничение - отзывчивость интерфейса. В случае ошибки при выполнении скрипт не блокируется, выдается сообщение "Ошибка скрипта!" на клиент, с которого был вызван скрипт. Потенциал переиспользования низкий, так как в скрипте обычно используются локальные переменные и другие сущности проекта (идентификаторы экранов, элементов, пользователей и тд)
- **Скрипты - функции, выполняемые на клиенте.** Решают задачи дополнительной обработки данных на уровне визуализации. Логика должна быть минимальной.
- **REST API скрипты** для обработки входящих запросов к серверу IntraSCADA. Служат для обмена данными с другими системами. Могут как передавать, так и принимать данные. Имеют доступ к устройствам и большинству функций системы. Асинхронные операции внутри скрипта широко используются.

Чтобы дать возможность переиспользовать пользовательский код, начиная с версии системы v5.18 были добавлены [Библиотеки функций](#)

# Диагностика

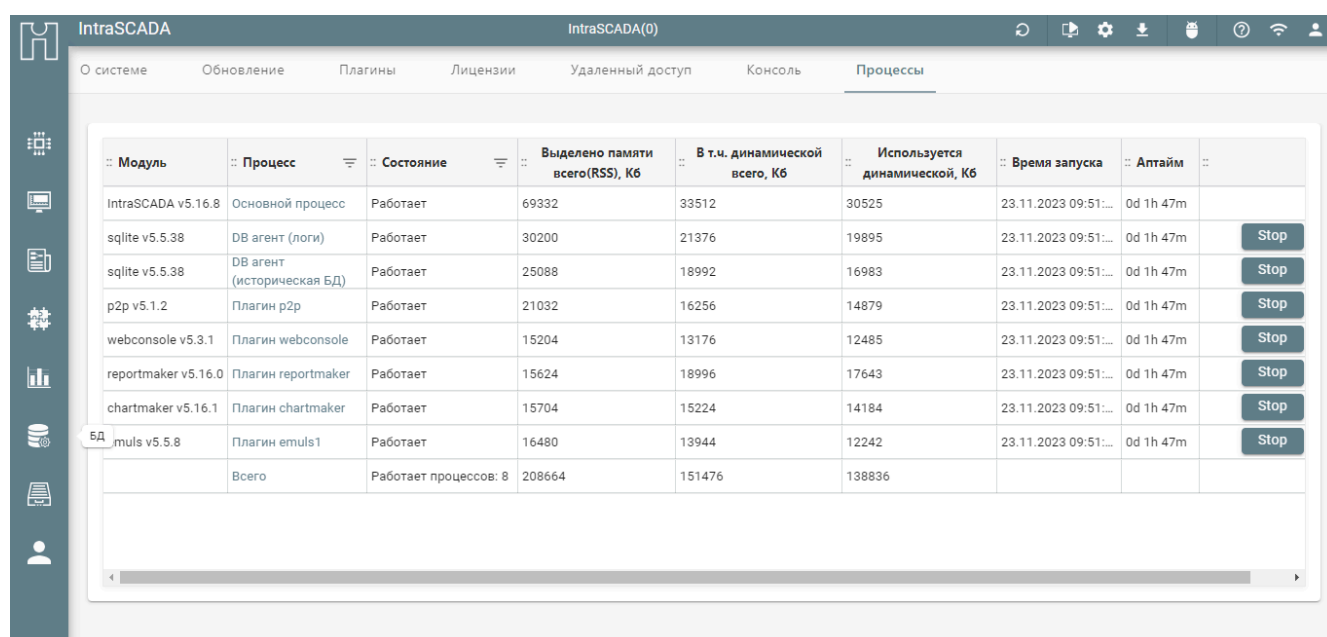
Инструменты диагностики - это логи, журналы и средства отладки и трассировки. Также механизмы диагностики позволяют автоматически восстанавливать работу плагинов, или наоборот, блокировать работу скриптов с ошибкой.

## Диагностика процессов/плагинов

Основной процесс системы запускает **плагины** как дочерние процессы.

### Таблица процессов

Информация обо всех процессах системы доступна на вкладке **Процессы** основного дашборда.



Модуль	Процесс	Состояние	Выделено памяти всего(RSS), Кб	В т.ч. динамической всего, Кб	Используется динамической, Кб	Время запуска	Аптайм	
IntraSCADA v5.16.8	Основной процесс	Работает	69332	33512	30525	23.11.2023 09:51:...	0d 1h 47m	
sqlite v5.5.38	DB агент (логи)	Работает	30200	21376	19895	23.11.2023 09:51:...	0d 1h 47m	Stop
sqlite v5.5.38	DB агент (историческая БД)	Работает	25088	18992	16983	23.11.2023 09:51:...	0d 1h 47m	Stop
p2p v5.1.2	Плагин p2p	Работает	21032	16256	14879	23.11.2023 09:51:...	0d 1h 47m	Stop
webconsole v5.3.1	Плагин webconsole	Работает	15204	13176	12485	23.11.2023 09:51:...	0d 1h 47m	Stop
reportmaker v5.16.0	Плагин reportmaker	Работает	15624	18996	17643	23.11.2023 09:51:...	0d 1h 47m	Stop
chartmaker v5.16.1	Плагин chartmaker	Работает	15704	15224	14184	23.11.2023 09:51:...	0d 1h 47m	Stop
emuls v5.5.8	Плагин emuls1	Работает	16480	13944	12242	23.11.2023 09:51:...	0d 1h 47m	Stop
	Всего	Работает процессов: 8	208664	151476	138836			

Плагин имеет постоянную связь с основным процессом и передает диагностическую информацию (состояние, использование памяти). При остановке (зависании) плагина основной процесс перезапускает его в соответствии с заданными настройками (время перезапуска плагина).

В таблице выводятся время последнего запуска и аптайм каждого процесса.

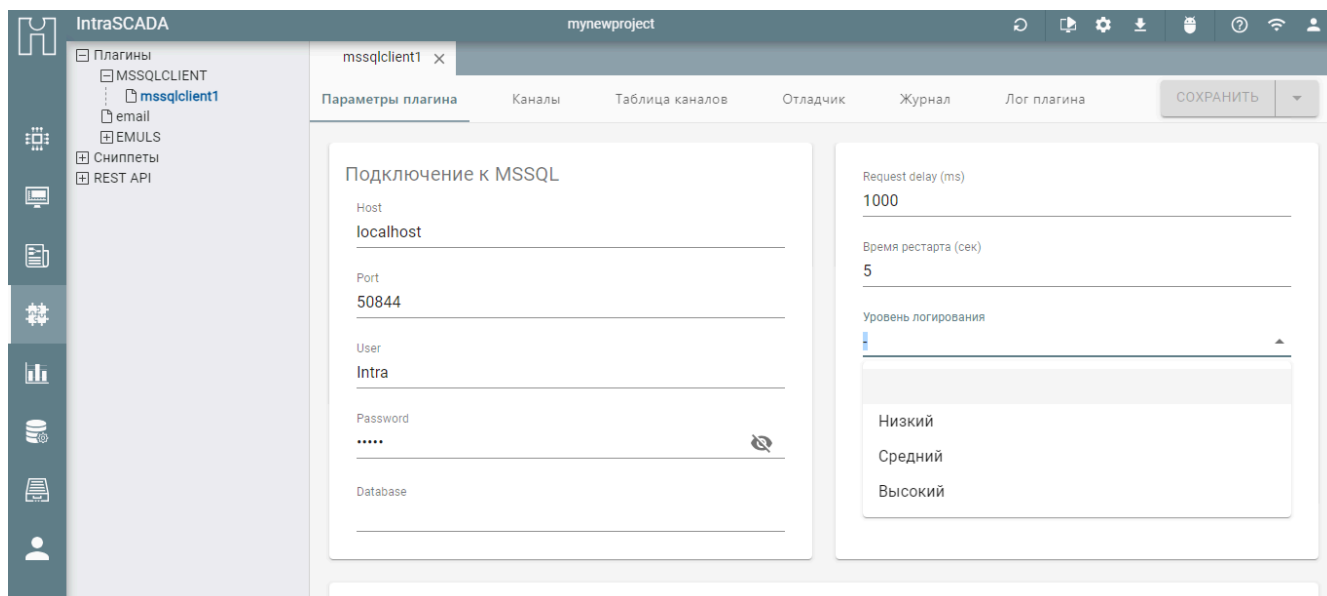
Можно также вручную запустить или остановить процесс кнопкой Start/Stop.

### Журнал

При запуске и остановке каждого дочернего процесса сервер выполняет запись в журнал плагинов. Этот журнал хранится в БД. Срок хранения сообщений настраивается в разделе Аналитика -> Журналы. Просмотреть записи можно в разделе Источники данных -> Плагины на вкладке Журнал для всех плагинов или каждого отдельно.

### Лог

Каждый процесс ведет лог, просмотреть который можно по ссылке из таблицы **Процессы**. Уровень логирования, от которого зависит объем сообщений лога, задается для каждого экземпляра плагина.

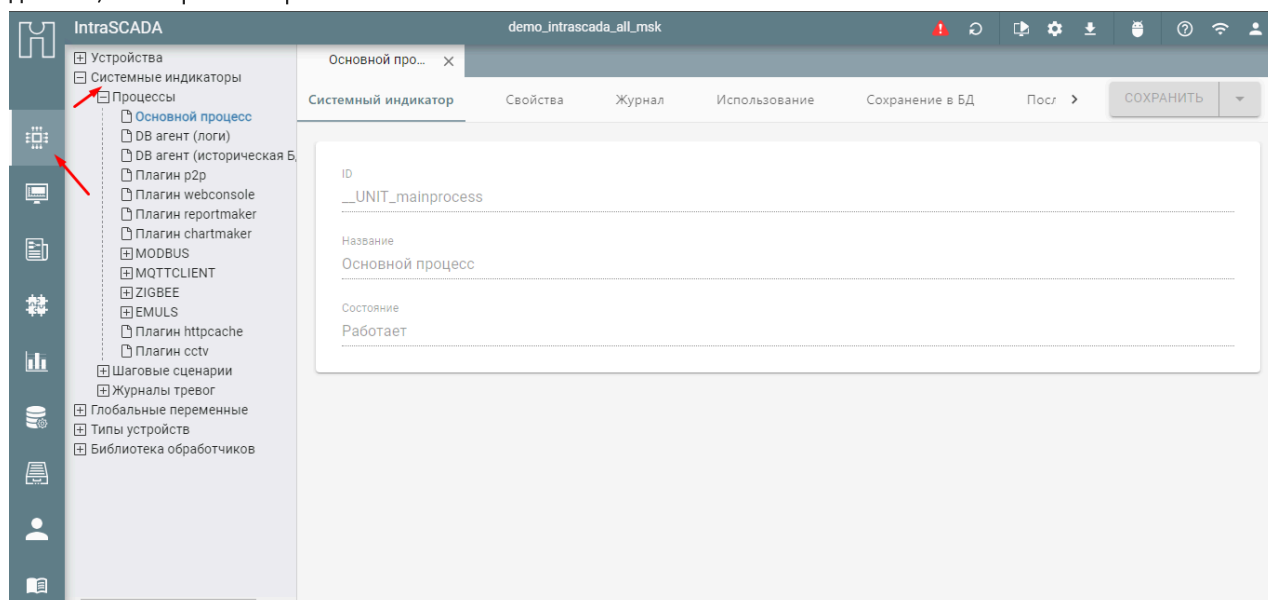


## Индикатор процесса

Для каждого процесса в системе автоматически создается **Системный индикатор**. Системные индикаторы доступны в дереве РМ рядом с узлом **Устройства**. По сути это виртуальные устройства со свойствами, определяемыми плагином.

Использовать значения свойств системных индикаторов можно, как и свойства обычных устройств:

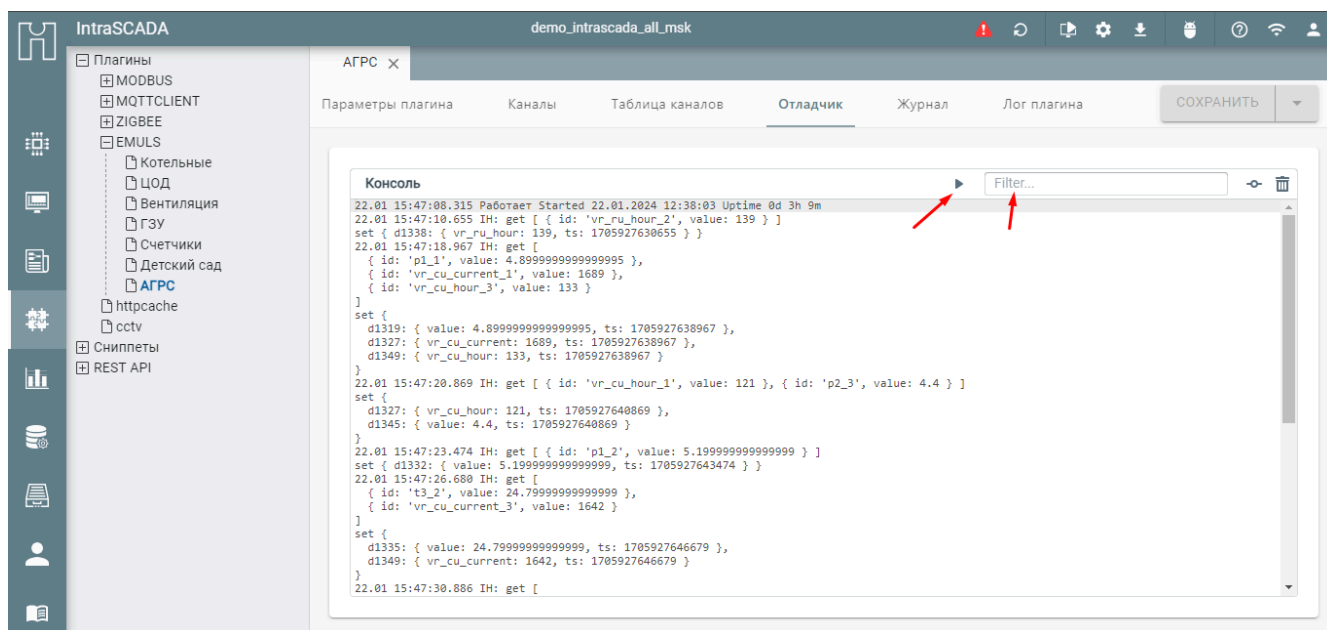
- выводить на индикацию на интерфейс
- записывать в БД
- создать сценарии, которые слушают и анализируют системные индикаторы и формируют сообщения Для каждого индикатора есть вкладки Лог и Журнал, на которых также можно увидеть соответствующие данные, о которых говорилось выше.



## Отладчик плагина

Работу каждого плагина можно наблюдать, используя **отладчик плагина**. В отладчике выводятся все сообщения плагина (независимо от уровня логирования), а также информация от сервера о получении данных от плагина и об отправке команды.

Если информации очень много, можно воспользоваться фильтром. Также можно перезапустить экземпляр плагина прямо из дерева плагинов, наблюдая в отладчике процесс запуска.



## Диагностика устройства/глобальной переменной

### Журнал

Устройство имеет свой журнал, который часто используется для отладки/диагностики. Этот журнал хранится в БД, но он самый короткоживущий, так как ограничен максимальным числом записей для одного устройства. Записи, используемые для диагностики работы конкретного устройства, не загромождают главный журнал системы.

Есть функция **Очистить журнал устройства** для конкретного устройства в дереве устройств. Число записей на одно устройство по умолчанию равно 100, настраивается в свойствах проекта. Чтобы сохранять информацию о значениях свойств, а также о выполнении команд, нужно поставить галочку **Сохранять в журнал устройства**. В журнале будет сохраняться информация об отправителе команды, это может быть пользователь или сценарий.

### Диагностика обработчиков

Обработчики устройств и глобальных переменных создаются, используя встроенный редактор кода. Все обработчики выполняются в отдельном от основного процесса потоке.

Это позволяет контролировать заикливание или слишком частый вызов обработчика

### Проверка при сохранении

После сохранения обработчика выполняется синтаксический анализ кода.

При ошибке выводится диагностическое сообщение. Скрипт с ошибкой блокируется.

### Блокировка при запуске

В случае ошибки выполнения обработчик блокируется.

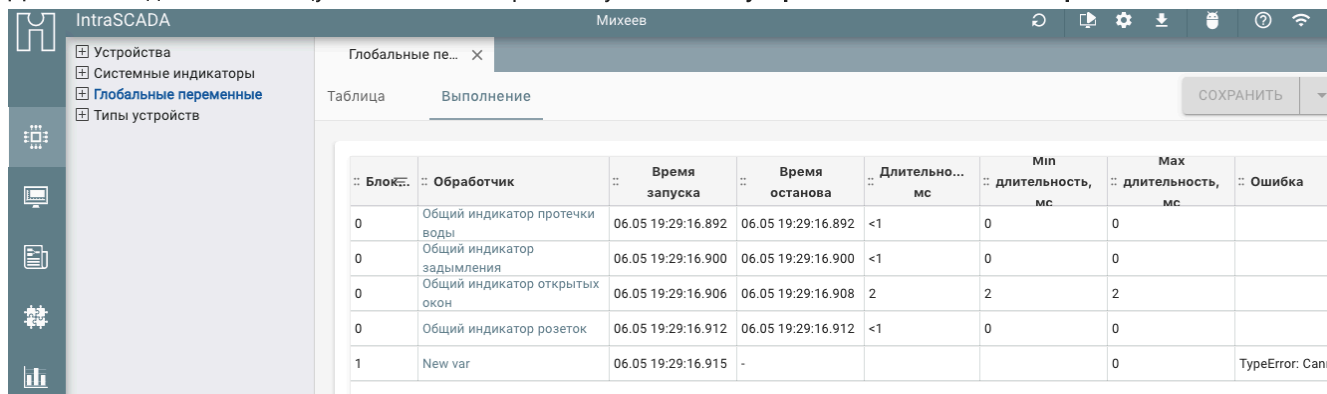
Также обработчик будет заблокирован, если он долго не отдает управление или слишком часто вызывается.

Сообщение об ошибке и флаг блокировки выводится в таблице Выполнение.

## Таблица Выполнение

Для каждого обработчика фиксируется время последнего запуска/завершения и число запусков.

Данные сведены в таблицу **Выполнение** корневых узлов **Типы устройств** и **Глобальные переменные**



Блок	Обработчик	Время запуска	Время останова	Длительность... мс	Мин длительность, мс	Мак длительность, мс	Ошибка
0	Общий индикатор протечки воды	06.05 19:29:16.892	06.05 19:29:16.892	<1	0	0	
0	Общий индикатор задымления	06.05 19:29:16.900	06.05 19:29:16.900	<1	0	0	
0	Общий индикатор открытых окон	06.05 19:29:16.906	06.05 19:29:16.908	2	2	2	
0	Общий индикатор розеток	06.05 19:29:16.912	06.05 19:29:16.912	<1	0	0	
1	New var	06.05 19:29:16.915	-			0	TypeError: Cani

## Диагностика скриптов

### Сценарии

Сценарии выполняются в отдельном от основного процесса потоке.

Это позволяет контролировать заикливание или слишком частый вызов сценария

### Проверка при сохранении

После сохранения сценария выполняется синтаксический анализ кода.

При ошибке выводится диагностическое сообщение. Скрипт с ошибкой блокируется.

### Блокировка при запуске

В случае ошибки выполнения сценарий блокируется. Также сценарий будет заблокирован, если он долго не отдает управление или слишком часто вызывается.

Сообщение об ошибке фиксируется в таблице **Выполнение**.

## Таблица Выполнение

Для каждого сценария (в случае мультисценария - для каждого экземпляра) фиксируется время последнего запуска/завершения и число запусков. Все сценарии сведены в таблицу **Выполнение** корневого узла **Сценарии**.

### Отладчик

В нижней части окна Редактора кода есть консоль, в которой можно видеть информацию о состоянии сценария и сработках триггеров.

Если сценарий запущен, по ходу выполнения выводятся трассировочные сообщения о таймерах, слушателях, выполненных командах. Можно вывести свои отладочные сообщения. Трассировка ведется в разрезе каждого экземпляра.

## Скрипты визуализации

Скрипты визуализации выполняются в основном процессе.

## Проверка при сохранении

После сохранения скрипта выполняется синтаксический анализ кода.

При ошибке выводится диагностическое сообщение, но скрипт не блокируется.

## Отладчик

В нижней части окна Редактора кода есть консоль, в которой можно видеть информацию о работе скрипта.

Скрипт может быть запущен для нескольких клиентов и, при наличии асинхронных операций в коде скрипта, выполняться параллельно. При запуске выводятся идентификатор и входные данные от клиента.

Можно вывести свои отладочные сообщения, используя в коде функцию **debug()**

## Регламент выхода новых версий

Разработка системы IntraSCADA выполняется в соответствии концепцией жизненного цикла разработки ПО (SDLC).

Новые версии платформы IntraSCADA выпускаются регулярно по мере добавления нового функционала.

Выпуск обновлений по устранению уязвимостей производится не реже 1 раза в месяц.

Выпуск обновлений по устранению критических уязвимостей занимает не более 7 дней.

Уведомление об обнаружении и устранении различных уязвимостей выполняется по электронной почте при наличии действующего договора на техническую поддержку.



# Технические требования

## Требования к серверу

Аппаратная конфигурация сервера подбирается исходя из масштаба автоматизированной системы.

Наименование	Минимальные требования	От 2500 до 10000 тегов	От 10000 тегов
Процессор	ARM Cortex, 800 МГц	от Intel Atom, 1 ГГц	от Intel® Core™ i5, 3 ГГц
Оперативная память (ОЗУ)	1 Гб	от 2 Гб	от 8 Гб
Жесткий диск	8 Гб	от 32 Гб	от 100 Гб

Большое значение имеет количество ядер процессора. Если планируется использование нескольких экземпляров плагинов и/или интенсивная запись в базу данных, увеличение числа ядер повысит скорость работы системы.

Размер жесткого диска зависит от количества тегов, частоты записи и времени хранения.

В системе IntraSCADA для каждого тега можно настроить периодичность сохранения в базу данных и срок хранения.

Для расчета можно исходить из значения 100 байт на одну запись в базе SQLite или 50 байт для PostgreSQL.

В качестве примера, если записывать 1000 тегов каждые 10 секунд, то необходимо 26 Гб в месяц.

Если применить умные алгоритмы записи, размер БД существенно сокращается.

## Требования к рабочим станциям и гаджетам

## Минимальные требования

Браузер	Chrome, Firefox, Safari или Яндекс Браузер актуальной версии
---------	--

# Установка системы IntraSCADA

Систему IntraSCADA можно установить на любой компьютер на архитектуре x64, arm, arm64 с операционными системами Linux, Windows, macOS

## Linux

Процедура установки системы IntraSCADA на операционные системы:

- [ALT Linux](#)
- [Astra Linux](#)
- [CentOS](#)
- [Debian](#)
- [Raspberry Pi OS](#)
- [Red Hat](#)
- [Ubuntu](#)
- [РЕД ОС](#)
- [POCA Linux](#)

## macOS

Установка системы на компьютеры с операционной системой macOS

[Процедура установки](#)

## Windows

Установка системы на компьютеры с операционной системой Windows

[Процедура установки](#)

## Контроллеры Wiren Board

Поддерживаемые контроллеры [Wiren Board](#):

- Wiren Board 6

- Wiren Board 7

[Процедура установки](#)

## Контроллеры JetHome

Поддерживаемые контроллеры [JetHome](#):

- JetHome JetHub D1+

[Процедура установки](#)

## Переустановка

Обычно переустанавливать систему не требуется. Обновление выполняется штатными средствами в интерфейсе самой системы.

В случае необходимости переустановить систему рекомендуем выполнить сохранение проекта.

См. [Перенос проекта](#)

# ALT Linux (RPM)

Варианты установки системы IntraSCADA на компьютеры с операционными системами:

- ALT Linux

## Установка

### Вариант 1 - Онлайн установка

На компьютере с доступом в интернет нужно в терминале выполнить следующие команды:

Войти в систему от пользователя root:

```
su -
```

Установить систему IntraSCADA:

```
wget https://rpm.ih-systems.com/altlinux/latest/intrascada/intrascada_x86-64.rpm  
apt-get install ./intrascada_x86-64.rpm
```

Будет установлена последняя версия, доступная на данный момент.

### Вариант 2 - Офлайн установка

На компьютере (сервере) без доступа в интернет установить систему IntraSCADA можно следующим способом:

1. **Скачать** на компьютер, где есть интернет, файл с rpm пакетом из репозитория для операционной системы:

- [ALT Linux](#)

Для разных архитектур процессора есть свои rpm пакеты системы IntraSCADA. Для определения архитектуры можно в терминале выполнить команду: **uname -m**

2. Скопировать загруженный файл на сервер.
3. Установить пакет на сервере

```
apt-get install ./intrascada_x86-64.rpm
```

Команда установки должна выполняться из папки, где находится файл с rpm пакетом.

## Удаление

- Для полного удаления системы IntraSCADA выполнить команду:

```
apt-get remove intrascada
```

# Astra Linux

## Установка

### Вариант 1 - Онлайн установка

Для операционной системы

- **Astra Linux Common Edition**

В терминале выполнить следующие команды:

1. Добавить репозиторий deb.ih-systems.com:

```
sudo wget -O - http://deb.ih-systems.com/setup | sudo -E bash
```

Эта команда выполняется один раз. Нет необходимости добавлять уже однажды добавленный репозиторий.

2. Установить систему IntraSCADA:

```
sudo apt update  
sudo apt install -y intrascada
```

Будет установлена последняя стабильная версия.

### Установка определенной версии

Следующей командой можно установить систему определенной версии:

```
sudo apt install -y intrascada=5.18.2
```

В этом примере 5.18.2 - номер версии для установки.

Если нужно установить версию ниже той, что установлена у вас, команда будет выглядеть так:

```
sudo apt install -y intrascada=5.18.2 --allow-downgrades
```

## Вариант 2 - Офлайн установка

Для операционных систем

- Astra Linux Common Edition
- Astra Linux Special Edition

На компьютере (сервере) без доступа в интернет установить систему IntraSCADA можно следующим способом:

1. Скачать на компьютер, где есть интернет, файл с deb пакетом:
  - [IntraSCADA для Astra Linux](#)

Для разных архитектур процессора есть свои deb пакеты системы IntraSCADA. Для определения архитектуры можно в терминале выполнить команду: **uname -m**

2. Скопировать загруженный файл на сервер.
3. Установить пакет IntraSCADA:

```
sudo apt install ./intrascada_5.18.2_amd64.deb
```

Вместо **intrascada\_5.18.2\_amd64.deb** поставьте имя файла загруженного deb пакета.  
Команда установки должна выполняться из папки, где находится файл с deb пакетом.

Также существует альтернативная команда для офлайн установки вместо `sudo apt install`:

```
sudo dpkg -i ./intrascada_x86-64.deb
```

**Не забудьте настроить FIREWALL для доступа к веб интерфейсу системы!!!**

## Удаление

- Для удаления системы IntraSCADA выполнить команду:

```
sudo apt remove intrascada
```

Будут удалены все файлы системы кроме проектов и плагинов.

- Для полного удаления системы IntraSCADA выполнить команду:

```
sudo apt purge intrascada
```



# Linux (Deb)

Варианты установки системы IntraSCADA на компьютеры с операционными системами:

- Debian
- Ubuntu
- Raspberry Pi OS

## Установка

### Вариант 1 - Онлайн установка

На компьютере с доступом в сеть интернет нужно в терминале выполнить следующие команды:

1. Добавить репозиторий deb.ih-systems.com:

```
sudo wget -O - http://deb.ih-systems.com/setup | sudo -E bash
```

Эта команда выполняется один раз. Нет необходимости добавлять уже однажды добавленный репозиторий.

2. Установить систему IntraSCADA:

```
sudo apt update  
sudo apt install -y intrascada
```

Будет установлена последняя стабильная версия.

### Установка определенной версии

Следующей командой можно установить систему определенной версии:

```
sudo apt install -y intrascada=5.18.2
```

В этом примере 5.18.2 - номер версии для установки.

Если нужно установить версию ниже той, что установлена у вас, команда будет выглядеть так:

```
sudo apt install -y intrascada=5.18.2 --allow-downgrades
```

## Вариант 2 - Офлайн установка

На компьютере (сервере) без доступа в сеть интернет установить систему IntraSCADA можно следующим способом:

1. **Скачать** на компьютер, где есть интернет, файл с deb пакетом из репозитория для операционной системы:
  - [Debian](#)
  - [Raspberry Pi OS](#)
  - [Ubuntu](#)

Для разных архитектур процессора есть свои deb пакеты системы IntraSCADA. Для определения архитектуры можно в терминале выполнить команду: **uname -m**

2. Скопировать загруженный файл на сервер.
3. Установить пакет на сервере

```
sudo apt install ./intrascada_5.18.2_amd64.deb
```

Вместо **intrascada\_5.18.2\_amd64.deb** поставьте имя файла загруженного deb пакета. Команда установки должна выполняться из папки, где находится файл с deb пакетом.

Также существует альтернативная команда для офлайн установки вместо `sudo apt install`:

```
sudo dpkg -i ./intrascada_x86-64.deb
```

## Удаление

- Для удаления системы IntraSCADA выполнить команду:

```
sudo apt remove intrascada
```

Будут удалены все файлы системы кроме проектов и плагинов.

- Для полного удаления системы IntraSCADA выполнить команду:

```
sudo apt purge intrascada
```

Видеоуроки по установке системы:

- [VK Видео](#)
- [Youtube](#)
- [Rutube](#)

# Linux (RPM)

Варианты установки системы IntraSCADA на компьютеры с операционными системами:

- Red Hat
- CentOS
- Fedora

Для корректной работы системы на Fedora Linux нужно разрешить порт в firewalld командами:

```
sudo firewall-cmd --add-port=8088/tcp --permanent  
sudo firewall-cmd --reload
```

## Установка

### Вариант 1 - Онлайн установка

На компьютере с доступом в сеть интернет нужно в терминале выполнить следующие команды:

1. Добавить репозиторий rpm.ih-systems.com:

```
sudo wget -O - http://rpm.ih-systems.com/setup | sudo bash
```

Эта команда выполняется один раз. Нет необходимости добавлять уже однажды добавленный репозиторий.

2. Установить систему IntraSCADA:

- Для версий Fedora 21 и ниже, Red Hat/CentOS 7 и ниже:

```
sudo yum install -y intrascada
```

- Для версий Fedora 22 и выше, Red Hat/CentOS 8 и выше:

```
sudo dnf install -y intrascada
```

Будет установлена последняя стабильная версия.

## Установка определенной версии

Следующей командой можно установить систему определенной версии:

- Для версий Fedora 21 и ниже, Red Hat/CentOS 7 и ниже:

```
sudo yum install -y intrascada=5.18.2
```

- Для версий Fedora 22 и выше, Red Hat/CentOS 8 и выше:

```
sudo dnf install -y intrascada=5.18.2
```

В этом примере 5.18.2 - номер версии для установки.

Если нужно установить версию ниже той, что установлена у вас, команда будет выглядеть так:

- Для версий Fedora 21 и ниже, Red Hat/CentOS 7 и ниже:

```
sudo yum install -y intrascada=5.18.2 --allow-downgrades
```

- Для версий Fedora 22 и выше, Red Hat/CentOS 8 и выше:

```
sudo dnf install -y intrascada=5.18.2 --allow-downgrades
```

## Вариант 2 - Офлайн установка

На компьютере (сервере) без доступа в сеть интернет установить систему IntraSCADA можно следующим способом:

1. **Скачать** на компьютер, где есть интернет, файл с rpm пакетом из репозитория для операционной системы:

- [Red Hat](#)
- [CentOS](#)
- [Fedora Linux](#)

Для разных архитектур процессора есть свои rpm пакеты системы IntraSCADA. Для определения архитектуры можно в терминале выполнить команду: **uname -m**

2. Скопировать загруженный файл на сервер.

### 3. Установить пакет на сервере

- Для версий Fedora 21 и ниже, Red Hat/CentOS 7 и ниже:

```
sudo yum install ./intrascada-5.18.2-1.x86_64.rpm
```

- Для версий Fedora 22 и выше, Red Hat/CentOS 8 и выше:

```
sudo dnf install ./intrascada-5.18.2-1.x86_64.rpm
```

Вместо **intrascada-5.18.2-1.x86\_64.rpm** поставьте имя файла загруженного rpm пакета.  
Команда установки должна выполняться из папки, где находится файл с rpm пакетом.

## Удаление

- Для полного удаления системы IntraSCADA выполнить команду:
- Для версий Fedora 21 и ниже, Red Hat/CentOS 7 и ниже:

```
sudo yum remove intrascada
```

- Для версий Fedora 22 и выше, Red Hat/CentOS 8 и выше:

```
sudo dnf remove intrascada
```

# Windows

## Установка

Систему IntraSCADA можно установить на любой компьютер с операционной системой Windows 10, Windows 11 или Windows Server

[Скачать систему IntraSCADA](#)

Установка системы выполняется около 2 минут.

Для дальнейшей работы необходимо [войти в систему](#)

## Удаление

Удаление системы IntraSCADA производится штатными средствами операционной системы Windows.

## Обновление

Для обновления системы IntraSCADA необходимо скачать новую версию и установить ее поверх существующей.

# Контроллеры Wiren Board

Установка системы IntraSCADA.

Поддерживаются контроллеры [Wiren Board](#):

- Wiren Board 6
- Wiren Board 7

## Установка

### Вариант 1 - Онлайн установка

На контроллере с доступом в сеть интернет нужно в терминале выполнить следующие команды:

```
apt update
apt install -y intrascada
```

Будет установлена последняя стабильная версия.

### Установка определенной версии

Следующей командой можно установить систему определенной версии:

```
apt install -y intrascada=5.18.2
```

В этом примере 5.18.2 - номер версии для установки.

Если нужно установить версию ниже той, что установлена у вас, команда будет выглядеть так:

```
apt install -y intrascada=5.18.2 --allow-downgrades
```

### Вариант 2 - Офлайн установка

На контроллере без доступа в сеть интернет установить систему IntraSCADA можно следующим способом:

1. Скачать на компьютер, где есть интернет, файлы с deb пакетами:
  - [zip и libatomic1](#)
  - [IntraSCADA](#)
2. Скопировать загруженные файлы на контроллер.
3. Установить пакет zip на контроллере:



```
apt install ./zip_3.0-11_armhf.deb
```

4. Установить пакет libatomic1 на контроллере:

```
apt install ./libatomic1_6.3.0-18_armhf.deb
```

5. Установить пакет IntraSCADA на контроллере:

```
apt install ./intrascada_5.11.34_amd64.deb
```

Вместо **intrascada\_5.11.34\_amd64.deb** поставьте имя файла загруженного deb пакета.  
Команда установки должна выполняться из папки, где находится файл с deb пакетом.

## Удаление

- Для удаления системы IntraSCADA выполнить команду:

```
apt remove intrascada
```

Будут удалены все файлы системы кроме проектов и плагинов.

- Для полного удаления системы IntraSCADA выполнить команду:

```
apt purge intrascada
```

# Контроллеры JetHome

Установка системы IntraSCADA на контроллеры [JetHome](#)

Поддерживаются контроллеры:

- JetHome JetHub D1+

## Установка

### Вариант 1 - Онлайн установка

На контроллере с доступом в сеть интернет нужно в терминале выполнить следующие команды:

```
sudo wget -O - http://deb.ih-systems.com/setup | sudo -E bash  
apt update  
apt install -y intrascada
```

Будет установлена последняя стабильная версия.

### Установка определенной версии

Следующей командой можно установить систему определенной версии:

```
apt install -y intrascada=5.18.2
```

В этом примере 5.18.2 - номер версии для установки.

Если нужно установить версию ниже той, что установлена у вас, команда будет выглядеть так:

```
apt install -y intrascada=5.18.2 --allow-downgrades
```

### Вариант 2 - Офлайн установка

На контроллере без доступа в сеть интернет установить систему IntraSCADA можно следующим способом:

1. Скачать на компьютер, где есть интернет, файл с deb пакетом:
  - [IntraSCADA](#)
2. Скопировать загруженный файл на контроллер.
3. Установить пакет IntraSCADA на контроллере:

```
apt install ./intrascada_5.18.2_amd64.deb
```

Вместо **intrascada\_5.18.2\_amd64.deb** поставьте имя файла загруженного deb пакета.  
Команда установки должна выполняться из папки, где находится файл с deb пакетом.

Также существует альтернативная команда для офлайн установки вместо apt install:

```
sudo dpkg -i ./intrascada_x86-64.deb
```

## Удаление

- Для удаления системы IntraSCADA выполнить команду:

```
apt remove intrascada
```

Будут удалены все файлы системы кроме проектов и плагинов.

- Для полного удаления системы IntraSCADA выполнить команду:

```
apt purge intrascada
```

# macOS

Установка системы IntraSCADA на компьютерах с операционной системой macOS

## Вариант установки через GUI

Скачать файл и запустить двойным кликом:

```
http://macos.ih-systems.com/latest/intrascada/
```

Будет установлена последняя актуальная версия

## Вариант установки через консоль

На компьютере с доступом в сеть интернет нужно в терминале выполнить следующие команды:

```
sudo wget http://macos.ih-systems.com/intrascada/intrascada-x64-5.18.20.pkg
sudo installer -pkg ./intrascada-x64-5.18.20.pkg -target /
```

Обратите внимание, что для установки системы через терминал может потребоваться установка утилиты **wget**.  
В данном примере показана установка версии 5.18.20, вы так же можете установить другую версию, поменяв ее номер в команде

## Удаление

Для удаления системы IntraSCADA выполнить команды остановки процесса и удаления папок:

```
sudo launchctl unload /Library/LaunchDaemons/intrascada.plist
sudo launchctl stop intrascada

sudo rm -r /Library/intrascada
sudo rm -r /var/lib/intrascada
sudo rm /Library/LaunchDaemons/ih-v5.plist
```

# РЕД ОС

Варианты установки системы IntraSCADA на компьютеры с операционными системами:

- РЕД ОС

## Установка

### Вариант 1 - Онлайн установка

На компьютере с доступом в сеть интернет нужно в терминале выполнить следующие команды:

1. Добавить репозиторий rpm.ih-systems.com:

```
sudo wget -O - http://rpm.ih-systems.com/setup | sudo bash
```

Эта команда выполняется один раз. Нет необходимости добавлять уже однажды добавленный репозиторий.

2. Установить систему IntraSCADA:

Для РЕД ОС 7.3 и ниже:

```
sudo yum install -y intrascada
```

Будет установлена последняя стабильная версия.

Командой **sudo yum install intrascada-5.17.80-1.x86\_64** можно установить систему определенной версии. В этом примере 5.17.80 - номер версии для установки.

Для РЕД ОС 8 и выше:

```
sudo dnf install -y intrascada
```

Будет установлена последняя стабильная версия.

Командой **sudo dnf install intrascada-5.17.80-1.x86\_64** можно установить систему определенной версии. В этом примере 5.17.80 - номер версии для установки.

## Вариант 2 - Офлайн установка

На компьютере (сервере) без доступа в сеть интернет установить систему IntraSCADA можно следующим способом:

1. **Скачать** на компьютер, где есть интернет, файл с rpm пакетом из репозитория для операционной системы:

- [РЕД ОС](#)

Для разных архитектур процессора есть свои rpm пакеты системы IntraSCADA. Для определения архитектуры можно в терминале выполнить команду: **uname -m**

2. Скопировать загруженный файл на сервер.

3. Установить пакет на сервере

Для РЕД ОС 7.3 и ниже:

```
sudo yum install ./intrascada-5.17.80-1.x86_64.rpm
```

Для РЕД ОС 8 и выше:

```
sudo dnf install ./intrascada-5.17.80-1.x86_64.rpm
```

Вместо **intrascada-5.17.80-1.x86\_64.rpm** поставьте имя файла загруженного rpm пакета. Команда установки должна выполняться из папки, где находится файл с rpm пакетом.

## Удаление

- Для полного удаления системы IntraSCADA выполнить команду:

Для РЕД ОС 7.3 и ниже:

```
sudo yum remove intrascada
```

Для РЕД ОС 8 и выше:

```
sudo dnf remove intrascada
```

## Установка РЕД ОС

[Подробная статья](#) по установке РЕД ОС и системы IntraSCADA в PDF формате

# Первый запуск

В систему можно войти с любого компьютера в вашей локальной сети через браузеры Chrome, Safari или Firefox.

## Пользовательский интерфейс

Для входа в пользовательский интерфейс в адресной строке браузера наберите строку запроса:

<http://xxx.xxx.xxx.xxx:port>

xxx.xxx.xxx.xxx — IP адрес сервера в вашей локальной сети

port - номер порта, по умолчанию 8088

Если вход выполняется локально на том же компьютере, где установлена система:

<http://localhost:8088>

или

<http://127.0.0.1:8088>

В окне авторизации введите имя пользователя и пароль:

имя пользователя: admin

пароль: 202020

## Интерфейс разработчика

Для входа в интерфейс разработчика (Project Manager) наберите строку запроса:

<http://xxx.xxx.xxx.xxx:port/admin>

Так же, как для входа в интерфейс пользователя, но в конце добавить **/admin**

В целях безопасности после установки системы настоятельно рекомендуем [изменить имя и пароль](#)

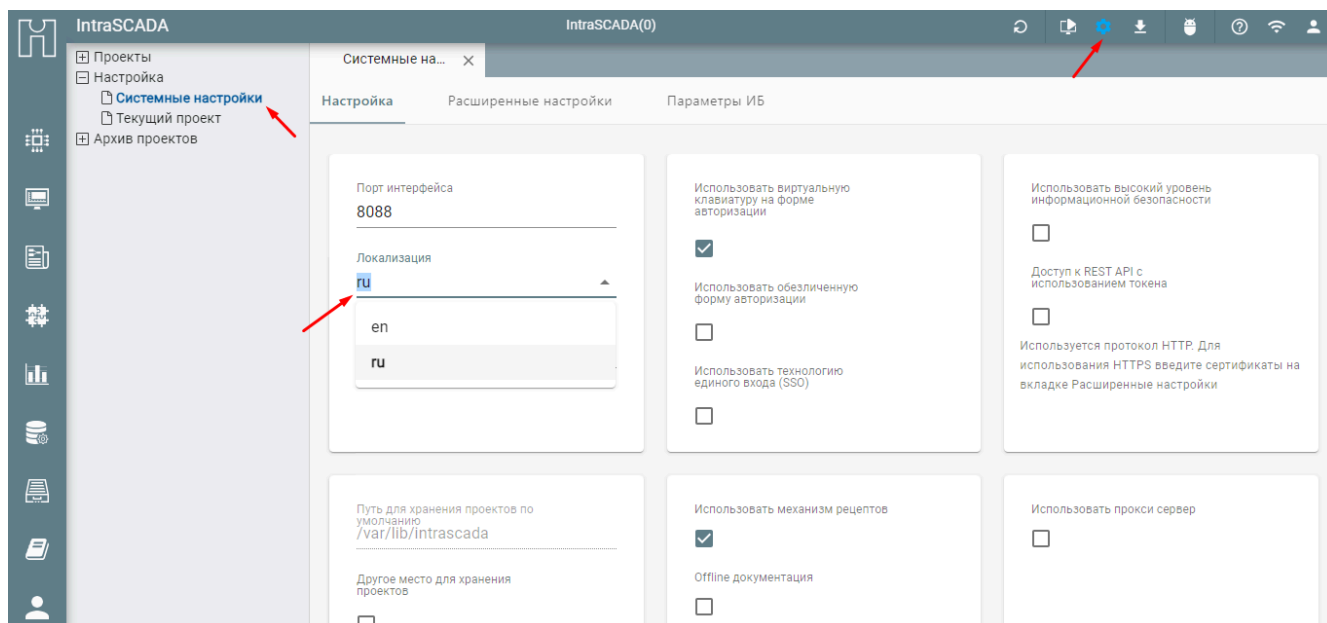
## Локализация

Для переключения языка необходимо нажать шестеренку в правом верхнем углу.

В разделе Системные настройки доступно поле **Локализация**. Для применения настроек локализации требуется перезагрузить систему.

Нажать кнопку "Перезагрузить сервер с этими настройками":





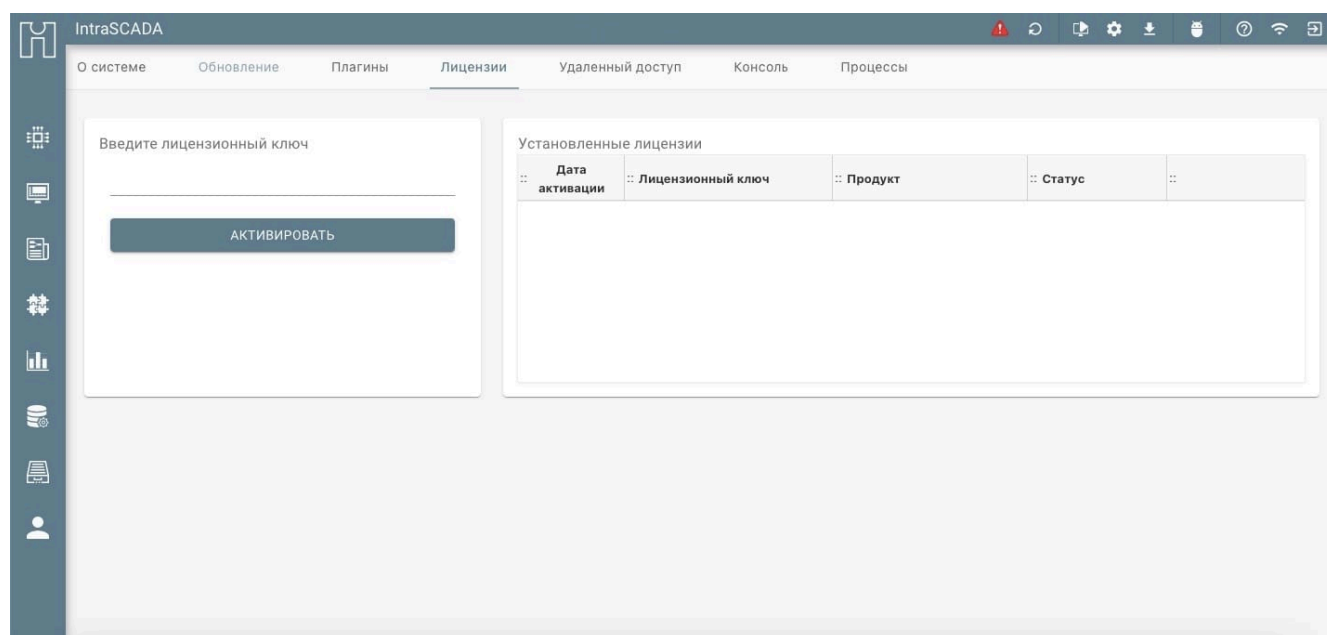
# Лицензирование

## Процедура лицензирования **online**

Для установки лицензии **online** необходим доступ в интернет.

Для работы системы с уже установленной лицензией доступ в интернет не требуется.

Установка лицензионных ключей выполняется на вкладке "Лицензии":

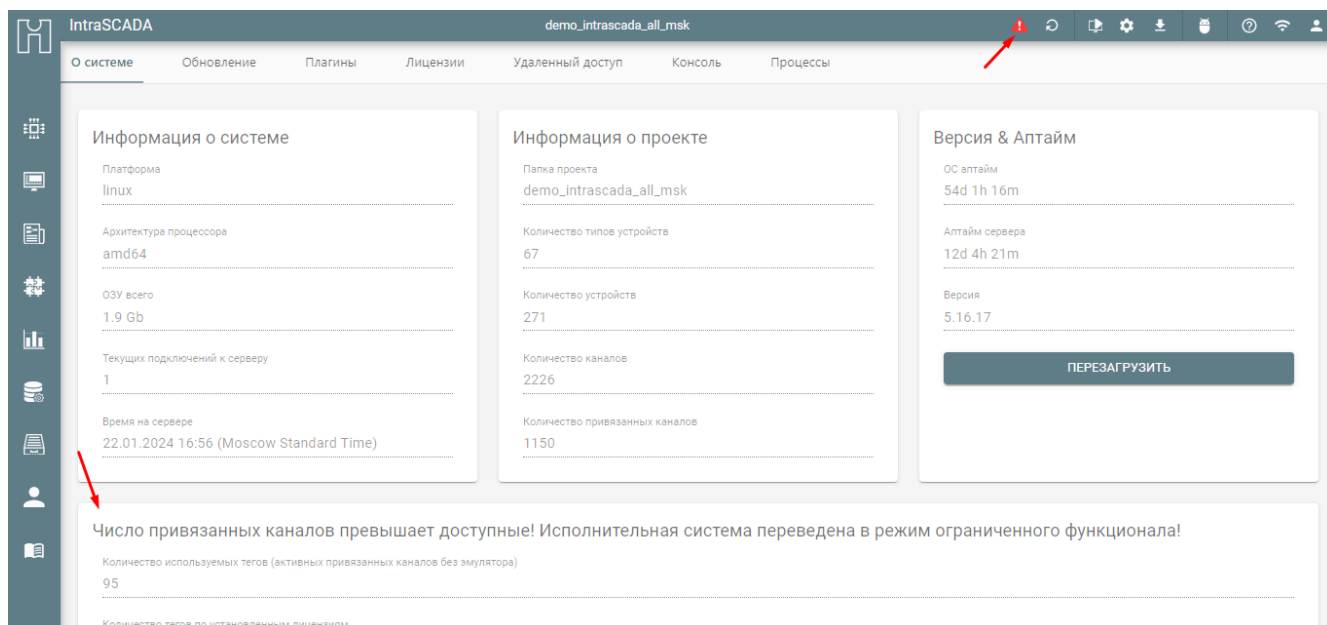


Для активации лицензии необходимо ввести лицензионный ключ и нажать кнопку "Активировать"

Необходимое количество лицензионных ключей можно приобрести в [интернет-магазине](#)

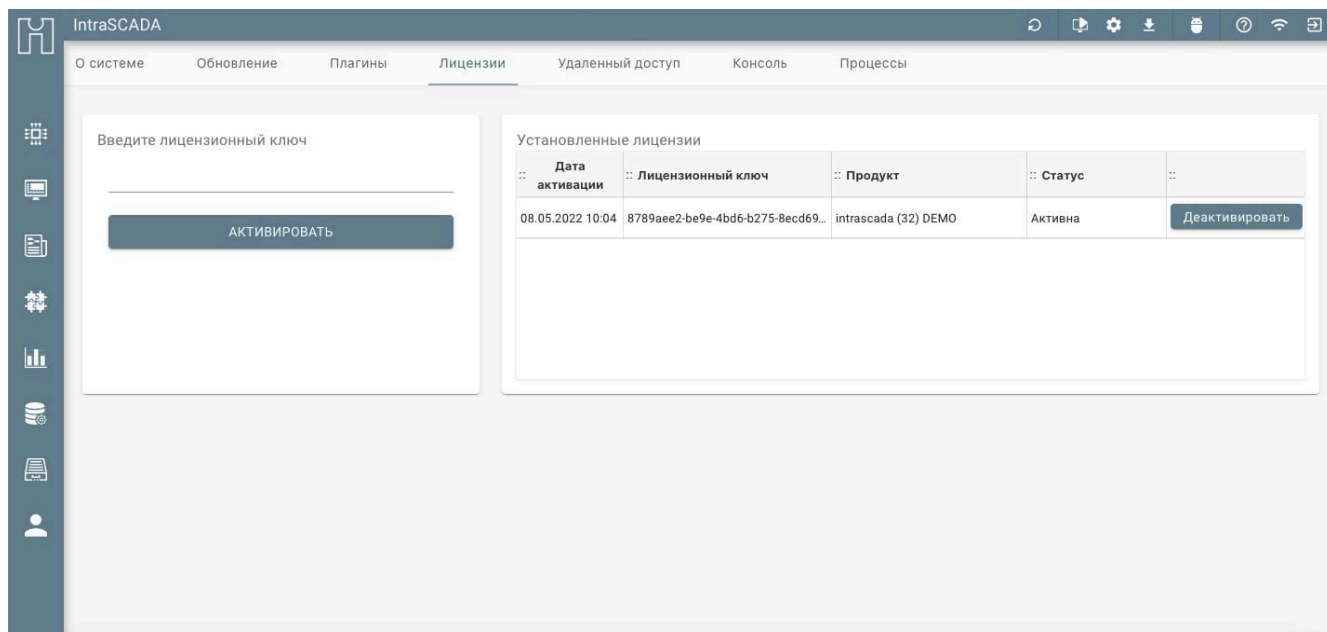
Расширение системы выполняется простым добавлением нового лицензионного ключа.

При отсутствии лицензионных ключей система выводит предупреждение:



Без лицензионных ключей система IntraSCADA работает в полном объеме, за исключением связи с каналами подключенных устройств.

Для деактивации лицензии необходимо нажать кнопку "Деактивировать":



Деактивация лицензии может быть востребована при переносе системы с одного компьютера на другой. В этом случае достаточно деактивировать лицензию на старом компьютере и активировать на новом.

## Процедура лицензирования **offline**

При невозможности обеспечить доступ в интернет воспользуйтесь процедурой **offline** лицензирования.

Она состоит из 3 шагов:

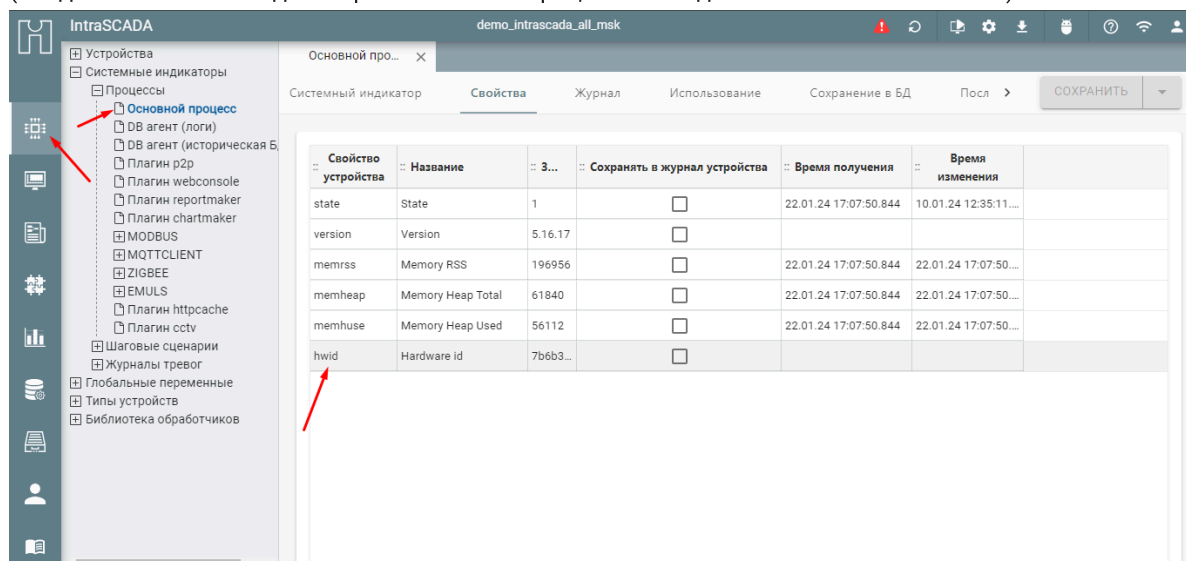
1. Если система еще не установлена, установите систему на ваш сервер.

2. Отправьте запрос на электронную почту [support@ih-systems.com](mailto:support@ih-systems.com)

В запросе укажите:

- Hardware id сервера, на который будет установлена лицензия.

(Раздел Системные индикаторы - Основной процесс - Вкладка Свойства - Hardware id)



- лицензионный ключ, для которого требуется **offline** лицензирование
- реквизиты организации, которой принадлежит лицензия (название, ИНН, контактное лицо)
- дату приобретения лицензии

В ответ будет выслано письмо, содержащее файл, позволяющий активировать лицензию на указанном сервере

3. Полученный файл загрузите на ваш сервер через кнопку Импорт (меню наверху справа).

В случае успешной активации в окне загрузки будет выведено сообщение: **Лицензия успешно активирована**

На вкладке Лицензии появится новая лицензия.

## Введение

Это руководство предназначено для новичков, которые впервые знакомятся с системой IntraSCADA — программной платформой для систем диспетчеризации и мониторинга. Цель — помочь избежать типичных проблем при установке, запуске и начальной настройке. Мы пройдем пошагово от установки до подключения устройств на основе протокола Modbus, включая визуализацию. IntraSCADA — это клиент-серверная SCADA-система, поддерживающая веб-интерфейс для мониторинга и управления. Она работает на Linux, Windows, macOS, ARM-устройствах (например, Raspberry Pi, Wiren Board, JetHome). Без лицензии система функционирует в ограниченном режиме, но также доступна демо-лицензия на 100 тегов сроком на 30 дней. Emuls-плагин установлен по умолчанию для симуляции устройств, Modbus можно установить в разделе *Плагины* основного дашборда.

### Важные замечания перед началом

- Сервер запускается на порту 8088 по умолчанию.
- Логин по умолчанию: **admin**, пароль: **202020** (измените после первого входа для безопасности).
- Если нет интернета, используйте оффлайн-методы (описаны ниже).

## Технические требования

Аппаратные требования зависят от масштаба проекта. Минимальные выглядят так:

Для сервера:

- Процессор: ARM Cortex 800 МГц или Intel Atom 1 ГГц.
- ОЗУ: 1 ГБ
- Диск: 8 ГБ
- ОС: Linux (ALT, Astra, Debian, Ubuntu, Red Hat, CentOS, РЕД ОС, Fedora), Windows 10+, macOS, ARM-платформы (Wiren Board 6+, JetHome JetHub D1+, Raspberry Pi).

Для клиента:

- Любой ПК, планшет или смартфон с браузером.

## Установка системы

Установка простая, занимает 2–5 минут.

Систему IntraSCADA можно установить на почти любой компьютер на архитектуре x64, arm, arm64 с операционными системами Linux, Windows, macOS

Методы установки для разных операционных систем доступны по [ссылке](#)

Сервер запустится автоматически после установки.

Возможные проблемы при установке:

- Ошибка версии: Проверьте архитектуру (uname -m)
- Недостаточно места на диске

## Переустановка системы

Обычно переустанавливать систему не требуется. Обновление выполняется штатными средствами в интерфейсе самой системы. В случае необходимости переустановить систему рекомендуем выполнить сохранение проекта. (См. [Перенос проекта](#)).

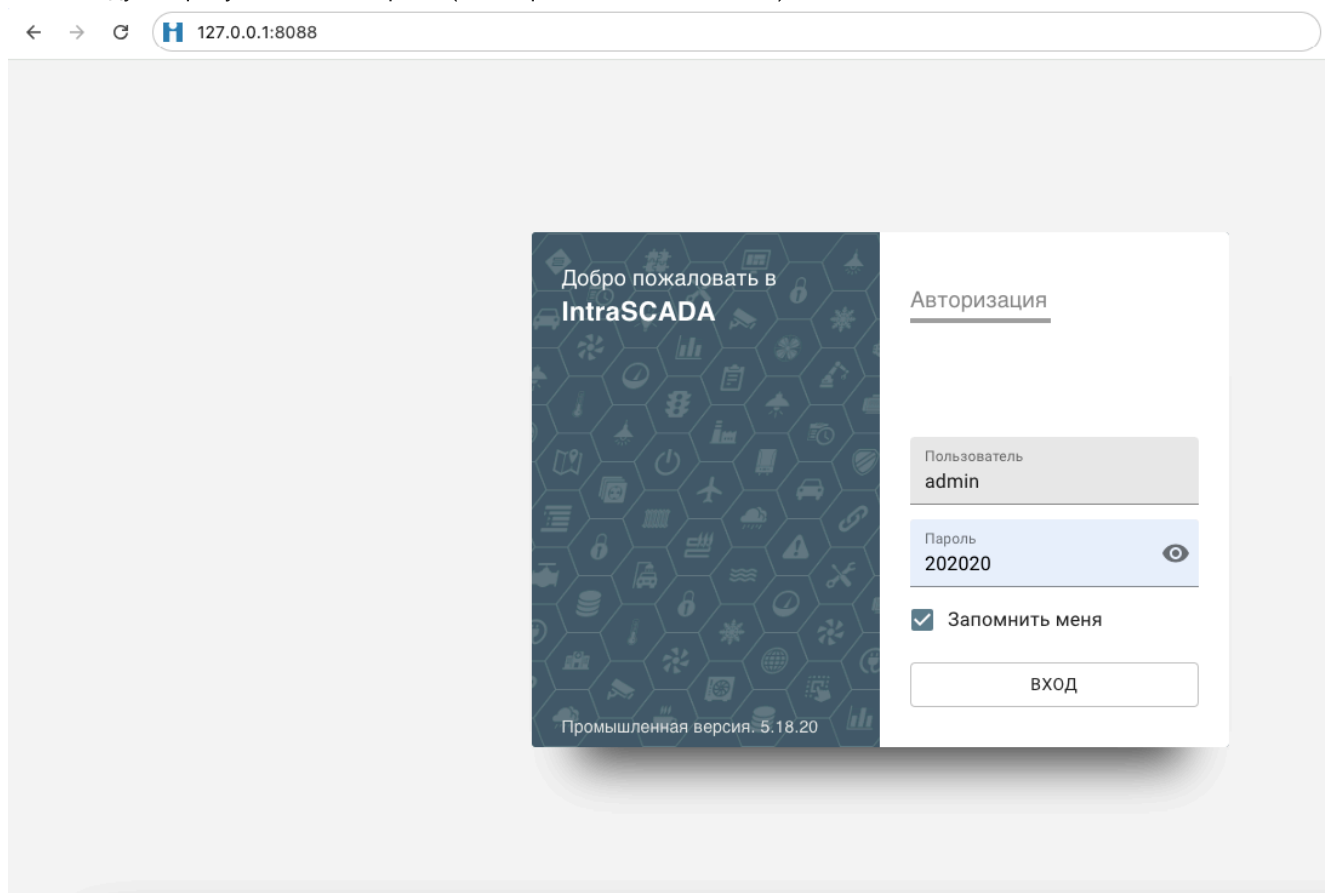
## Первый запуск

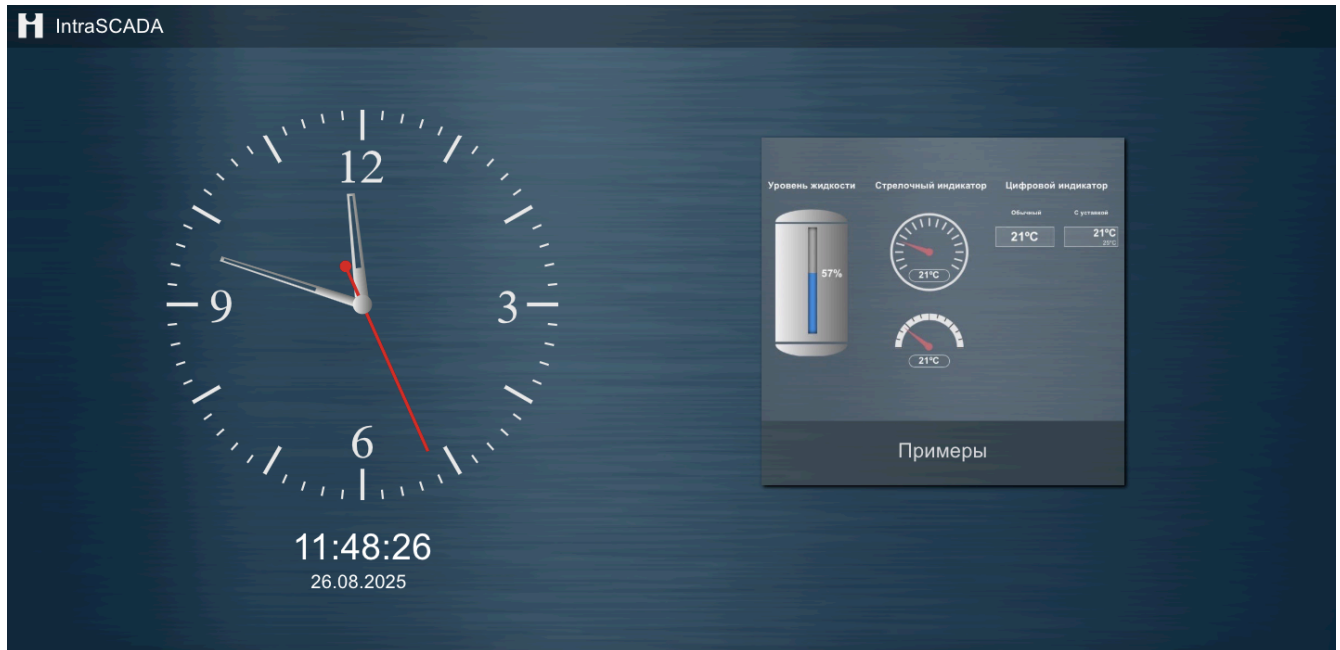
Сервер запускается автоматически как сервис.

Откройте браузер и перейдите по адресу: <http://localhost:8088> (локально) или <http://<IP-сервера>:8088> (из сети).

Войдите: Логин — admin, пароль — 202020.

Рекомендуем сразу изменить пароль (в настройках пользователя).





Вы в пользовательском интерфейсе. Для перехода в интерфейс разработчика добавьте в конце адреса /admin: <http://localhost:8088/admin>.

Обратите внимание, что с установкой системы также устанавливается демо-проект, который вы можете использовать в качестве примера для своих решений.

Локализация: Нажмите шестеренку вверху справа, выберите язык в "Системные настройки" → "Локализация", перезагрузите сервер.

Возможные проблемы при первом запуске:

- Firewall блокирует: Разрешите порт 8088:

```
sudo firewall-cmd --add-port=8088/tcp --permanent
sudo firewall-cmd --reload
```

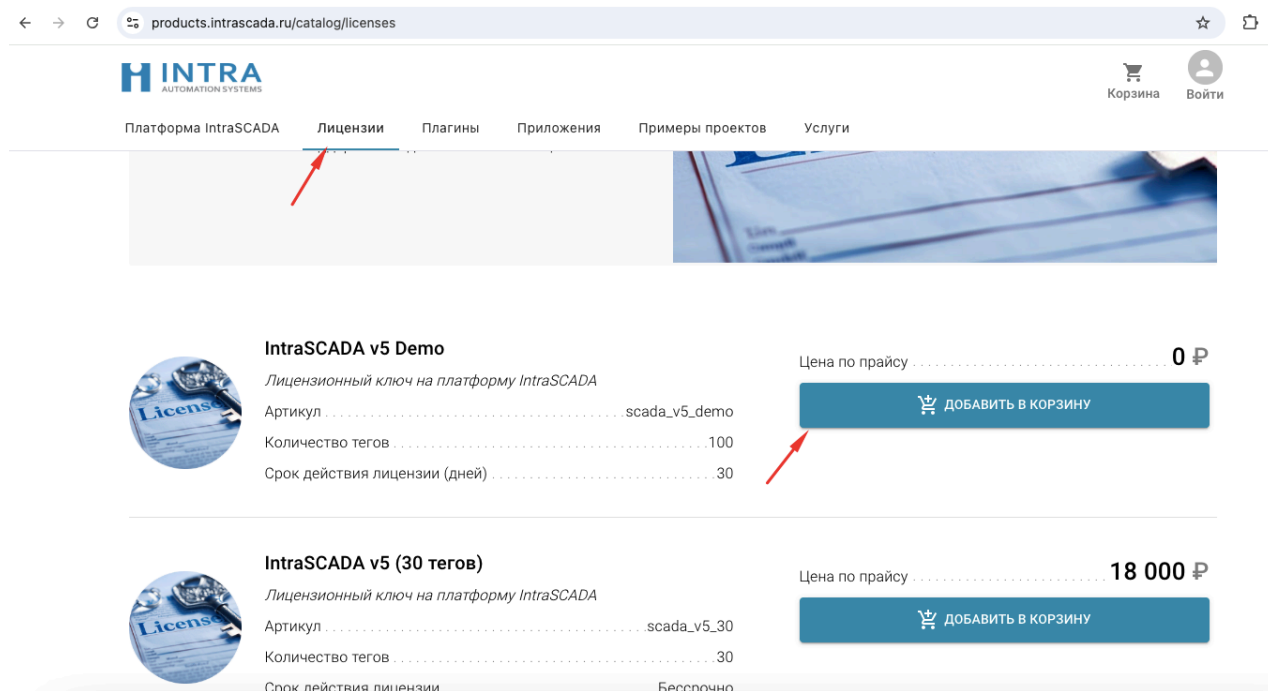
## Получение демо-лицензии

Лицензия нужна для того, чтобы подключать физические устройства ввода-вывода, стоимость лицензии зависит от количества тегов.

Тег — это дискретная или аналоговая переменная, считываемая из физического устройства ввода-вывода.

В системе IntraSCADA есть возможность получить бесплатную демо-лицензию на 100 тегов сроком на 30 дней.

- Заходим на сайт [products.intrascada.ru](https://products.intrascada.ru) в раздел Лицензии:



- Выбираем демо-лицензию и жмём **Добавить в корзину**



- Переходим в корзину и жмём **Оформить заказ**

The screenshot shows the IntraSCADA v5 Demo product page on the left and a shopping cart on the right. The product page lists two items: 'IntraSCADA v5 Demo' (100 units, 30-day license) and 'IntraSCADA v5 (30 тегов)' (30 units, lifetime license). The shopping cart shows the 'IntraSCADA v5 Demo' item with a quantity of 1 and a price of 0 RUB. A red arrow points to the 'Оформить заказ' button in the bottom right corner of the cart.

**Корзина** Очистить корзину ×

**IntraSCADA v5 Demo**  
Лицензионный ключ на платформу IntraSCADA

0 RUB / шт.

**Итого: 0 RUB** Оформить заказ

- Заполняем контактную информацию и жмём **Оформить**

The screenshot shows the checkout process with four steps: 1. Контактная информация, 2. Информация о заказе, 3. Заказ, and 4. Оплата. A red arrow points to the '1. Контактная информация' step. The 'Состав заказа' section shows the 'IntraSCADA v5 Demo' item with a price of 0 RUB and a quantity of 1. The 'Купон' section has a 'Ваш купон' input field and a 'ПРИМЕНИТЬ' button. The 'Итого' section shows 'К оплате: 0 RUB' and a red arrow pointing to the 'Оформить' button.

**1. КОНТАКТНАЯ ИНФОРМАЦИЯ** **2. ИНФОРМАЦИЯ О ЗАКАЗЕ** 3. ЗАКАЗ

**Состав заказа**

**IntraSCADA v5 Demo**  
Лицензионный ключ на платформу IntraSCADA  
Цена по прайсу: 0 RUB  
Количество: 1 шт.  
Сумма: 0 RUB

**Купон**

Ваш купон ПРИМЕНИТЬ

**Итого**

К оплате: 0 RUB Оформить

- Готово! Наш заказ оформлен, лицензионный ключ придет на электронную почту, которую мы указали при оформлении

## Спасибо!

Ваш заказ № 25217 успешно оформлен.

### Основное

Номер заказа: 25217  
 Дата оформления заказа: 26.08.2025  
 К оплате: 0 Р

### Состав заказа



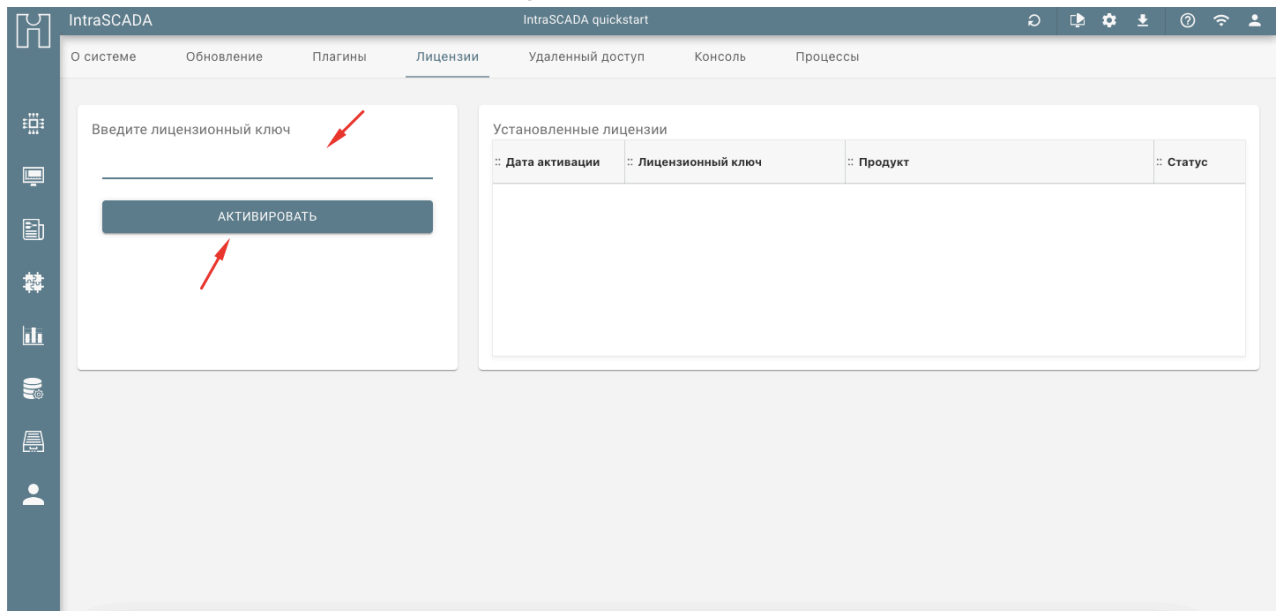
IntraSCADA v5 Demo

Цена по прайсу: 0 Р  
 Количество: 1 шт.

Сумма: 0 Р

- Далее переходим в дашборд системы на вкладку **Лицензии**

Вводим лицензионный ключ и жмём **Активировать**



Готово! Демо-лицензия успешно активирована!

О системе

Обновление

Плагины

Лицензии

Удаленный доступ

Консоль

Процессы

IntraSCADA

IntraSCADA quickstart

Введите лицензионный ключ

АКТИВИРОВАТЬ

Установленные лицензии

Дата активации	Лицен... ключ	Продукт	Статус	
02.09.2025 10:09	4c54152e-c...	DEMO intrascada (100)	Активна до 02.10.2025...	Удалить

Лицензия успешно активирована

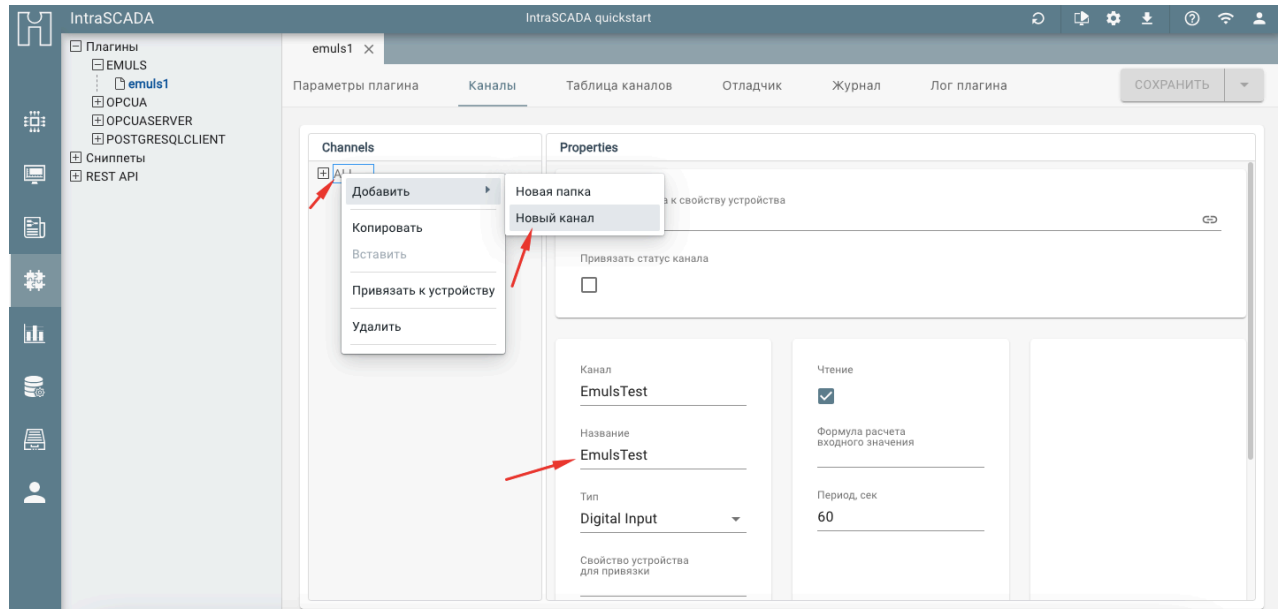
## Настройка устройств

### Настройка с эмулятором

Плагин эмулятор(emuls) установлен по умолчанию в системе.

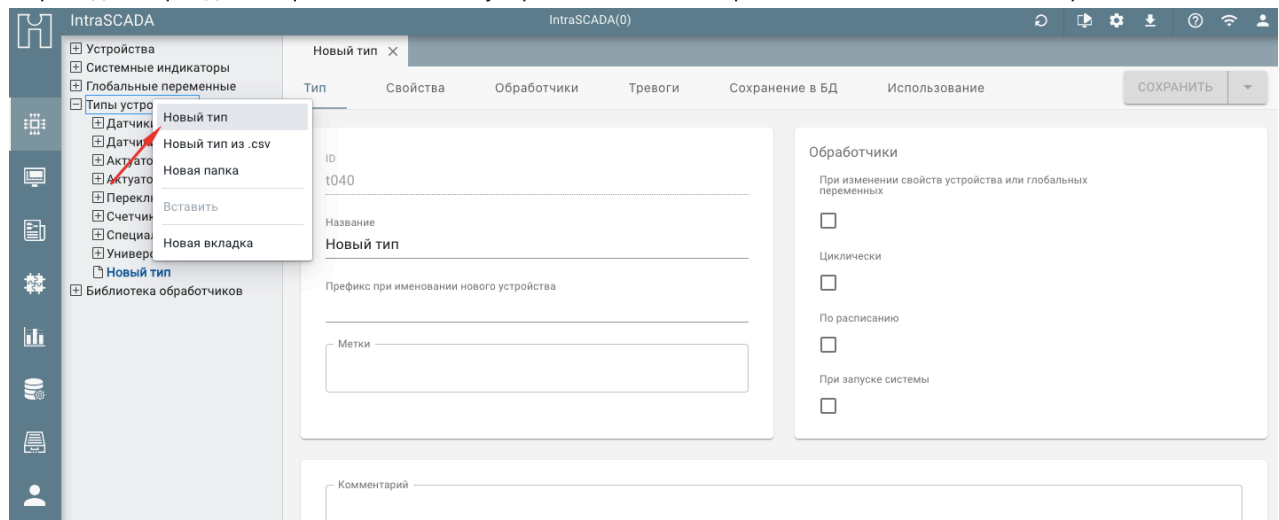
Для настройки переходим в раздел Источники данных -> Плагины -> Emuls

- Добавляем новый канал, назовём его *EmulsTest*

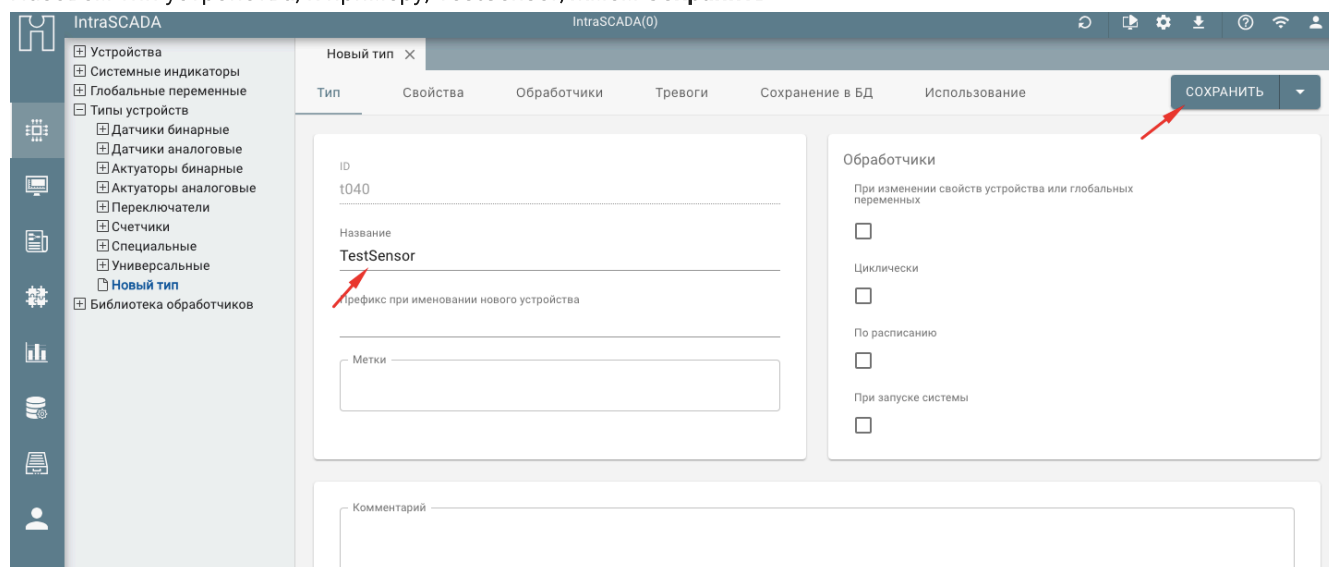


- Создаем тип устройства

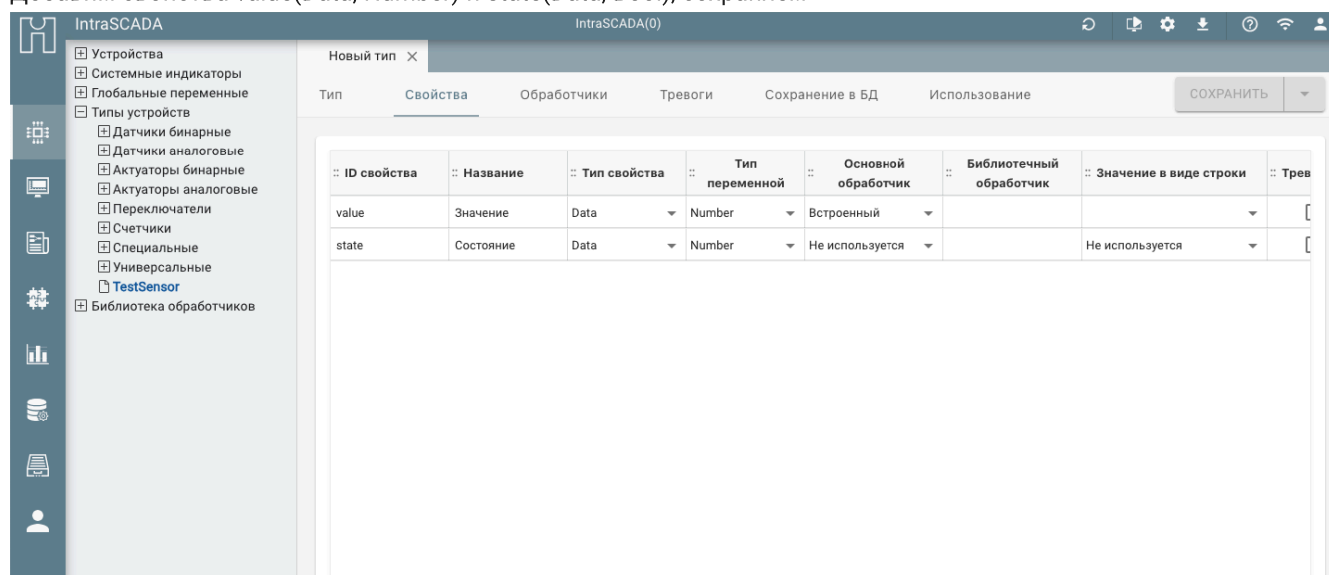
Переходим в раздел Устройства -> Типы устройств, кликаем правой кнопкой мыши и выбираем **Новый тип**



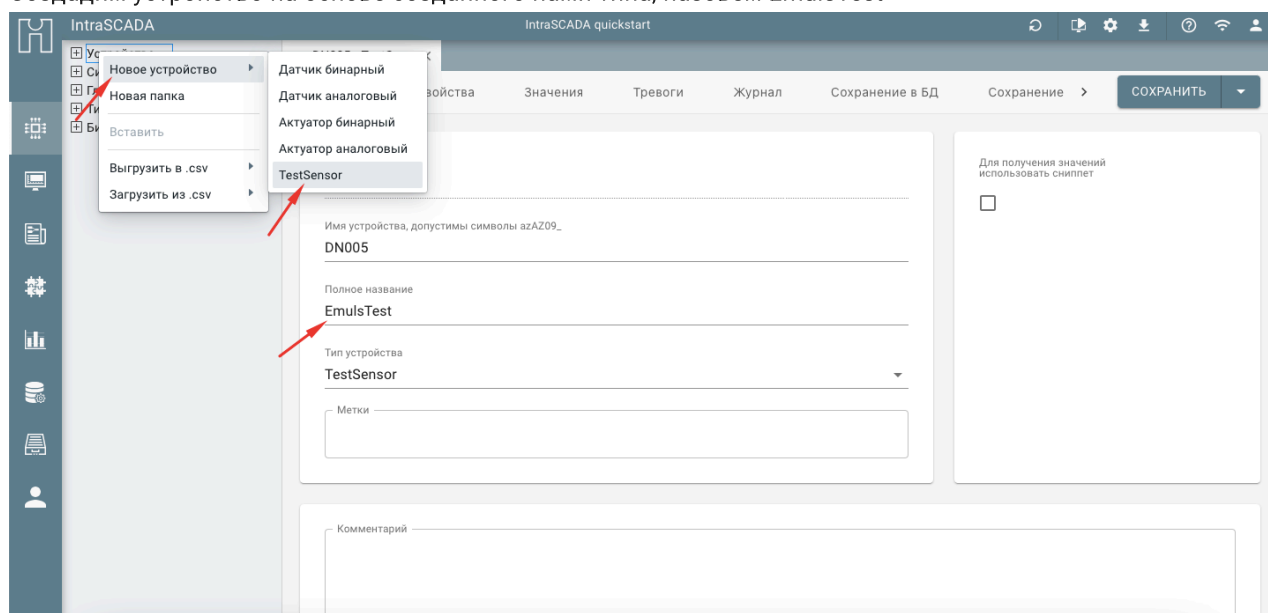
Назовем тип устройства, к примеру, *TestSensor*, жмём **Сохранить**



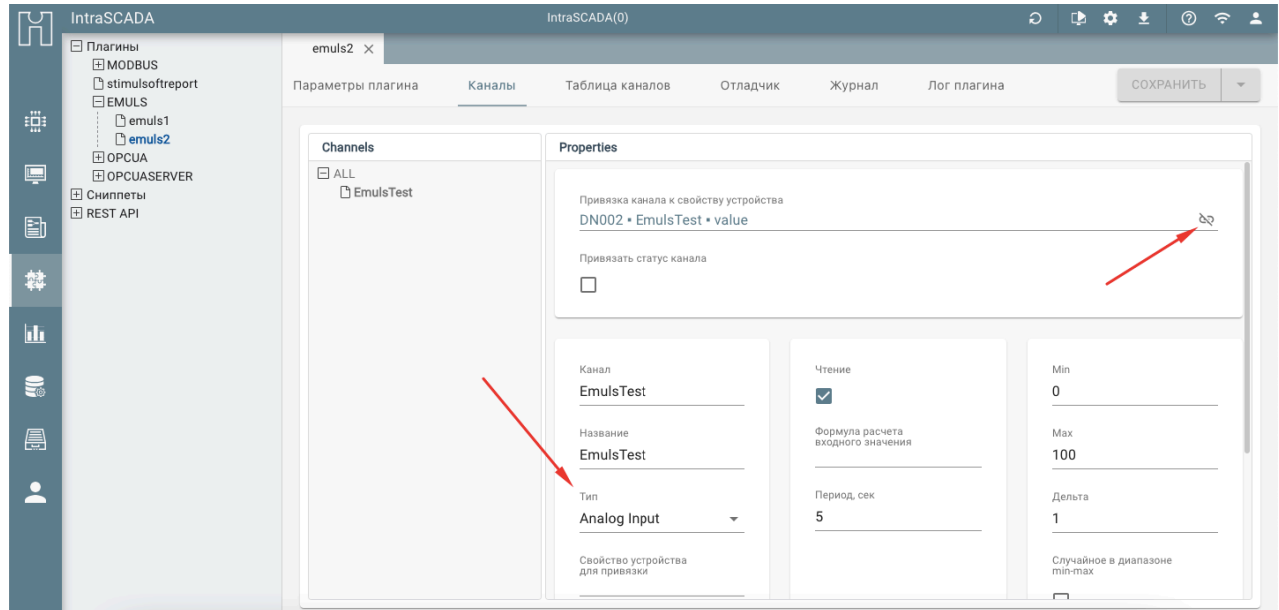
Добавим свойства `value(Data, Number)` и `state(Data, Bool)`, сохраняем



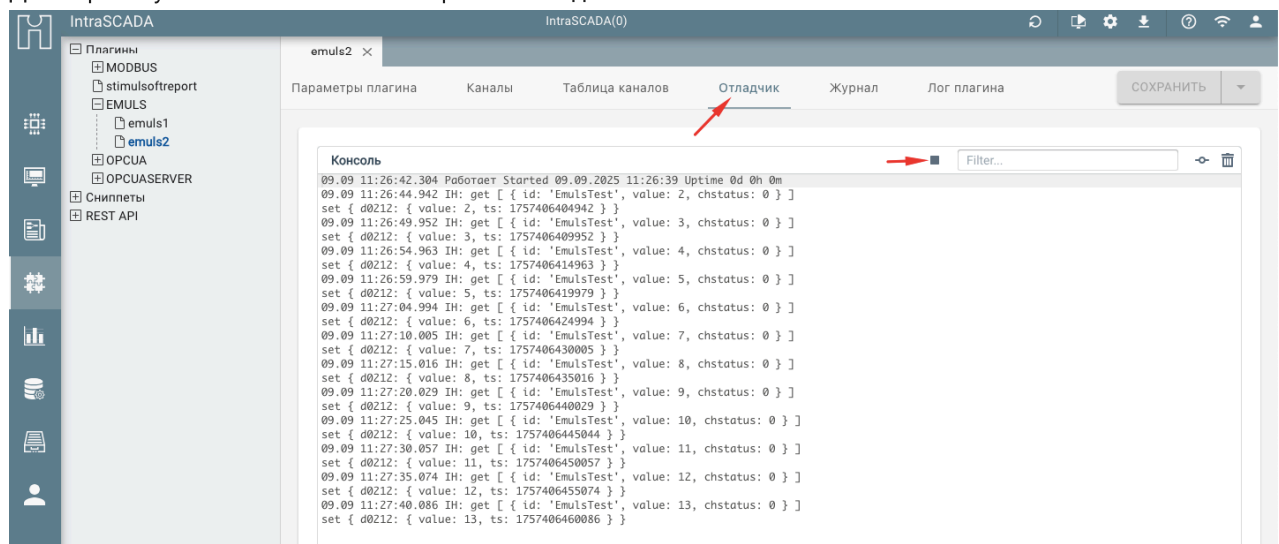
- Создадим устройство на основе созданного нами типа, назовём EmulsTest



- Привяжем свойство устройства **value** к созданному нами каналу эмулятора
- Настроим канал: Тип - Аналоговый; Период опроса - 5 секунд; min,max - 0,100; Дельта - 1.



- Далее работу плагина можно посмотреть в отладчике



Как мы видим, плагин успешно эмулирует работу устройства с заданными нами параметрами

## Визуализация

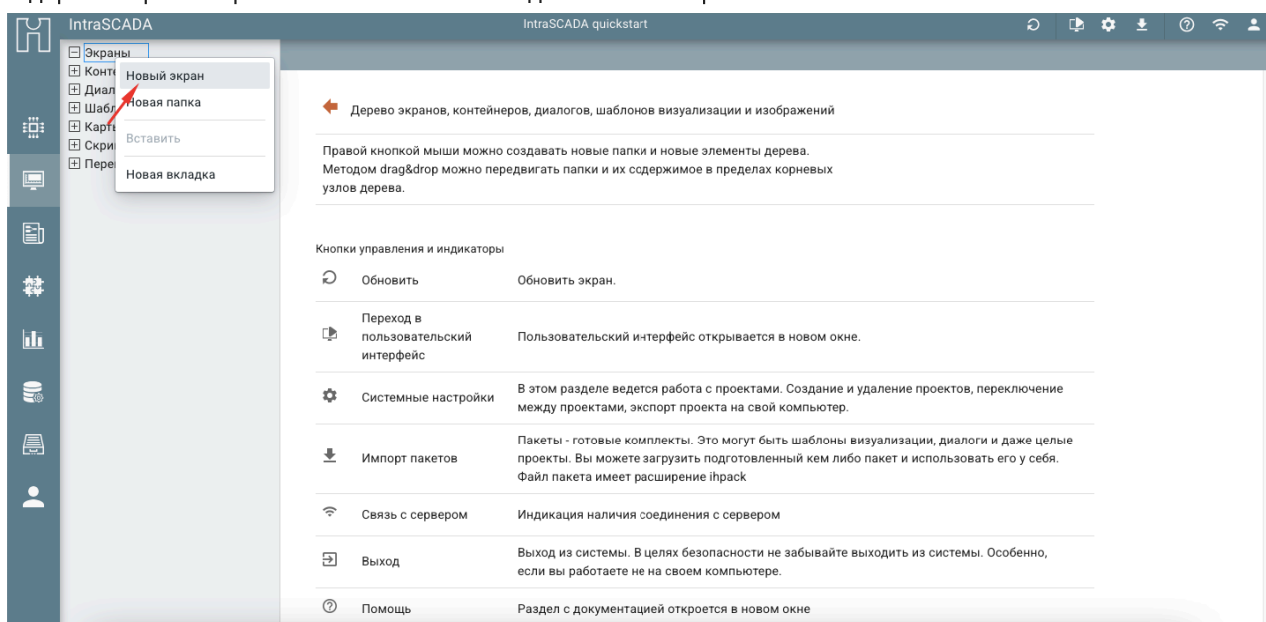
Визуализация в IntraSCADA позволяет создавать эффективные интерфейсы для мониторинга и управления. Основные компоненты: экраны (layouts), контейнеры, диалоги, шаблоны визуализации, элементы (basic, interactive, view, charts, widgets).

Привязки связывают элементы с данными устройств, скрипты добавляют логику, переменные клиента — для динамики.

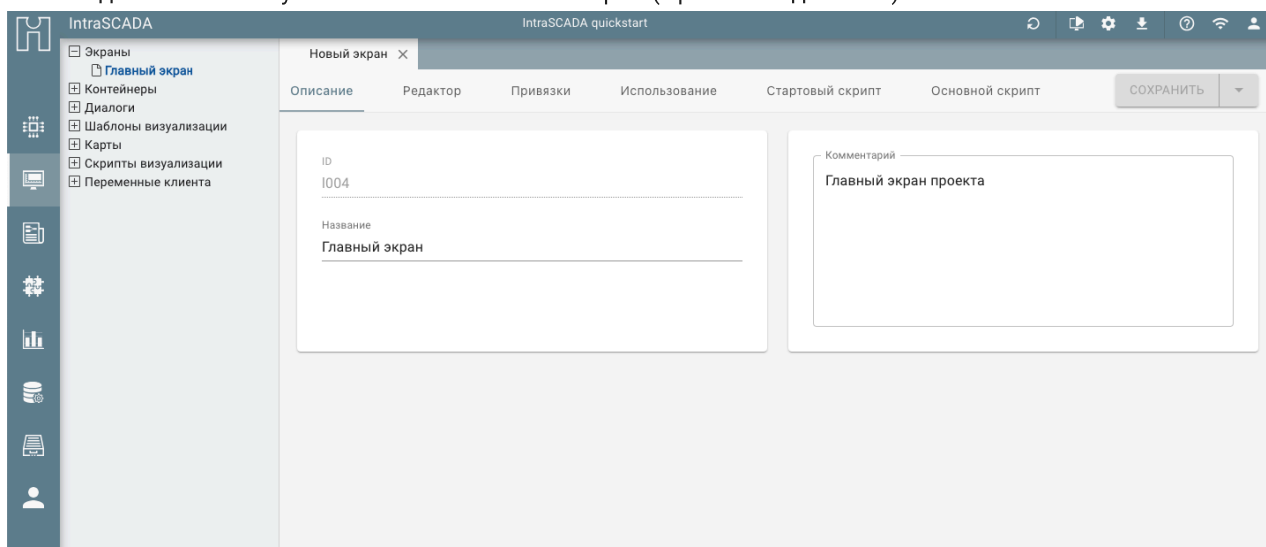
### Создание экранов

Экран — основной компонент визуализации. На нем размещаются контейнеры и элементы для компоновки (рекомендуется использовать контейнеры для масштабирования).

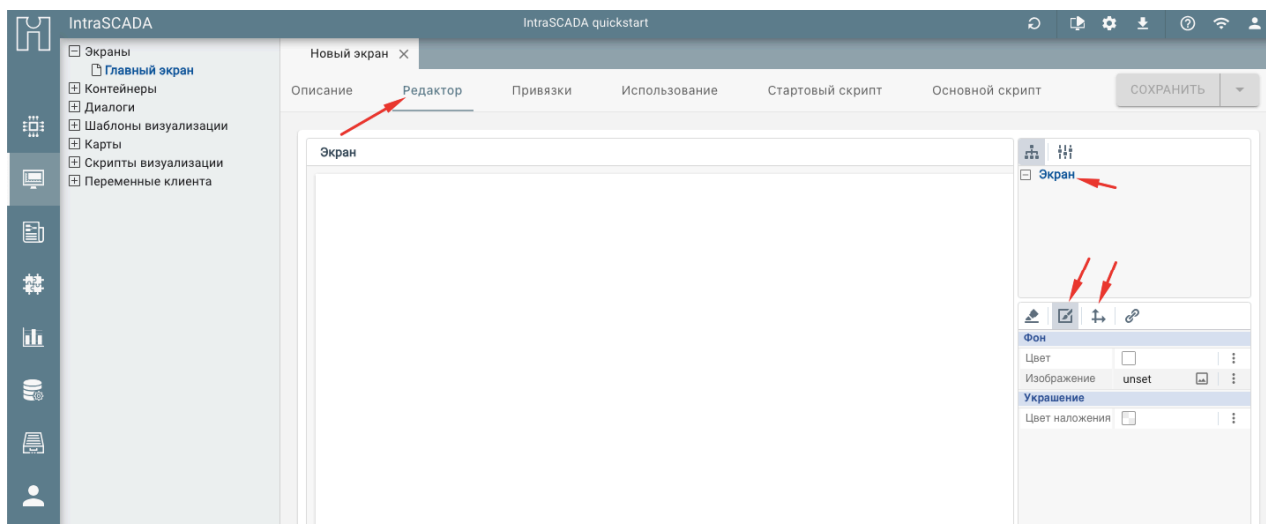
- В дереве экранов правой кнопкой мыши создаем новый экран



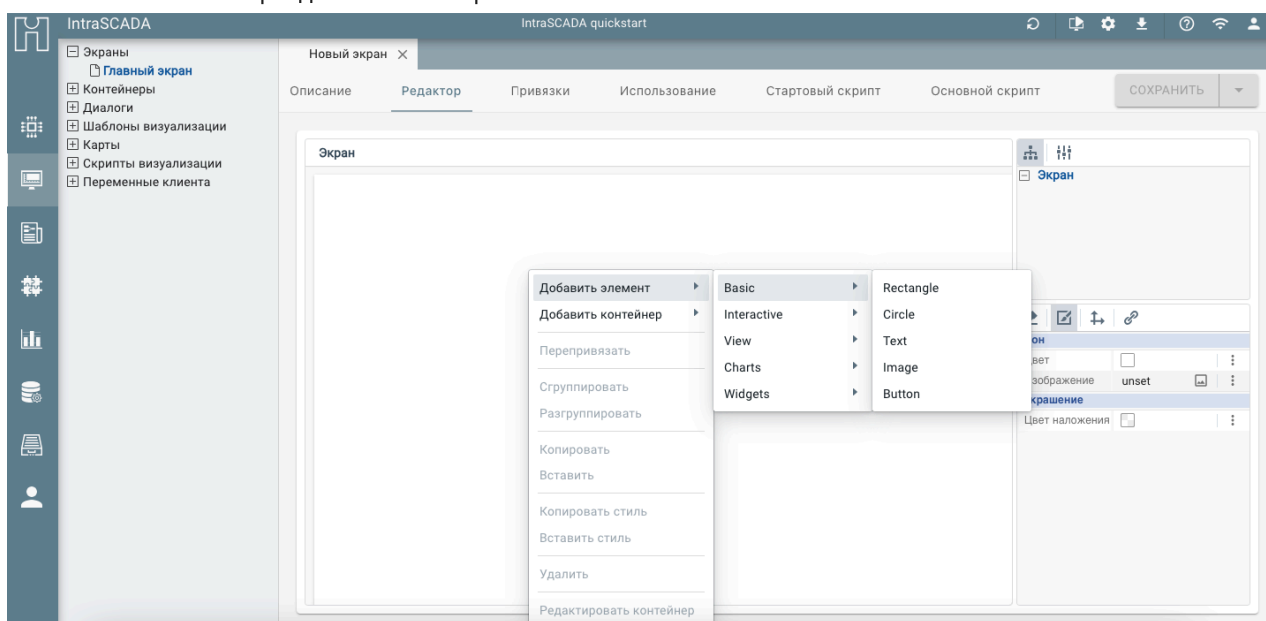
- На вкладке "Описание" указываем имя и комментарии (При необходимости)



- На вкладке "Редактор" настраиваем фон (цвет или изображение), размеры (ширина, высота для адаптации под устройства)



- Элементы и контейнеры добавляются правой кнопкой мыши



- Масштабирование: Элементы растягиваются под разрешение клиента. Для фиксированного вида используйте контейнеры.

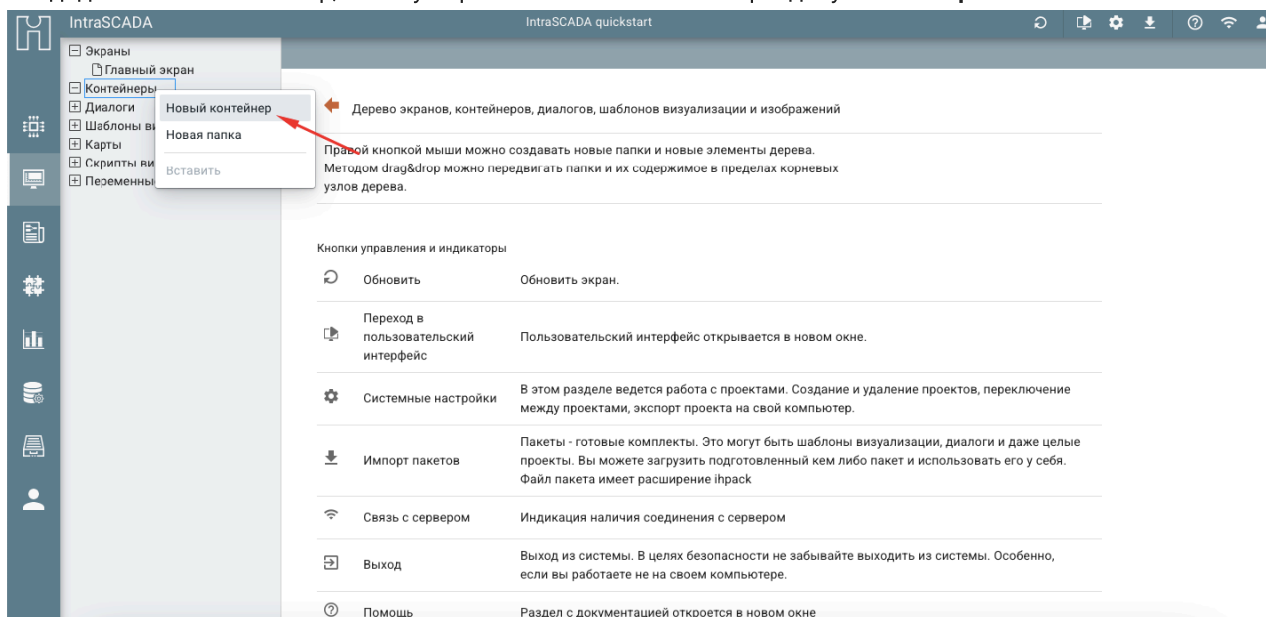
[Пример масштабирования](#)

## Контейнеры

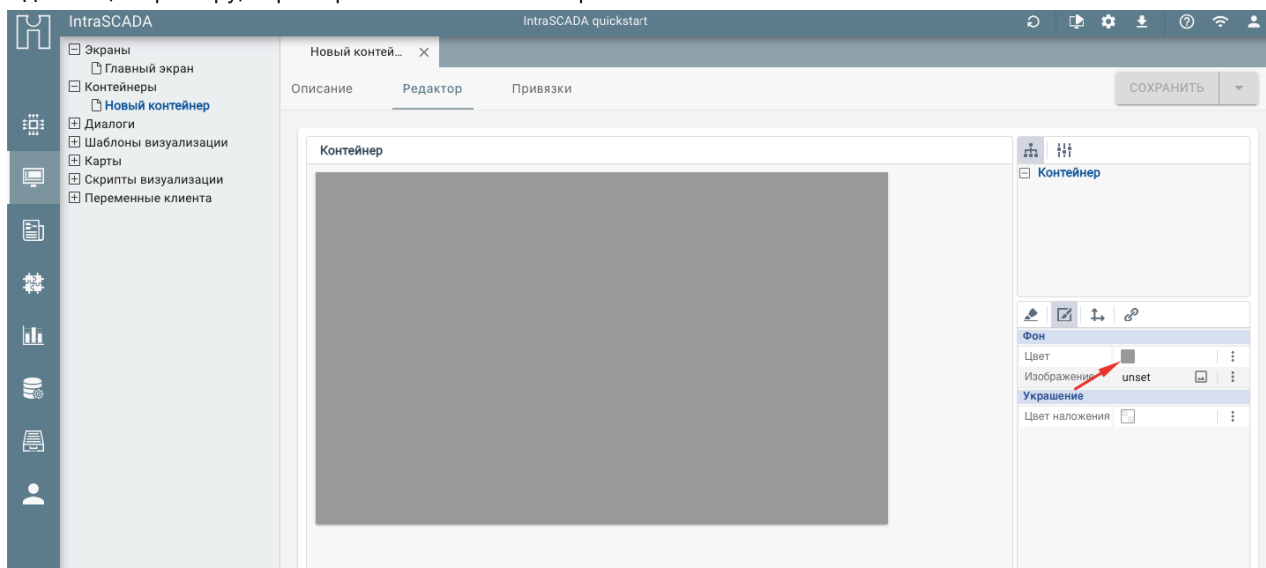
В контейнерах размещаются шаблоны визуализации и отдельные элементы. Также есть возможность добавить контейнер внутри уже готового контейнера.



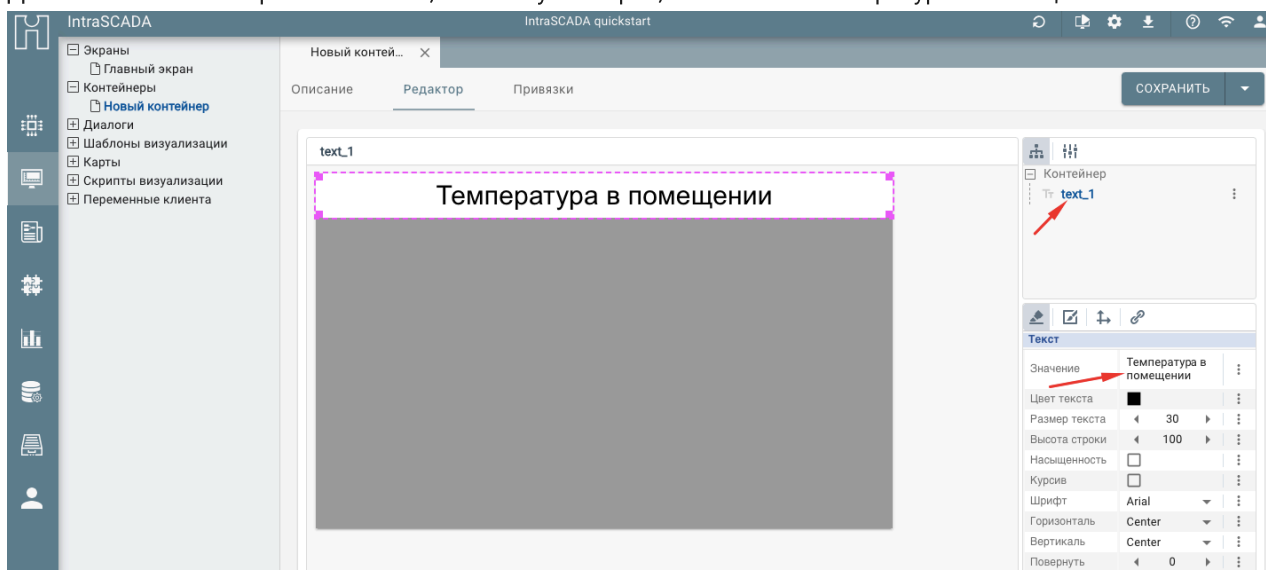
- Создадим новый контейнер, кликнув правой кнопкой мыши по разделу **Контейнеры**



- Сделаем, к примеру, серый фон нашего контейнера

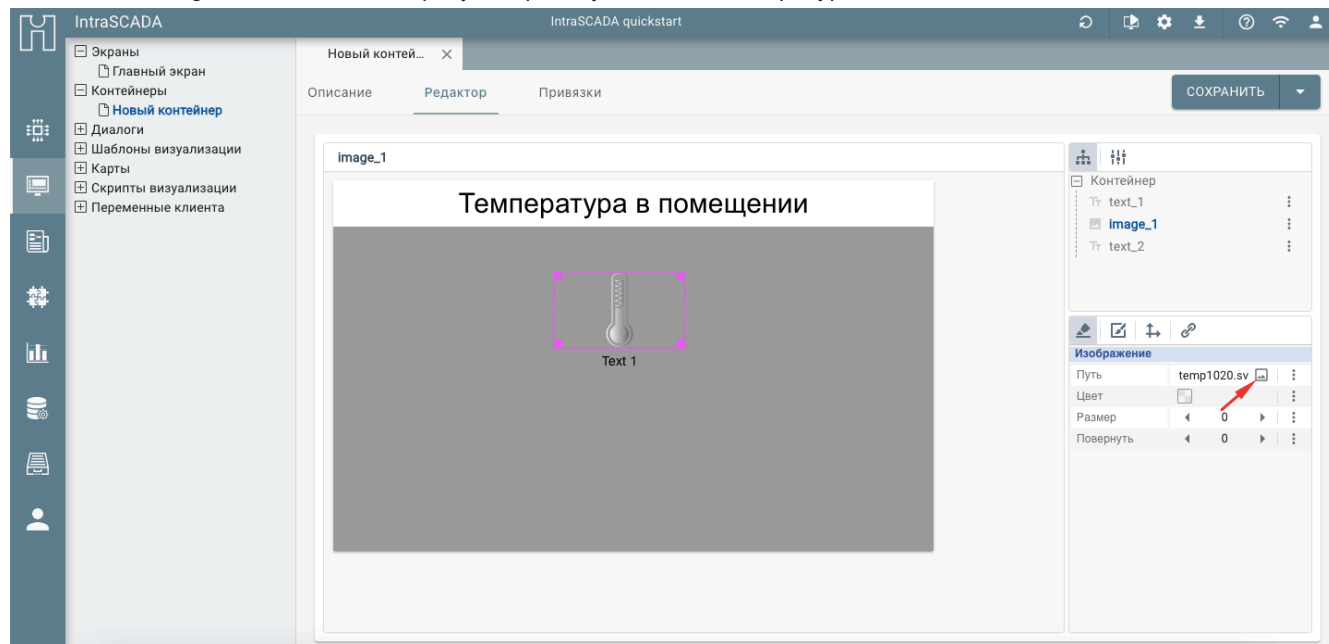


- Добавим на контейнер элемент Text, меняем у него фон, и напишем Температура в помещении

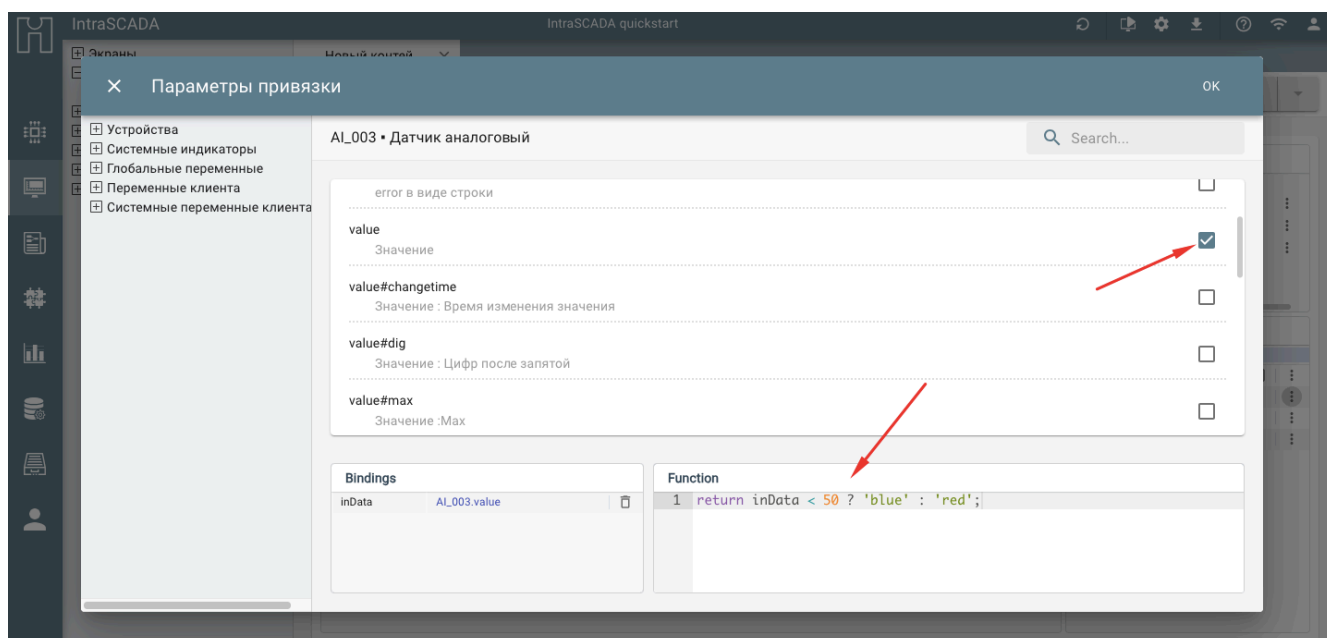
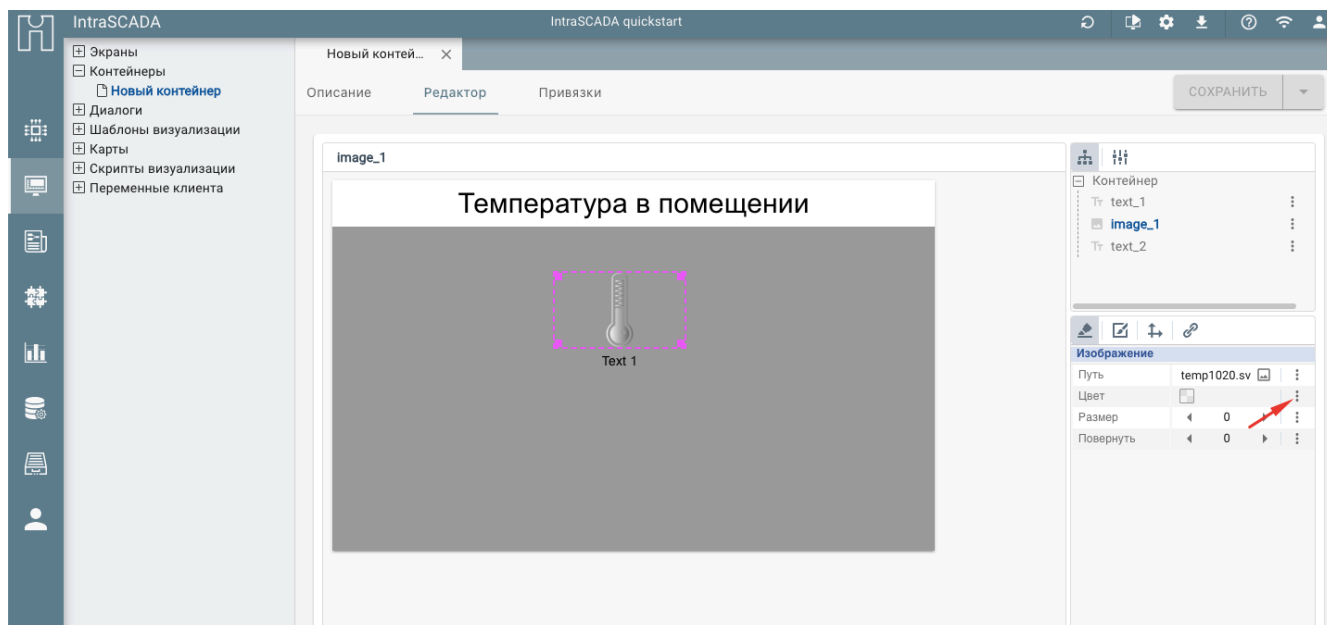


- Добавим еще 2 элемента: Image и Text

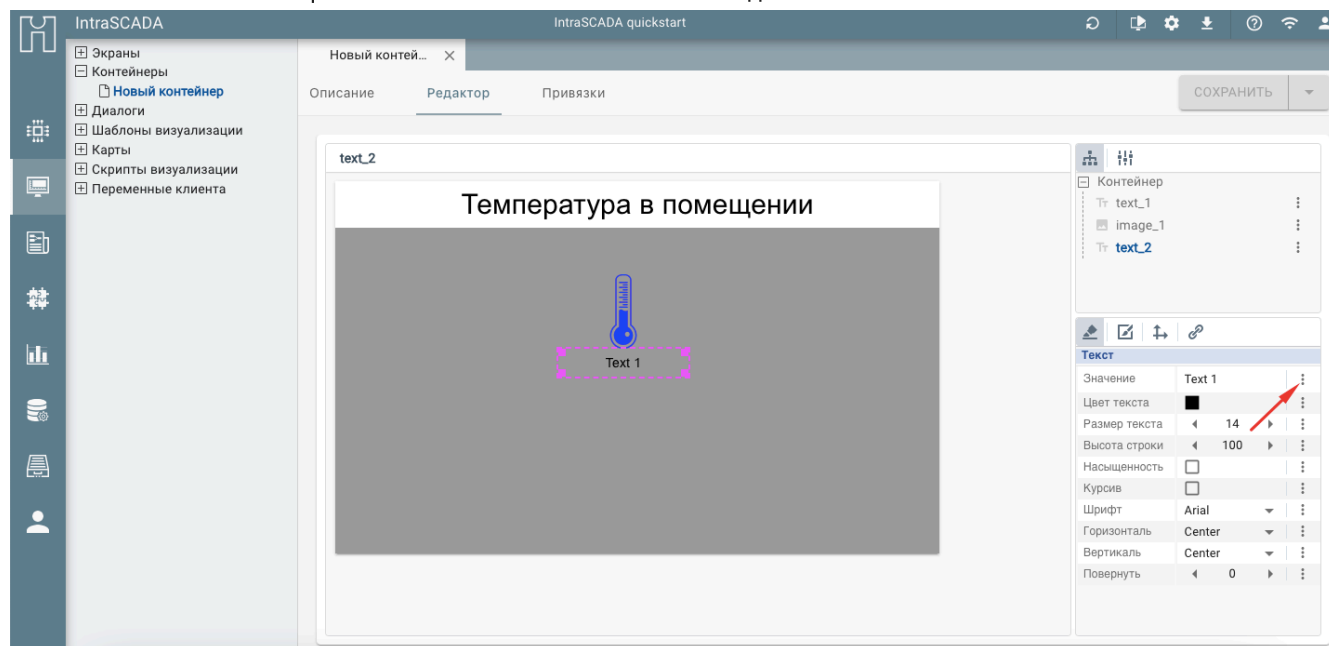
У элемента Image поставим стандартную картинку датчика температуры



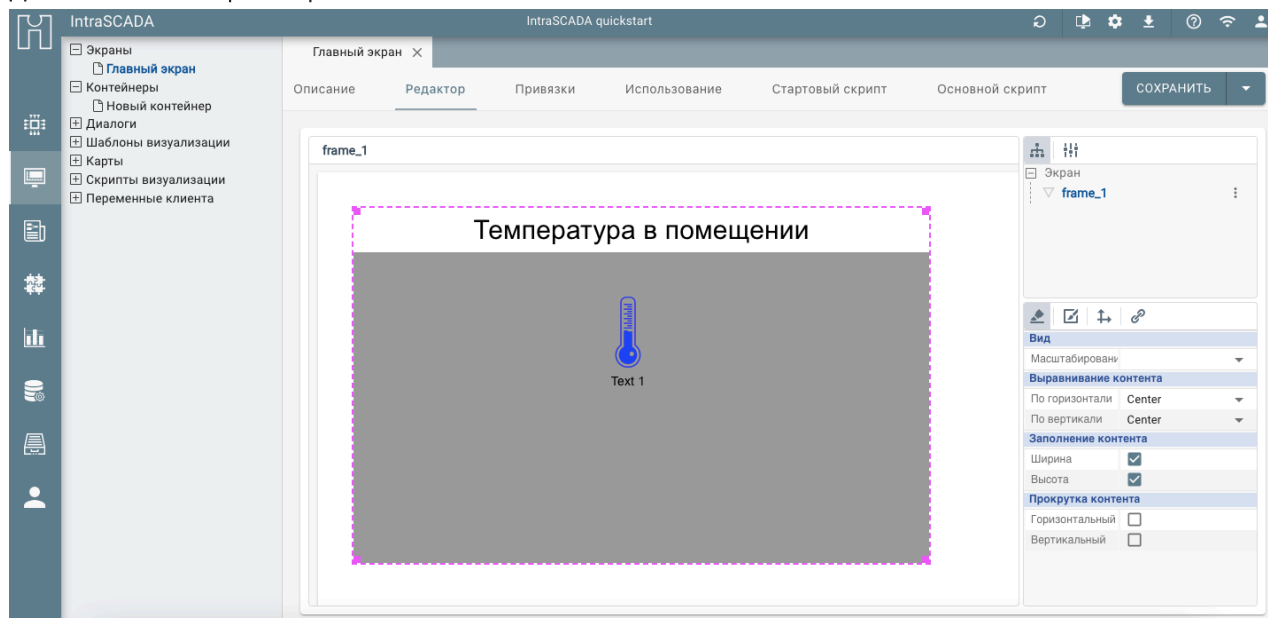
Цвет элемента привяжем к значению 'value' аналогового датчика и напишем функцию, чтобы при значении ниже 50 цвет был синий, выше - красный.



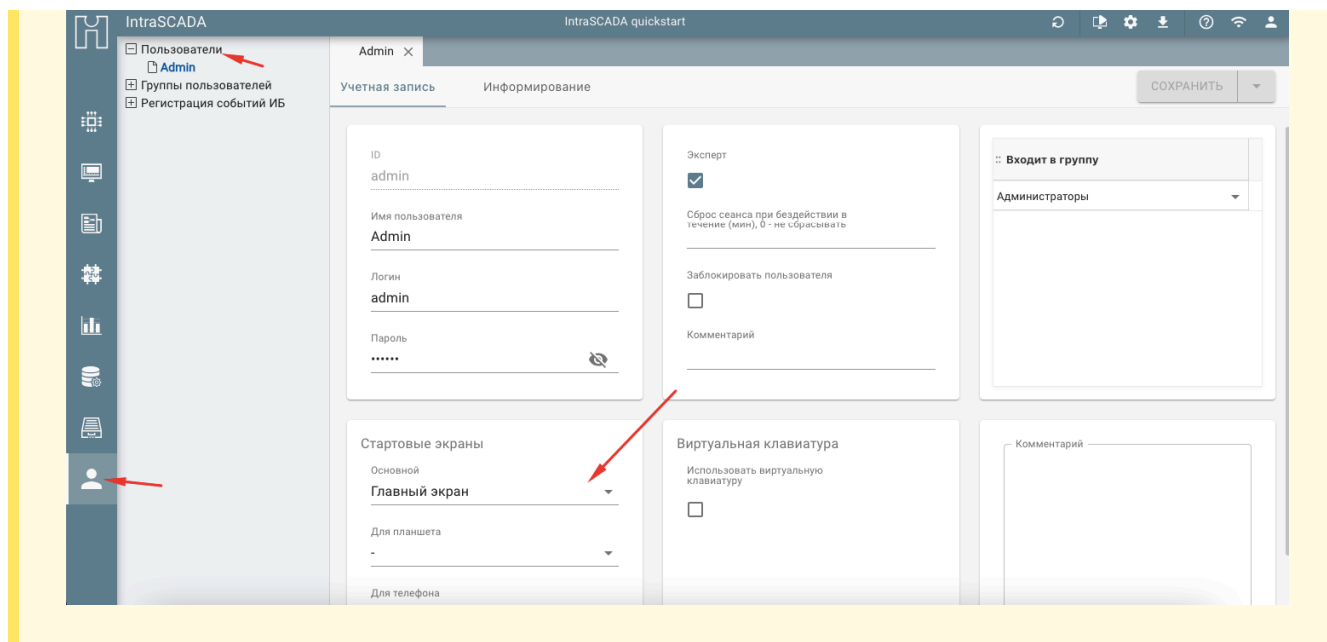
Значение элемента Text привяжем так же к 'value' аналогового датчика



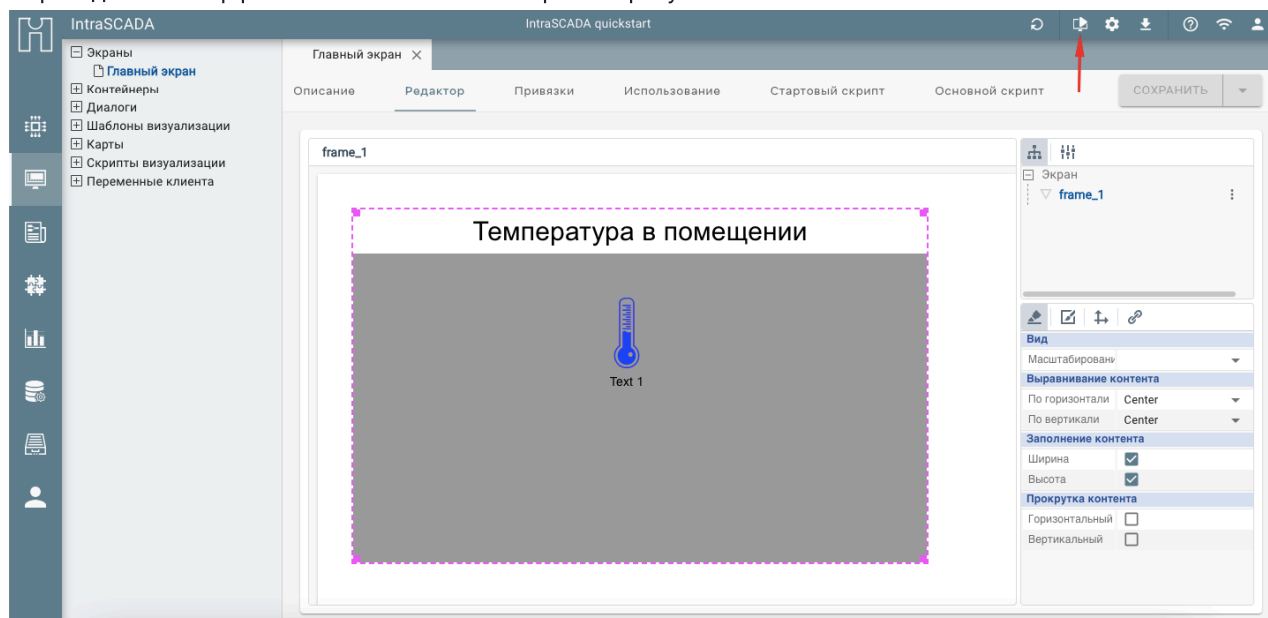
- Добавим контейнер на экран

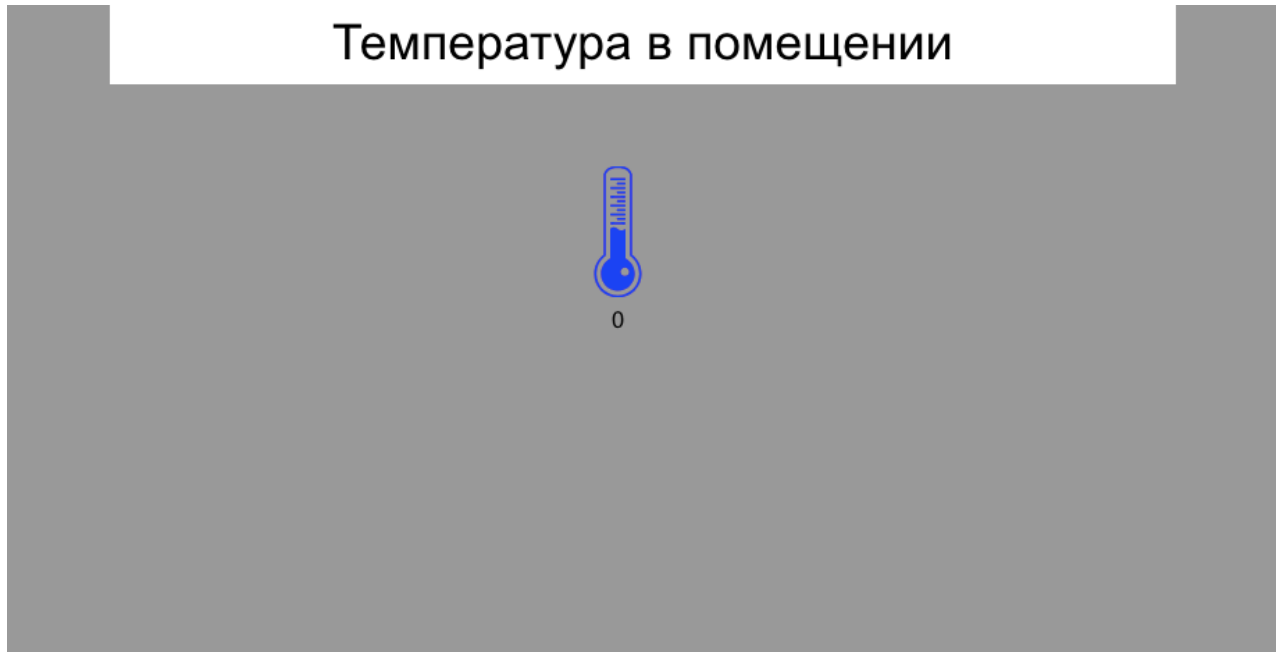


Важно! Чтобы экран отображался при переходе в Интерфейс пользователя, нужно его назначить в разделе Доступ -> Пользователи



- Переходим в Интерфейс пользователя и смотрим на результат





Готово! Теперь цвет датчика и значение будет меняться в зависимости от значения аналогового датчика

[Более подробно про контейнеры](#)

## Диалоги

Диалоги (диалоговые окна) вызываются командой Показать диалог .

Примеры диалогов: всплывающие окна с настройками параметров устройств, информационные сообщения в центре экрана.

[Более подробно про диалоги](#)

## Шаблоны визуализации

Шаблоны визуализации - это элементы интерфейса, размещаемые в контейнерах для визуализации однотипных устройств. Используются для создания большого количества устройств с одинаковыми свойствами. Можно создать один Шаблон визуализации, разместить необходимое количество этих шаблонов на мнемосхеме и привязать каждый отдельный шаблон к свойствам различных устройств.

[Более подробно про шаблоны визуализации](#)

## Элементы визуализации

Элементы - это те компоненты, из которых создаются шаблоны, диалоги, контейнеры и экраны. Текст, изображение, кнопка, слайдер, поле ввода, график - все это элементы.

У элементов есть общие свойства и индивидуальные, присущие конкретному элементу.

Любое свойство элемента можно связать с устройством.

[Более подробно про элементы](#)

## Скрипты визуализации

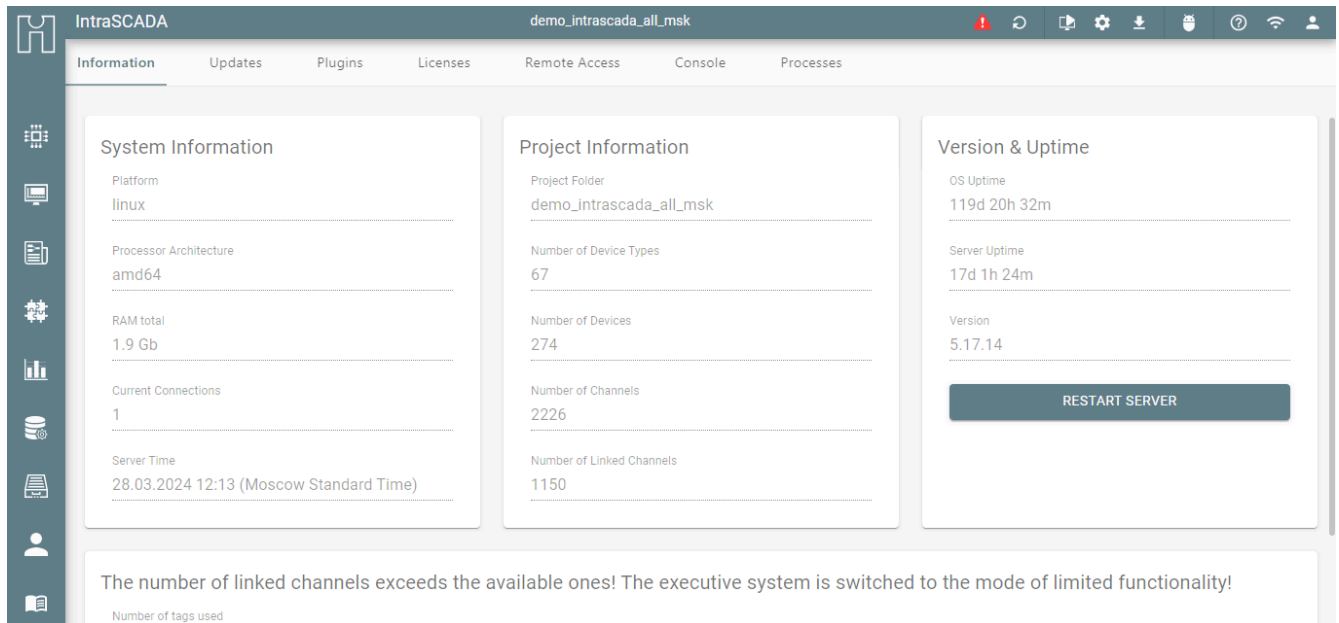
Скрипты визуализации позволяют добавить логику и интерактивность вашим интерфейсам.

[Более подробно про скрипты визуализации](#)

# Интерфейс разработчика

Интерфейс разработчика (Project Manager) — среда разработки, отладки и редактирования проектов автоматизации.

Для входа в интерфейс разработчика наберите строку запроса: <http://xxx.xxx.xxx.xxx:port/admin>



## Левое меню



Настройка структуры разрабатываемого проекта. Группировка устройств, локальных и глобальных переменных по функциональному признаку или по месту расположения.



Настройка визуализации с помощью экранов и контейнеров.



Создание сценариев поведения системы.



Работа с плагинами и сниппетами. Подключение физических устройств.



Аналитика. Настройка графиков и отчетов





Работа с базой данных



Ресурсы



Настройка доступа к системе.

## Строка состояния



Открыть вкладку браузера с пользовательским интерфейсом



Работа с проектами и общие настройки системы.



Импорт пакетов. Система позволяет импортировать и экспортировать наборы готовых элементов визуализации, типов устройств и прочих компонентов проекта в файлах с расширением ihpack



Переход на документацию.



Индикатор соединения с сервером.



Выход. Если при входе в систему вы установили галку "Запомнить меня", то после закрытия браузера и следующем входе система не будет запрашивать имя и пароль. Для сброса автоматического входа в систему нажмите кнопку Выход.

## О системе

Основная информация о системе, архитектуре, проекте и количестве тегов

### Информация о системе

- Платформа (linux, windows, darwin(macOS))
- Архитектура процессора (x64, arm, arm64)
- Количество оперативной памяти
- Количество клиентов, подключенных к серверу
- Время на сервере

### Информация о проекте

- Папка проекта
- Количество типов устройств в проекте
- Количество экземпляров устройств в проекте
- Количество каналов в плагинах проекта
- Количество тегов (активных каналов), привязанных к свойствам устройств (с учетом бесплатных каналов эмулятора)

### Версия и Аптайм

- ОС аптайм - время с момента запуска операционной системы
- Аптайм сервера - время с момента старта сервиса
- Текущая версия системы
- Кнопка перезагрузить сервис

[Интерфейс разработчика](#) / [Инфопанель](#) / [Обновление](#)

## Обновление

На этой вкладке отображается текущая версия системы.

По галочке **Отображать историю версий с сайта разработчика** можно получить **Release Notes** (информацию об изменениях, включая последнюю доступную версию).

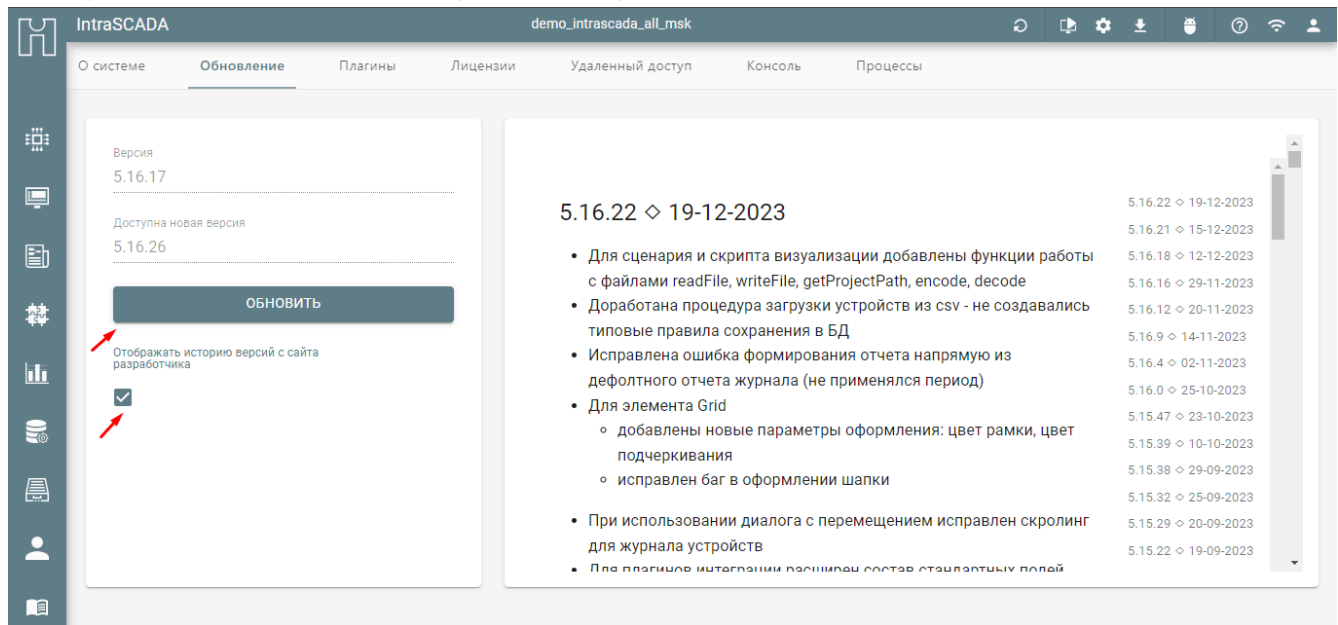
Остальное содержимое этой вкладки зависит от ОС.

### Linux и macOS

Для системы, установленной на Linux и macOS, доступна кнопка **Проверить обновления**, с помощью которой проверяется наличие новой версии.

Если новая версия доступна, появится кнопка **Обновить**, при нажатии на которую обновления будут установлены.

После установки обновлений сервис будет перезагружен.



## Windows

Для системы, установленной на Windows, интерактивное обновление не предусмотрено.

Для обновления системы установите [новую версию](#)

## Откат обновления

После обновления на новую версию можно откатиться назад, выполнив повторно [Установку](#) с указанием конкретной версии системы.

# Плагины

С помощью данного раздела вы можете установить плагины в систему либо обновить уже установленные плагины. Для обновления списка актуальных плагинов необходимо нажать на кнопку **Проверить обновления**, после этого в таблице появится список плагинов с актуальными версиями на текущий момент.

The screenshot displays the 'Плагины' (Plugins) section of the IntraSCADA interface. The top navigation bar includes 'О системе', 'Обновление', 'Плагины', 'Лицензии', 'Удаленный доступ', 'Консоль', and 'Процессы'. The 'Плагины' tab is active, showing a table of plugins. The table has columns for 'Плагин', 'Описание', 'Версия', 'Последняя версия', and 'Действия'. The table lists several plugins, including 'applehome', 'applehomekit', 'bacnetip', 'cctv', 'codesys2v', 'email', 'emuls', 'ethernetip', 'http', 'httpcache', and 'iec60870p'. The 'Действия' column contains buttons for 'Удалить' (Delete) and 'Установить' (Install). A red arrow points to the 'ПРОВЕРИТЬ ОБНОВЛЕНИЯ' (Check for updates) button located on the right side of the table.

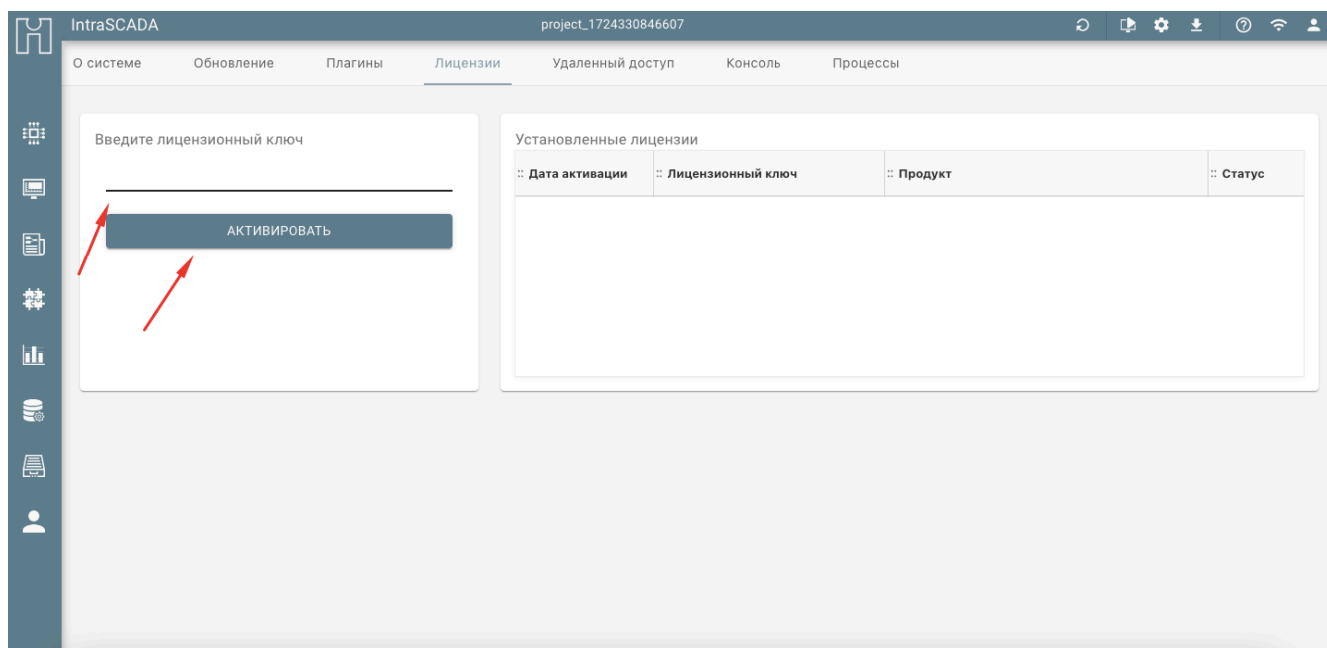
Плагин	Описание	Версия	Последняя версия	Действия
applehome	Apple Home Plugin	5.2.1	5.2.1	Установлена последняя вер... Удалить
applehomekit	Apple HomeKit Plugin		5.0.8	Доступно для установки Установить
bacnetip	Bacnet IP plugin		5.5.2	Доступно для установки Установить
cctv	CCTV plugin		5.7.3	Доступно для установки Установить
codesys2v	Codesys plugin		5.5.8	Доступно для установки Установить
email	Email plugin	5.5.3	5.5.3	Установлена последняя вер... Удалить
emuls	Emulator	5.5.8	5.5.8	Установлена последняя вер... Удалить
ethernetip	Ethernet/IP plugin		5.0.16	Доступно для установки Установить
http	HTTP plugin		5.5.6	Доступно для установки Установить
httpcache	HTTP cache plugin		5.0.0	Доступно для установки Установить
iec60870p	Client for 60870-5-104		5.5.7	Доступно для установки Установить

Установлено плагинов 4  
Доступно обновлений 1  
Доступно для установки 26  
ПРОВЕРИТЬ ОБНОВЛЕНИЯ

# Лицензии

В данном разделе представлены все лицензии системы, а так же срок их действия, если они не бессрочные.

[Подробная информация по лицензированию](#)



# Удаленный доступ

Механизм удаленного доступа через сервис p2p позволяет подключиться к пользовательскому интерфейсу и к интерфейсу разработки без необходимости использовать белый IP адрес или VPN. С помощью данного механизма так же можно прокинуть порты в локальную систему для доступа, например, к контроллерам или другим сервисам.

Плагин P2P

Работает

---

Ключ P2P

---

Состояние связи

connected

---

Адрес сервиса для удаленного входа: <https://p2p.ih-systems.com>

Настройка портов для p2p подключений

Использовать выделенный диапазон портов

☒

Номер первого порта диапазона

9001

---

Номер последнего порта диапазона

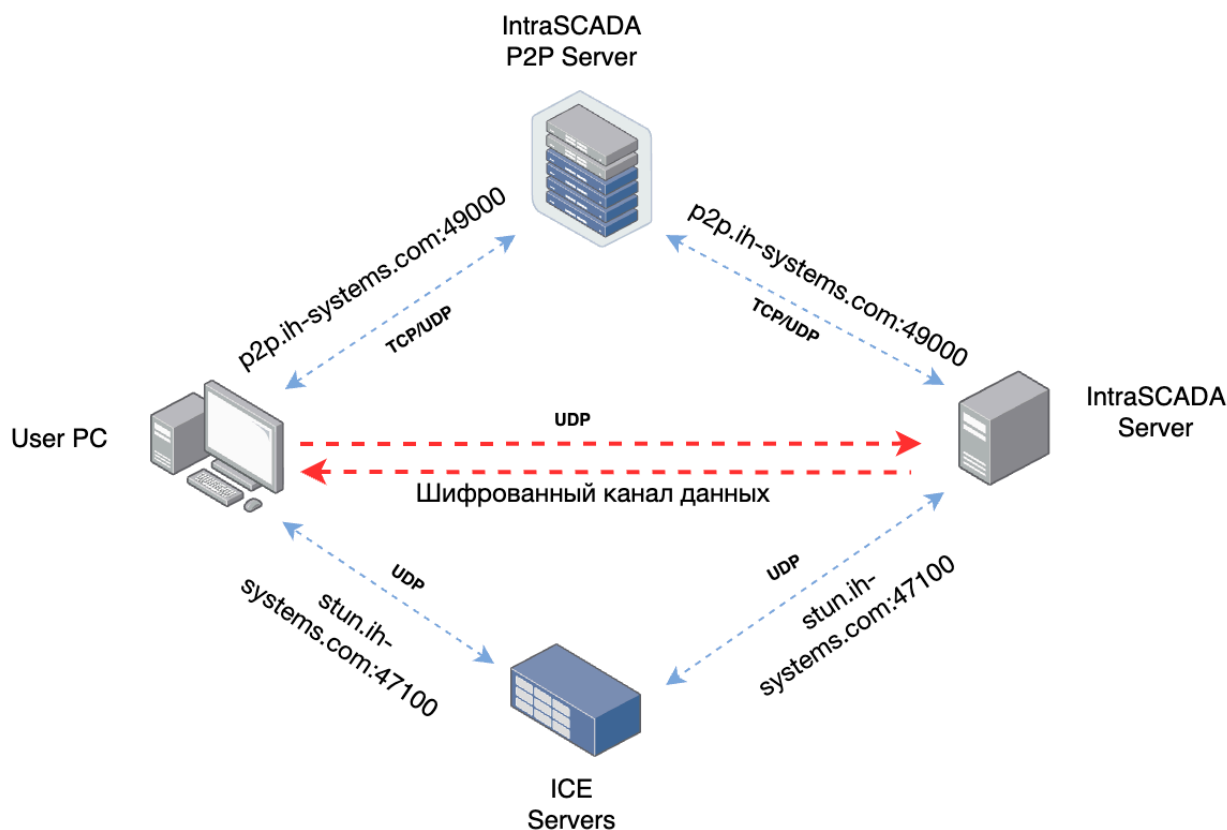
9010

---

СОХРАНИТЬ НАСТРОЙКИ

## Схема работы P2P сервиса

Для организации подключения через p2p, необходимо предоставить доступ в интернет серверу IntraSCADA. Плагин p2p, обнаружив подключение к интернету, регистрируется на сигнальном сервере ([p2p.ih-systems.com](https://p2p.ih-systems.com)) для обмена кандидатами с будущими клиентами. Когда вы захотите подключиться к удаленному серверу, вы вводите уникальный ключ из 9 цифр. При инициализации соединения, с помощью ICE серверов определяются кандидаты и осуществляется обмен между сервером IntraSCADA и клиентом с помощью сигнального сервера. После обмена кандидатами весь трафик идет напрямую по зашифрованному каналу данных. Подключение к вспомогательным серверам (ICE, P2P) завершается.



Для того, чтобы данный сервис заработал в сетях с межсетевым экраном необходимо открыть доступ к следующим серверам:

1. `p2p.ih-systems.com(89.23.116.120):49000` (TCP/UDP)
2. `stun.ih-systems.com(89.23.116.120):47100` (UDP)
3. `traffic` (UDP). Настраивается в системе. По умолчанию порты 0-65535.

Необходимо разрешить исходящие UDP пакеты со стороны сервера IntraScada во вне, а так же разрешить обратные входящие пакеты по этому порту в сторону IntraScada. Для этого необходимо настроить `masquerade srcnat` для каждого правила.

Пример правил настройки файрвола на роутере Микротик:

- 192.168.0.100 - IP адрес сервера IntraSCADA на предприятии
- 9001-9010 - ограниченный диапазон портов, указывается в настройка P2P подключения.

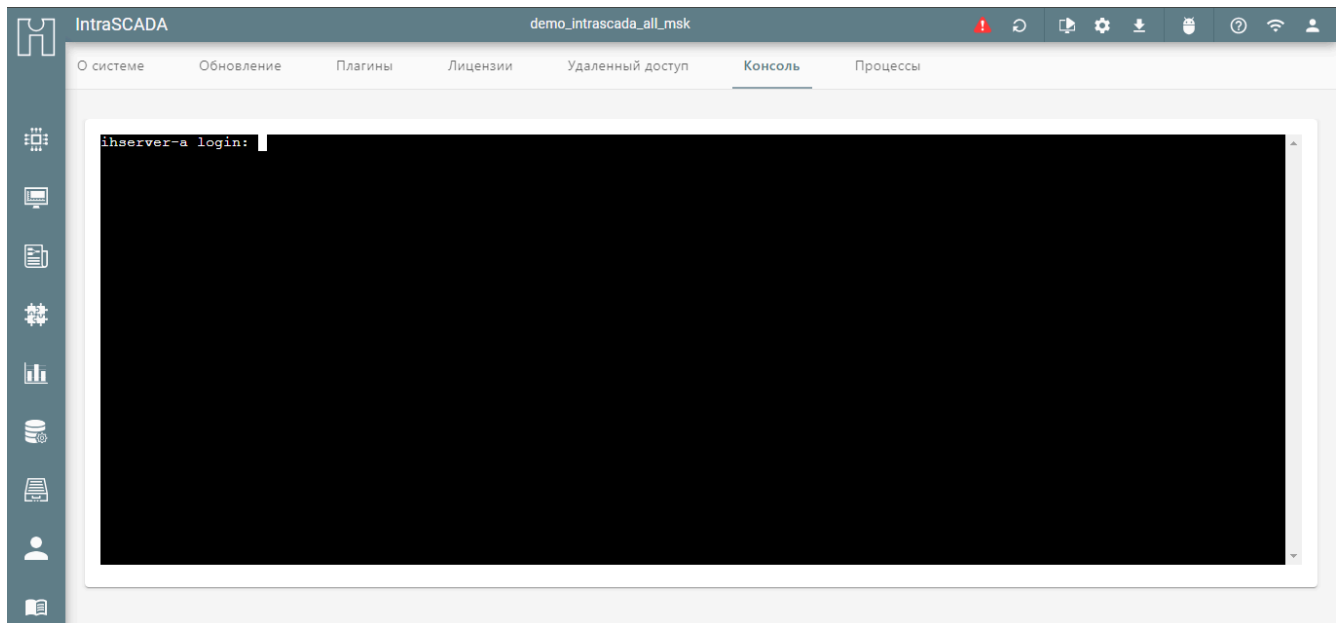
```
add action=masquerade chain=srcnat \  
    dst-address=89.23.116.120 dst-port=49000 \  
    protocol=udp src-address=192.168.0.100  
  
add action=masquerade chain=srcnat \  
    dst-address=89.23.116.120 dst-port=49000 \  
    protocol=tcp src-address=192.168.0.100  
  
add action=masquerade chain=srcnat \  
    dst-address=89.23.116.120 dst-port=47100 \  
    protocol=udp src-address=192.168.0.100  
  
add action=masquerade chain=srcnat \  
    port=9001-9010 protocol=udp \  
    src-address=192.168.0.100
```

При ограничении диапазона портов вы ограничиваете количество одновременных подключений, один порт одно соединение



# Консоль

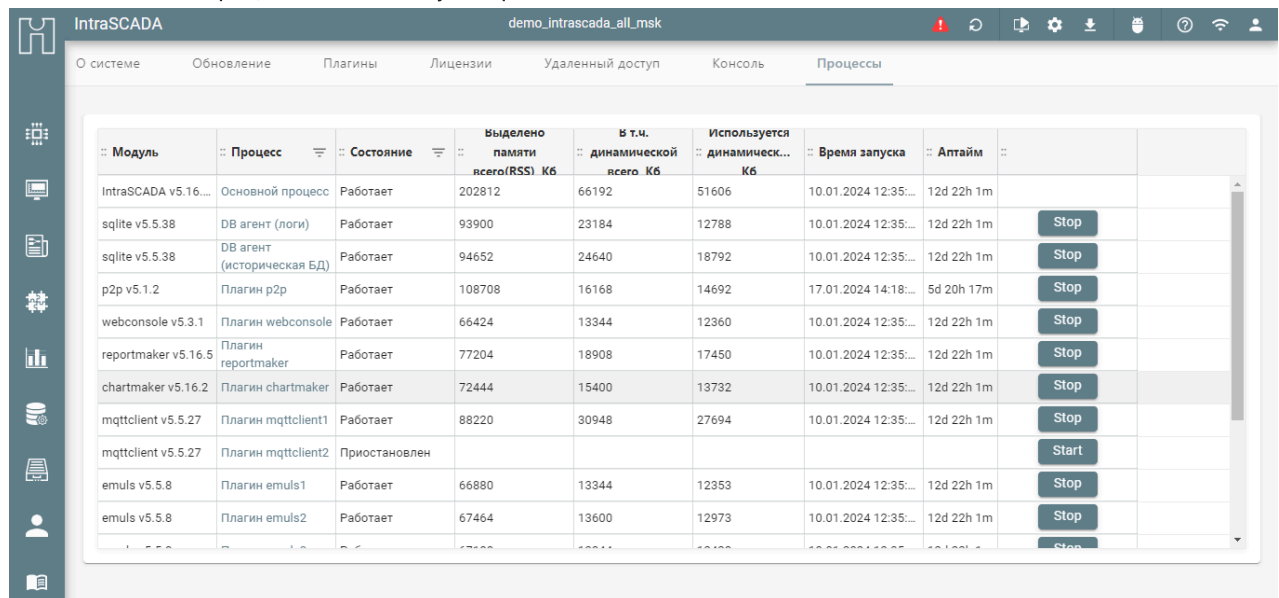
В данном разделе вы можете подключиться к консоли операционной системы, на которой установлена IntraSCADA. Доступ осуществляется по логину и паролю пользователя операционной системы. Если в системе только root пользователь, то вам необходимо создать пользователя для доступа через консоль.



# Процессы

Система работает на микроядерной архитектуре для равномерного распределения нагрузки на процессор и для обеспечения высокого уровня отказоустойчивости. В данной вкладке отображаются все процессы, запущенные сервисом, потребление памяти, время запуска и работы. С данной панели вы можете останавливать или перезапускать процессы. Нажав на название процесса, вы перейдете в его лог. Основные процессы, которые запускаются после установки :

- Основной процесс - процесс ядра системы
- DB агент (логи) - процесс для записи логов в базу данных
- DB агент (историческая БД) - процесс для записи данных временных рядов в базу данных
- Плагин r2p - процесс для обеспечения удаленного доступа
- Плагин reportmaker - процесс для работы с отчетами
- Плагин webconsole - процесс для доступа к консоли сервера
- Плагин chartmaker - процесс для графиков
- Плагины emuls - процесс плагина эмулятора



Модуль	Процесс	Состояние	Выделено памяти всего(RSS) Кб	В т.ч. динамической всего Кб	Используется динамическ... Кб	Время запуска	Аптайм	
IntraSCADA v5.16...	Основной процесс	Работает	202812	66192	51606	10.01.2024 12:35:...	12d 22h 1m	
sqlite v5.5.38	DB агент (логи)	Работает	93900	23184	12788	10.01.2024 12:35:...	12d 22h 1m	Stop
sqlite v5.5.38	DB агент (историческая БД)	Работает	94652	24640	18792	10.01.2024 12:35:...	12d 22h 1m	Stop
p2p v5.1.2	Плагин r2p	Работает	108708	16168	14692	17.01.2024 14:18:...	5d 20h 17m	Stop
webconsole v5.3.1	Плагин webconsole	Работает	66424	13344	12360	10.01.2024 12:35:...	12d 22h 1m	Stop
reportmaker v5.16.5	Плагин reportmaker	Работает	77204	18908	17450	10.01.2024 12:35:...	12d 22h 1m	Stop
chartmaker v5.16.2	Плагин chartmaker	Работает	72444	15400	13732	10.01.2024 12:35:...	12d 22h 1m	Stop
mqttclient v5.5.27	Плагин mqttclient1	Работает	88220	30948	27694	10.01.2024 12:35:...	12d 22h 1m	Stop
mqttclient v5.5.27	Плагин mqttclient2	Приостановлен						Start
emuls v5.5.8	Плагин emuls1	Работает	66880	13344	12353	10.01.2024 12:35:...	12d 22h 1m	Stop
emuls v5.5.8	Плагин emuls2	Работает	67464	13600	12973	10.01.2024 12:35:...	12d 22h 1m	Stop

## Устройства

Любой объект автоматизации содержит набор устройств. Это различные датчики и исполнительные механизмы (актуаторы).

Устройства можно сгруппировать по месту нахождения или по функционалу.

## Системные индикаторы

Системные индикаторы - это индикаторы плагинов. Эти индикаторы появляются в дереве автоматически при установке плагинов. Соответственно они пропадают в дереве при удалении плагина.

## Глобальные переменные

Глобальные переменные можно создавать для различных целей. Например, можно сделать переменную "Режим" и использовать ее для задания режима работы системы.

## Типы устройств

Это шаблоны для создания экземпляров устройств.

Система поставляется с некоторым количеством стандартных типов.

- Любой тип можно отредактировать или создать новые типы.
- Любой тип можно выгрузить и загрузить в другой проект.

## Устройства

Устройства — ключевой компонент системы.

Вокруг устройств крутятся все механизмы системы. Любые свойства устройств можно:

Отобразить в визуализации на экранах и мнемосхемах

Подключить с помощью плагинов к физическим каналам

Обработать сценариями

Записать в базу данных и получить аналитическую информацию



Устройства системы - это сущности, позволяющие моделировать объекты реального мира для мониторинга и управления.

Разработчик проекта может создавать устройства с нужным ему набором свойств и степенью детализации.

Например, беспроводной датчик температуры/влажности/давления можно смоделировать как:

- четыре отдельных датчика: температуры, влажности, давления, заряд батареи
- одно устройство, включающее все эти свойства
- часть устройства "Гостиничный номер", в котором, кроме датчика, есть и другие элементы

Конечно, было бы утомительно создавать каждый экземпляр устройства с нуля.

Экземпляры устройств создаются на базе [Типа устройства](#). В дальнейшем тип устройства можно изменить

Сами типы устройств не являются жестко заданными.

Система поставляется с набором стандартных типов, при этом любой тип можно добавить или изменить.

Тип определяет набор свойств и логику устройства.

## Свойства

В системе нет жесткой связи физического сигнала и элемента визуализации.

Свойство устройства - это элемент системы, к которому, с одной стороны, может быть привязан физический канал (и образуется активный тег). С другой стороны, свойство привязывается к элементам визуализации.

Свойства определяются типом. В зависимости от функционала это могут быть:

### 1. Данные (Data)

Это обычно данные, которые поступают с плагина (железа) - текущие показания датчиков, счетчиков, состояние устройств. Каждое поступление данных фиксируется, но событие изменения генерируется только при изменении поступившего значения.

### 2. Событие (Event)

Это поступающие данные, для которых важен сам факт поступления. Событие изменения генерируется при каждом поступлении данных.

### 3. Параметр (Parameter)

Такие данные как уставки, настройки регуляторов, шкалы масштабирования для датчиков, верхний и нижний пределы срабатывания аварий и т. д.

#### 4. Вычисляемое значение (Calculate)

На основе поступающих значений можно сразу формировать расчетные значения

#### 5. Команда (Command)

## Каналы

Связь с физическими устройствами выполняется через [каналы](#). С каналами работают [Плагин](#)ы.

Любое свойство, кроме **Вычисляемого значения**, можно привязать к каналу.

Чтобы начать получать данные с физических устройств, нужно установить соответствующий плагин и настроить каналы.

К одному свойству можно привязать только один канал.

## Визуализация

К одному свойству можно привязать множество элементов визуализации на одном или разных экранах.

При изменении значения привязанного свойства все привязанные элементы будут обновляться.

Существует несколько способов привязки, от точечной привязки конкретного свойства устройства к отдельному атрибуту до использования шаблонов и привязок контекста.

### Привязка к элементам

Свойство устройства можно привязать напрямую к атрибутам [элемента визуализации](#).

При привязке можно задать формулу или даже написать небольшой скрипт, если нужно преобразовать значение, полученное с сервера.

Причем можно сделать привязку как к основному атрибуту визуального элемента (например, к значению элемента **Text/ Input/ Slider/ Checkbox**), так и к атрибуту оформления (Положению элемента/Размеру/Цвету текста/фона/рамки и т. д).

Например, можно выводить показание датчика в текстовом поле плюс на этом же или других элементах менять цвет текста и/или рамки, размер элемента или даже выполнять анимацию, используя функцию преобразования в зависимости от значения.

### Привязка к шаблонам

Второй вариант привязки - использование [шаблонов](#).

Здесь привязка свойства устройства выполняется к абстрактному свойству шаблона, так называемой переменной состояния.

Однажды созданный шаблон можно использовать для создания большого количества устройств с одинаковыми свойствами.

Все формулы преобразования и визуальные эффекты уже учтены при создании шаблона.

### Использование переменных клиента

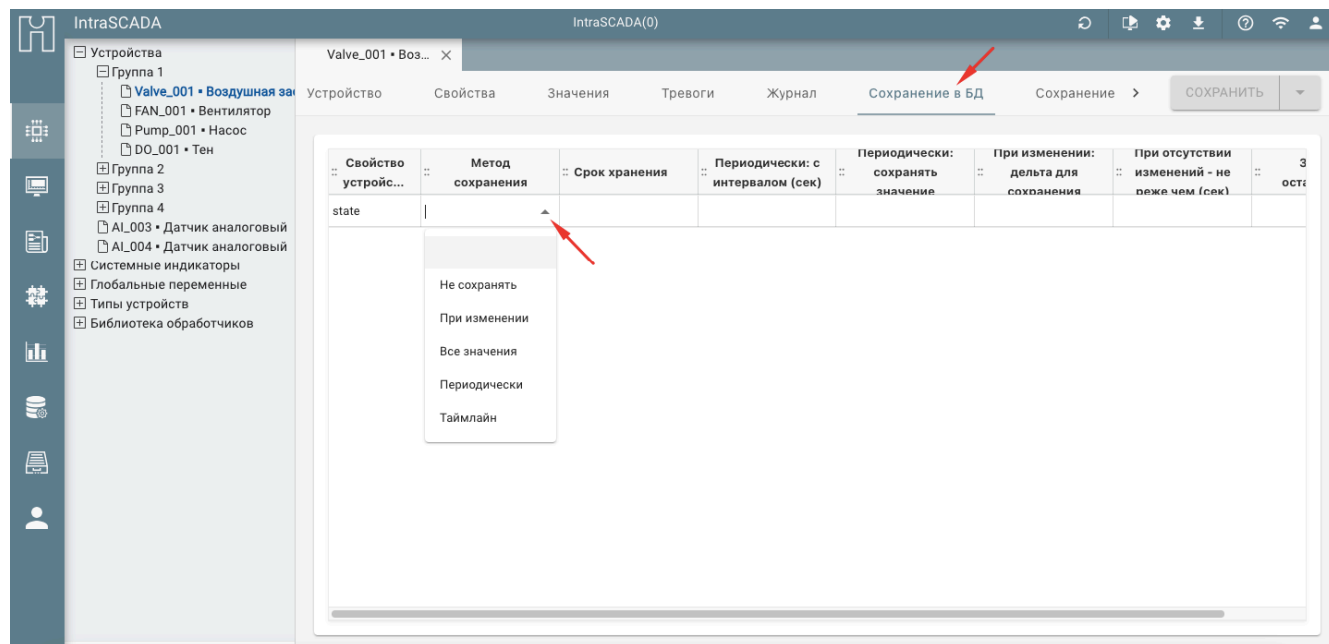
В первых двух вариантах в привязке участвует конкретное устройство.

Если однотипных устройств много и/или они динамически добавляются, можно применить [Динамическую привязку устройств](#)

Этот способ позволяет не привязывать жестко элемент визуализации или шаблон к свойству конкретного устройства, ускоряет создание проекта и его дальнейшее масштабирование.

## Сохранение в БД

Правила сохранения данных устройств в БД настраиваются на вкладке **Сохранение в БД**



## Экземпляры устройств

Экземпляры устройств создаются на базе **Типа устройства**. С помощью правой кнопки мыши в дереве устройств вы можете создать устройство необходимого вам типа.

При создании экземпляра вы можете поместить его в иерархическое дерево устройств, которое будет соответствовать, например, его местоположению (Площадка 1 -> Цех 1).

Для упрощения задачи добавления устройств в проект существует механизм загрузки и выгрузки устройств через CSV файл.

Чтобы воспользоваться данным механизмом, необходимо правой кнопкой нажать на узел **Устройства** в дереве устройств и выбрать **Выгрузить ВСЕ устройства** или **Загрузить (заменить) ВСЕ устройства**

## Вкладки экземпляра

### Устройство

При создании экземпляра данные этой вкладки будут сгенерированы на основе выбранного типа.

Все параметры, кроме внутреннего ID устройства, можно изменить в любой момент существования проекта:

- **Имя устройства** - допустимы только символы azAZ09\_  
Имя устройства должно быть уникально для проекта.  
*Если вы меняете имя устройства, которое уже используется в сценарии, его нужно будет изменить и там.*
- **Полное название** - любое текстовое описание
- **Тип устройства** - можно изменить тип существующего устройства, но имейте в виду, что при этом изменятся свойства устройства
- **Метки** - ключевые слова. Используются в дальнейшем в различных фильтрах.  
При клике на поле Метки будет показан список существующих меток.  
Для создания новой метки необходимо ввести ключевое слово и нажать Ввод.

### Свойства

Набор свойств определяется типом. На уровне экземпляра можно изменить параметры свойств:

- **Сохранять в Главный журнал** - любое изменение будет попадать в [Главный журнал](#)
- **Сохранять в Журнал устройства** - любое изменение будет попадать в [Журнал устройства](#)
- **Минимальное значение**
- **Максимальное значение**
- **Цифр после запятой**
- **Единица измерения**

### Сохранение в БД

Большой объем непрерывно поступающих в систему данных может вести к быстрому исчерпанию дискового пространства.

Система не требует записи всех данных подряд в историческую базу данных.

Имеет смысл сохранять значения только тех свойств, которые в дальнейшем будут использоваться для анализа, построения графиков и отчетов.

Можно выбрать различные **Методы сохранения** и настроить **Срок хранения данных** для каждого свойства.

Это позволяет рационально использовать хранилища.

## Методы сохранения

1. **При изменении.** При выборе данного метода данные сохраняются в БД только при изменении.

Для аналоговых значений можно настроить абсолютную дельту отклонения от ранее записанного значения.

Можно также настроить время записи в БД при отсутствии изменений.

2. **Все значения.** В БД будут записываться все поступающие значения.

3. **Периодически.** Данные в БД сохраняются с определенным интервалом и по определенному правилу:

- o First - первое значение в интервале
- o Last - последнее значение в интервале
- o Min - минимальное значение в интервале
- o Max - максимальное значение в интервале
- o Min/Max - минимальное и максимальное значение в интервале
- o Mean - среднее значение в интервале

4. **Таймлайн.** В БД будут записываться любые изменения для построения графиков типа [Таймлайн](#). Данный график предназначен для отображения изменения состояния, например Включение/Выключение насоса.

5. **Не сохранять.** При выборе данного метода данные сохраняться в БД не будут.

Уже сохраненные данные не удаляются.

*Добавлено v5.18.1.*

Начиная с **v5.18** есть возможность сохранять значения не только числовых, но и строковых свойств.

Если у устройства есть свойства типа **String**, на вкладке будет выведена отдельная табличка для формирования правил сохранения таких свойств.

В этой табличке меньше столбцов, так как для строковых значений нет методов сохранения **Периодически** и **Таймлайн**.

Числовые значения хранятся в таблице БД **records**, строковые в **strrecords**. При использовании БД **PostgreSQL** для этих таблиц применяется механизм гипертаблиц.

## Отладчик

В отладчике можно проверить работу обработчиков для этого экземпляра устройства. Более подробно про [Отладчик](#)

## Журнал

Журнал экземпляра содержит записи о вызове команд и изменении значений для тех свойств, для которых стоит галочка **Сохранять в журнал устройства**.

## Использование

В данной вкладке можно отследить, в каком месте визуализации используется данное устройство, и в каком сценарии оно задействовано.



## Последние записи в БД

Отображаются последние 100 записей исторической БД, относящихся к устройству.

*Добавлено v5.18.1.*

Добавлен столбец с именем таблицы, в которой хранятся данные.

Для числовых значений это **records**, для строковых **strrecords**.

## Привязка каналов к свойствам

Связь виртуального устройства с физическим миром выполняется через каналы.

- любое свойство устройства, кроме calculate, можно привязать к каналу плагина
- к команде можно привязать отдельный канал "только для записи"
- можно внутри устройства привязывать разные свойства к каналам разных плагинов.
- привязку можно сделать как из канала, так и из свойства устройства

Структуру данных для настройки канала определяет плагин.

Для MQTT это топики и сообщения. Для Modbus - адреса регистров и функции....

Каналы имеют флаги Чтение (R) и Запись (W), то есть можно создать канал "только для чтения", "для чтения/записи", "только для записи". Все зависит от оборудования. Иногда есть разные варианты решения задачи.

### Пример 1: Реле 1 через MQTT - один канал R/W

По документации устройство имеет топики:

Топик чтения статуса: `/devices/<deviceid>/0` присылает on или off

Топик для включения: `/devices/<deviceid>/0/command on`

Топик для выключения: `/devices/<deviceid>/0/command off`

Создаем канал MQTT, ставим галочки Чтение и Запись, заполняем:

Топик для подписки: `/devices/<deviceid>/0`

Формула извлечения значения: `value == "on" ? 1 : 0`

Топик для публикации: `/devices/<deviceid>/0/command`

Сообщение для публикации: `value == 1 ? "on" : "off"`

Используем устройство стандартного типа **Актuator бинарный**, которое имеет свойства:

- state - Data Bool - состояние реле
- on, off, toggle - команды

Привязываем созданный канал к свойству state. На этом привязка закончена. Встроенные обработчики команд on/off будут переключать свойство state через привязанный канал.

Команды к каналам привязывать не надо

### Пример 2: Реле 2 через MQTT - три разных канала: R, W, W

Бывает, что для команд on и off предназначены разные топики:

Топик чтения статуса: `/devices/<deviceid>/0` присылает on или off аналогично примеру 1

Топик для включения: `/devices/<deviceid>/0/on` не имеет сообщения, но топики разные

Топик для выключения: `/devices/<deviceid>/0/off`

В этом случае нужно создать 3 отдельных канала:

1. Канал получения состояния state, галка Чтение

Топик для подписки: `/devices/<deviceid>/0`

Формула извлечения значения: `value == "on" ? 1 : 0`

Канал привяжем к свойству state устройства

2. Канал команды on, галка Запись

Топик для публикации: `/devices/<deviceid>/0/on`

Сообщение для публикации: не заполняем

Канал привяжем к команде on устройства

3. Канал команды off, галка Запись

Топик для публикации: `/devices/<deviceid>/0/off`

Сообщение для публикации: не заполняем

Канал привяжем к команде off устройства

## Пример 2.1 Реле 1 - три разных канала: R, W, W

Для Реле 1 из Примера 1 также можно было использовать 3 канала,

1. Канал получения состояния state, галка Чтение

Топик для подписки: `/devices/<deviceid>/0`

Формула извлечения значения: `value == "on" ? 1 : 0`

Канал привяжем к свойству state устройства

2. Канал команды on, галка Запись

Топик для публикации: `/devices/<deviceid>/0/command` Топики одинаковые, сообщение заполняем

Сообщение для публикации: `on`

Канал привяжем к команде on устройства

3. Канал команды off, галка Запись

Топик для публикации: `/devices/<deviceid>/0/command`

Сообщение для публикации: `off`

Канал привяжем к команде off устройства

## Пример 3: Привязка устройства к каналам разных плагинов

Счетчик получает значение по MQTT, но сбросить его можно по Modbus.

Создаем тип устройства Счетчик с командой Сброс:

- value - Data Number - показания счетчика - привязка к каналу плагина MQTT (R)
- offset -Parameter Number - базовое показание, которое будет добавляться к value
- clear - Command - сброс счетчика - привязка к каналу плагина Modbus (W)

## Пример 4: Реле на MegaD.

Создаем типовой канал MegaD, ставим галочки Чтение и Запись.

Используем стандартный тип устройства - Актуатор бинарный.

Привязываем канал к свойству state. На этом привязка закончена. Встроенные обработчики команд on/off будут переключать свойство state через привязанный канал.

Команды к каналам привязывать не надо

Примеры [различных диммеров и привязка к каналам](#)

## Состояние канала

### Атрибут chstatus

У привязанных к каналам свойств появляется значение атрибута **Состояние (ошибка) канала**.

Устройство	Свойства	Значения	Тревоги	Журнал	Сохранение в БД	Последние з:	СОХРАНИТЬ
:: ID свойства	:: Название	:: Значение	:: Канал	Состояние (ошибка) канала	:: Блокиро... канал	:: Значение строка	::
state	Состояние	3	modbus1.PLC_1.ch	1	<input type="checkbox"/>	Закрыт	
auto	Режим Авто	0	emuls1.DP_001	0	<input type="checkbox"/>	OFF	
timeOn	Таймаут включения	5				5	

Состояние канала содержит атрибут **#chstatus**.

Например, для свойства **value** это **value#chstatus**, для свойства **auto** - **auto#chstatus**.

Если плагин не работает, все связанные с ним каналы находятся в состоянии ошибки (=1). После запуска плагина при получении данных ошибка сбрасывается.

После запуска плагина статус канала переключится в 0 только **после получения данных** по этому каналу

Обычно плагин присылает chstatus:1, если по какой-то причине значение с канала перестало приходить (например, не отвечает конкретный адрес по ModbusRTU)

```
{id:'ch1', value:42, chstatus:0} // Значение с канала получено
{id:'ch99', chstatus:1} // Значение не получено, возникла ошибка
```

Для некоторых протоколов (например, OPC) передается **quality** - информация о состоянии и достоверности данных. Это может быть не 0/1, а другое целое число. Значение quality также будет записано в атрибут chstatus.

### Сохранение в БД

При поступлении **chstatus>0** в БД пишется **null**. В этом случае на графике будет разрыв, в отчетах значение не будет учитываться.

### Использование состояния канала на визуализации

Состояние канала можно использовать на визуализации:

auto#changetime	Режим Авто : Время изменения значения	<input type="checkbox"/>
auto#chstatus	Режим Авто : Статус (ошибка) канала	<input checked="" type="checkbox"/>
auto#chstatuschangetime	Режим Авто : Время изменения статуса канала	<input type="checkbox"/>
auto#propTitle	Режим Авто : Название свойства	<input type="checkbox"/>
auto#refreshtime	Режим Авто : Время обновления значения	<input type="checkbox"/>

Bindings			Function
inData	GV2.auto#chstatus		1 return inData;

Для индикации ошибки на мнемосхеме можно использовать разные стратегии:

- Вариант 1.  
Вместо значения выводить например прочерк или знак вопроса Для этого к тексту для показа привязать оба атрибута (#string и #chstatus), результат возвращать в зависимости от состояния канала:

AI\_1057 • Датчик аналоговый 1057 🔍 Search...

value#changetime	Значение : Время изменения значения	<input type="checkbox"/>
value#chname	Значение : Привязанный канал	<input type="checkbox"/>
value#chstatus	Значение : Статус (ошибка) канала	<input checked="" type="checkbox"/>
value#chstatuschangetime	Значение : Время изменения статуса канала	<input type="checkbox"/>
value#dbsuspend	Значение Флаг блокировки сохранения в БД	<input type="checkbox"/>

Bindings			Function
inData	AI_1057.value#string		1 return inData1>0 ? '???' : inData;
inData1	AI_1057.value#chstatus		

- Вариант 2.  
Значение выводить как обычно, а рядом выводить индикатор, что текущее значение недостоверно  
Например, в качестве индикатора ошибки использовать image, и скрывать его, если ошибки нет
- Другие варианты с использованием привязки значения и статуса канала к элементам визуализации...

## Использование состояния канала в скриптах

В скриптах и сценариях можно получить статус канала с помощью метода устройства **getPropValue**:

```
// Получить статус канала, привязанного к свойству auto
device.getPropValue('auto#chstatus')
```

## Привязка статуса канала к отдельному свойству устройства

Иногда бывает удобно привязать статус канала к отдельному свойству устройства.

Это бывает полезно в случае:

- если изменение статуса должно быть триггером для сценария
- если планируется использовать единый флаг для индикации состояния связи всех каналов плагина

The screenshot shows the 'Channels' tab in the IntraSCADA interface. On the left, under 'Channels', there is a list with 'Частотник2' and 'Новый канал'. The main area is divided into 'Properties' and 'Channels' sections. In the 'Properties' section, there are two rows of configuration:

- Row 1: 'Привязка канала к свойству устройства' (Inv\_002 • Частотник EuraDrive • frequency) with a copy icon.
- Row 2: 'Привязать статус канала' (checked checkbox) with a blue arrow pointing to it, followed by 'Привязка статуса канала к свойству устройства' (Inv\_002 • Частотник EuraDrive • nolink) with a copy icon.

Below these, there are input fields for 'Unit ID (адрес устройства на шине)' (value: 2), 'Канал' (value: frequency), and 'Свойство для привязки' (value: frequency). To the right, there are checkboxes for 'Чтение' (checked) and 'Запись' (unchecked).

Здесь значение с канала связано со свойством **frequency**, а статус канала помещен в свойство **nolink**, которое будет использоваться как индикатор наличия связи.

Имя свойства для привязки к статусу может быть любым (не нужно только использовать системное свойство **error**, у него другие цели).

Теперь есть возможность использовать эту информацию в системе (для целей визуализации, в сценариях, ...). Имеет смысл привязать статус к наиболее часто меняющемуся каналу.

Не все плагины позволяют привязать статус канала к отдельному свойству устройства. На текущий момент этот механизм есть в плагинах **modbus** и **opcua**.

В манифесте такой плагин имеет атрибут:

```
linkChstatus:1
```



## Блокировка каналов

*Добавлено v5.14.27*

Иногда бывает необходимо вводить значения свойства вручную (для отладки или проверки функционала). Если свойство привязано к каналу, при поступлении данных с канала значение, введенное вручную, будет заменено.

Блокировка позволяет временно заблокировать поступление данных, не отвязывая при этом канал (фактически, данные с плагина приходят, но не передаются в свойство).

Для реализации такой возможности свойству устройства добавлен атрибут `#blkchannel`.

**Этот атрибут появляется только у свойства, привязанного к каналу.**

Данные, вводимые вручную, будут обработаны, как если бы они поступали от физического канала (будут отработаны тревоги, сценарии, выполнена запись в журнал и БД, ...).

Сразу после снятия блокировки в свойство будет передано последнее значение, поступившее в канал.

Заблокировать или разблокировать канал можно:

- в РМ для устройства на вкладке **Значения**
- в UI - привязав атрибут **#blkchannel**
- через сценарий - функция устройства **setBlkChannel**

### Блокировка в РМ

Для устройства на вкладке **Значения** в столбце **Блокировать канал** галочкой можно включить и отключить блокировку канала.

Галочка доступна только для свойств, привязанных к каналу.

### Блокировка в UI

При привязке визуальных компонентов можно выбрать атрибут **#blkchannel**. Значение 1 означает, что канал заблокирован.

### Блокировка через сценарий

Используется метод устройства **setBlkChannel(<имя свойства>, <1-блокировать,0-разблокировать>)**

```
const dev = Device("AI_001")

const script = {
  start() {
    dev.setBlkChannel('temp1', 1);
    dev.setBlkChannel('temp2', 1);
    dev.setBlkChannel('temp3', 1);
    dev.setBlkChannel('state', 1);
  }
};
```

Этот функционал предназначен для использования при разработке или тестировании!

Убедитесь, что в оперативном режиме блокировка снята.

Информация о заблокированных каналах есть в общей таблице устройств:

PM -> корневой узел Устройства -> Таблица -> столбец Есть заблокированные каналы

# Примеры создания каналов

В качестве примеров для простоты рассмотрим подключения различных диммеров.

Можно создать диммер, используя стандартный тип **Аналоговый актуатор**. Смотрим, какие свойства есть у этого типа.

Свойства - значения:

- value - свойство типа Data Number  
Можно использовать для чтения/записи текущего значения яркости
- state - свойство типа Calculate Bool  
Вычисляется состояние по формуле `device.value>0 ? 1 : 0`
- setpoint - свойство типа Parameter Number  
Это уставка, используется при выполнении команды on. Уставку можно задавать интерактивно с интерфейса, менять сценарием...

Свойства - команды. Каждая команда имеет функцию-обработчик для реализации своего функционала.

- on - Функция-обработчик: `device.setValue('value', device.setpoint);`
- off - Функция-обработчик: `device.setValue('value', 0);`
- toggle - Функция-обработчик: `device.value ? device.off() : device.on();`

В таком виде виртуальный диммер вполне работоспособен. Управление устройством происходит через изменение value. Теперь рассмотрим диммеры реального мира

## Modbus диммер 0-10v

С точки зрения железа, считывается и записывается значение яркости по заданному адресу.

- value - свойство типа Data - привяжем к каналу modbus. Для канала задаем адрес и функцию. Ставим галки Чтение и Запись.

С физического устройства можно считать только яркость. Состояние нужно будет вычислять, и это у нас есть. Больше привязывать нечего (про on/off этот диммер ничего не знает). Для такого диммера стандартный тип подходит.

## MQTT Shelly Dimmer

С точки зрения железа, имеются топики для чтения яркости и состояния. Кроме того, устройство имеет команду set для установки яркости, и может выполнить команды on/off.

Итак, состояние тоже будем считывать с устройства, а не вычислять. Поэтому создаем новый тип устройства **Диммер с состоянием**

За основу берем аналоговый актуатор.

Все, что нужно сделать - это изменить тип свойства **state** на Data вместо Calculate

- state - свойство типа Calculate => Data

Создаем каналы:

1. Канал для чтения и установки яркости. Ставим галки Чтение и Запись.

Топик для подписки: `shellies/shellydimmer-<deviceid>/light/0/status`

Формула извлечения значения: `JSON.parse(value).brightness`

Топик для публикации: `shellies/shellydimmer-<deviceid>/light/0/set`

Сообщение для публикации: `{"brightness": ${value}, "turn": "on"}`

Канал привязываем к свойству **value**

2. Канал для чтения состояния. Ставим галку Чтение.

Топик для подписки: `shellies/shellydimmer-<deviceid>/light/0`

Формула извлечения значения: `value == "on" ? 1 : 0`

Канал привязываем к свойству **state**

3. Канал команды off. Ставим галку Запись.

Топик для публикации: `shellies/shellydimmer-<deviceid>/light/0/command`

Сообщение для публикации: `off`

Канал привязываем к свойству **off**

4. Для команды on есть варианты:

**Вариант 1** - создать канал on и привязать к команде on (аналогично команде off)

Канал команды on. Ставим галку Запись.

Топик для публикации: `shellies/shellydimmer-<deviceid>/light/0/command`

Сообщение для публикации: `on`

Канал привязываем к свойству **on**

В этом варианте при включении будет применена сохраненная на диммере яркость (выполнится встроенная команда)

**Вариант 2** - Можно не создавать канал для on и ничего не привязывать к свойству on.

Использовать опцию, что этот диммер включается, если выдать ненулевую яркость в Канал 1 (set).

Если ничего не привязывать, по команде on сработает функция-обработчик

`device.setValue('value', device.setpoint)` . Произойдет запись в канал, привязанный к свойству value, и яркость установится в значение уставки setpoint.

## "Хитрый диммер" по HTTP

Подключаем диммер, у которого есть отдельно управление состояниями: включить и выключить, а также яркостью.

При этом яркостью можно управлять и в выключенном состоянии диммера.

При включении устанавливается заданная яркость.

Пробуем применить тип **Диммер с состоянием**

- value - свойство типа Data, привязка к каналу http для получения и задания яркости (чтение, запись)

- state - свойство типа Data, привязка к каналу http для получения состояния (чтение)
- on - привязка к каналу http - отправить команду on
- off - привязка к каналу http - отправить команду off

Минусы: Диммер выключен, а value показывает не нулевую яркость.

Здесь нужно рассматривать яркость диммера как setpoint, а value вычислять.

Придется создать новый тип устройства **Хитрый диммер**, у которого изменить типы свойств

- value - свойство Data=> Calculate: `device.state ? device.setpoint : 0;`
- setpoint - свойство типа Parameter => Data, привязка к каналу http для получения и задания яркости (чтение, запись)

Обратите внимание, несмотря на изменения в типах и каналах, на визуализации у нас все будет работать без изменений.

# Отладчик

## Для чего используется

Цель - отладка **пользовательских обработчиков**.

Консоль аналогична отладчику плагинов, данные выводятся в реальном времени.

## Что выводится автоматически

При нажатии кнопки "Play" в консоли выводится список пользовательских обработчиков этого устройства. Это могут быть:

- основные обработчики свойств для приема данных (Data, Param) или вычисления (Calc).  
Возвращают результат - новое значение свойства;
- форматирующие обработчики свойств.  
Возвращают результат - новое строковое значение свойства;
- обработчики для команд, результат не возвращают;
- обработчики, относящиеся к устройству в целом, результат не возвращают.

Пример устройства, имеющего 3 основных пользовательских обработчика для свойств, форматирующий обработчик для state, а также обработчики, работающие циклически, по расписанию и при изменении заданных свойств.

Пользовательские обработчики устройства:

```
Обработчик state
Обработчик форматирования state
Обработчик p_cooling
Обработчик p_light
Обработчик "При изменении" свойств: state,p_cooling,p_light
Обработчик "Циклически" :10 сек
Обработчик "По расписанию" :minutely
```

Далее в консоль будет выводиться информация о запуске и завершении этих обработчиков в реальном времени: Результат, возвращаемый обработчиками свойств, будет выведен после знака ==>.

```
04.08 21:30:22.033 □ Обработчик state запущен
04.08 21:30:22.033 ✓ Обработчик state завершен ==> 1
04.08 21:30:22.033 □ Обработчик форматирования state запущен
04.08 21:30:22.033 ✓ Обработчик форматирования state завершен
                        ==> 'Включено'
```

## Как вывести другую информацию

Можно выводить любые значения /сообщения, включив в обработчик команду **debug(<строка>)**

```
/**
 * Функция вычисления значения свойства "Input_P"
 *   device – Устройство (Object)
 *
 */
module.exports = function(device, prop, value, debug) {
  debug(
    'Сумма L1=' + device.Input_L1 +
    ', L2=' + device.Input_L2 +
    ', L3=' + device.Input_L3
  )
  return device.Input_L1 + device.Input_L2 + device.Input_L3;
}
```

В консоли получим:

```
04.08 21:32:27.033 □ Обработчик Input_P запущен
04.08 21:32:27.033 Сумма L1=15, L2=15, L3=12
04.08 21:32:27.033 ✓ Обработчик Input_P завершен => 42
```

## Аргументы функции-обработчика

Обратите внимание!

Чтобы использовать функцию debug, она должна быть прописана в аргументах!

Заново создаваемые обработчики устройств теперь имеют больше аргументов:

**function(device, prop, value, debug, senderObj)** // четвертый аргумент - функция debug

Если вы хотите использовать debug в уже написанных скриптах, имеющих меньшее число аргументов, просто добавьте необходимые аргументы, соблюдая новый порядок.

Если отладка не нужна, можно ничего не изменять. Функции JavaScript поддерживают переменное количество аргументов.

## Как выводятся ошибки скрипта

Если при выполнении функции произошла ошибка, это будет выведено в консоль.

**Пример 1.** В коде вызвали debug, но не добавили в аргументы:

```

module.exports = function(device, prop, value) {

  debug(
    'Сумма L1=' + device.Input_L1 +
    ', L2=' + device.Input_L2 +
    ', L3=' + device.Input_L3
  );

  return (
    device.Input_L1 +
    device.Input_L2 +
    device.Input_L3
  );
};

```

В отладчике будет выведена ошибка:

```

04.08 18:47:08.100 Обработчик p_light БЛОКИРОВАН
Ошибка ReferenceError: debug is not defined

```

**Пример 2.** Допущена ошибка в названии метода устройства (device.of вместо device.off):

```

module.exports = function(device, prop, value, debug) {
  if (device.state) {
    device.of();
  } else {
    device.on();
  }
}

```

В отладчике будет выведена ошибка:

```

04.08 18:54:21.205 Обработчик toggle БЛОКИРОВАН
Ошибка TypeError: device.of is not a function

```

Если при запуске отладчика обработчик уже с ошибкой и, соответственно, блокирован, это будет отражено в списке, который выводится при старте отладки:



Пользовательские обработчики устройства:

Обработчик `state`

Обработчик `p_cooling`

Обработчик `p_light` БЛОКИРОВАН

Обработчик `"При изменении"` свойств: `state,p_cooling,p_light`

## Как исправить ошибку и снять блокировку

Перейдите на вкладку **Обработчики** в **Типе устройства**, в дереве выберите свойство. Сообщение о блокировке и описание ошибки выводится над скриптом. Исправьте ошибку и сохраните скрипт.

После сохранения, если синтаксический контроль пройден, блокировка сбрасывается.

Нужно отметить, что ошибки скрипта могут быть 2 видов:

- Синтаксические ошибки, которые отлавливаются при сохранении скрипта. В этом случае скрипт блокируется сразу, выполнения не будет.
- Ошибка времени выполнения. Самая распространенная ошибка выполнения - отсутствие переменной или функции.

Это относится к любым скриптам системы, не только к обработчикам.

## Журнал устройств

Журнал устройств - это журнал последних изменений свойств устройств с ограниченным количеством записей. Количество записей в журнале настраивается в настройках системы, на вкладке **Текущий проект**. Данные журналы находятся постоянно в оперативной памяти.

Для вывода Журнала устройства на визуализацию необходимо добавить Виджет **Device Log** и в привязке указать ссылку на устройство. Ссылку так же можно указать на Переменную клиента для динамического отображения журнала, например в диалогах.

## Системные индикаторы

Под узлом **Устройства** в дереве находятся **Системные индикаторы**.

Это специально генерируемые устройства, содержащие информацию об объектах системы.

Они появляются в дереве автоматически, и также автоматически удаляются. В отличие от обычных устройств, они имеют встроенные свойства только для чтения и идентификатор, который нельзя изменить.

Как и любое свойство устройства, свойство индикатора можно сохранить в журнал, БД и вывести на визуализацию. Системные индикаторы можно использовать в сценариях как обычные устройства.

Системные индикаторы разбиты на папки, которые не могут быть изменены. Внутри повторяется структура папок исходной сущности.

## Процессы

В папке **Процессы** находятся индикаторы плагинов, агентов БД и основного процесса. Индикаторы плагинов появляются в дереве автоматически при установке плагинов и пропадают при удалении плагина.

## Журналы тревог

Для каждого Журнала тревог раздела Аналитика создается Системный индикатор журнала.

Он содержит данные о количестве активных предупреждений и аварий, и записях, требующих квитирования. Это данные можно использовать на визуализации и в сценариях.

## Шаговые сценарии

Экспериментальная возможность. Для каждого Шагового сценария создается индикатор, позволяющий отслеживать текущее состояние, включая активный шаг.

## Системные индикаторы плагинов

### Основные свойства

Системные индикаторы для каждого экземпляра плагина имеют следующие свойства:

- state - Состояние работы плагина
- version - Текущая версия плагина
- memrss - Общий объем в Кб места, занимаемого в оперативной памяти
- memheap - Объем в Кб выделенной динамической памяти (куча) всего
- memhuse - Объем в Кб выделенной динамической памяти использованной

### Состояние работы плагина

Состояние плагина (свойство state) устанавливается сервером:

- 0 - плагин не установлен
- 1 - плагин работает
- 2 - остановлен пользователем
- 3 - остановлен с ошибкой и будет перезапускаться системой в соответствии с настройками
- 4 - остановлен
- 5 - заблокирован
- 10 - находится в процессе запуска

### Дополнительные свойства

При разработке плагина есть возможность добавить плагину [дополнительные свойства](#).

## Системные индикаторы агентов БД

К свойствам системных индикаторов агентов БД помимо свойств системных индикаторов плагинов добавляются несколько дополнительных свойств:

### DB агент (логи):

- size - Объем в Мб, занимаемый под журналы mainlog, pluginlog, devicelog
- mainlog - количество записей в основном журнале системы
- pluginlog - количество записей в журнале плагинов системы
- devicelog - количество записей в журнале устройств системы
- dbname - название используемого ДБ агента (по умолчанию sqlite)

### DB агент (историческая БД):

- size - Объем в Мб, занимаемый под исторические данные, таймлайны и пользовательские таблицы
- overflow - Индикатор переполнения БД.

Если индикатор имеет состояние 1, то это означает, что выделенная вами память под исторические данные в разделе [настройки БД](#) закончилась и ДБ агент перестал записывать в базу данных. Для восстановления записи данных в БД вам необходимо либо увеличить лимит, либо удалить данные через настройки системы или SQL Навигатор. После проведения всех манипуляций необходимо перезагрузить ДБ агент из процессов.

- lastMaxTimeWrite - максимальное время записи в БД в мс
- lastMaxCountWrite - максимальное количество записей за максимальное время записи
- lastMaxTimeRead - максимальное время чтения из БД в мс
- lastMaxCountRead - максимальное количество записей за максимальное время чтения
- dbname - название используемого ДБ агента (по умолчанию sqlite)

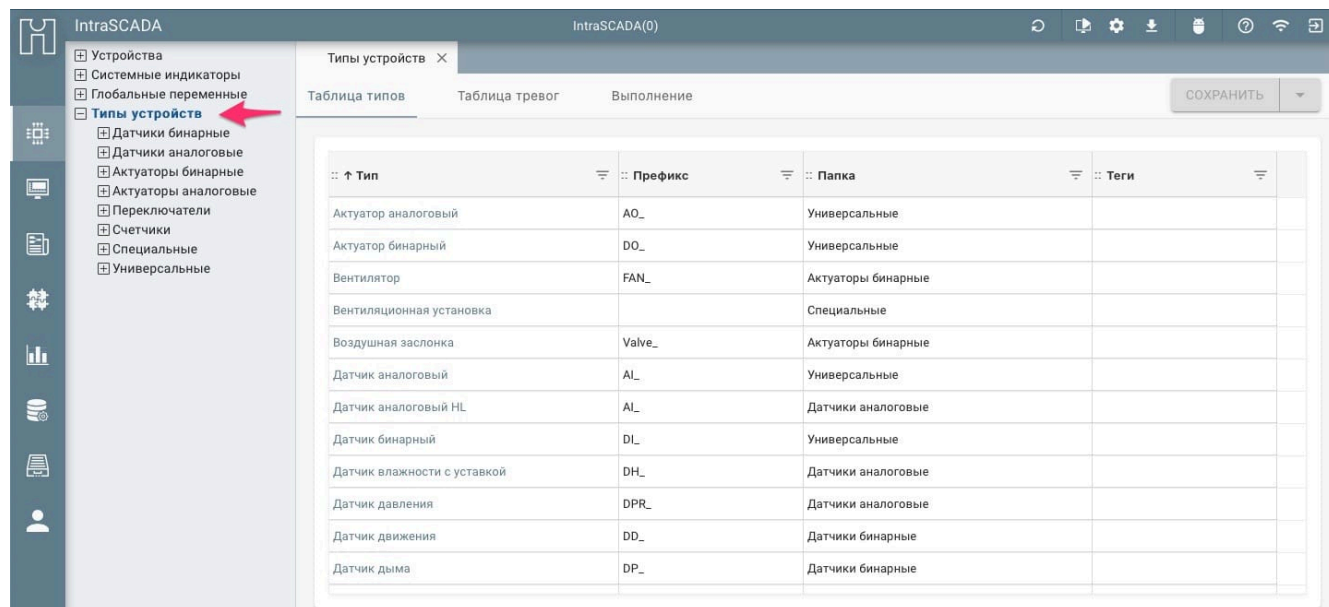
## Типы устройств

Экземпляры устройств создаются на базе **Типа устройства**. Фактически **Тип** - это шаблон, содержащий описание устройства.

Можно взять готовый тип, можно создать новый, можно доработать существующий.

Любой тип можно выгрузить и использовать в других проектах.

Типы можно (но не обязательно) разложить по папкам.



### Что определяет тип

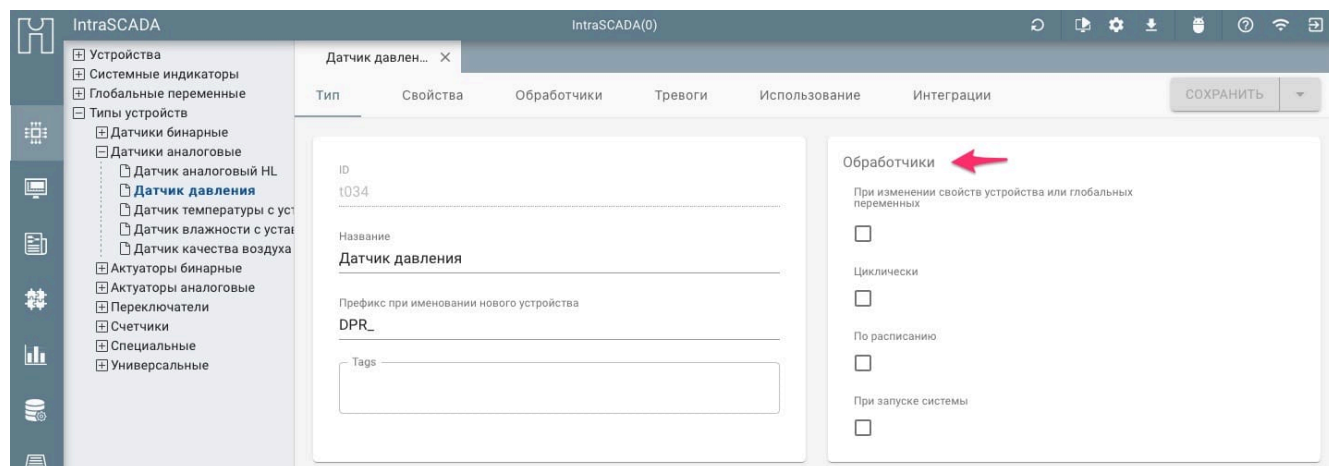
Тип определяет набор свойств устройства, и, возможно, внутреннюю логику.

Для задания логики предназначены скрипты-обработчики.

Можно создавать обработчики на уровне отдельного свойства.

Есть возможность создать несколько обработчиков на уровне всего устройства, которые будут запускаться:

- при изменении свойств устройства
- циклически по заданному правилу
- по расписанию
- при старте системы



Все обработчики являются частью типа и будут перенесены при переносе типа.

Также для типа можно настроить механизм тревог.

### Что не определяет тип

Хотя можно создать тип, ориентированный на конкретное оборудование, обычно тип является абстрактным по отношению к железу.

Также тип не определяет визуализацию.

## Свойства

Свойств у одного устройства может быть сколько угодно. Набор свойств определяется типом.

The top screenshot shows the 'Датчик давл...' (Pressure Sensor) type. The 'Свойства' (Properties) tab is active, displaying a table with the following data:

ID свойства	Название	Тип свойства	Тип переменной	Основной обработчик	Значение в виде строки	Тревоги	Mir
value	Значение	Data	Number	Встроенный	Встроенный	<input type="checkbox"/>	
setpoint	Уставка	Parameter	Number	Встроенный	Встроенный	<input type="checkbox"/>	

The bottom screenshot shows the 'Вентиляцион...' (Ventilation) type. The 'Свойства' (Properties) tab is active, displaying a table with the following data:

ID свойства	Название	Тип свойства	Тип переменной	Основной обработчик	Значение в виде строки
state	Состояние установки	Data	Bool	Встроенный	Пользовательский
on	Включить	Command		Встроенный	
off	Выключить	Command		Встроенный	
temp1	Температура наружного воздуха	Data	Number	Встроенный	Встроенный
temp2	Температура приточного воздуха	Data	Number	Встроенный	Встроенный
temp3	Температура обратного воздуха	Data	Number	Встроенный	Встроенный
temp4	Температура вытяжки	Data	Number	Встроенный	Встроенный
temp5	Температура обратной воды	Data	Number	Встроенный	Встроенный
setpoint	Уставка температуры приточного воздуха	Parameter	Number	Встроенный	Встроенный
fan1	Вентилятор приточки	Data	Bool	Не используется	Встроенный
fan2	Вентилятор вытяжки	Data	Bool	Не используется	Встроенный
filter1	Фильтр приточки	Data	Bool	Не используется	Не используется

Для каждого свойства в типе задаются:

- идентификатор (допустимы только латинские буквы, цифры и знак подчеркивания)
- текстовое название - любое текстовое описание
- тип свойства - **Data, Parameter, Event, Calculate, Command**

Свойство, содержащее данные (типа **Data, Parameter, Event**), может быть связано с каналом для получения (read) и/или отправки (write) значения.

Вместе с тем, можно не привязывать свойство к каналу, а хранить там данные ввода пользователя или присваивать значения сценарием.

- **Number** - число: целые и вещественные числа; дата в виде timestamp

*В JavaScript максимальное целое число без потери точности это  $2^{53}-1$  или 9 007 199 254 740 991*

- **String** - строка: строковые значения; возможно хранение объектов в JSON или даты в строковом формате
- **Bool** - бинарное значение: допустимые значения только 1 и 0
- **BigInt** - *Добавлено в v5.18.7*

BigInt позволяет работать с целыми числами без ограничения по максимальному значению.

Следует учитывать, что BigInt - это не обычные целые числа, работа с ними имеет свои особенности, подробнее [JavaScript. Очень большие числа](#)

Практически в SCADA BigInt бывает востребован для значений INT64.

## Свойства типа **Data**

Обычно хранят значения, поступающие с каналов, данные могут приходить очень часто.

Поступающие данные обрабатываются обработчиком (встроенным или специально написанным).

Если значение изменилось - генерируется событие изменения.

По событию изменения происходят различные действия - обновление элементов визуализации, запуск сценариев, обработка тревог и т.д. Основным источником данных этого типа - плагины.

Если есть необходимость сохранения значения при перезагрузке, то установите галочку **Сохранить в журнал устройства** либо используйте тип устройства **Parameter**.

## Свойства типа **Event**

Отличаются от **Data** лишь тем, что любое поступление данных генерирует событие изменения.

Бывает полезно для различных кнопок, которые присылают только один сигнал при нажатии (отжати).

## Свойства типа **Parameter**

Предназначены для хранения всевозможных параметров и настроек. Гарантируется восстановление значения при перезагрузке.

Рекомендуется применять для свойств, которые меняются не слишком часто.

## Вычисляемые свойства **Calculate**

Свойства автоматически пересчитываются при изменении других свойств.

Для выполнения расчета пишется обработчик свойства.

Привязать к каналу это свойство нельзя.

## Свойства типа **Command**

Реализуют выполнение команд. [Более подробно про команды](#) Могут привязываться к каналу на запись. Канал на чтение для них смысла не имеет.

## Пример



Для простоты приведем пример устройства, принцип работы которого знаком всем. Это **Диммер**. Можно создать диммер, используя стандартный тип **Аналоговый актуатор** (устройство, управляемое изменением свойства **value**)

- **value** типа **Data Number** - содержит текущую яркость (0-100)
- **state** типа **Calculate Bool** - будем вычислять состояние 0/1, в зависимости от яркости
- **setpoint** типа **Parameter Number** - содержит уставку
- Команды (свойства типа **Command**)
  - **on** - выдает значение **setpoint** в канал **value**
  - **off** - выдает значение 0 в канал **value**
  - **toggle** - в зависимости от **state** вызывает команды **on** и **off**

Это один из вариантов реализации устройства **Диммер**.

Раздел [Механизм каналов->Примеры](#) содержит варианты диммеров реального мира.

## Параметры свойств

Для свойства при желании можно настроить параметры. Это единица измерения, мин и max значение, число цифр после запятой.

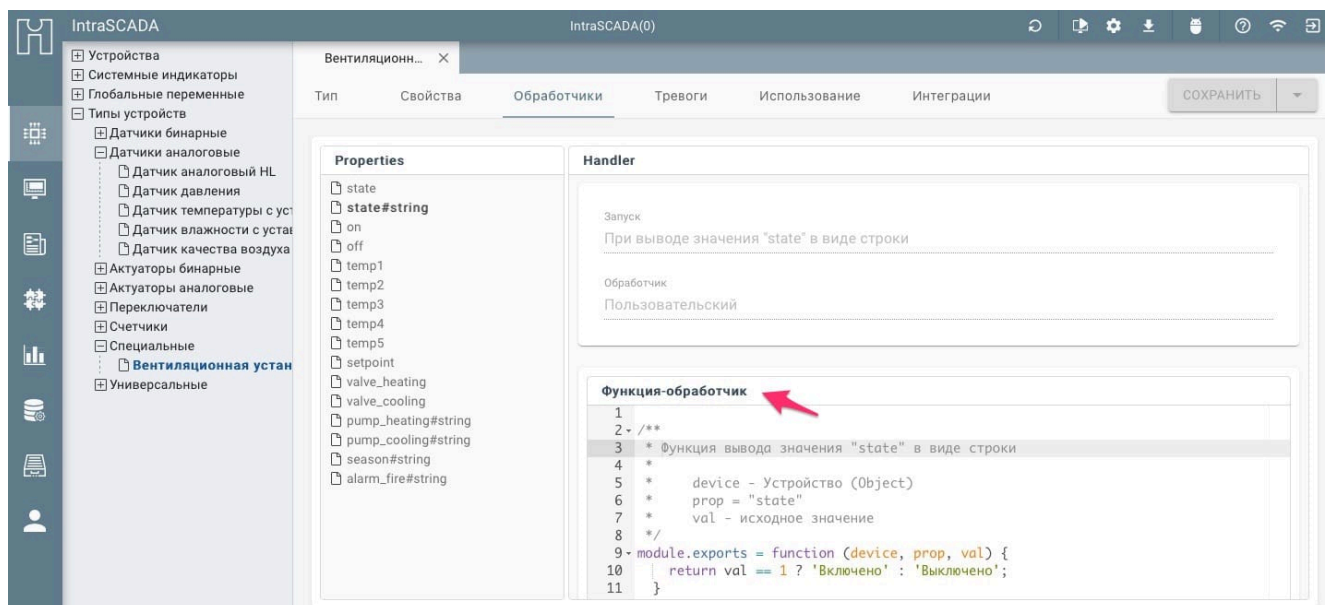
Тип переменной	Основной обработчик	Значение в виде строки	Min	Max	Значение по умолчанию	Цифр после запятой	Ед. изм.
Bool	Встроенный	Пользовательский					
	Встроенный						
	Встроенный						
Number	Встроенный	Встроенный	-50	100		0	°C
Number	Встроенный	Встроенный	0	100		0	°C
Number	Встроенный	Встроенный	0	100		0	°C
Number	Встроенный	Встроенный	0	100		0	°C
Number	Встроенный	Встроенный	0	100		0	°C
Number	Встроенный	Встроенный	0	100		0	°C
Bool	Не используется	Встроенный					

Обычно это имеет смысл для числовых свойств. Также можно присвоить значение по умолчанию. Параметры будут перенесены в устройство при его создании.

В дальнейшем все эти параметры можно отредактировать на уровне экземпляра.

## Обработчики

**Обработчики** - это скрипты, позволяющие настроить логику работы устройства. Задаются в типе. Существуют Встроенные и Пользовательские обработчики.



Каждый Пользовательский обработчик попадает в таблицу **Выполнение** в папке **Типы устройств**, где можно отследить количество запусков и время выполнения каждого обработчика.

Также существует [Отладчик](#), позволяющий отслеживать работу **Пользовательских обработчиков** для конкретного экземпляра устройства.

# Команда устройства

Команда устройства - это свойство "только на запись" для выполнения действий с устройством. Результат выполнения команды предсказуем и четко определен.

## Для чего нужны команды как отдельные свойства?

Чтобы управлять физическим устройством, обычно нужно выполнить **запись** в канал.

Например, по какому-то адресу есть потенциальное реле.

Выполняя чтение по этому адресу, получаем состояние 1/0

Записывая 1/0 по этому адресу, выполняем его переключение

Таким образом, минимальный тип устройства может содержать единственное свойство **state** с привязкой к каналу. И команды вроде не нужны.

- state - Data Bool - состояние устройства -> привязка к каналу, например, Modbus с возможностью чтения/записи

Реализация позволяет выполнить команды on/off, специально не определяя их. Будем просто переключать состояние:

```
LAMP1.setValue('state',1); // включаем... если не инверсный выход...
LAMP1.setValue('state',0); // выключаем... возможно
```

Даже в таком простом случае есть детали, которые нужно учесть: если выдаем 1, мы включаем или выключаем? В каждом сценарии, в каждой точке интерфейса нужно это учитывать. А если мы выполняем групповую операцию?

Пойдем дальше - концепция поменялась, вместо потенциальных используем импульсные реле. Теперь нужно учитывать текущее состояние. И адрес для выдачи импульса обычно другой.

Ничего, добавим новое свойство для записи:

- pulse - Data Bool -импульс -> привязка к каналу на выдачу импульса (пусть плагин сам формирует импульс)

```
// Если нет комментария, то понять, что мы делаем, сложно
if (!LAMP1.state) (LAMP1.setValue('pulse',1);
```

Придется внести изменения везде, где нужно включить LAMP1 из сценария или с интерфейса.

## Команда уменьшает сложность

Попробуем по-другому. Добавим в тип новые свойства-команды: on и off

Мы можем использовать их где угодно - в сценарии, в интерфейсе, в групповых командах. Теперь целевое действие строго определено.

При изменении физической привязки логический уровень не будет затронут.

```
LAMP1.on(); // включаем
LAMP1.off(); // выключаем
```

Таким образом, команда абстрагирует от деталей, добавляет однозначность и уменьшает сложность при разработке проекта и особенно при эксплуатации. Использование команд не обязательно, но очень рекомендуется

## Как работает команда

Команда реализуется функцией-обработчиком или прямой записью в канал.

Обычно обработчик выполняет команду через запись в одно из свойств Data и не требует привязки команды непосредственно к каналу

Рассмотрим обработчики для команд on и toggle для дискретного актуатора

**Обработчик on:** пишет в канал свойства state.

Если изменится логика включения, это единственное место, где нужно будет внести изменения

```
function(device) {
  device.setValue('state', 1);
}
```

**Обработчик toggle:** в зависимости от состояния выдает команды on или off

```
function(device) {
  if (device.state) {
    device.off();
  } else {
    device.on();
  }
}
```

Любой обработчик можно как изменить, так и отключить.

Допустим, актуатор подключен к MQTT реле, которое само может выполнять команды on/ off/ toggle через соответствующие топики.

Очень просто - создаем для каждой команды канал и привязываем каналы к командам устройства.

Наши обработчики тут не нужны, поэтому можно отключить обработчик, выбрав опцию Обработчик "Не используется"

Если для команды обработчик "Не используется", то

- если есть привязка к каналу, просто будет сразу запись в канал
- если нет привязки к каналу, команда ничего не делает

## Встроенные обработчики команд

Для упрощения разработки в системе есть встроенная реализация обработчиков для команд on/ off/ toggle

В этом обработчике учитывается, привязан ли к команде канал

- если привязан, отправляется команда
- если не привязан, команда реализуется через свойства устройства (state)

```
function(device) {  
  if (device.hasChannel('on')) {  
    // Есть канал на свойстве on – отправляем команду железу  
    device.writeChannel('on');  
  } else {  
    // Нет канала – переключаем через state  
    device.setValue('state', 1);  
  }  
}
```

Такой подход позволяет, не отключая обработчик, работать по любому варианту, в том числе если устройство полностью виртуальное

# Обработчики

Любое устройство принадлежит к какому-то типу. Тип задает набор свойств и логику работы:

- прием и проверку значений, обработку ошибок
- алгоритмы выполнения команд
- работу устройства с учетом внутреннего состояния и глобальных переменных

Обработчики - это функции, которые позволяют реализовать всю эту логику.

Обратите внимание, обработчики создаются не для экземпляра, а для типа.  
Если нужен функционал на уровне конкретного экземпляра, используйте сценарий.

Для решения стандартных задач в системе имеются встроенные обработчики, также можно формировать [Библиотеку обработчиков](#).

В зависимости от потребностей, при настройке можно выбрать вариант:

- "Встроенный" обработчик - код можно просмотреть, но нельзя отредактировать
- "Пользовательский" обработчик - можно написать код для выбранного свойства внутри типа на вкладке Обработчики
- "Библиотечный" обработчик - можно создать, а затем использовать один и тот же обработчик из библиотеки для разных свойств/типов
- "Не использовать" - отключить обработчик совсем

Обработчик может быть связан с конкретным свойством или работать с устройством в целом.

## 1. Обработчик свойства

В зависимости от типа свойства (Данные, Вычисление или Команда), обработчик выполняет разные задачи: проверку (преобразование) входных данных, вычисление свойств типа Calculate или выполнение команд.

Для стандартных свойств есть встроенные обработчики

## 2. Форматирующий обработчик

Применяется для преобразования значения свойства в текстовое представление для вывода в журнал/ на интерфейс.

Встроенные возможности - форматирование значения с добавлением единицы измерения, а также функции преобразования даты/ временного периода.

## 3. Обработчик уровня устройства

Это функция для реализации логики работы устройства. Имеет доступ к глобальным переменным. Встроенных обработчиков уровня устройства нет. Можно создать до 4 функций для одного устройства, которые будут запускаться:

- при изменении свойств устройства (или глобальной переменной)
- по расписанию
- циклически
- при запуске системы

Раздел [Виды обработчиков](#) содержит подробное описание и примеры кода.

## Настройка тревог

Встроенный механизм позволяет генерировать тревоги на уровне свойства устройства.

- Активные тревоги попадают в [журнал тревог](#), где может выполняться квитирование. Как только тревога снята, она удаляется из журнала тревог.
- Параллельно записи о тревогах в хронологическом порядке заносятся в [Главный журнал](#) и [журнал устройства](#). Туда же заносится запись о квитировании.

Настройка логики генерации и снятия тревоги выполняется в типе устройства.

## Уровни

Предусмотрено 2 уровня тревоги:

- Warning (Предупреждение)
- Alarm (Авария)

## Снятие тревоги

В системе предусмотрено 3 варианта снятия тревоги:

- При нормализации
  - Тревога снимается, если значение свойства перешло в Норма
- При квитировании
  - Тревога снимается, если выполнено квитирование, даже если сигнал не нормализовался
- Квитирование+нормализация
  - Для снятия требуются оба условия. Если сигнал нормализовался, но не квитирован, состояние алерта в [Журнале тревог](#) перейдет в *Закрыто, ожидает квитирования*

## Тревоги аналоговых свойств.

Используется общепринятая в системах автоматизации градация:

- Lo - понижение значение ниже допустимого (Предупреждение)
- LoLo - критическое понижение значения ниже допустимого (Авария)
- Hi - повышение значение выше допустимого (Предупреждение)
- HiHi - критическое повышение значение выше допустимого (Авария)

Автоматически отрабатываются переключения: Норма - Предупреждение - Авария.

Предыдущее состояние сбрасывается с индикацией "Повышение уровня"/"Понижение уровня".

При настройке можно выбрать, какие градации использовать для данного свойства. Например:

- при повышении температуры использовать обе градации:
  - выше 70 - уровень предупреждение



- о выше 90 - уровень авария
- при понижении - использовать только предупреждение (или не использовать совсем)

## Пороговые значения аналоговых свойств

В списке **Использовать для порога** можно выбрать варианты:

- Значение
- Свойство
- Атрибут *(Добавлено v5.14.20)*

Если выбрано **Значение**, то порог будет константой, значение нужно просто ввести в столбце **Пороговое значение**.

Если нужна возможность индивидуально настроить пороги для каждого экземпляра устройства, можно использовать две другие опции.

Оба варианта позволяют организовать настройку пороговых значений через UI.

- Можно создать специальное свойство-параметр для хранения порога, выбрать вариант **Свойство**, а в графе **Порог свойство** выбрать это свойство
- При выборе варианта **Атрибут** для свойства будет создан атрибут, доступный через #. В правиле больше ничего настраивать не надо.

Например, есть свойство temp1, контролируются нижние границы Lo, LoLo.

Для варианта **Свойство** нужно создать два дополнительных свойства устройства, например (названия произвольные):

- temp1\_lolo
- temp1\_lo

Затем для **LoLo** в графе **Порог свойство** выбрать **temp1\_lolo**, для **Lo** выбрать **temp1\_lo**

Для варианта **Атрибут** никакие свойства создавать не надо.

Если выбрать для **LoLo** и **Lo** опцию **Атрибут**, то основное свойство **temp1** будет иметь атрибуты **temp1#LoLo**, **temp1#Lo**, доступные в привязках визуализации. Значения этих атрибутов будут доступны в РМ для каждого **Устройства** на вкладке **Тревоги** в столбцах LoLo, Lo...

В данном случае названия атрибутов жестко связаны с правилом тревоги.

## Параметр Зона нечувствительности

Параметр **Зона нечувствительности** позволяет избавиться от дребезга при переходе через пороговые значения. Значение параметра - это дельта для создания интервала.

При попадании основного значения в интервал переход на другой уровень тревоги не происходит.

## Тревоги дискретных свойств.

Для значений 0/1 задается уровень:

Например:

- 0 - Норма
- 1 - Тревога (Авария или Предупреждение)

Или:

- 1 - Норма
- 0 - Тревога (Авария или Предупреждение)

## Текст сообщения тревоги

Для каждого уровня можно задать сообщение. Если сообщение не задано - оно будет сформировано автоматически.

Текст сообщения может содержать подстановки значений свойств выбранного устройства, например:

```
Критический уровень ${value#string}. ${name} ${dn}, ${placeStr}
// Будет выведено:
// Критический уровень 2 м. Датчик уровня DL_42, Цех 1 Стойка 17
```

Можно использовать свойства устройства, которые можно привязать на визуализации:

- статические свойства: \${dn}, \${name}, \${placeStr}, \${placePath}
- значения свойств: \${value},...
- строковые представления значений свойств: \${value#string},...

*Добавлено в 5.14.16.*

Чтобы вывести **название** текущего свойства, используется подстановка \${propTitle}

```
${propTitle} ${value#string}! ${name} ${dn}

// Если название свойства – "Значение", будет выведено:
// Значение 2 м! Датчик уровня DL_42

// Если название свойства – "Уровень воды", будет выведено:
// Уровень воды 2 м! Датчик уровня DL_42
```

Для состояния **Норма** также можно задать сообщение, тогда оно будет обработано (запись в журнал, информирование).

Если сообщение не задано, записи не будет

## Сценарий информирования

При срабатывании тревоги есть возможность вызвать сценарий.

Сценарий может сформировать сообщения и отправить их через email/telegram, вывести звуковые оповещения для активных клиентов, короче говоря - выполнить любые действия, доступные в сценарии.

В функцию **start** передается объект тревоги.

Сценарий можно вызывать и при нормализации.

Можно вызывать разные сценарии для разных строк правил (один - для Lo, другой - для Hi, третий - для Norm), а можно - один и тот же.

Сценарий должен быть без startOnChange().

Пример сценария:

При возникновении тревоги выведем объект в консоль.

```
const script = {
  start(aleObj) {
    this.log(JSON.stringify(aleObj));
  }
};
```

Привяжем сценарий к строкам Тревога и Норма дискретного датчика.

В отладчике видим объект, который содержит информацию о тревоге.

При сработке датчика:

```
21.02 11:20:58.196 scen004 Not active
21.02 11:21:07.708 ---Started by alert
21.02 11:21:07.708 LOG: {
  "status":"started",
  "did":"d0141",
  "prop":"state",
  "level":2,
  "aruleId":"Alert",
  "propTitle":"Состояние ",
  "devTitle":"Датчик CO (E_DP_001)",
  "txt":"Повышен уровень CO! Датчик CO (E_DP_001)",
  "tsStart":1679386867706
}
```

При сбросе датчика

```

21.03 11:24:08.021 ---Started by alert
21.03 11:21:17.725 LOG: {
  "status":"stopped",
  "did":"d0141",
  "prop":"state",
  "level":2,
  "aruleId":"Alert",
  "propTitle":"Состояние ",
  "devTitle":"Датчик дыма (E_DP_001)",
  "txt":"Нет задымления Датчик дыма (E_DP_001)",
  "tsStart":1679386867706,
  "tsStop":1679386877724
}

```

Объект содержит:

- "status": статус алерта - "started" или "stopped"
- "did": идентификатор устройства - "d0141"
- "devTitle": название устройства - "Датчик СО (D\_001)",
- "prop": имя свойства - "state"
- "propTitle": название свойства - "Состояние"
- "level": уровень тревоги (1-предупреждение, 2-авария)
- "aruleId": идентификатор тревоги ("LoLo","Lo","HiHi","Hi" - для аналоговых, "Alert"-для дискретных значений)
- "txt": сообщение из журнала тревог - "Повышен уровень СО! Датчик СО (DP\_001)"
- "tsStart": время возникновения - таймштамп
- "tsStop": время нормализации (для stopped) - таймштамп

Если нужны дополнительные данные устройства, можно получить их через **this.getDevice(aleObj.did)**

Пример сценария: При возникновении тревоги выдадим звуковое оповещение 'ding.wav', при нормализации - 'chimes.wav'. Также предусмотрим ситуацию, что датчик может быть заблокирован, и тогда звук выдавать не будем.

```
const script = {
  start(aleObj) {
    if (aleObj) {
      // устройство, которое сгенерировало тревогу
      const dev = this.getDevice(aleObj.did);
      if (dev.blk) {
        this.log(dev.name+' блокирован, звук не выдается');
      } else {
        const file = aleObj.status == 'stopped' ? 'chimes.wav' : 'ding.wav';
        this.info('sound', 'admin', file);
      }
    }
  }
};
```

В сценарии информирования не рекомендуется использование таймеров, слушателей и других асинхронных механизмов.

Причина - может пропуститься очередной вызов по событию, так как сценарий остался активен.

## Тревоги в UI

### Журналы тревог

[Журналы тревог](#) - это оперативные журналы, содержащие информацию о текущих тревогах.

Используя фильтры, можно создать журналы по отдельному устройству, объекту, группе устройств или в целом по проекту.

Журнал тревог имеет механизм квитирования, позволяющий квитировать отдельные тревоги.

### Системные индикаторы

Иногда бывает удобно не выводить журнал тревог постоянно на экран.

Для каждого журнала система автоматически создает служебное устройство - Системный индикатор журнала тревог, свойства которого содержат информацию о тревогах журнала:

- all - Количество всех записей в журнале
- active\_all - Количество активных записей (Активные предупреждения + Активные аварии)
- active\_w - Количество активных предупреждений
- active\_a - Количество активных аварий
- needack\_all - Количество всех записей, ожидающих квитирования
- needack\_w - Количество предупреждений, ожидающих квитирования
- needack\_a - Количество аварий, ожидающих квитирования

Работать с системным индикатором можно, как и с любым другим устройством - выводить значения свойств на визуализацию, привязывать к свойствам графических элементов, ...

Например, для объектов создать журналы тревог, а на общей схеме или карте подсвечивать индикатор объекта, на котором есть активные тревоги.

## Дополнительные атрибуты

Свойства устройства в системе могут иметь дополнительные атрибуты, которые можно использовать на визуализации.

Такие атрибуты доступны как **<имя свойства>#<имя атрибута>**. Если для свойства генерируется тревога, создается 4 атрибута: #ack, #alert, #alertmessage, #blkalert

- <имя свойства>#ack - Флаг квитирования тревоги. Значение 0/1
- <имя свойства>#alert - Уровень тревоги. Значения от -2 до 2 (LoLo/Lo/Norm/Hi/HiHi)
- <имя свойства>#alertmessage - Тревожное сообщение
- <имя свойства>#blkalert - Блокировка тревоги. Значение 0/1
- <имя свойства>#LoLo - Значение границы LoLo
- <имя свойства>#Lo - Значение границы Lo
- <имя свойства>#Hi - Значение границы Hi
- <имя свойства>#HiHi - Значение границы HiHi

Первые три атрибута только для чтения, а блокировку можно не только вывести, но и дать изменять интерактивно, с помощью, например, элемента checkbox.

Также есть возможность вызвать **команду квитирования** (#hexack), которая доступна, когда идет выбор Команд устройства:

```
off
    Выключить
on
    Включить
temp#hexack
    Температура приточки: Выполнить квитирование тревоги
```

## Квитировать ВСЁ

Выполнить квитирование всех алертов журнала можно в РМ -> Журналы тревог. Также можно использовать команду [скрипта визуализации](#) **core.ackAllAlerts(<ID журнала тревог>)**

## Доступ к свойствам

### Чтение

Все свойства, объявленные для устройства, доступны **для чтения**:

- основные свойства типа устройства;
- свойства, определяемые сценарием;
- системное свойство **error**.

Все свойства устройства можно увидеть на вкладке "Свойства".

Примеры:

- TEMP1.value
- DIMM1.setpoint
- LAMP1.timeoff
- TEMP2.error

Так же есть статические свойства, которые создаются вместе с устройством и не могут быть изменены:

Название	Описание
<b>TEMP1.name</b>	Имя экземпляра устройства
<b>DIMM1.dn</b>	ID экземпляра устройства
<b>TEMP1.placeStr</b>	Расположение устройства в дереве в виде строки например: Уровень-1 Уровень-2
<b>TEMP1.placePath</b>	Расположение устройства в дереве в виде полного пути например: Уровень-1 / Уровень-2
<b>TEMP1.userAccess</b>	Наличие доступа пользователя к свойству

У свойств типа DATA, PARAMETER, EVENT есть дополнительные атрибуты, которые можно вывести на визуализацию, они обозначаются после символа "#":

Название	Описание
<b>TEMP1.value#propTitle</b>	Название свойства
<b>TEMP1.value#mu</b>	Единица измерения
<b>TEMP1.value#changetime</b>	Время когда данные изменились
<b>TEMP1.value#refreshtime</b>	Время поступления данных
<b>TEMP1.value#string</b>	Строковое представление значения, активируется только в случае использования обработчика значения в виде строки

Для числовых свойств:

Название	Описание
<b>TEMP1.value#min</b>	Атрибут минимального значения свойства
<b>TEMP1.value#max</b>	Атрибут максимального значения свойства
<b>TEMP1.value#dig</b>	Число знаков после запятой

Для свойств, привязанных к каналам:

Название	Описание
<b>TEMP1.value#chname</b>	Имя канала в плагине
<b>TEMP1.value#chstatus</b>	Состояние (>0 - ошибка) канала
<b>TEMP1.value#chstatuschangetime</b>	Время изменения статуса канала
<b>TEMP1.value#blkchannel</b>	Флаг блокировки канала (1 - канал заблокирован)

Для свойств, с настроенными тревогами:



Название	Описание
<b>TEMP1.value#LoLo</b>	Уставка Нижней Аварийной границы
<b>TEMP1.value#Lo</b>	Уставка Нижней Предупредительной границы
<b>TEMP1.value#Hi</b>	Уставка Верхней Предупредительной границы
<b>TEMP1.value#HiHi</b>	Уставка Верхней Аварийной границы
<b>TEMP1.value#ack</b>	Число тревог, ожидающих квитирования
<b>TEMP1.value#alert</b>	Уровень тревоги. Значения от -2 до 2 (LoLo/Lo/Norm/Hi/HiHi)
<b>TEMP1.value#alertmessage</b>	Тревожное сообщение
<b>TEMP1.value#blkalert</b>	Блокировка тревоги.
<b>TEMP1.value#execack#userAccess</b>	Наличие доступа пользователя к команде квитирования тревоги.
<b>TEMP1.value#execack</b>	Команда квитирования тревоги.

## Запись

Операция **запись** выполняет изменение свойств устройства (запись значения или отправку команды).

Обратите внимание, что нельзя просто написать `TEMP1.setpoint = 42`.

Причина: любое изменение свойства - это процесс.

- Если свойство связано с каналом, происходит отправка команды на запись. Результат придет при выполнении операции чтения.  
Полученное значение проходит через обработчик входного значения, и только в этот момент мы получаем реальный результат.
- Если свойство не связано с железом, все еще работает механизм обработчиков и значение может быть отвергнуто или преобразовано

## Запись в свойство

**setValue(<свойство>, <значение>)**

```
TEMP1.setValue('setpoint', 42); // Уставка датчика
DIMM1.setValue('value', 100); // Яркость диммера
```

Используется для свойств типа Data и Parameter.

Если свойство связано с каналом, канал должен иметь возможность чтения и записи.

Для свойств типа **Command** и **Calculate** команда **setValue** будет проигнорирована.

## Отправка команды.

Команда - это свойство, которое имеет только возможность записи.

```
DIMM1.on(); // Включить диммер
METER1.clear(); // Сбросить счетчик
```

Любую команду устройства можно выполнить с интерфейса в рамках шаблона устройства либо привязать к клику на кнопке.

## Ошибка устройства

Любое устройство имеет системное свойство **error** типа Bool (0-нет ошибки, 1-есть ошибка).

Установка и сброс этого свойства происходит автоматически исходя из наличия ошибок отдельных свойств. У каждого свойства есть свое текстовое поле "Ошибка". Ошибки фиксируются в журнале устройства.

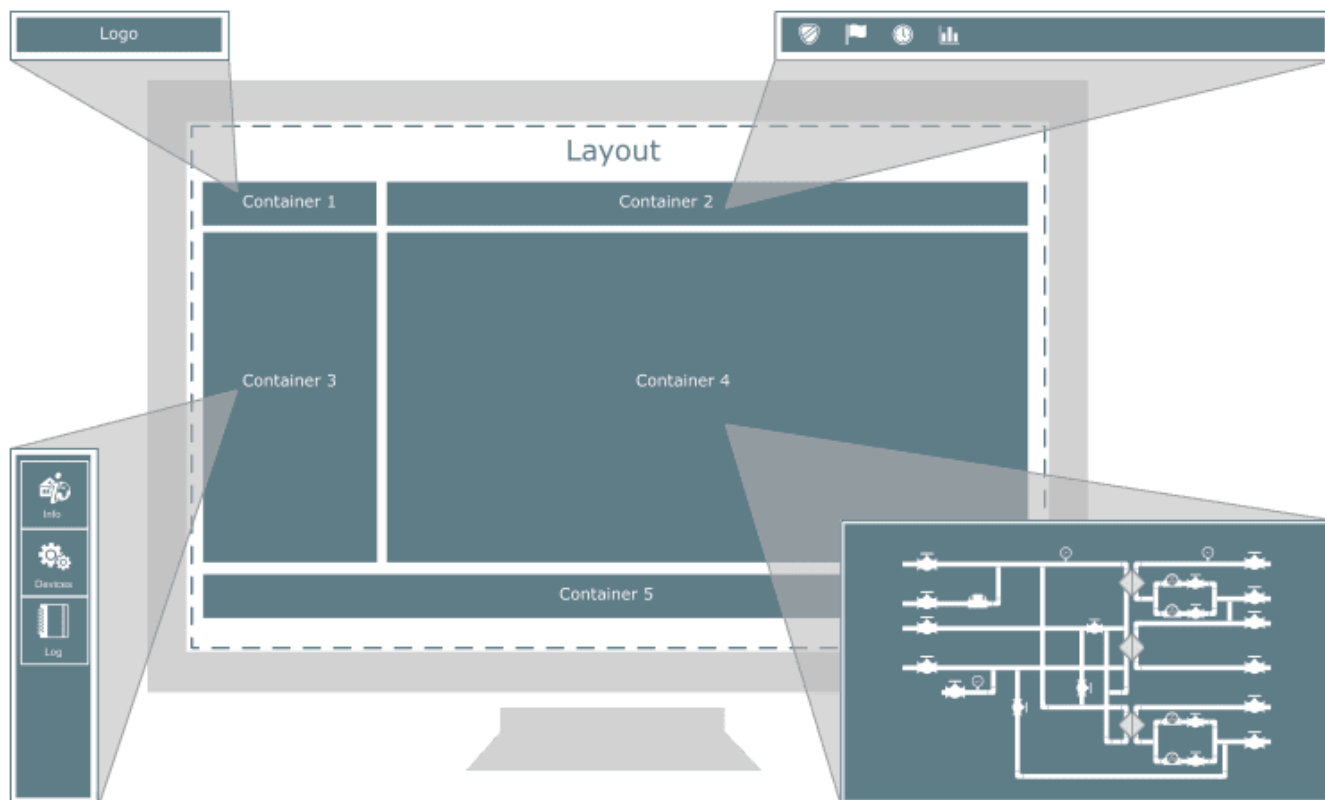
**Обработчик** может установить ошибку при приеме данных. В этом случае возвращается объект с полями:

- error: 'Текст ошибки'
- value: значение (опционально)

```
function(device, prop, value) {
  return value == 0 || value == 1
    ? Number(value)
    : {
      error: 'Допустимые значения 1/0. Получено ' + value
    };
}
```

# Визуализация

В разделе визуализации можно создавать весьма эффективные интерфейсы.



## Обратите внимание!

При работе с экранами, контейнерами, диалогами и шаблонами часто требуется возможность масштабирования и перемещения всего элемента в целом.

Для этого используются следующие [горячие клавиши](#)

Составные части визуализации:

**Экраны** - основной элемент визуализации.

На экранах размещаются Контейнеры и различные элементы (кнопки, тексты, изображения, прямоугольники ....).

### Контейнеры

В контейнерах размещаются Шаблоны визуализации и различные элементы.

**Диалоги** Специальные информационные роруп окна. Вызываются командой Show Dialog

### Шаблоны визуализации

Шаблоны визуализации размещаются в контейнерах для визуализации однотипных устройств.

Используются для создания большого количества устройств с одинаковыми свойствами.

**Элементы** - это те компоненты, из которых создаются шаблоны, диалоги, контейнеры и экраны. Текст, изображение, кнопка, слайдер, поле ввода, график - все это элементы.

### **Изображения**

Библиотека различных изображений.

## Экраны

Экраны - базовые элементы визуализации.

Экраны и контейнеры можно сгруппировать по месту нахождения или по функционалу.

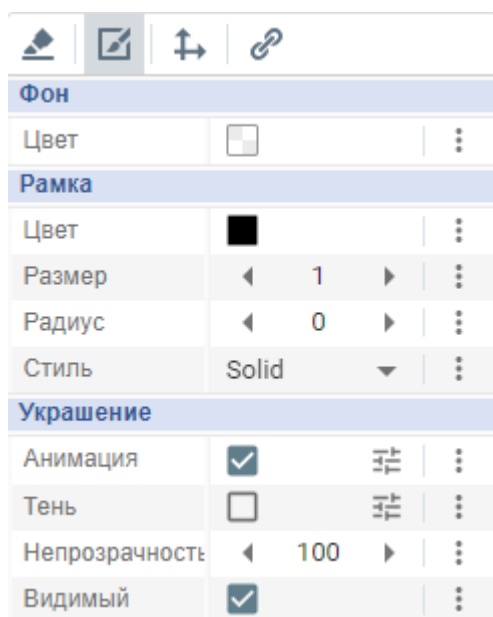
В левой части экрана вы видите дерево экранов и контейнеров.

Правой кнопкой мыши можно создавать новые папки и новые экраны и контейнеры.

Методом drag&drop можно передвигать папки и их содержимое.

Правой кнопкой мыши в поле редактора экрана можно добавить на текущий экран контейнер или отдельные элементы (кнопка, изображение, текст ...).

В правом меню редактора можно настроить свойства элемента или контейнера, такие как цвет, фон, размер, граница, анимация и т.д.



## Контейнеры

В [контейнерах](#) размещаются шаблоны визуализации и отдельные элементы. Все элементы добавляются правой кнопкой мыши.

Визуализация устройств может быть выполнена двумя способами: прямой привязкой любого элемента контейнера к свойству устройства или используя шаблоны.

## Шаблоны визуализации

[Шаблоны визуализации](#) - это элементы интерфейса, размещаемые в контейнерах для визуализации однотипных устройств. Используются для создания элементов управления большого количества устройств с одинаковыми свойствами. Можно создать один Шаблон визуализации, разместить необходимое количество этих шаблонов на мнемосхеме и привязать каждый отдельный шаблон к каналам различных устройств.

Для создания нового шаблона кликните правой кнопкой мыши по дереву слева и выберите «Новый шаблон». На вкладке «Редактор» кликнув правой кнопкой мыши по полю шаблона можно создать такие элементы как: квадрат, круг, текст, изображение и текст с изображением.

### Вкладки правого меню редактора



### Слои

Создав элемент можно изменить все его параметры в правом меню: цвет, размер, выравнивание, фон, граница, заливка, прозрачность, тень, анимация и т.д. Слои можно объединить в группы удерживая клавишу shift.

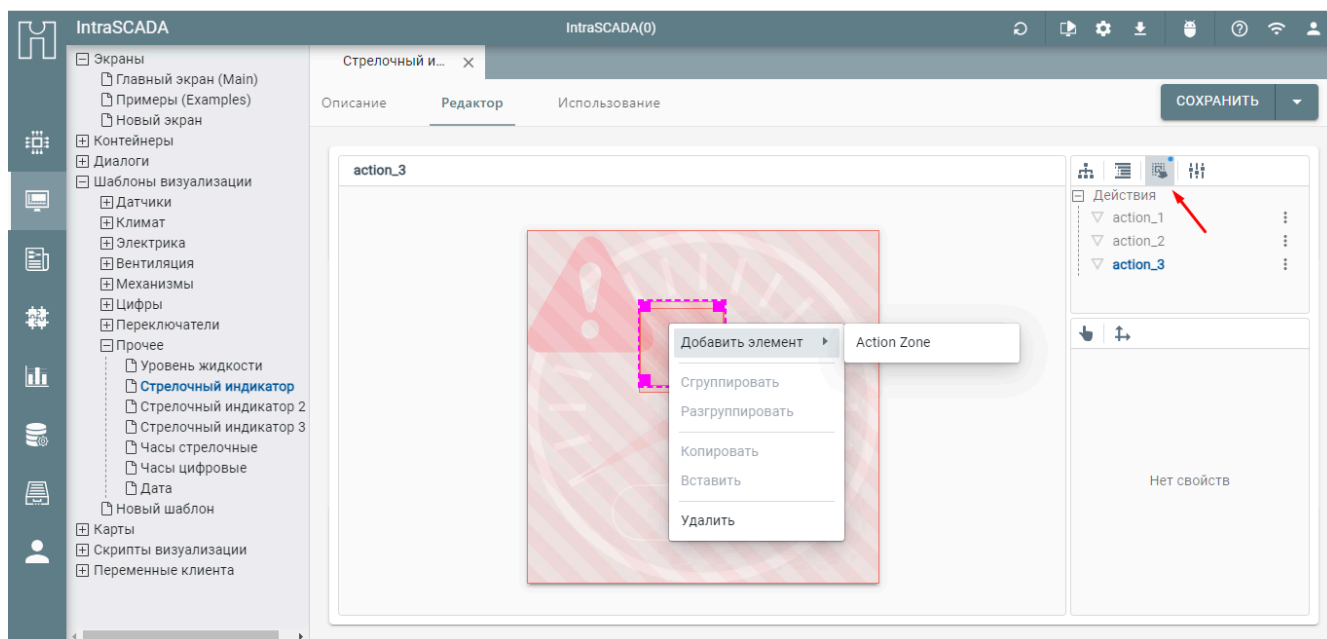
### Состояния

Для каждого состояния устройства (вкл/выкл/ошибка/автомат и т.д.) все параметры визуализации могут быть изменены - от изменения цвета надписи до замены изображения или включения анимации. По умолчанию создается две переменных состояния - state и error. Может быть добавлено неограниченное количество переменных. Пример: Шаблон светильника, включающегося по датчику движения может быть белым в выключенном состоянии и желтым во включенном. Для этого в state 0 выбираем цвет изображения - белый, а для state 1 - желтый. Также, наш светильник может находиться в автоматическом режиме. Для отображения автоматического режима добавим переменную auto и в состоянии 1 добавим надпись «АВТО» на шаблоне.

### Области действий

Для задания области, в которой будет срабатывать клик нажмите правой кнопкой мыши в поле редактора и выберите Добавить элемент -> Action Zone. Для большинства устройств область действия может быть задана во весь размер шаблона. Но иногда может быть полезно создание нескольких активных областей для управления разными каналами одного устройства.

Действия могут быть заданы для следующих событий для левой клавиши мыши (касания тачскрина): одинарный клик, двойной клик, долгий клик, нажатие клавиши мыши, отпускание клавиши мыши; для правой клавиши: одинарный клик.

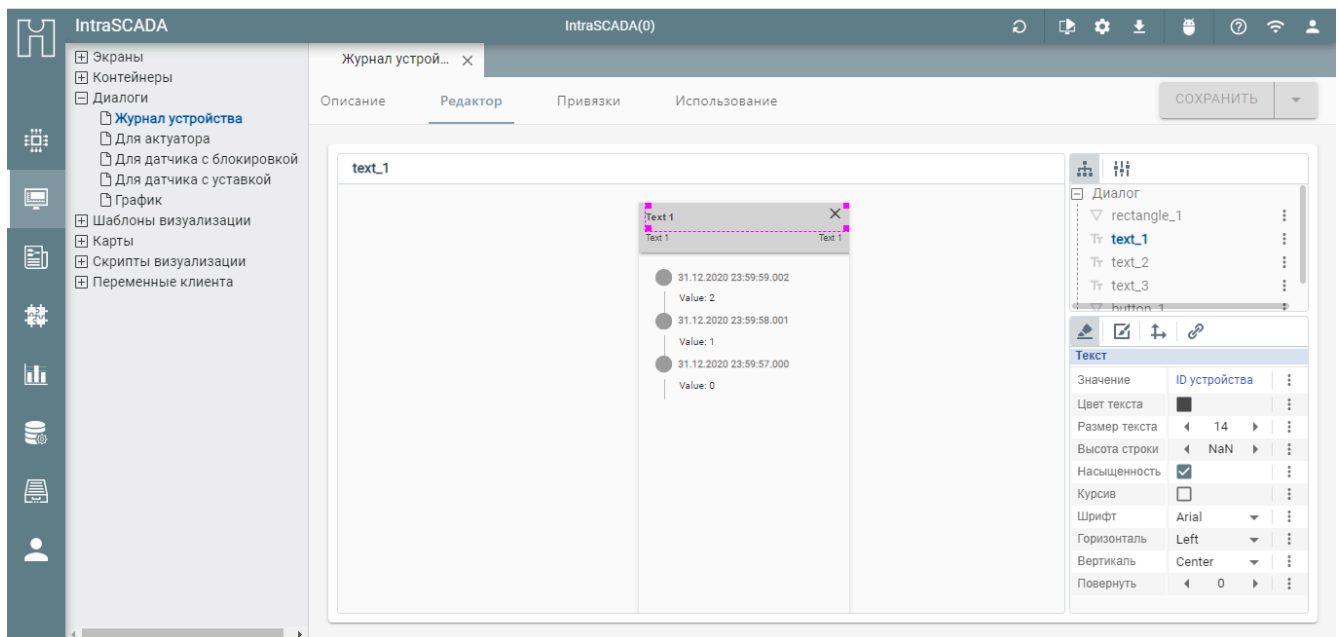


### Настройки экрана

В последнем меню редактора можно настроить размер сетки редактора, масштаб и цвет фона.

## Диалоги

Создание и редактирование диалогов.

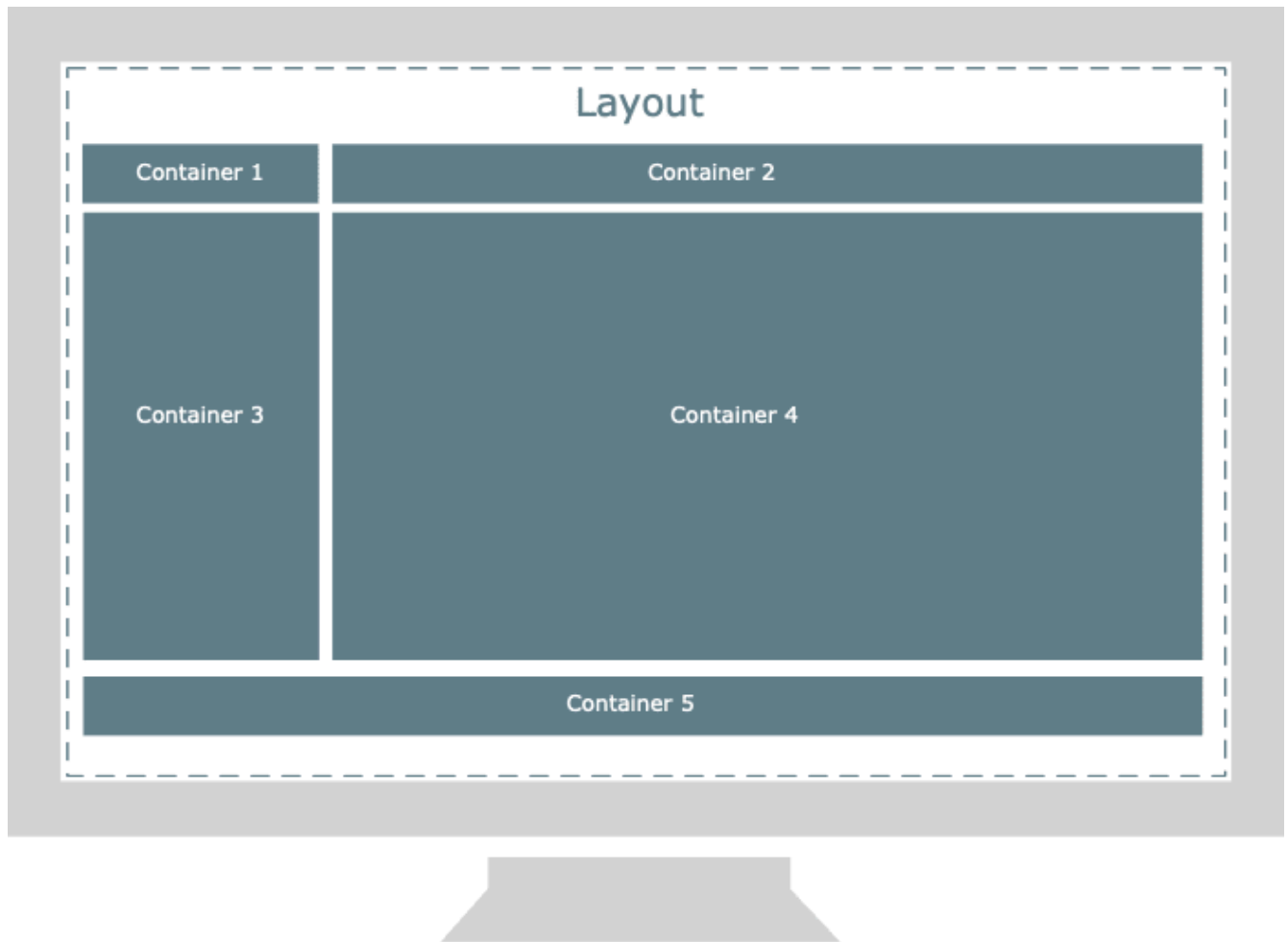


## Изображения

В системе используются следующие форматы [изображений](#): svg, png, jpg, gif. Правой кнопкой мыши в дереве можно добавить / удалить изображение, папку или экспортировать изображение на ПК.

# Экраны

Экран (Layout) — главный компонент визуализации.



На Экране выполняется размещение различных контейнеров и элементов.

Количество экранов не ограничено. Можно создавать экраны для разных устройств с разным разрешением (смартфоны, планшеты, компьютеры)

Хотя можно разместить элемент прямо на Экране, для компоновки элементов рекомендуется использовать контейнеры.

Элементы, не добавленные в контейнер, будут растягиваться или сжиматься в зависимости от разрешения клиентского устройства

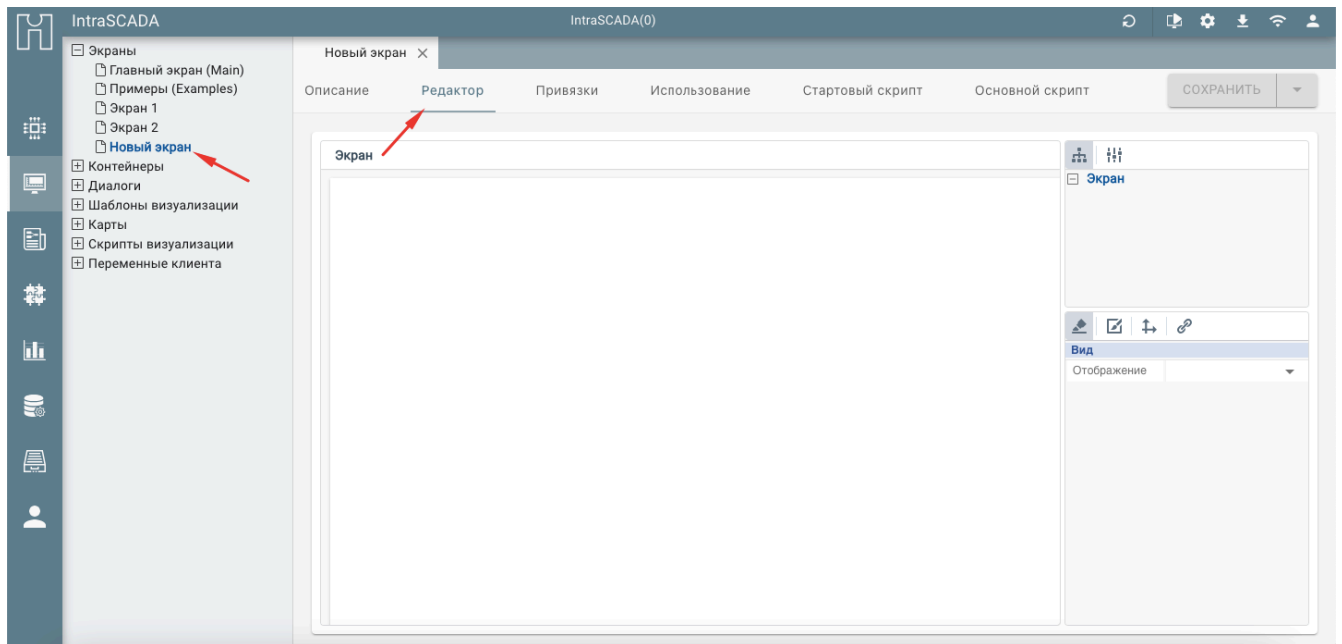
## Новый экран

Правой кнопкой мыши в дереве экранов создаем новый экран и, при необходимости, методом drag&drop перемещаем его в нужную папку. Папки предназначены для группировки экранов по месту нахождения или по функционалу.

На вкладке "Описание" можно переименовать экран и дать необходимые комментарии.



Вся работа с экраном ведется на вкладке "Редактор":

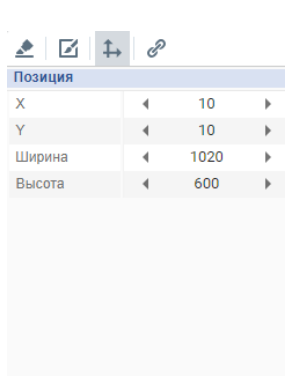


## Настройка экрана

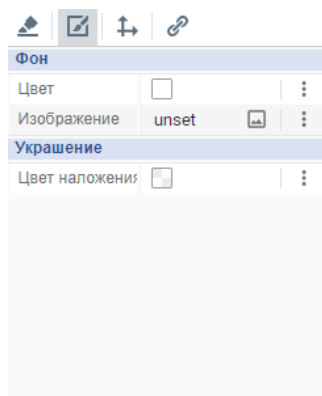


Выбрав экран в дереве справа, можно настроить следующие параметры:

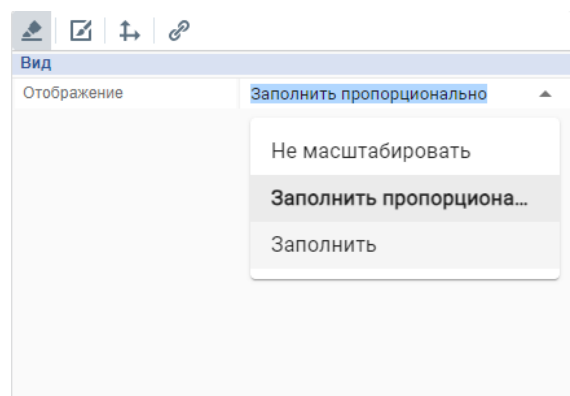
### Размеры экрана



### Оформление экрана



### Отображение (заполнение экрана)



## Оформление

Здесь можно задать цвет или Изображение для фона.

Запись unset означает, что изображение не выбрано.

Для выбора нажмите на иконку справа от unset, в диалоговом окне будут доступны все изображения проекта.

#### 📌 На заметку

В поле Изображение, вместо записи unset, можно ввести адрес изображения из сети интернет. Это дает возможность попробовать изображение, не загружая его в проект. Если результат вас устраивает, можно скачать файл и загрузить его в разделе [Ресурсы->Изображения](#).

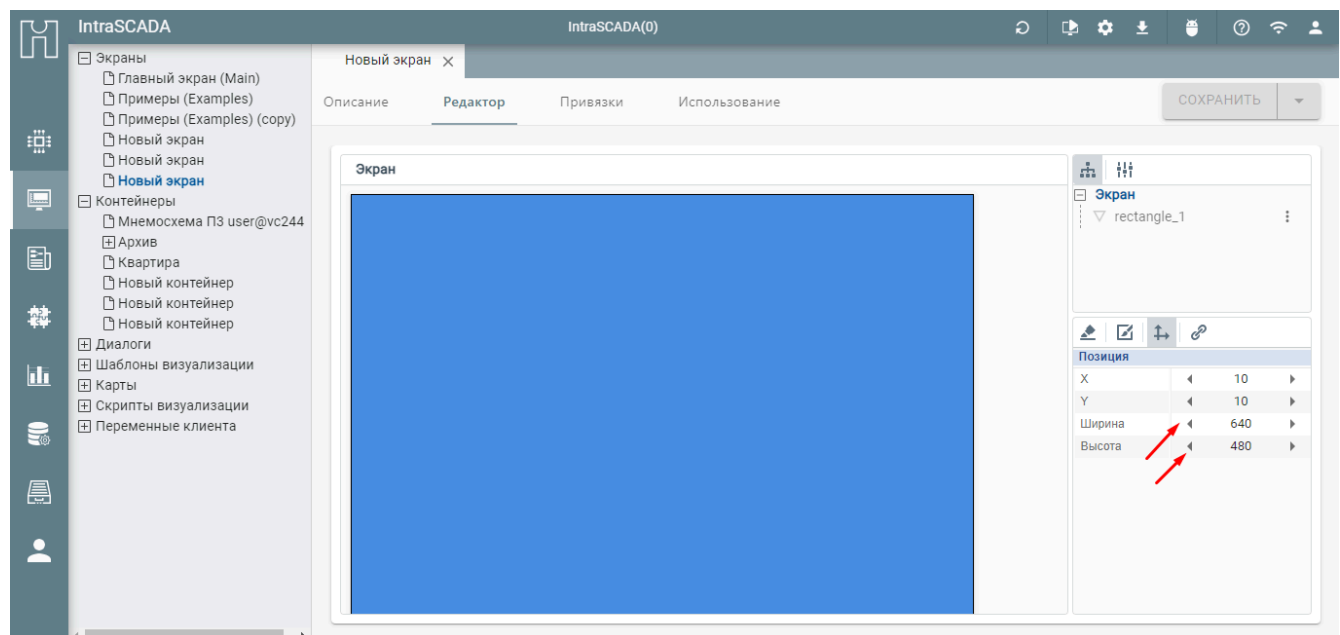
## Отображение

Этот важный параметр определяет, как будет отображаться экран на устройствах с различным разрешением.

Возможные опции:

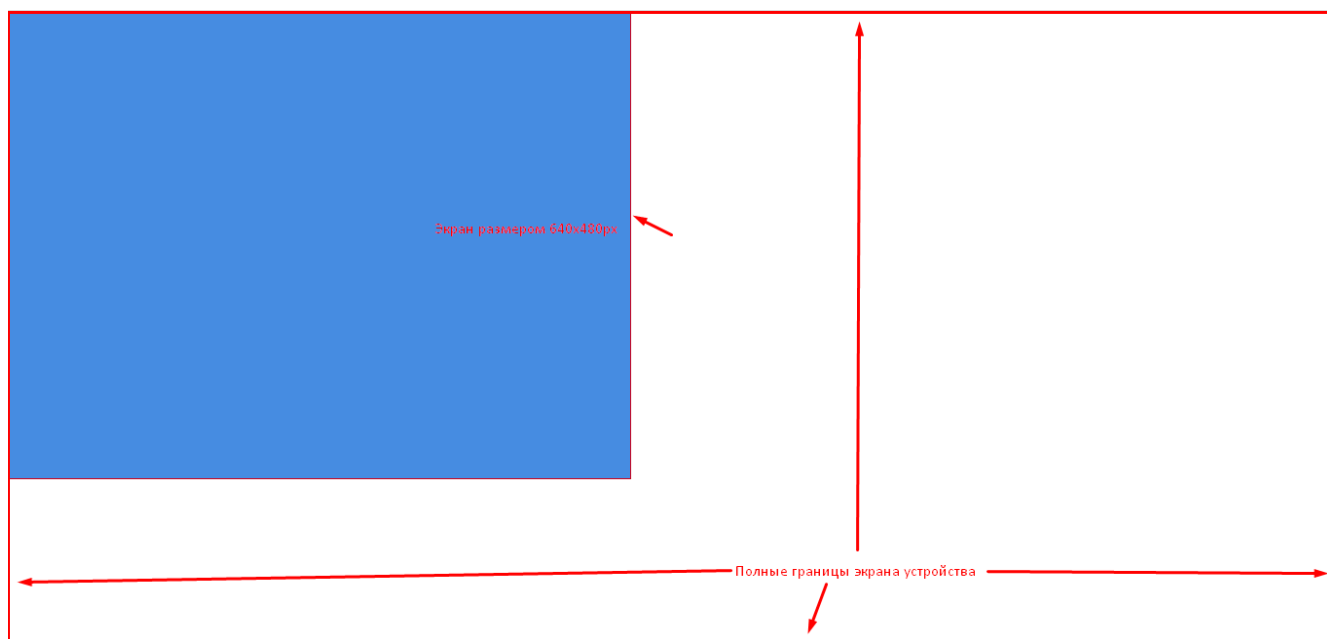
- **Не масштабировать** Экран будет отображаться в соответствии с заданными размерами в масштабе один к одному
- **Заполнить пропорционально** Экран будет заполнен (растянут) в соответствии с разрешением вашего устройства, при этом сохраняя заданные пропорции экрана
- **Заполнить** Экран будет заполнен в соответствии с разрешением вашего устройства как по горизонтали, так и по вертикали

Например, создадим экран размером 640x480 пикселей



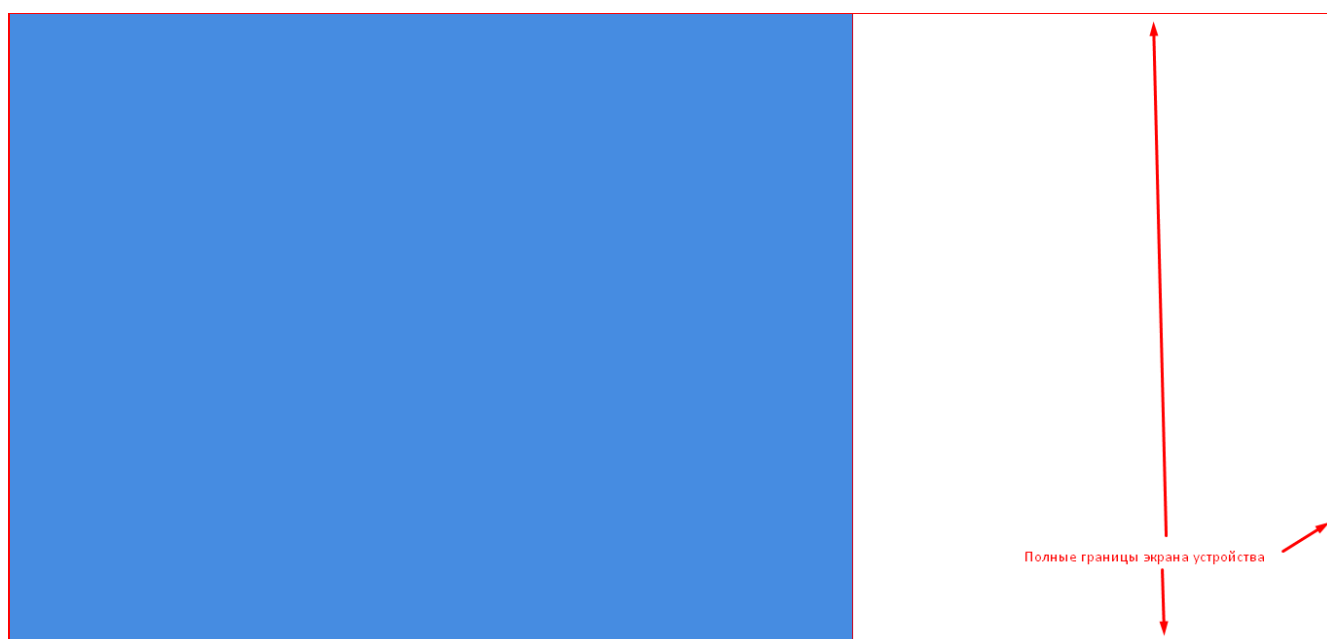
Будем переключать параметр **Отображение экрана**

### Не масштабировать



Экран отображается в соответствии с заданным разрешением в масштабе один к одному

### Заполнить пропорционально



Экран растянулся пропорционально заданному разрешению, при этом элементы, добавленные в экран, не искажаются (например, квадрат не станет прямоугольником).

### Заполнить



Экран заполнился в соответствии с разрешением экрана вашего устройства.

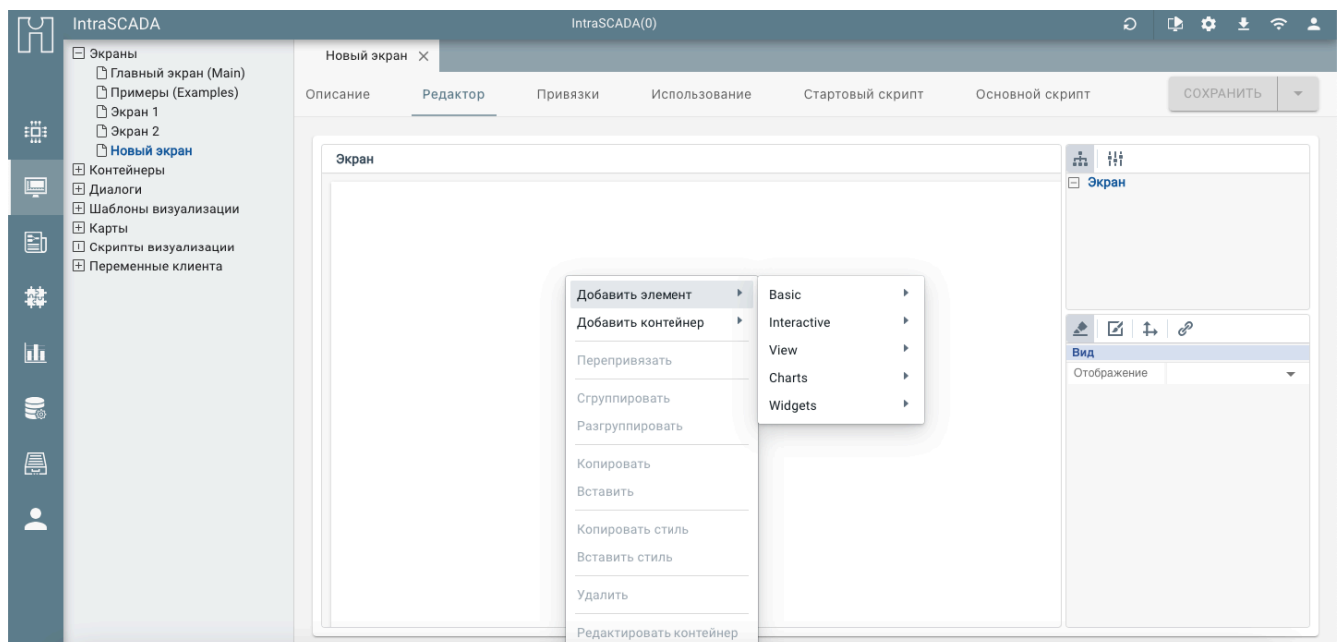


На заметку

Отображение (Заполнение экрана) изначально выставлено на Заполнить

## Наполнение экрана

По клику правой кнопкой мыши в области экрана вызывается меню:



**Добавить Элемент** - добавить на экран отдельные **элементы**

В правом меню редактора можно настроить **общие** и индивидуальные свойства элемента, такие как цвет, фон, размер, граница, анимация и т.д.

<b>Фон</b>			
Цвет			⋮
<b>Рамка</b>			
Цвет			⋮
Размер	◀ 1 ▶		⋮
Радиус	◀ 0 ▶		⋮
Стиль	Solid ▼		⋮
<b>Украшение</b>			
Анимация	<input type="checkbox"/>	⌵ ⌶	⋮
Тень	<input type="checkbox"/>	⌵ ⌶	⋮
Непрозрачность	◀ 100 ▶		⋮
Видимый	<input checked="" type="checkbox"/>		⋮

**Добавить Контейнер** - добавить контейнеры с установленными элементами и шаблонами визуализации. Если список контейнеров пустой, значит вы не создали еще ни одного контейнера.

В правом меню редактора можно настроить свойства контейнера:

Вид		Украсение	
Масштабирование	▼	Анимация	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<b>Выравнивание контента</b>		Тень	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
По горизонтали	Center ▼	Непрозрачность	◀ 100 ▶ <input type="checkbox"/>
По вертикали	Center ▼	Видимый	<input checked="" type="checkbox"/> <input type="checkbox"/>
<b>Заполнение контента</b>			
Ширина	<input checked="" type="checkbox"/>		
Высота	<input checked="" type="checkbox"/>		
<b>Прокрутка контента</b>			
Горизонтальный	<input type="checkbox"/>		
Вертикальный	<input type="checkbox"/>		

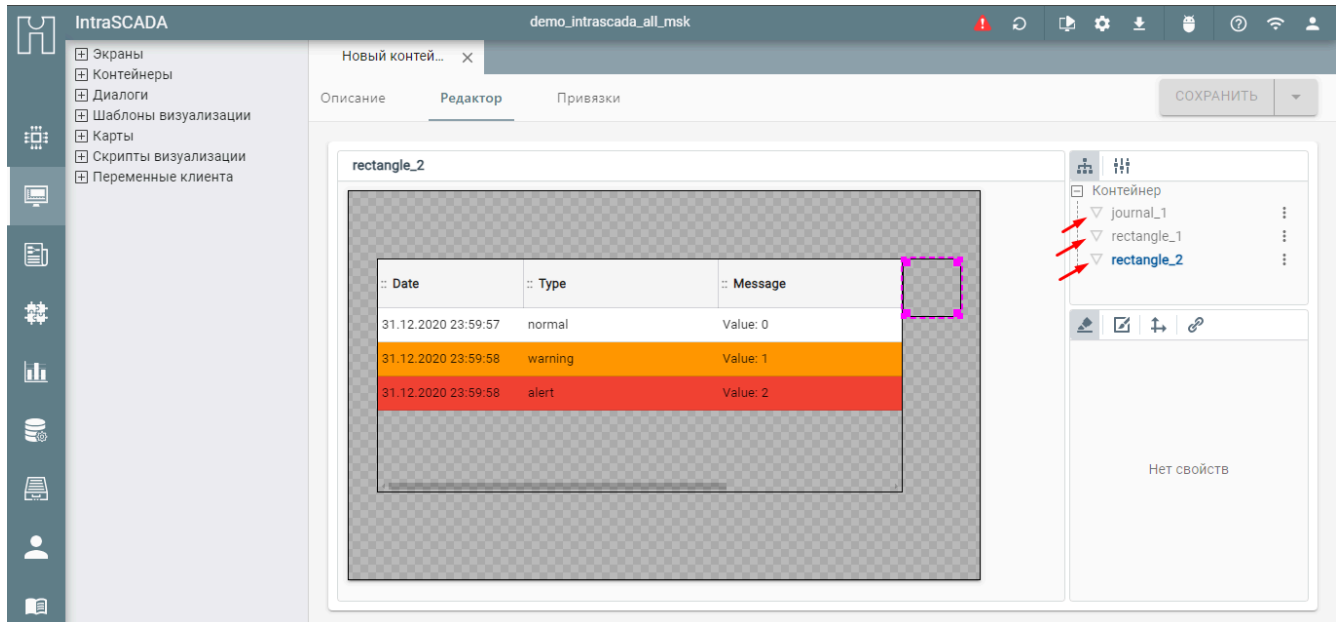
Позиция	
X	◀ 40 ▶ <input type="checkbox"/>
Y	◀ 40 ▶ <input type="checkbox"/>
Ширина	◀ 980 ▶ <input type="checkbox"/>
Высота	◀ 560 ▶ <input type="checkbox"/>
Положение эле	◀ 100 ▶ <input type="checkbox"/>
Обрезать	<input checked="" type="checkbox"/> <input type="checkbox"/>

## Свойства контейнера

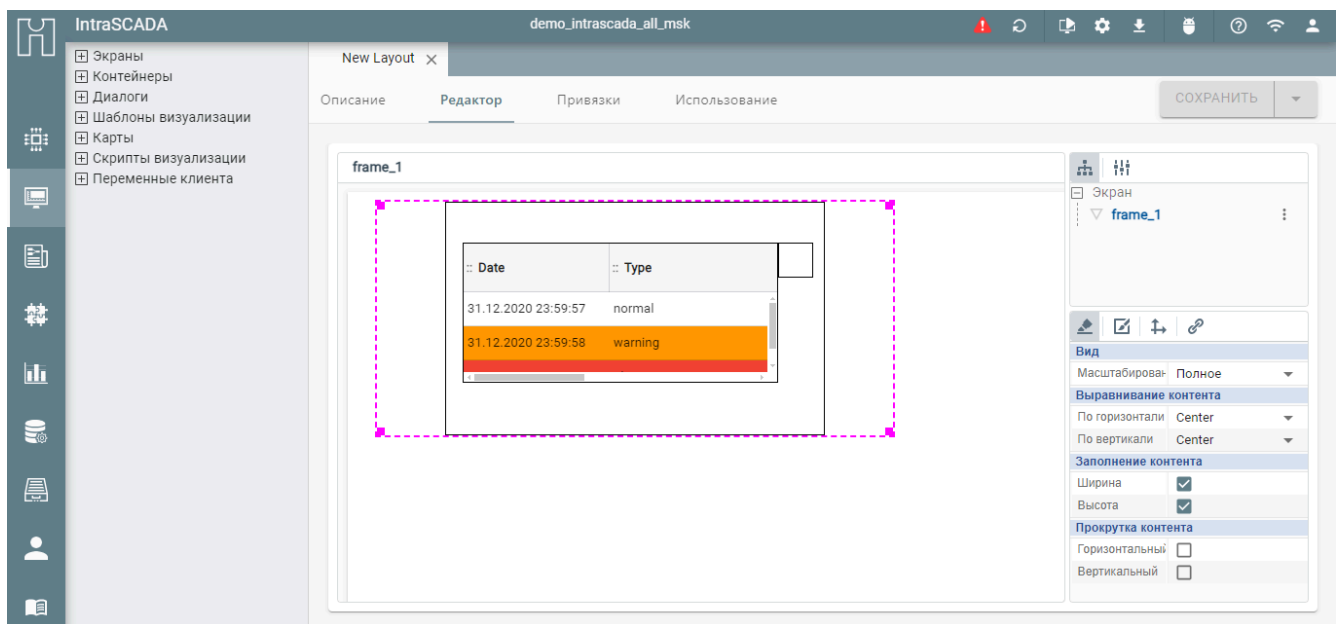
### Масштабирование

Пример :

- Создадим контейнер, добавим в него элементы journal и rectangle



- Привяжем элемент journal к любому существующему журналу
- Добавим контейнер на экран



Базовое масштабирование масштабирует элементы внутри только по параметрам X, Y, Ширина, Высота. Полное масштабирование масштабирует всё, включая текста, текста журналов и т.д.

Так будет выглядеть на экране базовое масштабирование:

Локация	Метки	Уров...
ЦОД/Серверная		Стан...
Boiler rooms	ВМК	Авар
Boiler rooms	ВМК	Пред

А так будет выглядеть полное масштабирование:

Локация	Метки	Урове...	Дата
ЦОД/Серверная		Предупрежд...	05.04.2024 11:35:00
Boiler rooms	ВМК	Предупрежд...	05.04.2024 11:34:54
Boiler rooms	ВМК	Предупрежд...	05.04.2024 11:34:49
Boiler rooms	ВМК	Авария	05.04.2024 11:34:44
ЦОД/Серверная		Стандартное...	05.04.2024 11:34:42

## Выравнивание и заполнение контента

Выравнивание контента работает в том случае, если снята хотя бы одна галка заполнения контента. Если выбрано заполнение контента только по ширине, то заполнение по высоте не учитывается и наоборот.

- Заполнение по ширине, выравнивание по центру:

Локация	Метки	Урове...	Дата
ЦОД/Серверная		Стандартное...	05.04.2024 11:41:31
Boiler rooms	ВМК	Предупрежд...	05.04.2024 11:41:25
ЦОД/Серверная		Предупрежд...	05.04.2024 11:41:17
Boiler rooms	ВМК	Предупрежд...	05.04.2024 11:41:15
Boiler rooms	ВМК	Предупрежд...	05.04.2024 11:41:05
ЦОД/Серверная		Авария	05.04.2024 11:40:55



- Заполнение по высоте, выравнивание по центру:

Локация	Метки	Уровни сообщений
Boiler rooms	ВМК	Предупреждение
Boiler rooms	ВМК	Авария
Boiler rooms	ВМК	Предупреждение

## Прокрутка контента

Прокрутка(скроллинг) контента применяется в том случае, если контент в контейнере не помещается на экране. Доступна вертикальная и горизонтальная прокрутка. В том случае, если обе галки прокрутки отключены, контент прокручиваться не будет.

Локация	Метки	Уровни сообщений
Boiler rooms	ВМК	Стандартное
ЦОД/Серверная		Предупреждение
Boiler rooms	ВМК	Предупреждение
Boiler rooms	ВМК	Авария

## Запуск скриптов до и после отрисовки экрана

### Стартовый скрипт

- Скрипт запускается перед загрузкой экрана
- Цель скрипта - присвоить значения переменным клиента, которые связаны с экраном
- Не предназначен для длительных и интерактивных команд, так как экран еще не загружен
- Скрипт должен вернуть объект, содержащий значения изменяемых переменных клиента

### Основной скрипт

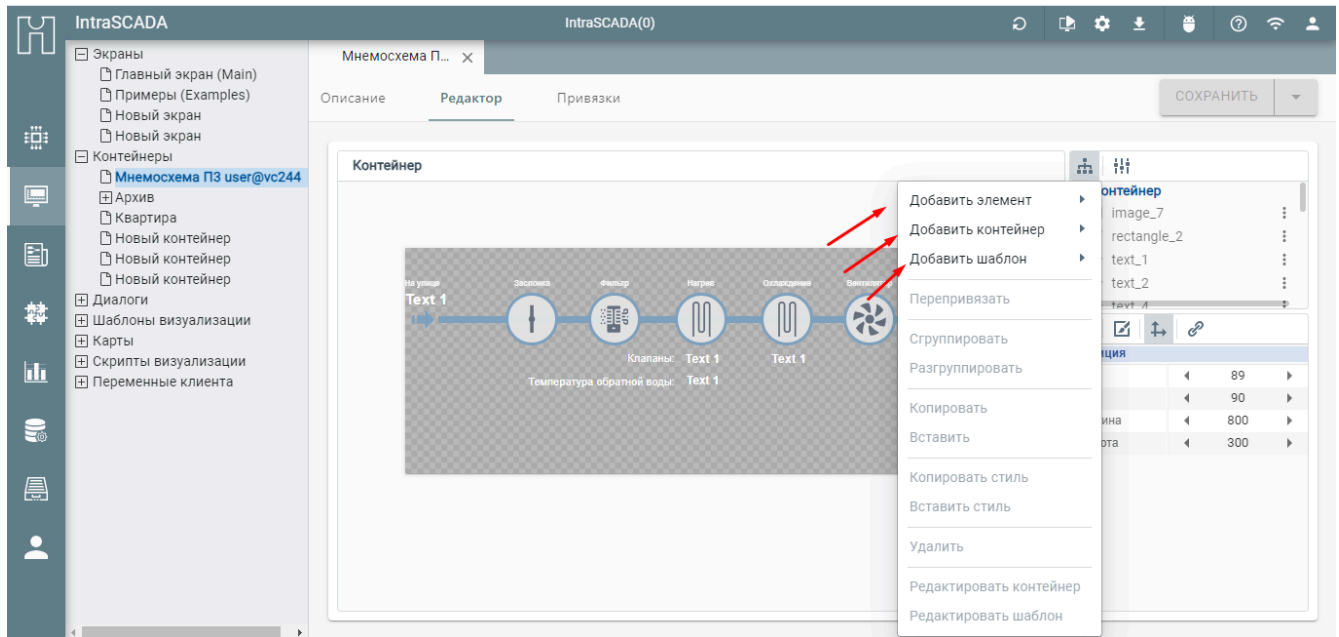
- Скрипт запускается после загрузки экрана
- Можно использовать любые функции скрипта визуализации
- Для обновления значений переменных клиента используйте `updateLocals({})`
- Скрипт может ничего не возвращать

# Контейнеры



В контейнерах размещаются шаблоны визуализации и отдельные элементы. Также есть возможность добавить контейнер внутрь уже готового контейнера.

Все элементы, шаблоны и контейнеры добавляются правой кнопкой мыши.



**Обратите внимание!** Публикации на экраны желательно выполнять посредством контейнеров, так как элементы, не добавленные в контейнер, имеют свойство растягиваться или сжиматься на главном экране в зависимости от разрешения

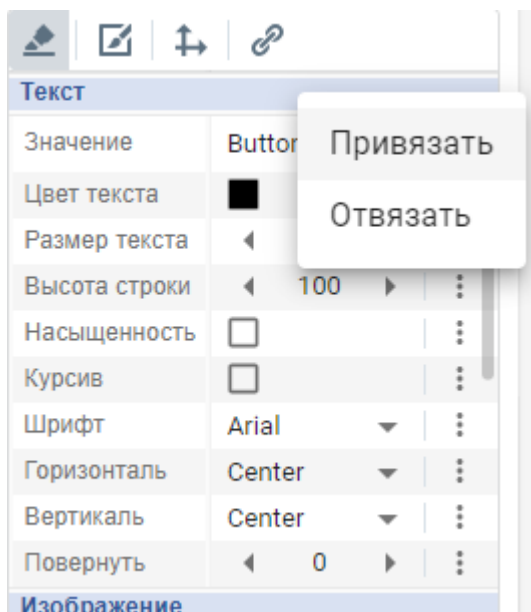
Визуализация устройств может быть выполнена двумя способами: прямой привязкой любого элемента контейнера к свойству устройства или используя шаблоны.

#### Способ 1. Прямая привязка любого элемента (кнопки, изображения, текста) к свойству устройства

Правой кнопкой мыши разместите в поле контейнера элемент: квадрат, круг, текст, изображение, текст с изображением, кнопку или график.

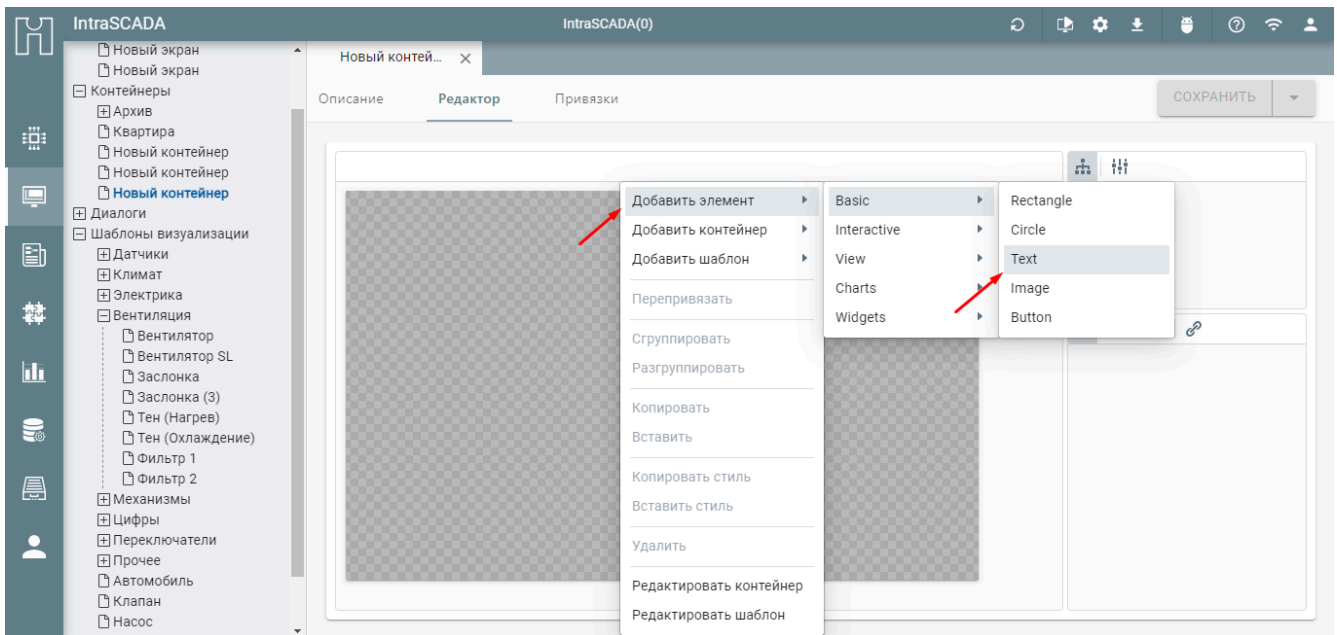
Свойства всех элементов настраиваются в правом меню редактора.

Любое из свойств элемента может быть привязано к каналу устройства:

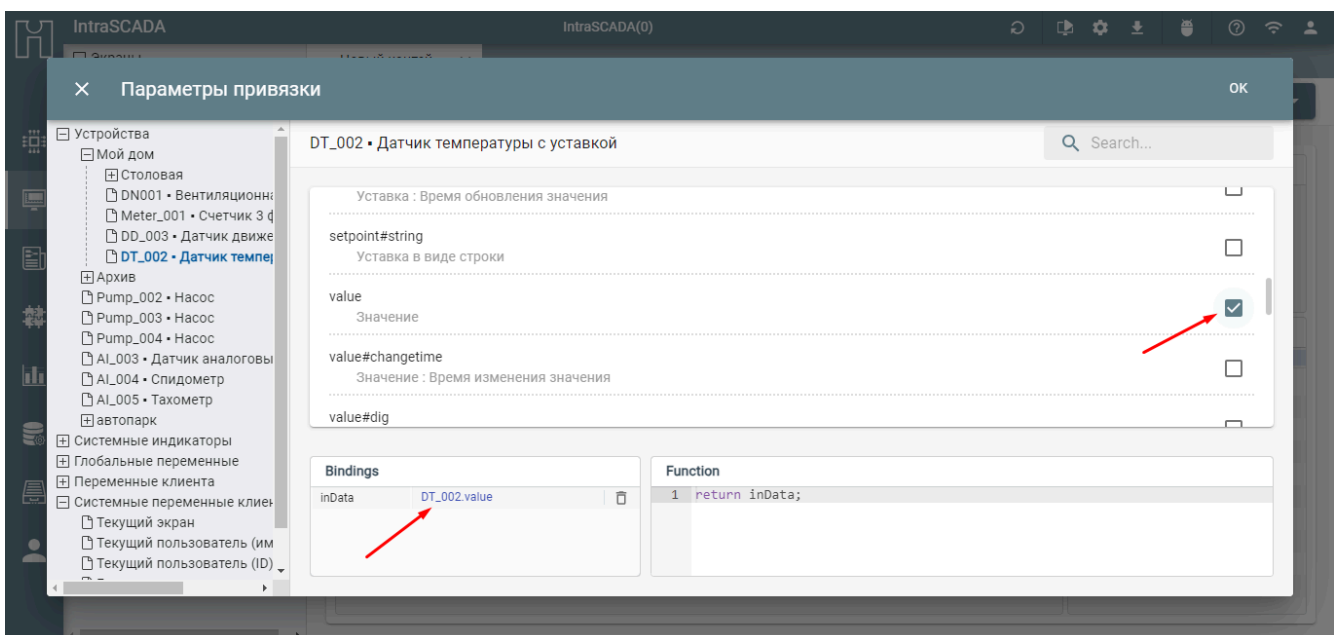


Например, выведем значение температуры разным цветом в зависимости от значения. А при ошибке датчика - изменим цвет рамки.

- Создаем элемент Текст

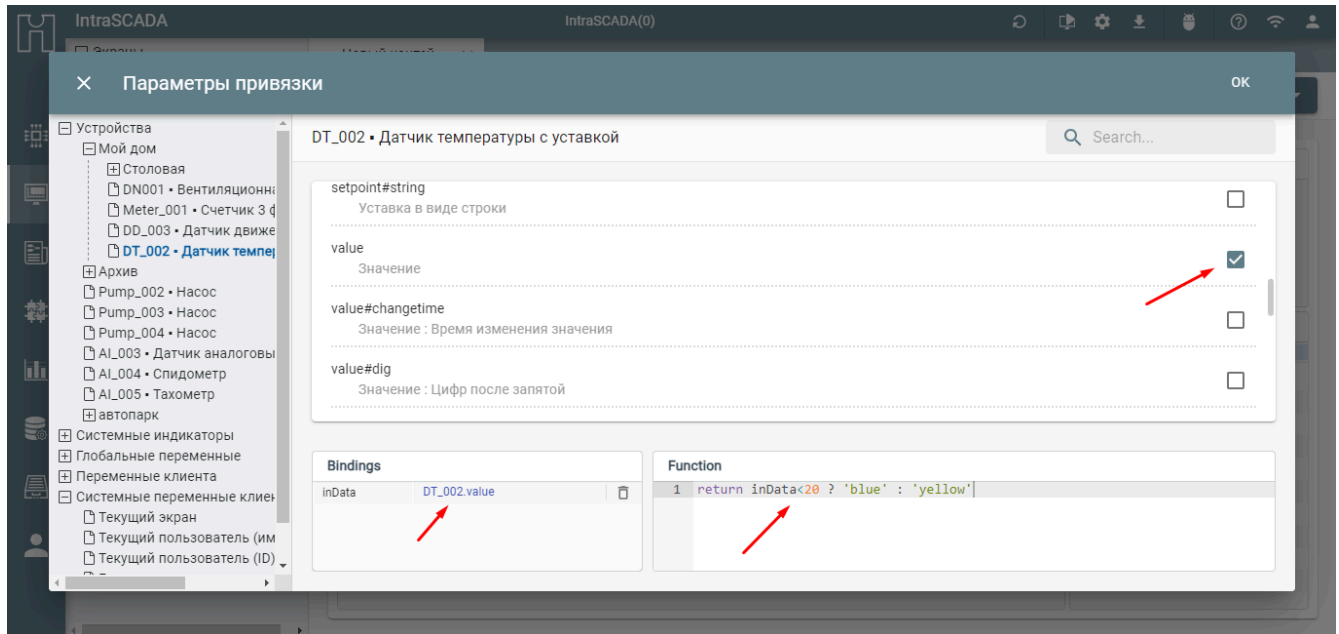


- Свойство "Текст" привяжем к каналу Датчика температуры VALUE



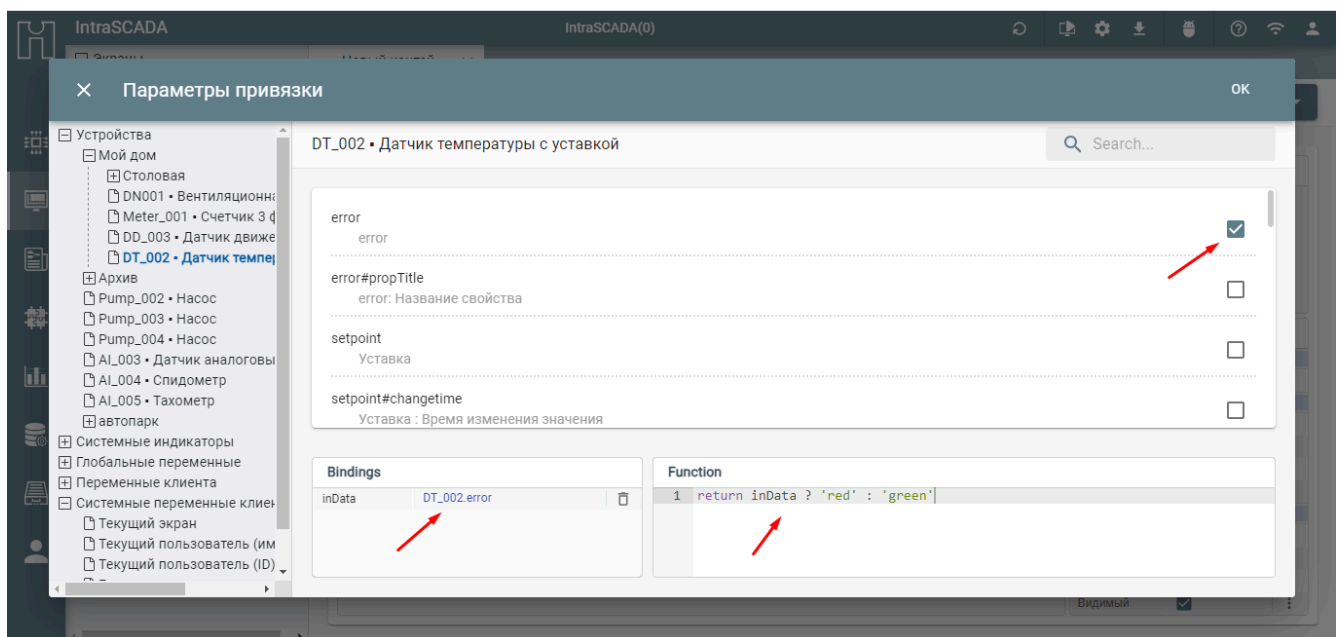
- Свойство "Цвет текста" также привяжем к каналу Датчика температуры VALUE и зададим функцию:

```
inData<20 ? 'blue' : 'yellow'
```



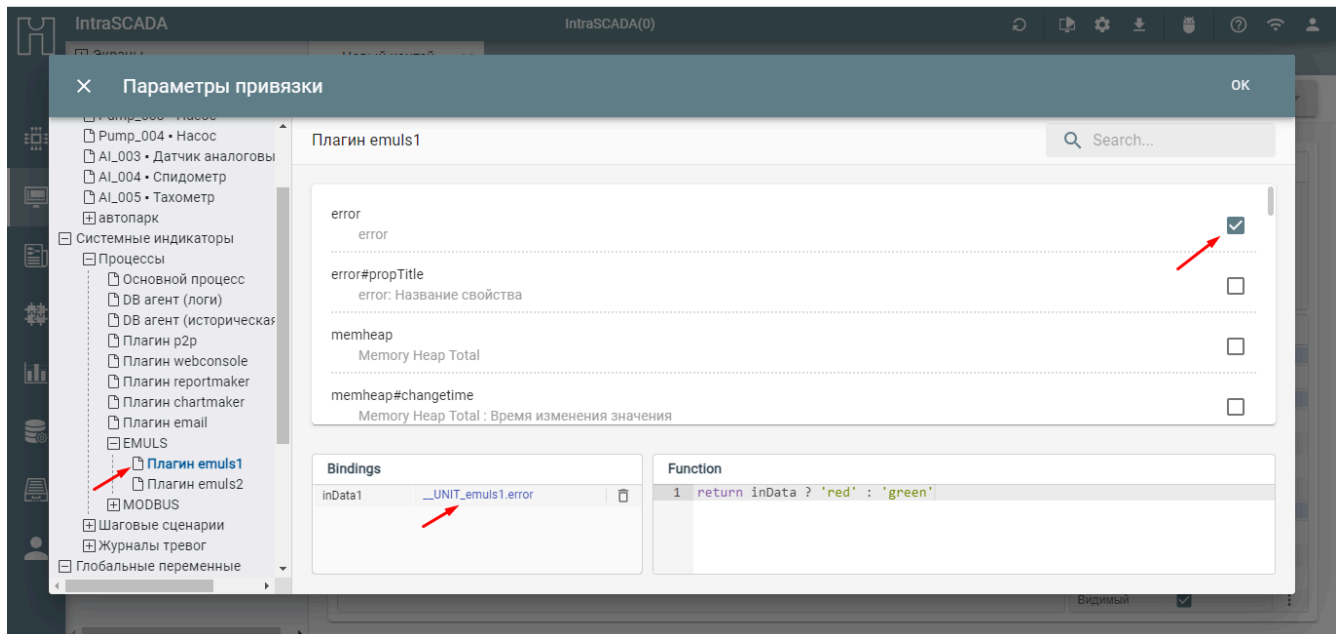
- Свойство "Цвет рамки" привяжем к каналу Датчика температуры ERROR и зададим функцию:

```
inData ? 'red' : 'green'
```



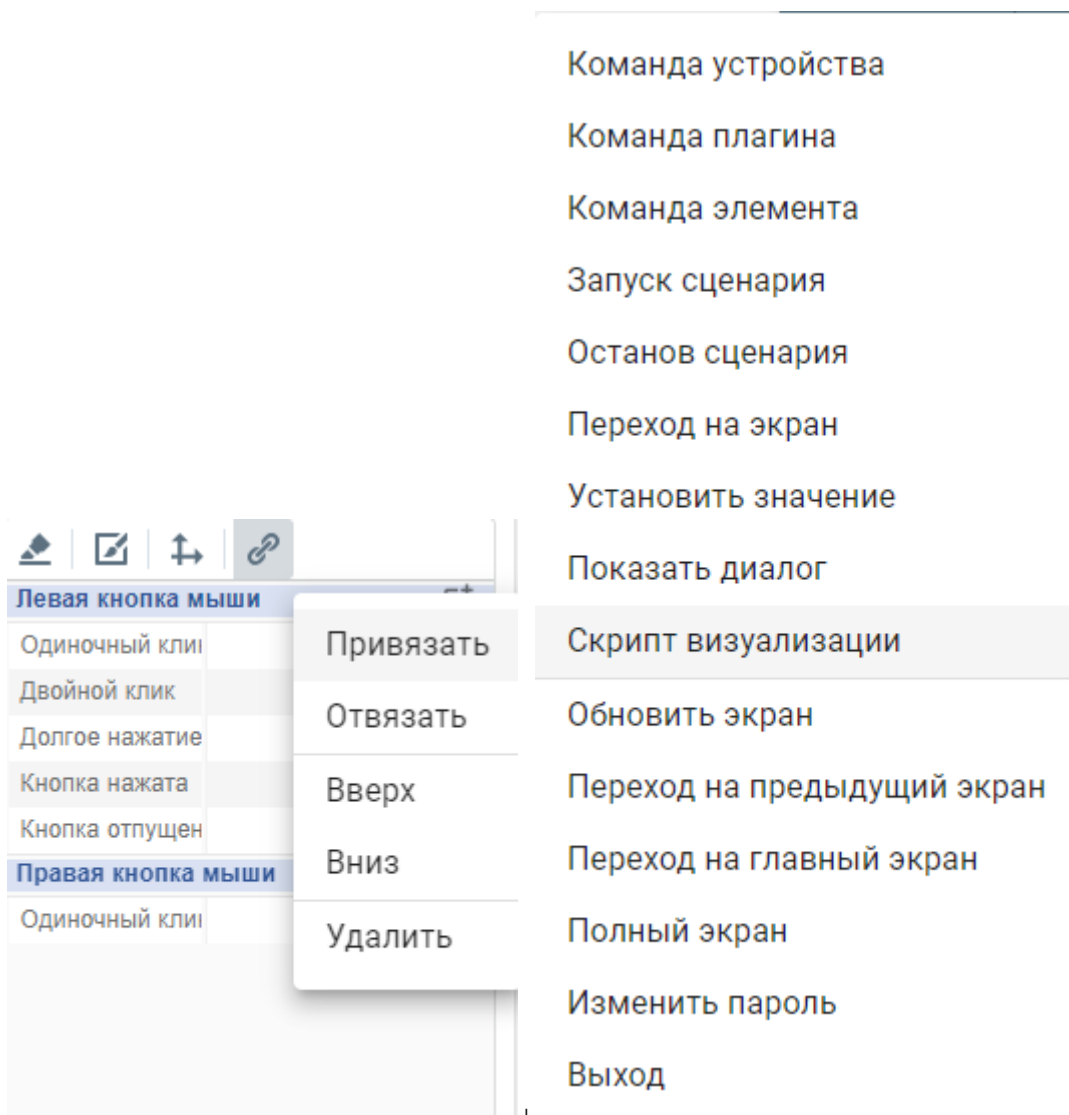
В конечном итоге, если температура будет выше 20, цвет текста будет жёлтым, если ниже - синим. Если датчик работает без ошибок, то рамка будет зелёной. При ошибке рамка будет красной. Цвет можно указать в разных форматах : текстовом ('red'), шестнадцатеричном ('#FF0000') или rgba ('rgba(255,0,0,1)')

- Внутри одного элемента можно выбирать любые свойства разных устройств. Например, "Цвет рамки" можно привязать не к ошибке датчика, а к ошибке плагина.



В последней вкладке осуществляется привязка действий мыши для элемента **Button**:

Все действия мыши, такие как одиночный клик, двойной клик, долгий клик, нажатие клавиши, отпускание клавиши или клик правой клавишей мыши могут быть привязаны к действиям, показанным на скриншоте :



**Способ 2. Использование шаблонов визуализации**

Это простой и не трудоемкий для пользователя вариант.

В системе есть набор готовых шаблонов визуализации. Кроме того, шаблоны можно создавать прямо в системе в разделе [Шаблоны визуализации](#).

**Способ 3. Использование переменных клиента для переключения контекста в контейнере**

Данный способ удобен в том случае, если у вас есть повторяющиеся объекты. Необходимо нарисовать одну мнемосхему с динамическими привязками, которые реализованы с помощью переменных клиента. Эти переменные доступны в рамках одной сессии для каждого пользователя индивидуально. В качестве самого простого примера использования переменных клиента является передача через них ID устройства для получения данных в диалоге. Более подробно как это реализовано можно прочитать по [ссылке](#)



# Диалоги

Диалоги (диалоговые окна) вызываются командой **Показать диалог**.

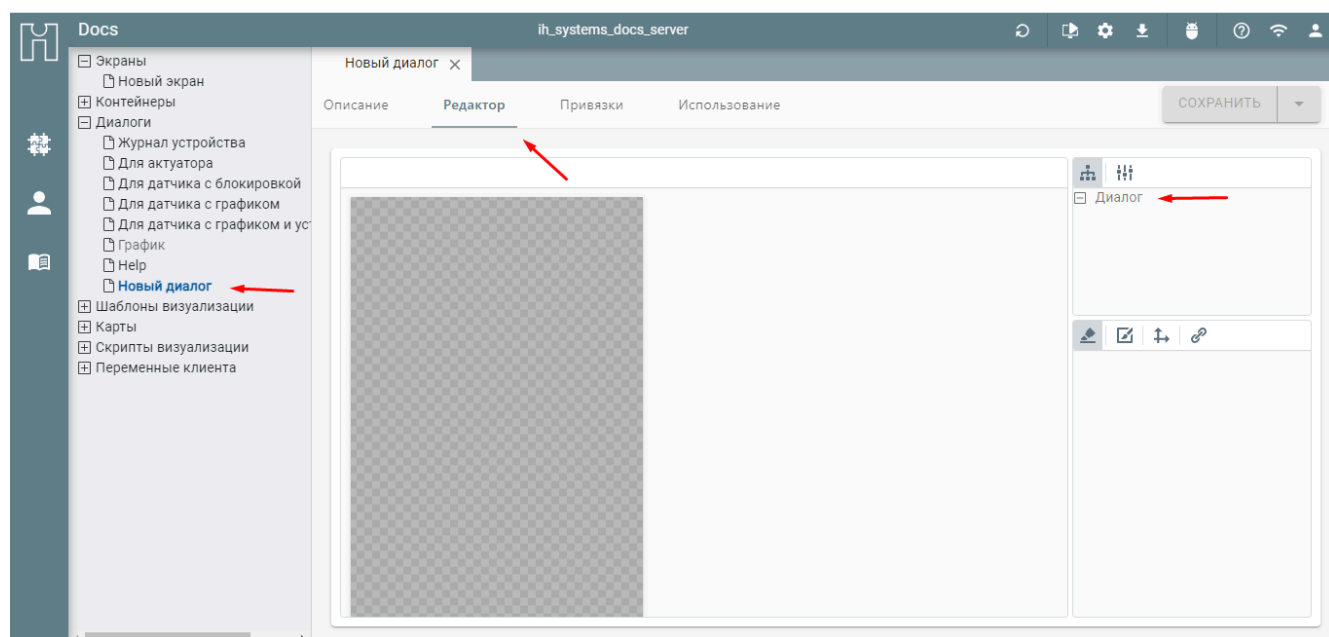
Примеры диалогов: всплывающие окна с настройками параметров устройств, информационные сообщения в центре экрана.

## Создание диалога

Для создания нового Диалога кликните правой кнопкой мыши в дереве Диалогов и выберите «Новый диалог».

На вкладке "Описание" можно ввести имя Диалога и комментарий.

На вкладке "Редактор" видим пустое окно Диалога. Кликнув кнопкой мыши на имени Диалог можно изменить его параметры:



## Основные настройки

Диалог

Позиция	Right
Процент заполн	◀ 0 ▶
Фиксировать пе	<input type="checkbox"/>
Модальный	<input checked="" type="checkbox"/>
Тень	<input checked="" type="checkbox"/>
Заккрыть по клин	<input checked="" type="checkbox"/>

Параметр	Описание
Позиция	Место, где будет выводиться диалог на экране: Сверху, Слева, Справа, Снизу, По центру, В месте клика
Процент заполнения экрана	Процент заполнения диалога на экране в зависимости от его расположения.
Фиксировать переменные клиента	Сохранять контекст переменной клиента при открытии нескольких диалоговых окон
Модальный	Данный параметр блокирует все, что находится под диалогом, пока его не закроешь
Тень	Тень под диалогом
Заккрыть по клику	Закрывать по клику вне диалога

Оформление

Фон

Цвет

Изображение

unset

Украшение

Цвет наложения

Параметр	Описание
Цвет	Цвет фона
Изображение	Изображение фона
Цвет наложения	Цвет перекрытия фона

Координаты

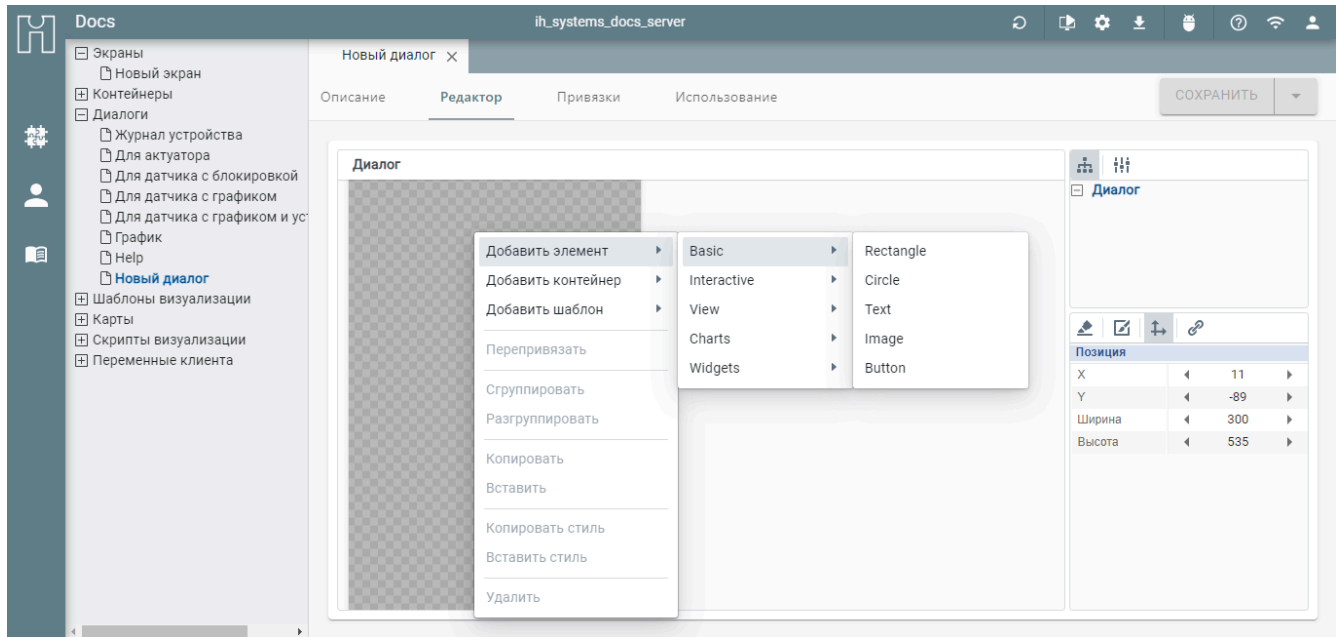
Позиция		
X	◀	10 ▶
Y	◀	10 ▶
Ширина	◀	300 ▶
Высота	◀	535 ▶

Параметр	Описание
Ширина	Ширина диалогового окна
Высота	Высота диалогового окна
X	Координата верхнего угла по горизонтали
Y	Координата верхнего угла по вертикали

## Редактирование диалога

Наполнение диалогового окна может состоять из различных элементов.

Правой кнопкой мыши можно добавить любые [элементы](#) из списка доступных:



## Пример

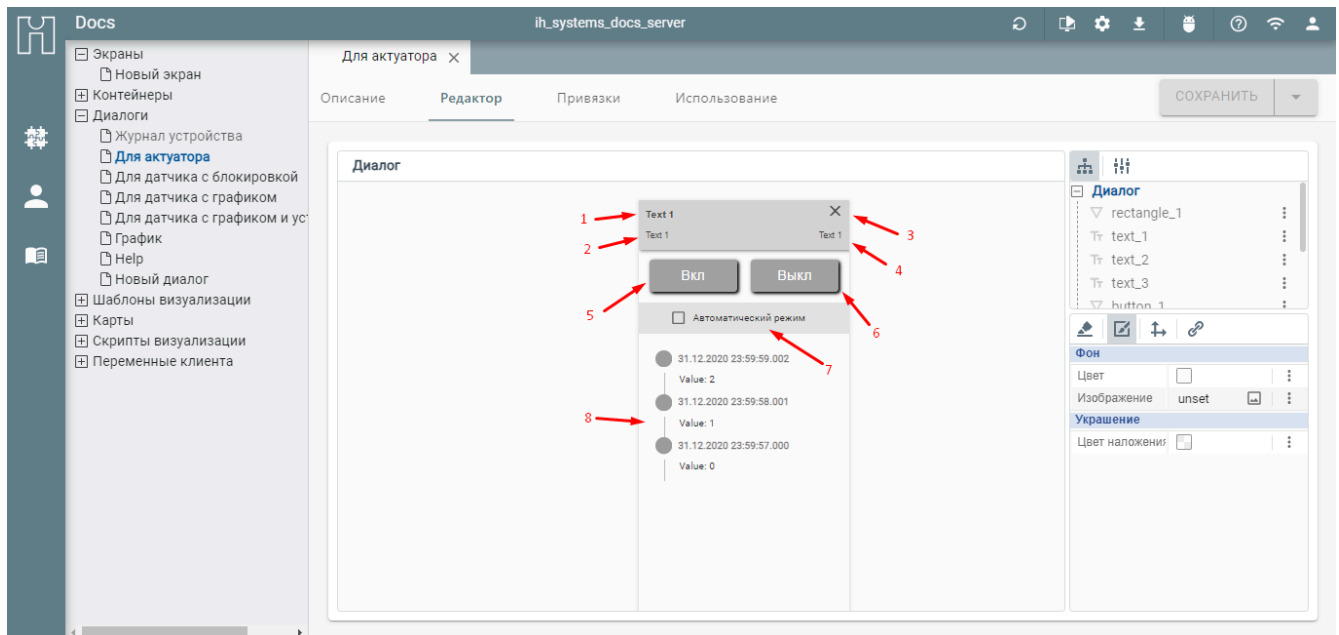
Сделаем диалог для актуаторов. Можно сделать диалог для конкретного устройства.

В данном примере мы сделаем универсальный Диалог для любых бинарных актуаторов.

В переменной клиента **ID устройства** (dev\_id) будет храниться идентификатор конкретного устройства.

## Элементы

Создаем новый диалог и правой кнопкой мыши добавляем на него несколько элементов:

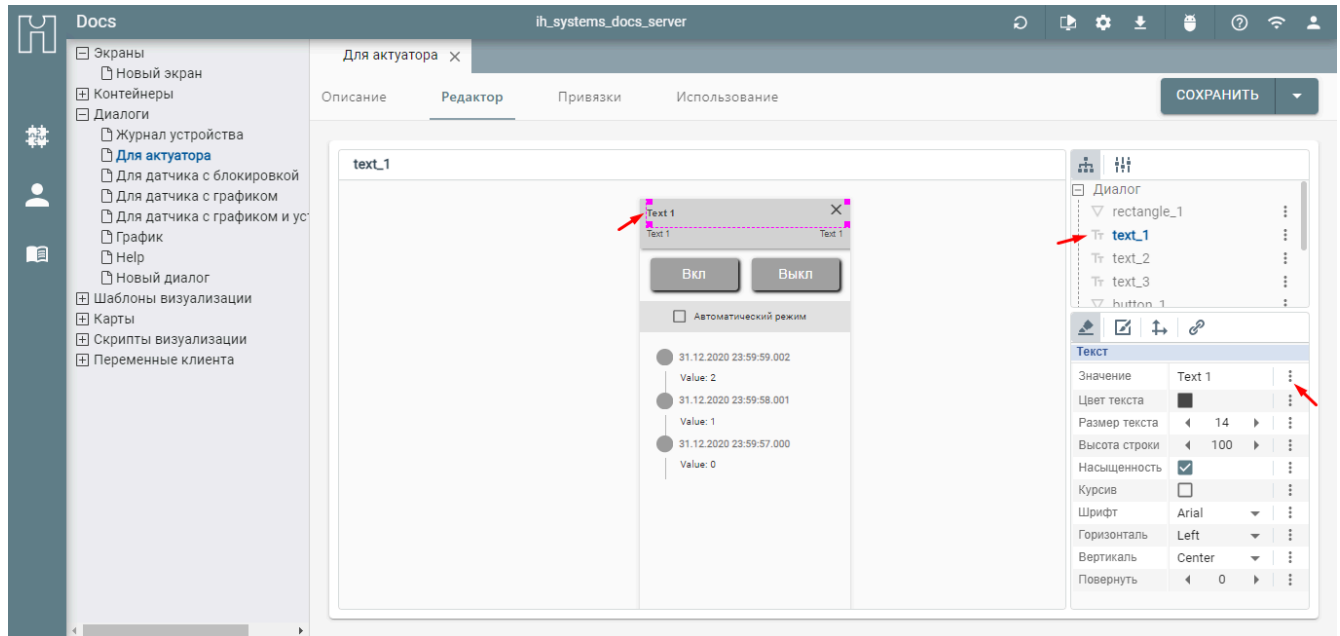


- 1,2,4 - Элементы Text
- 3 - Элемент Button для закрытия диалогового окна
- 5,6 - Элементы Button для управления актуатором
- 7 - Элемент checkbox для изменения режима работы
- 8 - Элемент Device Log для журнала

## Привязки

### 1. Привязываем свойства текстовых элементов

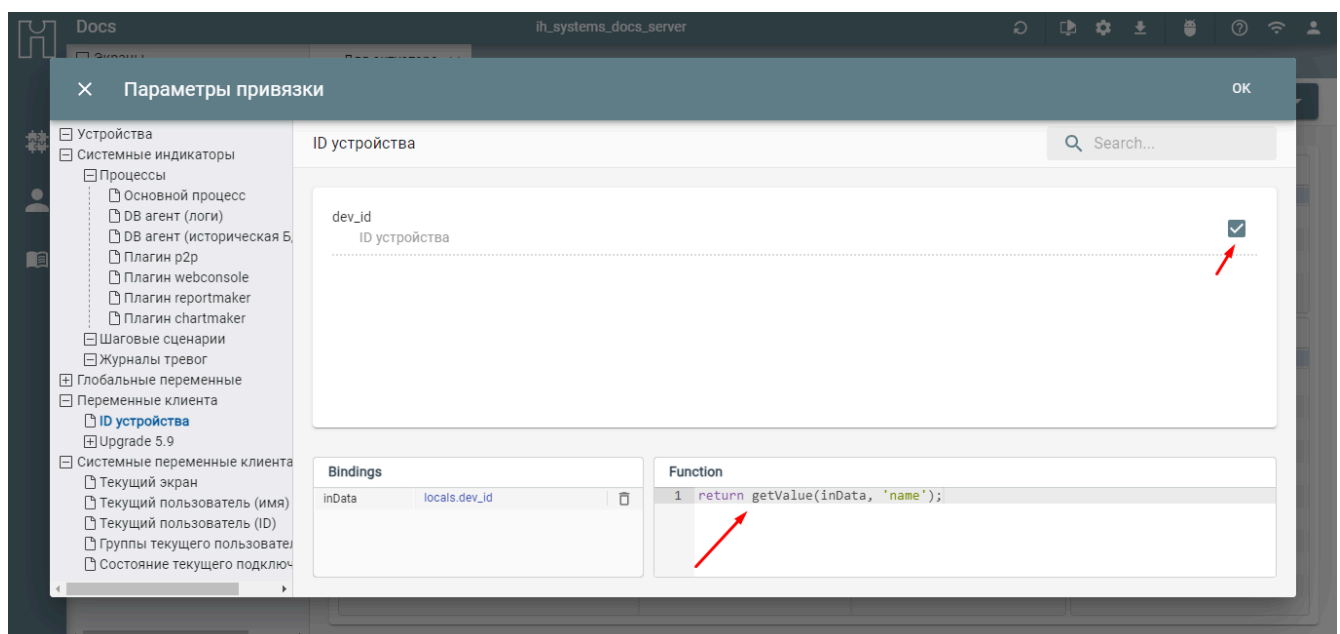
Нажимаем кнопку привязки для свойства Значение у элемента Text\_1:



В всплывающем окне выбираем "Переменную клиента" **ID устройства** и ставим галку напротив **dev\_id**. В формуле прописываем:

```
return getValue(inData, 'name');
```

Нажимаем кнопку Ok.



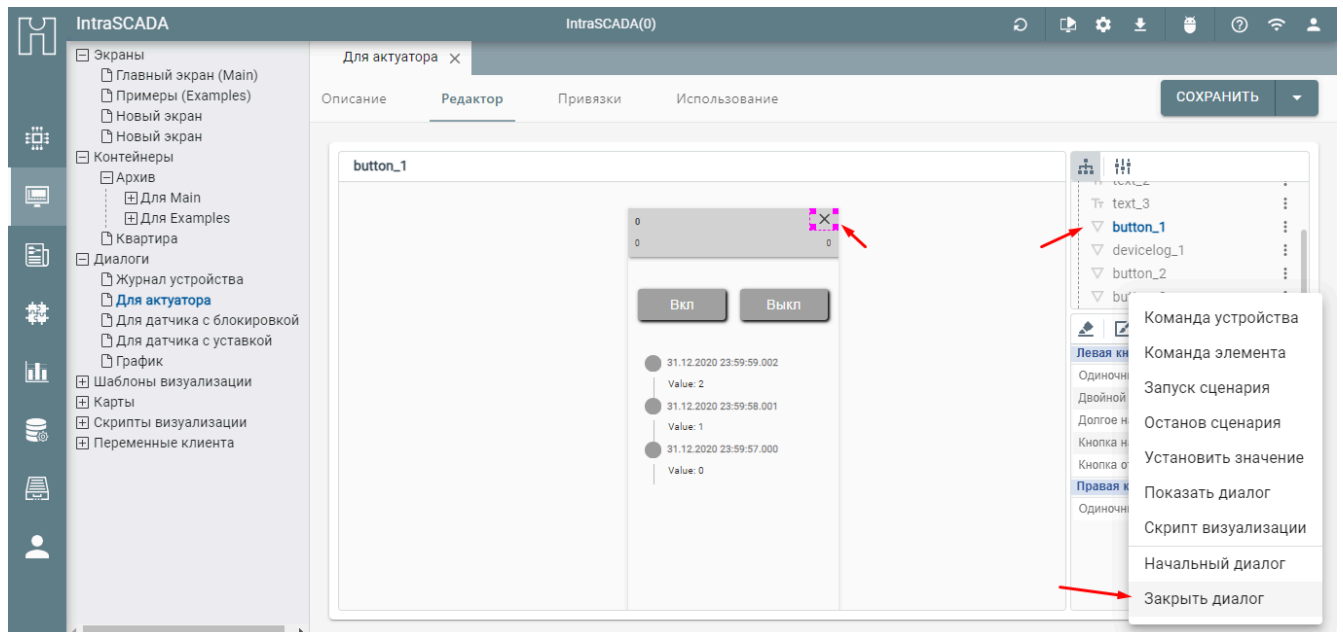
В результате в этом текстовом элементе будет отображаться наименование устройства.

Аналогично привязываем два других текстовых элемента (2,4) к свойствам:

PlacePath - место расположения устройства в соответствии с деревом устройств.

dn - Имя устройства

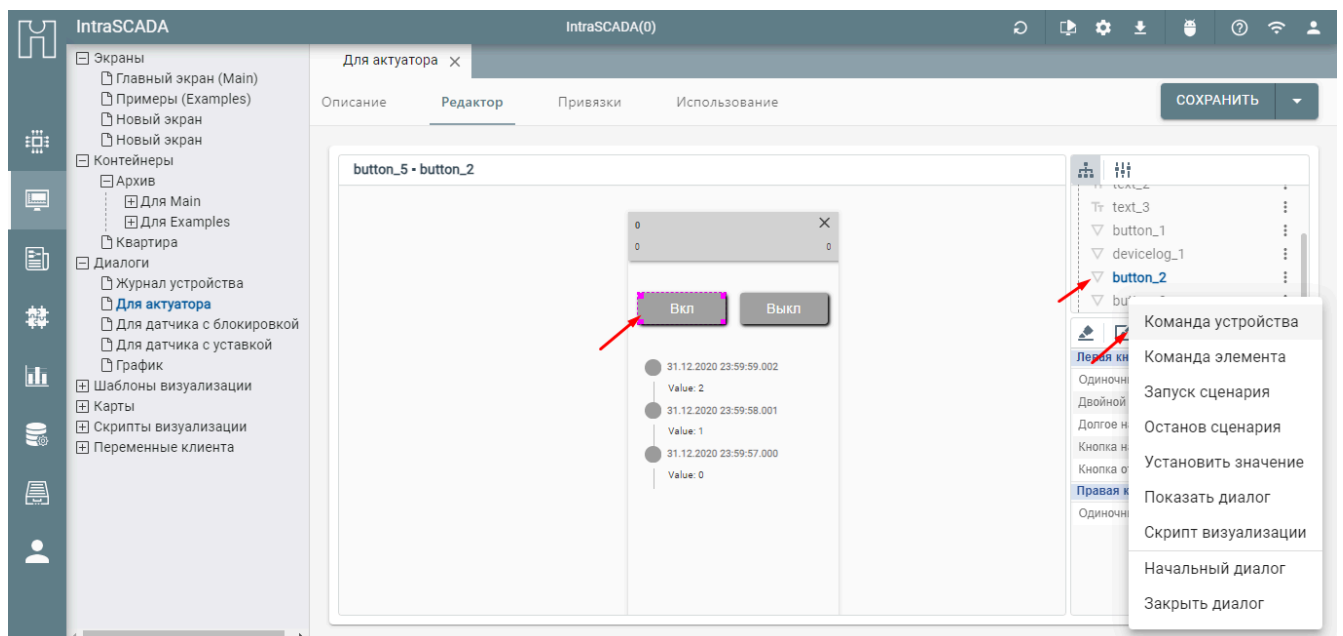
## 2. Привязываем элемент Button - Заккрытие диалога



При нажатии на кнопку button\_1 диалоговое окно закроется.

## 3. Привязываем кнопки управления актуатором

Выбираем кнопку button\_2 (Вкл.) и на действие мыши Одиночный клик привязываем **Команду устройства**.

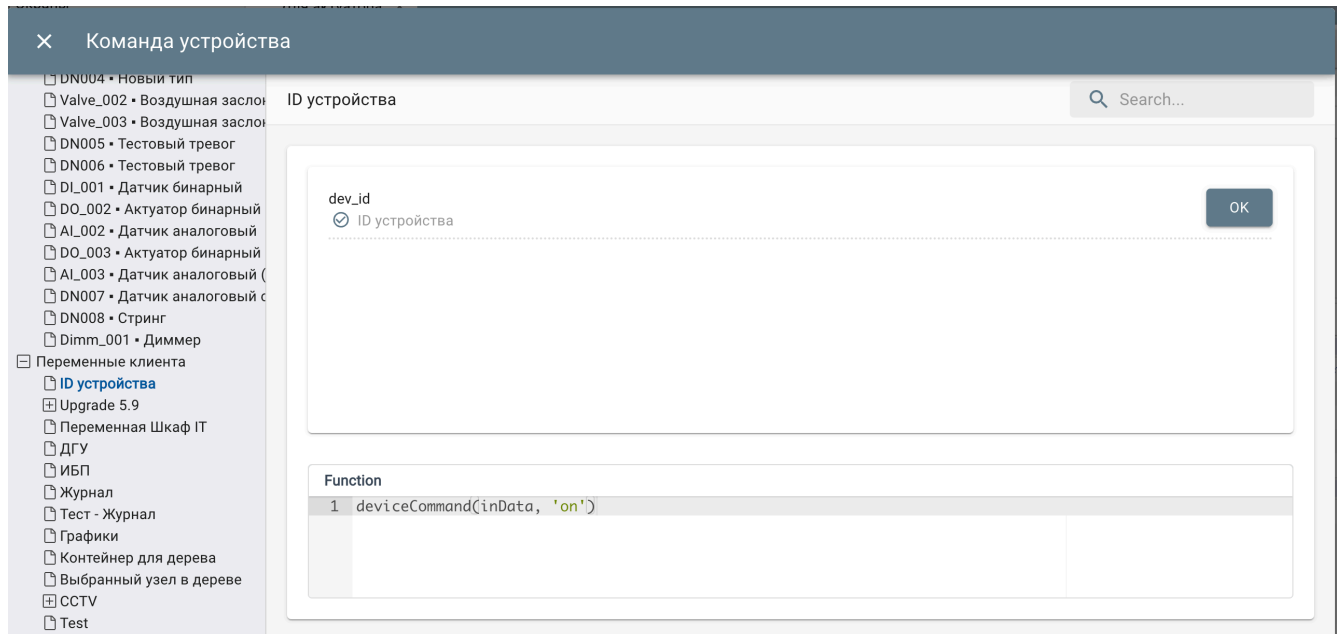


В всплывающем окне выбираем "Переменную клиента" **ID устройства** и ставим галку напротив **dev\_id**. В формуле прописываем:

```
deviceCommand(inData, 'on');
```

Нажимаем кнопку Ok.

**Обратите внимание.** В формуле при использовании функции **deviceCommand** return перед вызовом функции отсутствует, так как нет необходимости ничего возвращать на визуальный компонент.

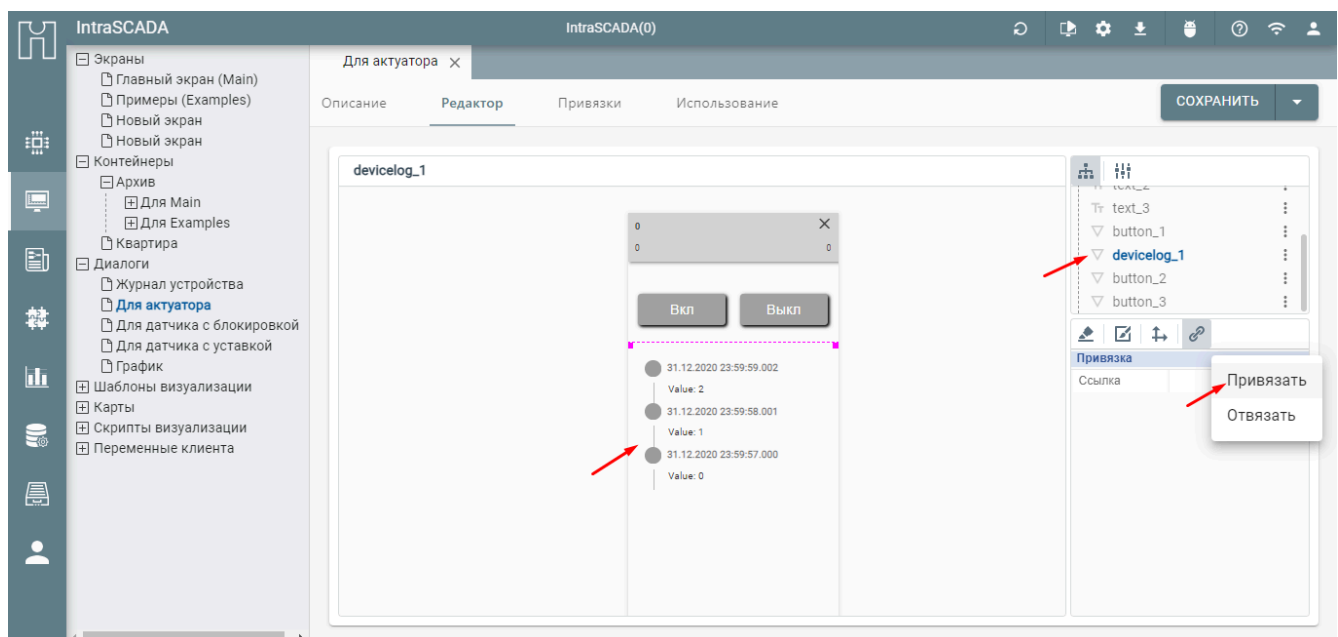


Аналогично привязываем кнопку для выключения актуатора.

```
deviceCommand(inData, 'off');
```

#### 4. Привязываем журнал

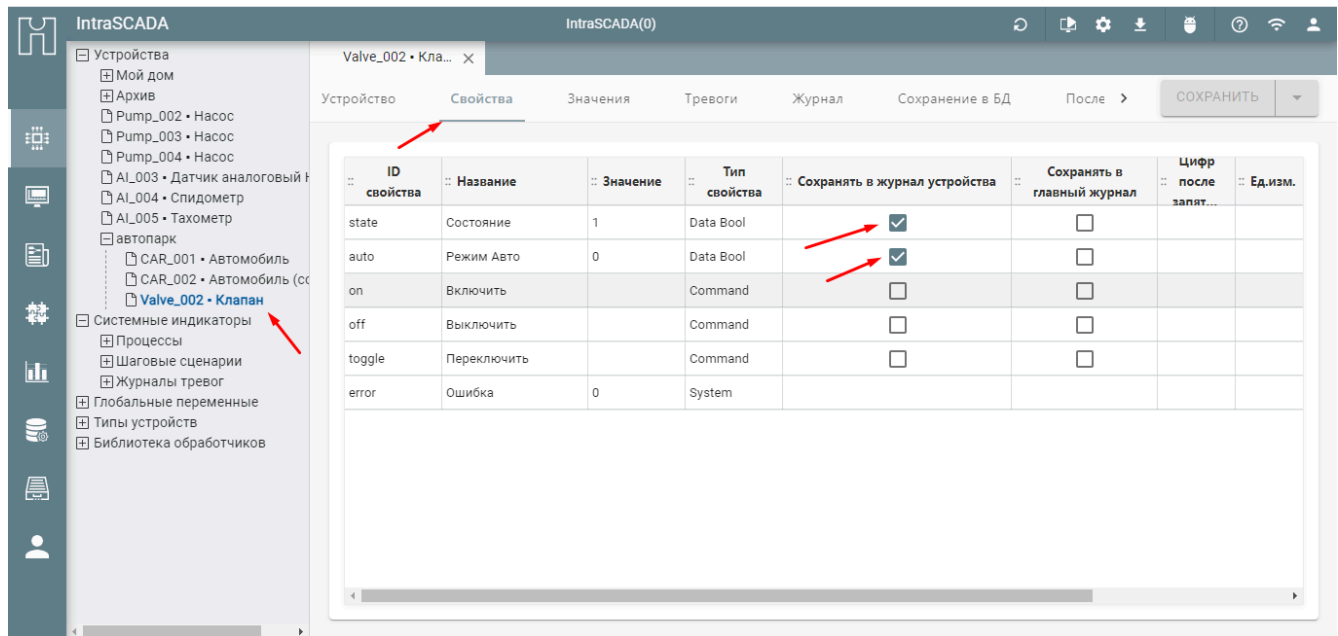
Выбираем элемент Device Log и нажимаем кнопку привязки:





В всплывающем окне выбираем "Переменную клиента" и нажимаем кнопку ОК

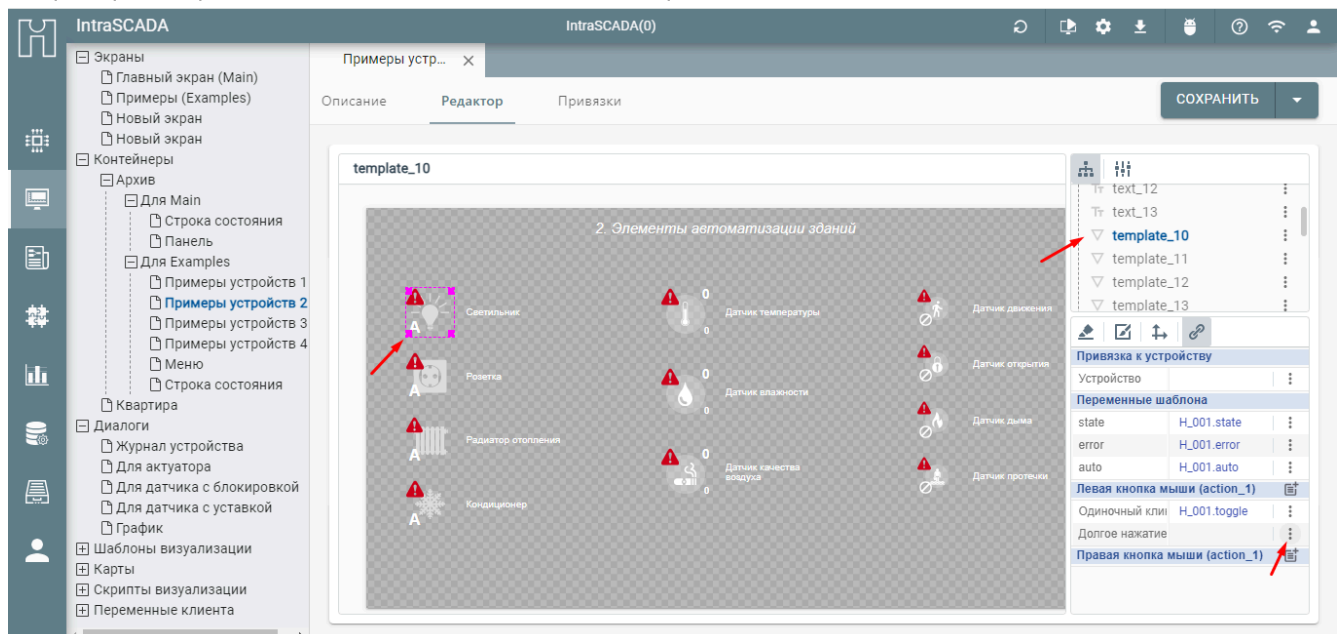
Обратите внимание, для того, чтобы в Device Log показывались данные, необходимо поставить галки в поле **Сохранять в журнал устройства** в списке устройств.

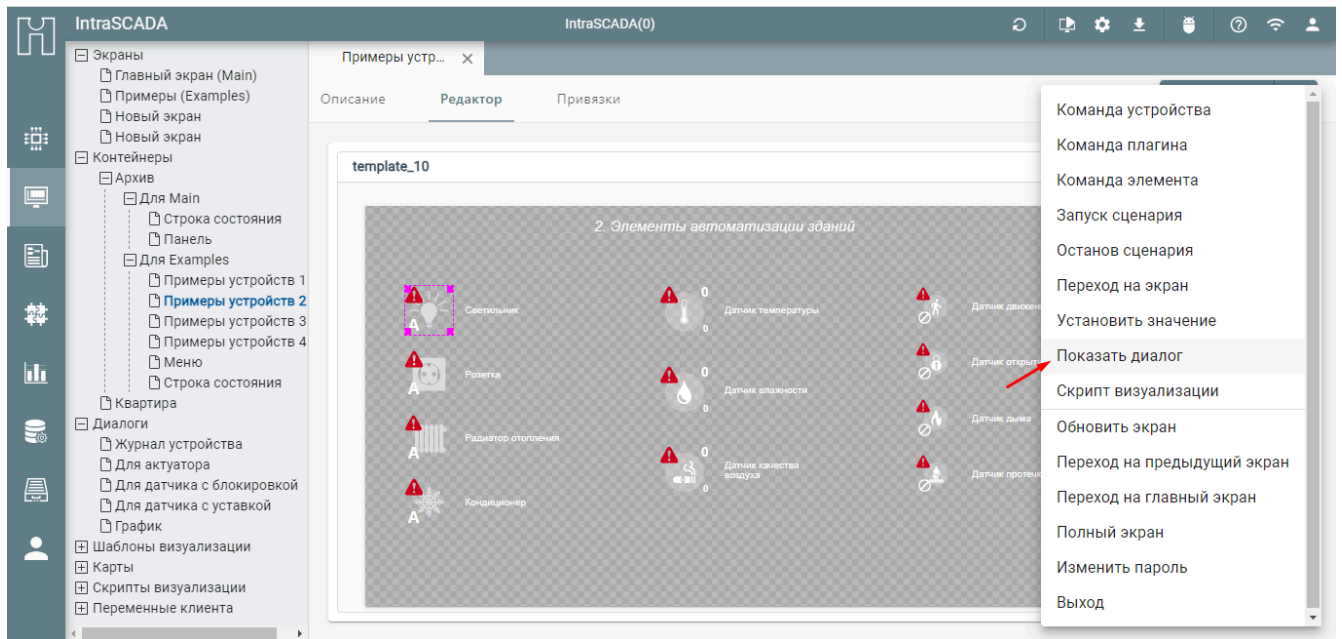


## Использование диалога

Вызов диалогового окна можно привязать к любой кнопке или к любому Шаблону визуализации.

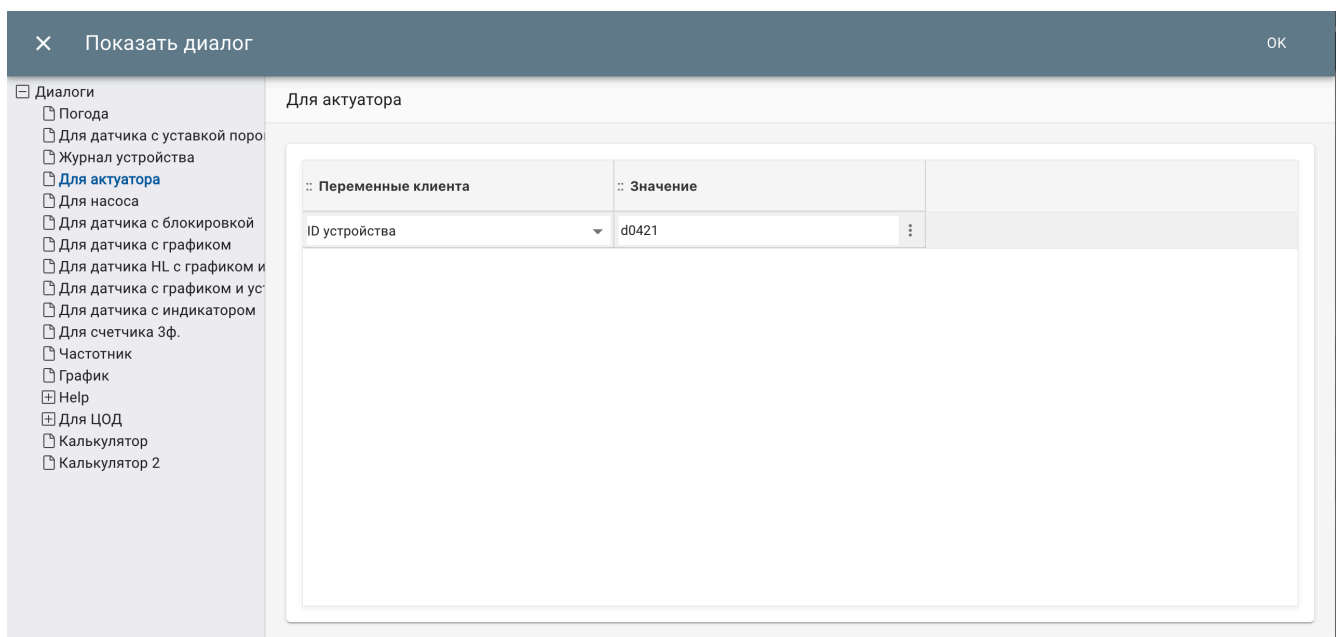
Например, выбираем светильник и на Долгое нажатие привязываем вызов Диалога



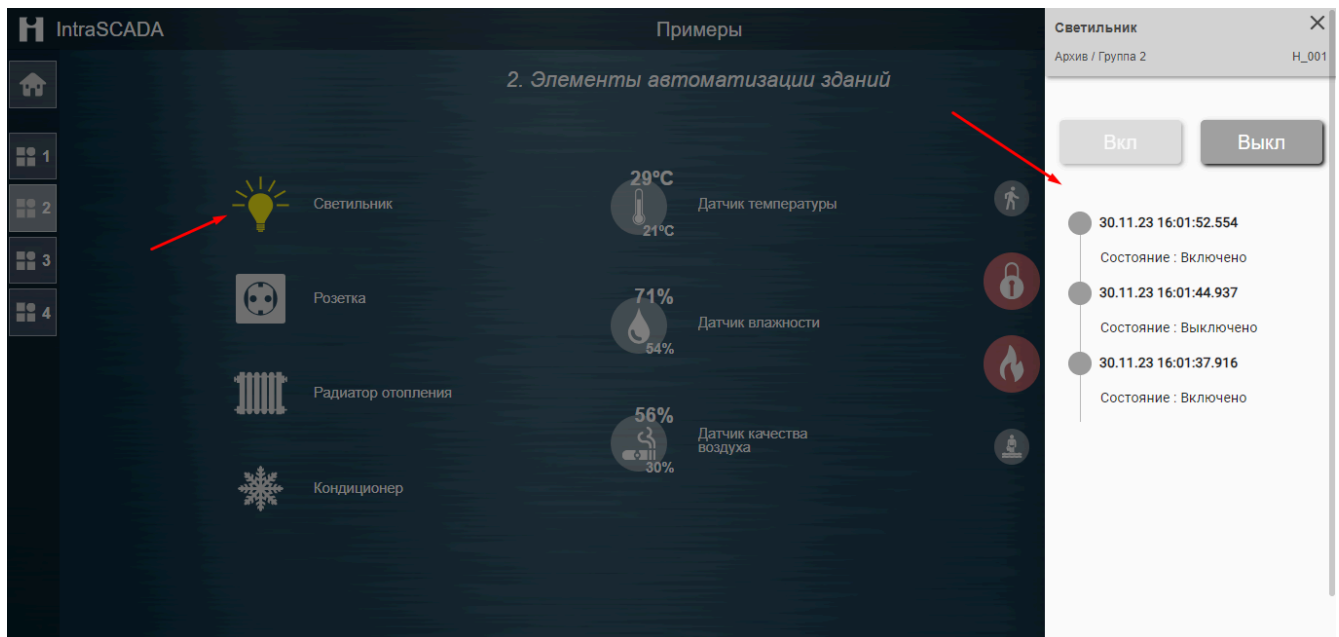


В всплывающем окне выбираем созданный выше диалог. В правой части окна с помощью правой кнопки мыши вызываем меню. Добавляем строку и вводим **ID устройства** и выбираем конкретное устройство.

Нажимаем Кнопку ОК.



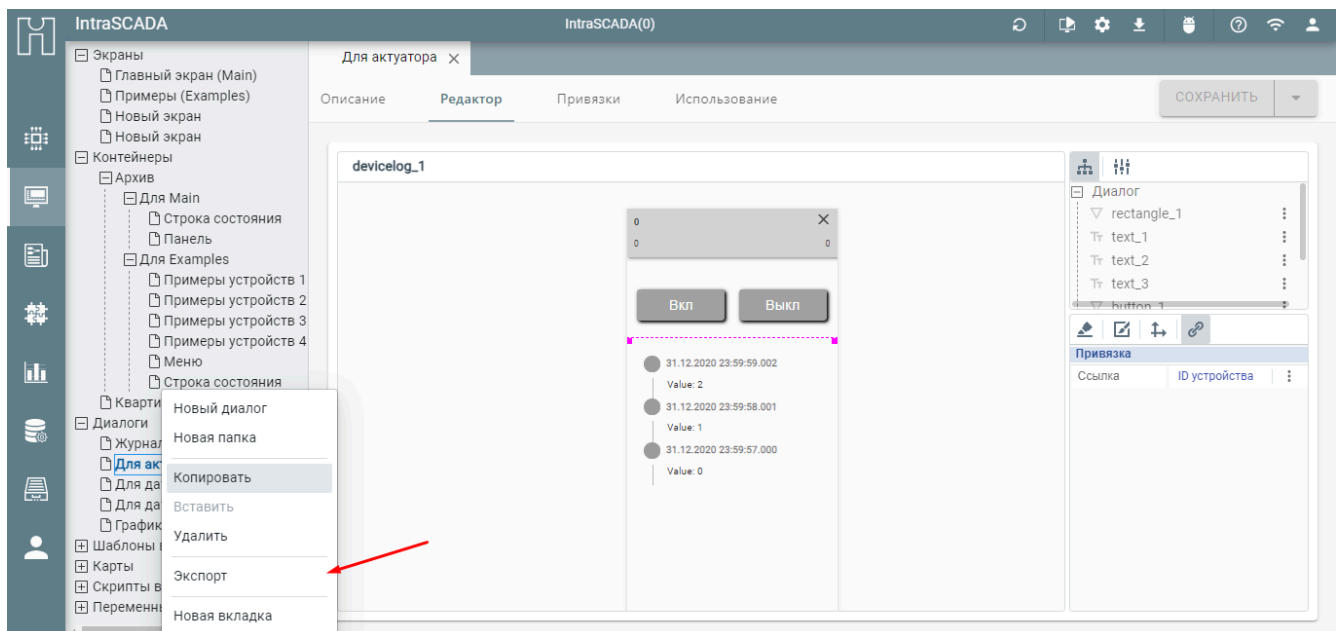
Теперь в пользовательском интерфейсе при долгом нажатии на устройстве будет открываться диалоговое окно и в **Переменной клиента** будет передано (присвоено) id устройства, который нам необходимо отобразить в диалоге.



## Экспорт/Импорт

### Экспорт диалога

Любое диалоговое окно можно экспортировать с сервера на свой компьютер. Это бывает необходимо для использования его в других проектах или чтобы поделиться с коллегами. Нажмите правой кнопкой мыши в дереве диалогов на нужный диалог и выберите пункт "Экспорт". Диалоговое окно будет сохранено на вашем компьютере с расширением `ihpack`



### Импорт диалога

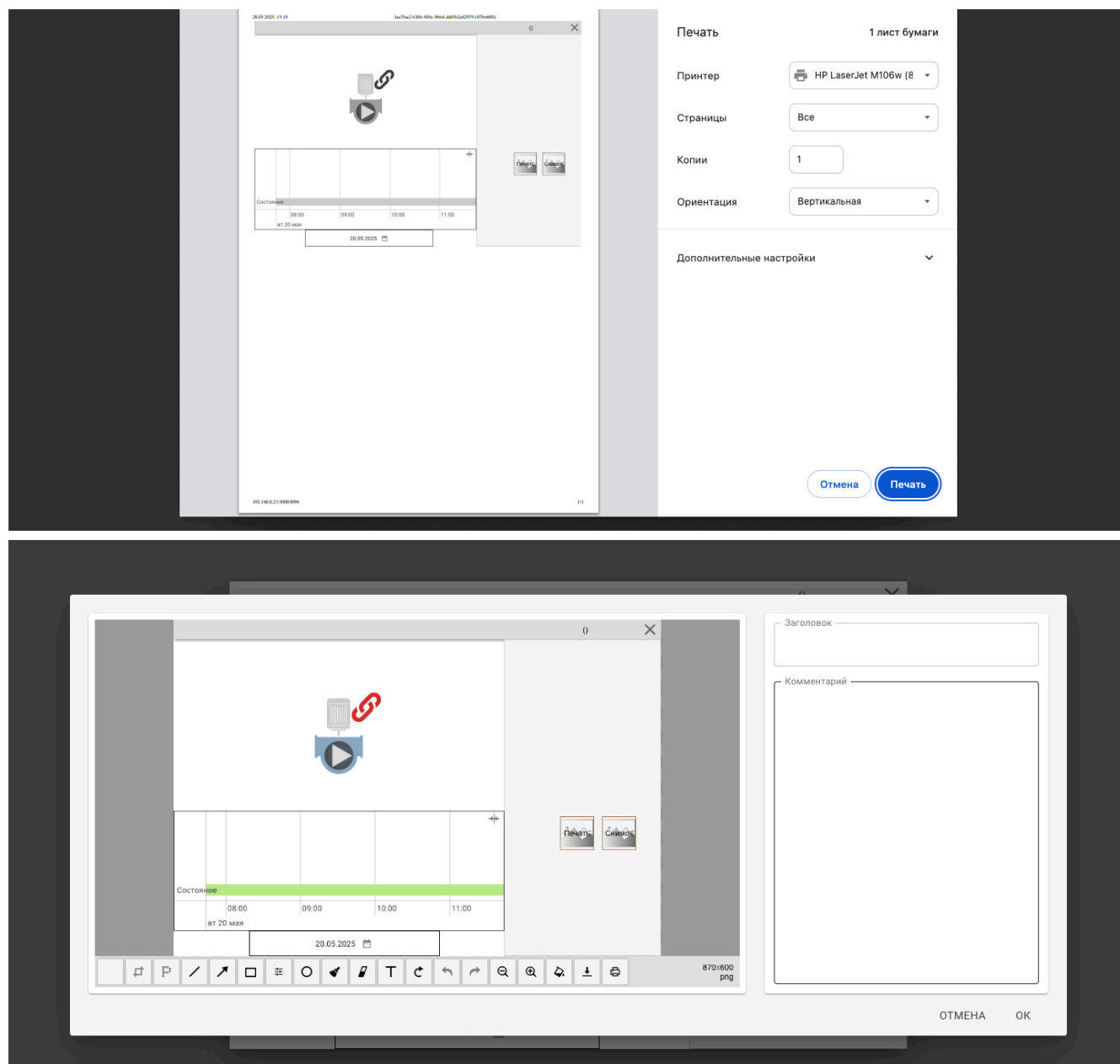
Для загрузки диалогового окна на сервер нужно нажать кнопку импорта в верхней строке главного меню системы и выбрать файл с расширением `ihpack`. Система сама определяет содержимое файла `ihpack` и, если там есть диалоговое окно, поместит его в дерево диалогов.



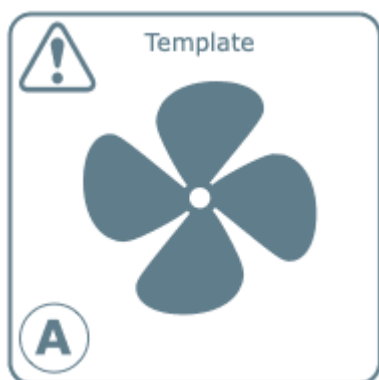
## Снимок / Печать диалогового окна

Добавлено v5.18.14

Начиная с версии 5.18.14 добавлена возможность снимка и печати диалогового окна



# Шаблоны визуализации



Шаблоны визуализации - это элементы интерфейса, размещаемые в контейнерах для визуализации однотипных устройств.

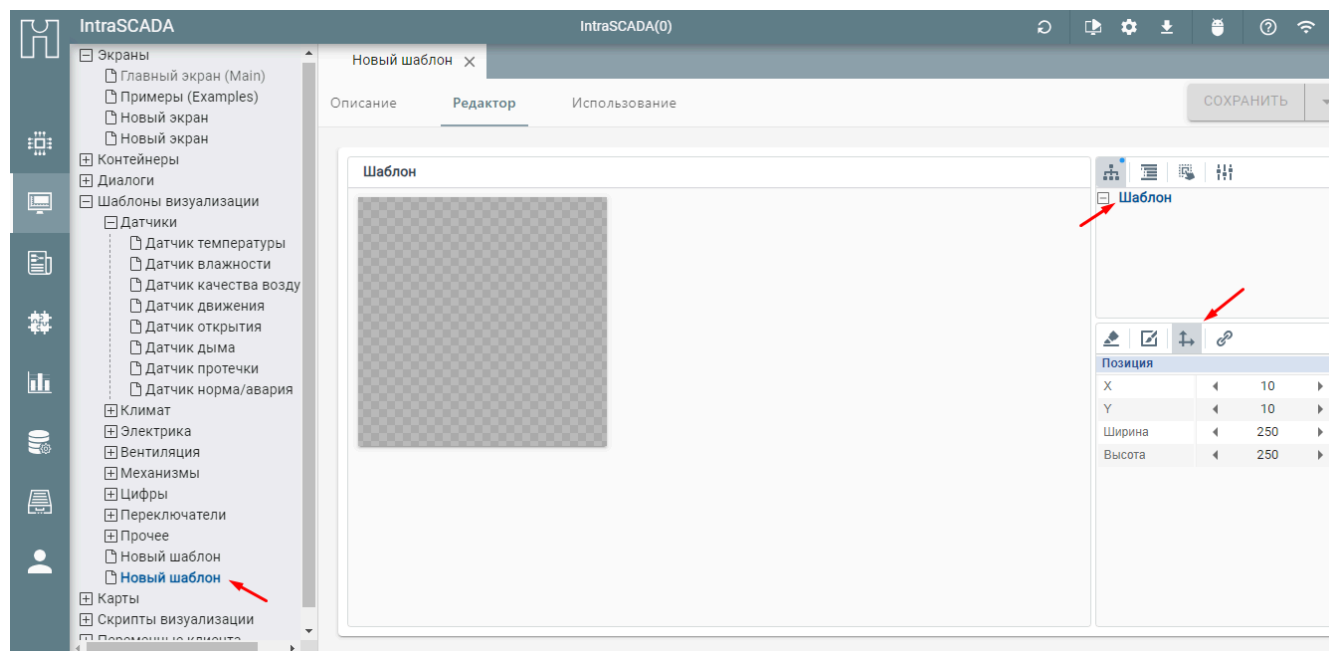
Используются для создания большого количества устройств с одинаковыми свойствами. Можно создать один Шаблон визуализации, разместить необходимое количество этих шаблонов на мнемосхеме и привязать каждый отдельный шаблон к свойствам различных устройств.

## Создание шаблона

Для создания нового шаблона, кликните правой кнопкой мыши в дереве шаблонов и выберите «Новый шаблон».

На вкладке "Описание" можно ввести имя шаблона и комментарий.

На вкладке "Редактор" видим пустой шаблон. Кликнув кнопкой мыши на имени шаблона можно изменить параметры шаблона:



Размеры шаблона

Позиция		
X	◀ 10 ▶	
Y	◀ 10 ▶	
Ширина	◀ 250 ▶	
Высота	◀ 250 ▶	

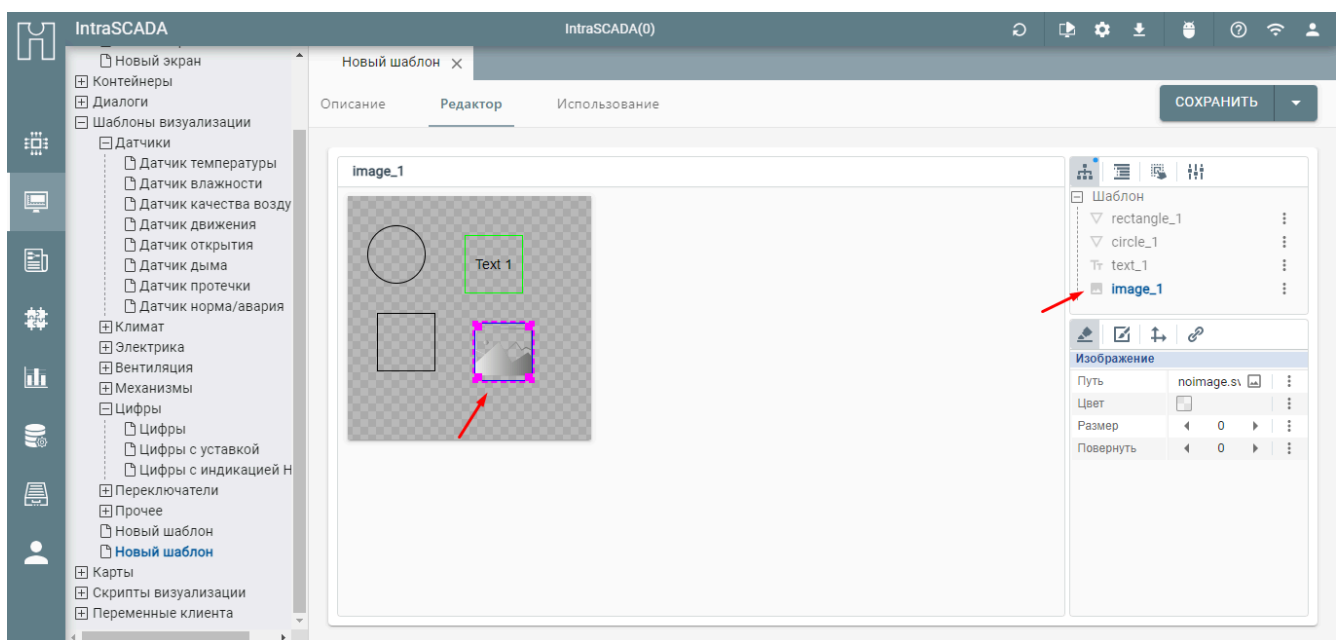
## Редактирование шаблона

### Дефолтное состояние



Кликнув правой кнопкой мыши по полю шаблона можно добавить [элементы](#).

Навигация по элементам в шаблоне выполняется кликами мыши по элементам в поле шаблона или дереве шаблона:




У каждого элемента есть настройки:

## Координаты, размеры


## Оформление

Специфические настройки  
для текста


## для изображения




Позиция			
X	◀	130	▶
Y	◀	130	▶
Ширина	◀	60	▶
Высота	◀	60	▶
Ширина справа	◀	60	▶
Высота снизу	◀	60	▶
Положение эл	◀	100	▶
Преобразовать			
Отразить по гор	<input type="checkbox"/>		
Отразить по вер	<input type="checkbox"/>		
Обрезать	<input checked="" type="checkbox"/>		



Фон			
Цвет	<input type="checkbox"/>		
Рамка			
Цвет	<input checked="" type="checkbox"/>		
Размер	◀	1	▶
Радиус	◀	0	▶
Стиль		Solid	
Украшение			
Анимация	<input type="checkbox"/>		
Тень	<input type="checkbox"/>		
Непрозрачность	◀	100	▶
Видимый	<input checked="" type="checkbox"/>		



Текст		
Значение	Text 1	
Цвет текста	<input checked="" type="checkbox"/>	
Размер текста	◀	14
Высота строки	◀	100
Насыщенность	<input type="checkbox"/>	
Курсив	<input type="checkbox"/>	
Шрифт	Arial	
Горизонталь	Center	
Вертикаль	Center	
Повернуть	◀	0



Изображение		
Путь	noimage.svg	
Цвет	<input type="checkbox"/>	
Размер	◀	0
Повернуть	◀	0

Набор элементов, установленных на шаблоне, образует исходное (дефолтное) состояние.

## Изменение состояний



Для каждого состояния устройства (вкл/выкл/ошибка/автомат и т.д.) все параметры визуализации могут быть изменены - от изменения цвета надписи до замены изображения или включения анимации.

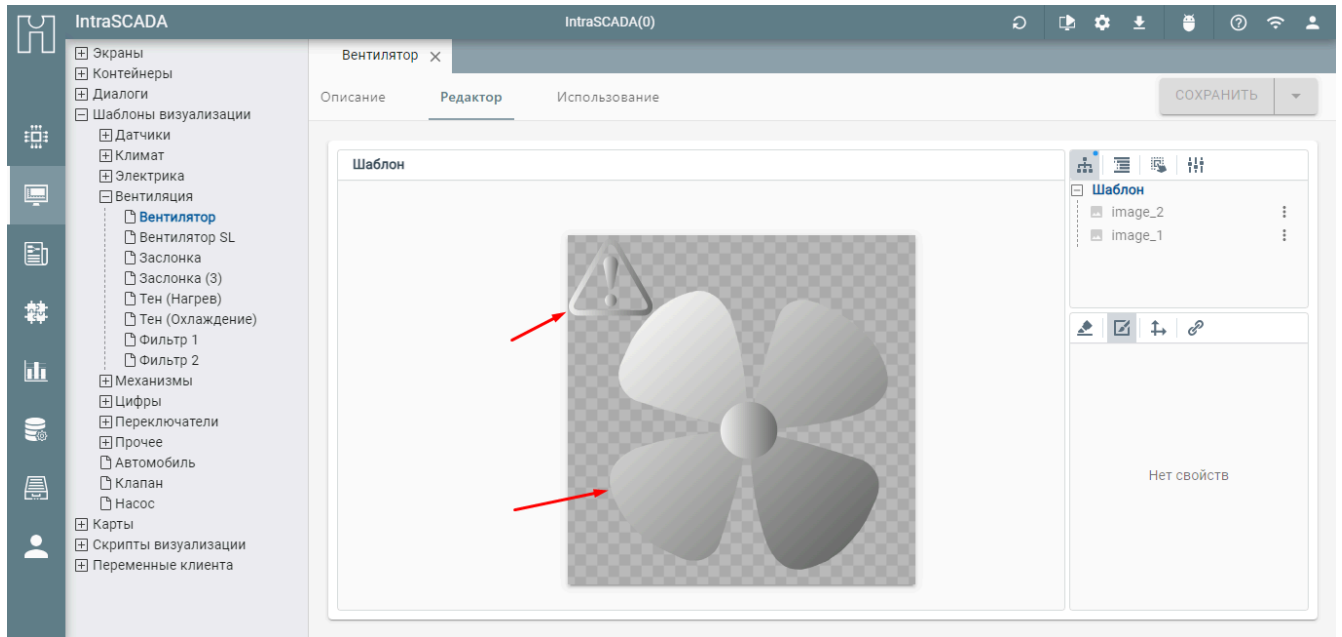
По умолчанию создается две переменных состояния - state и error. Может быть добавлено неограниченное количество переменных.

**Пример: Шаблон вентилятора с несколькими состояниями.**

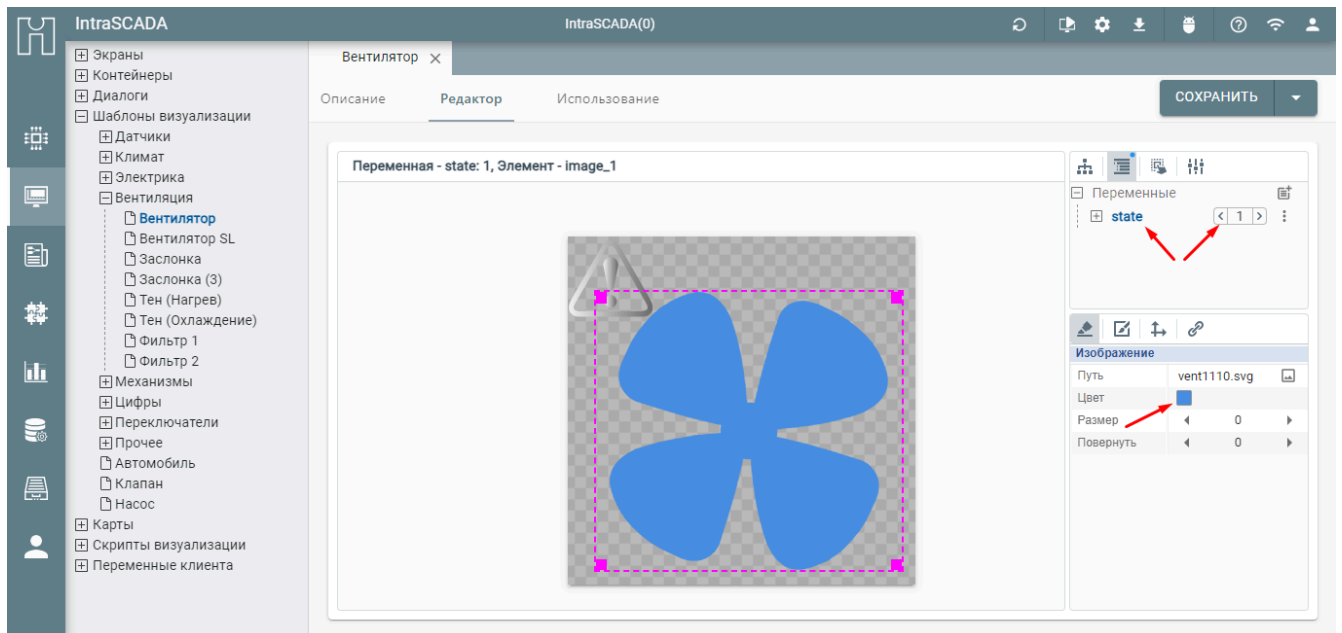
В дефолтном состоянии шаблон содержит два серых изображения:

**image\_1** - вентилятор

**image\_2** - ошибка



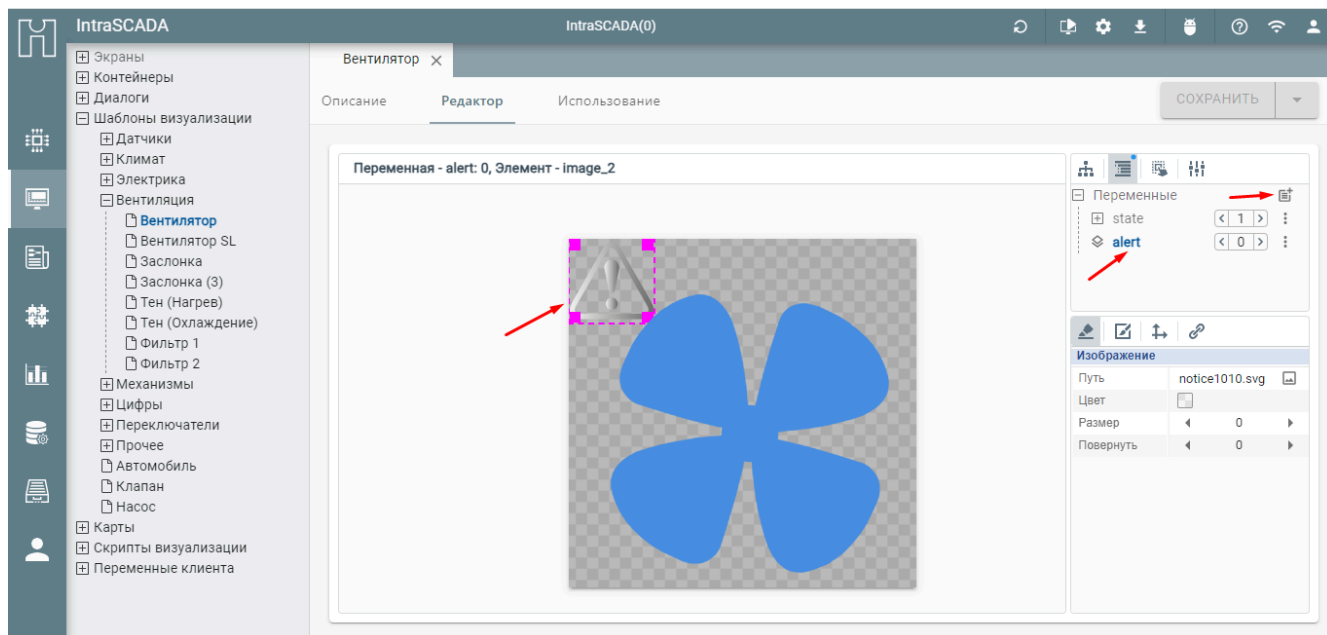
Переходим в настройку состояний, увеличиваем state на 1 (Состояние 1) и меняем цвет вентилятора на синий:



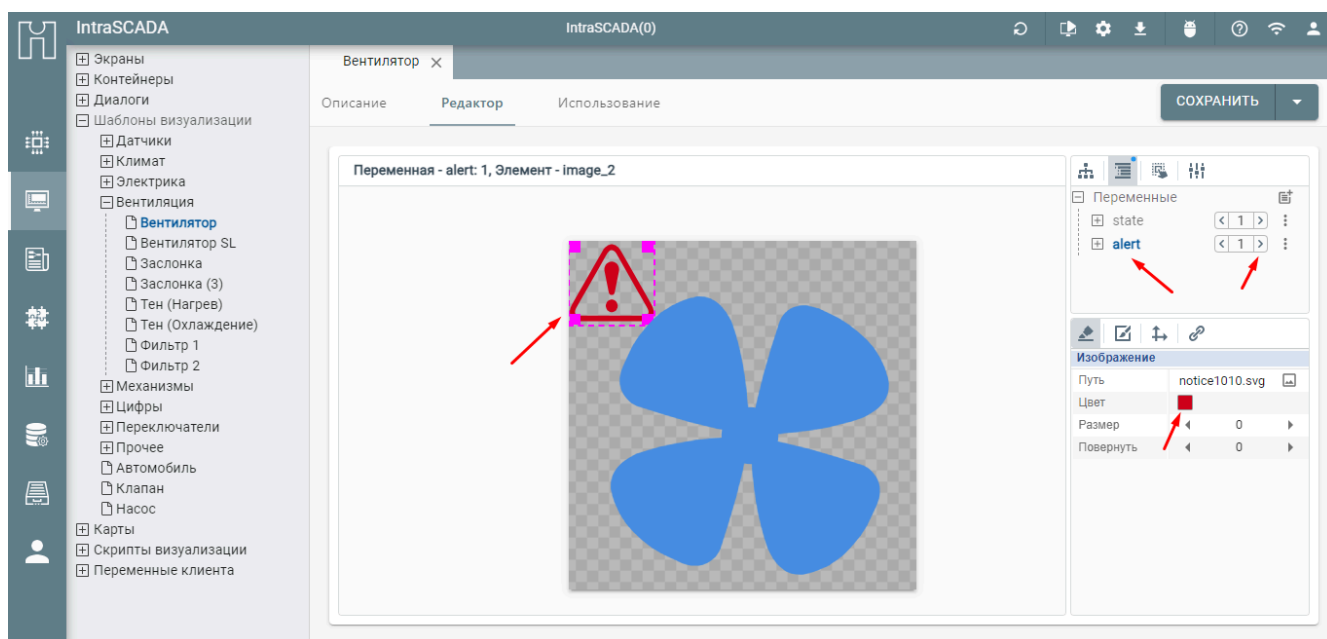
То есть в состоянии 0 вентилятор будет серым, а в состоянии 1 - синим.

Добавляем новую переменную "alert", для image\_2 (Ошибка вентилятора).





Теперь настроим визуальное отображение ошибки вентилятора. Кликаем мышкой на изображение ошибки. Увеличиваем alert на 1 и меняем цвет ошибки на красный:



Теперь можно попробовать изменять state и error и посмотреть как будет выглядеть устройство в различных комбинациях этих состояний.

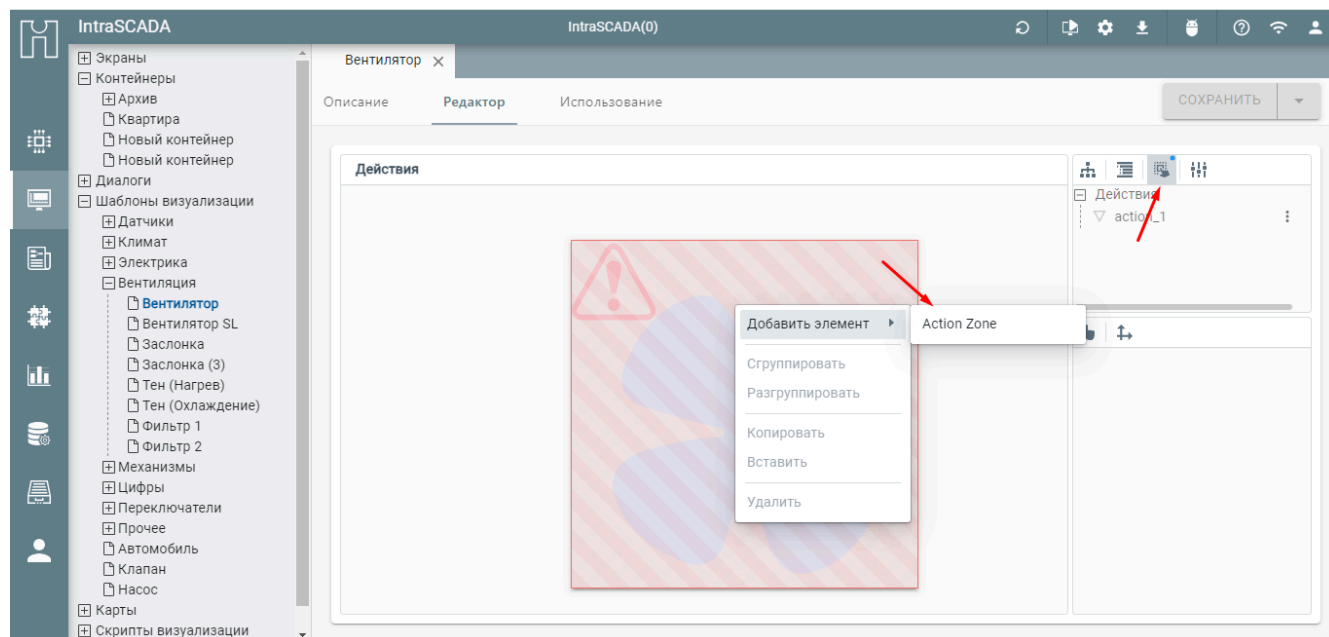
Механизм шаблонов очень мощный. Можно создать шаблон с многими элементами и разнообразными настройками состояний этих элементов - цвет, размер, поворот, анимация.

## Области действий



Любое устройство на экране в пользовательском интерфейсе может срабатывать по клику мыши. Если это необходимо, надо настроить область действий (Action Zone).

Для задания области нажмите правой кнопкой мыши в поле редактора и выберите Добавить элемент -> Action Zone.

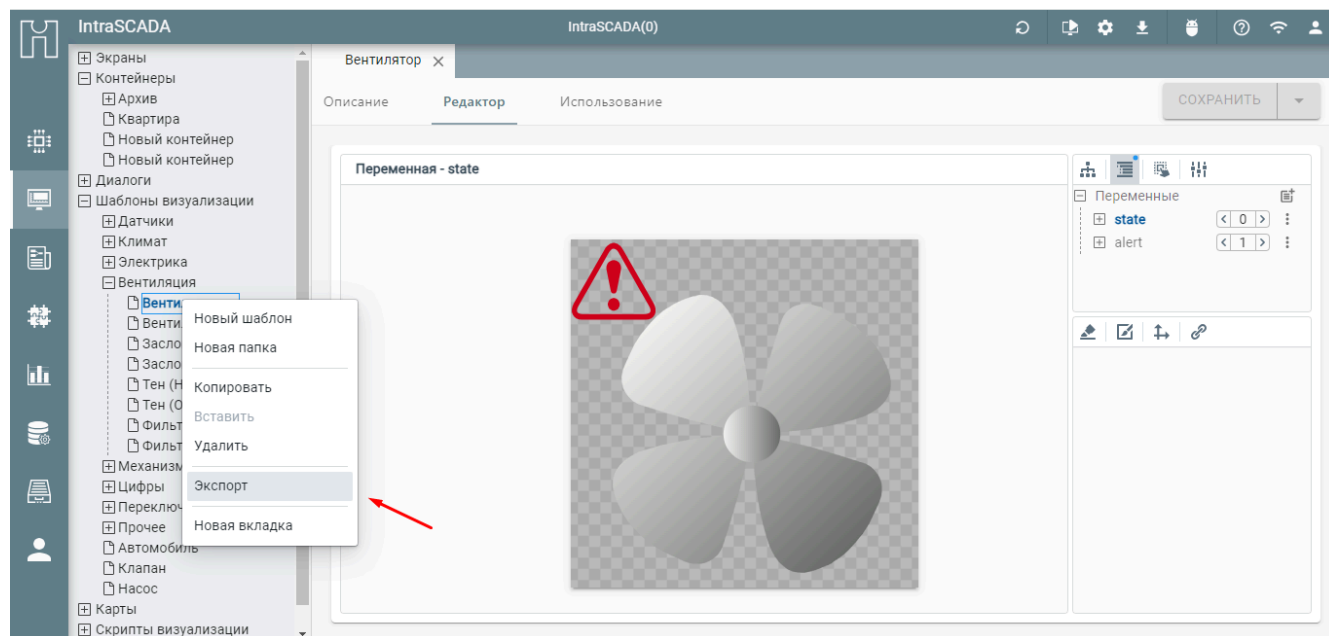


Для большинства устройств область действия может быть задана во весь размер шаблона. Но иногда может быть полезно создание нескольких активных областей для управления разными каналами одного устройства.

## Экспорт/Импорт

### Экспорт шаблона

Любой шаблон можно экспортировать с сервера на свой компьютер. Это бывает необходимо для использования его в других проектах или чтобы поделиться с коллегами. Нажмите правой кнопкой мыши в дереве шаблонов на нужный шаблон и выберите пункт "Экспорт". Шаблон будет сохранен на вашем компьютере с расширением `ihpack`



### Импорт шаблона

Для загрузки шаблона на сервер нужно нажать кнопку импорта в верхней строке главного меню системы и выбрать файл с расширением ihpasc. Система сама определяет содержимое файла ihpasc и, если там есть шаблон, поместит его в дерево шаблонов.



# Элементы

Элементы - это те компоненты, из которых создаются шаблоны, диалоги, контейнеры и экраны. Текст, изображение, кнопка, слайдер, поле ввода, график - все это элементы.

У элементов есть [общие свойства](#) и индивидуальные, присущие конкретному элементу.

Любое свойство элемента можно [связать](#) с устройством.

Наименование элемента	Описание	Примечание
<b>Basics</b>	<b>Основные элементы</b>	
<a href="#">Rectangle</a>	Прямоугольник	Прямоугольник имеет только <a href="#">общие свойства</a>
<a href="#">Circle</a>	Окружность	Этот элемент аналогичен прямоугольнику со скругленными углами. Окружность имеет только <a href="#">общие свойства</a>
<a href="#">Text</a>	Слово, строка или число	
<a href="#">Image</a>	Изображение (png, jpeg, svg, gif)	
<a href="#">Button</a>	Кнопка	Имеет множество свойств и действий. Управление устройствами, переключение экранов ...
<b>Interactive</b>	<b>Элементы ввода</b>	
<a href="#">Slider</a>	Слайдер	Предназначен для задания уставок или динамического изменения состояний устройства (диммеры, регулятор оборотов)
Input	Поле ввода	
Checkbox	Флажок	
Date Range	Выбор периода времени	
Calendar	Календарь	
<b>View</b>	<b>Элементы отображения информации</b>	
List	Список	



Table	Таблица	
Tree	Дерево	
<b>Charts</b>	<b>Графики</b>	
Chart Line	Однолинейный график	Отображает график для одного устройства
Chart Multiline	Многолинейный график	Отображает график для нескольких устройств на одной шкале
Chart Timeline	Таймлайн график	График в виде диаграммы Ганта
Chart Columns	Столбчатый график	График в виде столбцов
Chart Pie	Круговой график	График в виде круговой диаграммы
<b>Widgets</b>	<b>Виджеты</b>	
Iframe	Интерактивное окно	Возможно указать прямую ссылку для отображения
HTML	HTML компонент	Полностью кастомный html компонент
CCTV	Видео с камеры	Вывод изображения с камеры через плагин cctv
Device Settings	Настройки устройства	Используется для динамического создания полей настроек для устройств
Device Log	Журнал устройства	Используется для отображения журнала поведения устройства
Journal	Журнал	Вывод информации по событиям в зависимости от фильтров
Alert Journal	Журнал тревог	Вывод информации по аварийным / предупредительным событиям в зависимости от фильтров
Report	Отчет	Вывод отчетов в виде pdf

# Общие свойства

У элементов есть общие свойства, присущие любому элементу.

## 1. Оформление



Наименование	Описание	Примечание
<b>Фон</b>		
Цвет	Цвет фона	
<b>Рамка</b>		
Цвет	Цвет рамки	
Размер	Толщина рамки	
Радиус	Скругление рамки	
Стиль	Стиль рамки	
<b>Украсшение</b>		
Анимация		По умолчанию настроена анимация на вращение элемента. Для изменения анимации нужно нажать кнопку настроек: 
Тень		Для изменения свойств тени нужно нажать кнопку настроек: 
Непрозрачность		
Видимость		

## 2. Координаты



Наименование	Описание	Примечание
Позиция		
X	координата верхнего угла по горизонтали	
Y	координата верхнего угла по вертикали	
Ширина	Ширина	слева направо
Высота	Высота	сверху вниз
Ширина справа	Ширина	справа налево
Высота снизу	Высота	снизу вверх
Положение элемента	Положение элемента по оси Z	
Преобразовать		
Отразить по горизонтали	Перевернуть по горизонтали	
Отразить по вертикали	Перевернуть по вертикали	
Обрезать	Обрезать	обычно применяется для изображений
Повернуть	Повернуть	



# Привязки

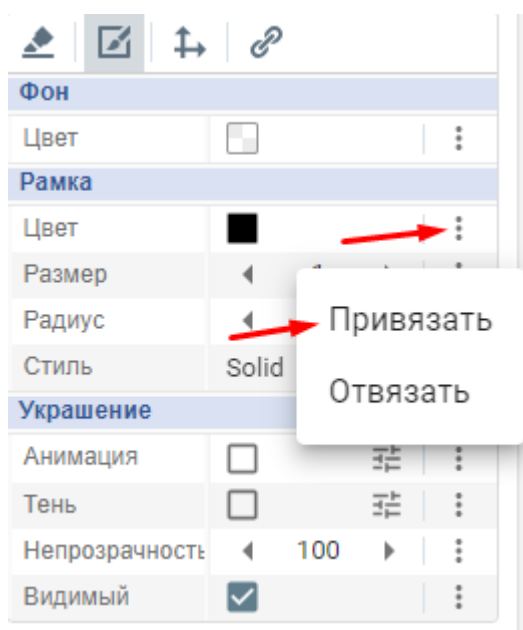
Каждое свойство элемента можно связать с устройством или с переменной клиента.

## Примеры

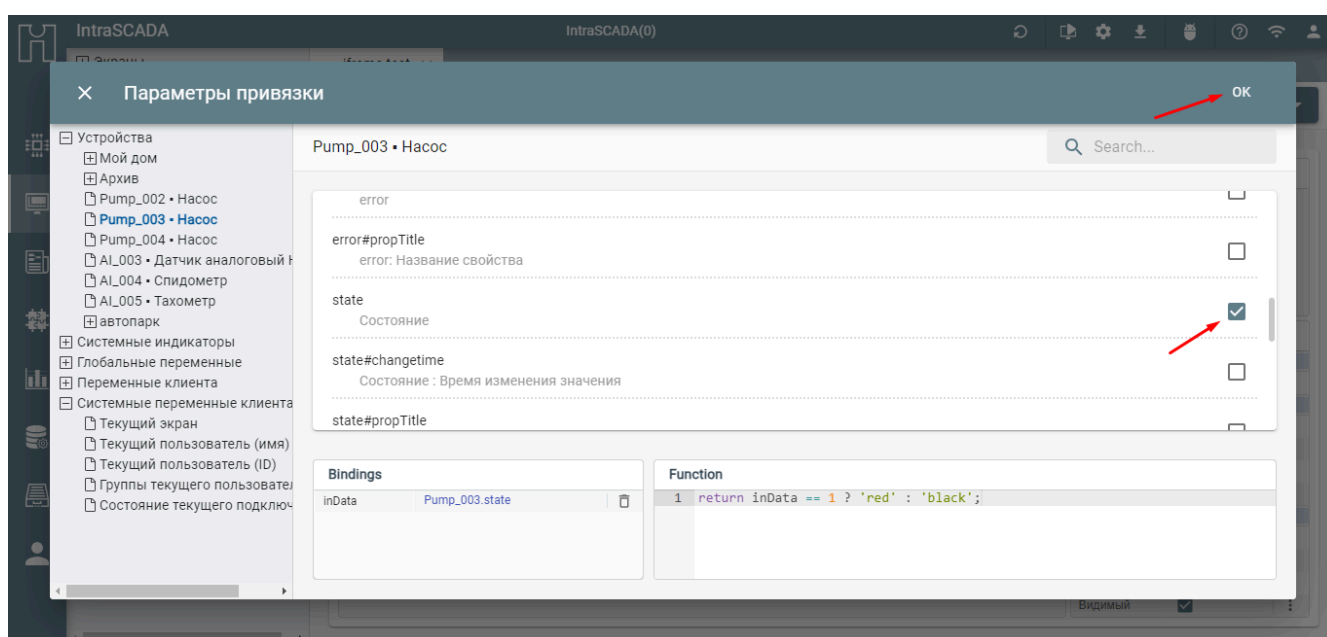
### Цвет рамки

Изменение цвета рамки в зависимости от состояния устройства.

В строке Цвет нажимаем кнопку привязки и выбираем Привязать:



Открывается окно привязки:



Выбираем устройство и его свойство и нажимаем кнопку ОК

В этом примере у насоса выбрано свойство state. При изменении этого свойства будет меняться цвет рамки.

В поле Function прописана тернарная условная операция:

```
return inData === 1 ? 'red' : 'black';
```

Если получаем входящее значение (inData) от свойства state равным 1, возвращаем red (красный), иначе black (черный)

Возвращаемое значение цвета можно прописать в шестнадцатеричном формате:

```
return inData === 1 ? '#FFF400' : '#FF0092';
```

```
return inData === 1 ? 'rgba(255,255,255,1)' : 'rgba(0,0,0,1)';
```

Если необходимо вернуть градиент

```
return inData === 1 ? 'linear-gradient(90deg, rgba(74,144,226,1) 0%, rgba(144,19,254,1) 100%)' : '#FFF400'
```

Если необходимо вернуть тень

```
return inData == 'vc123' ? '#2C2B2B inset 2px 2px 4px 0px' : '#2C2B2B 0px 0px 0px 0px';
```

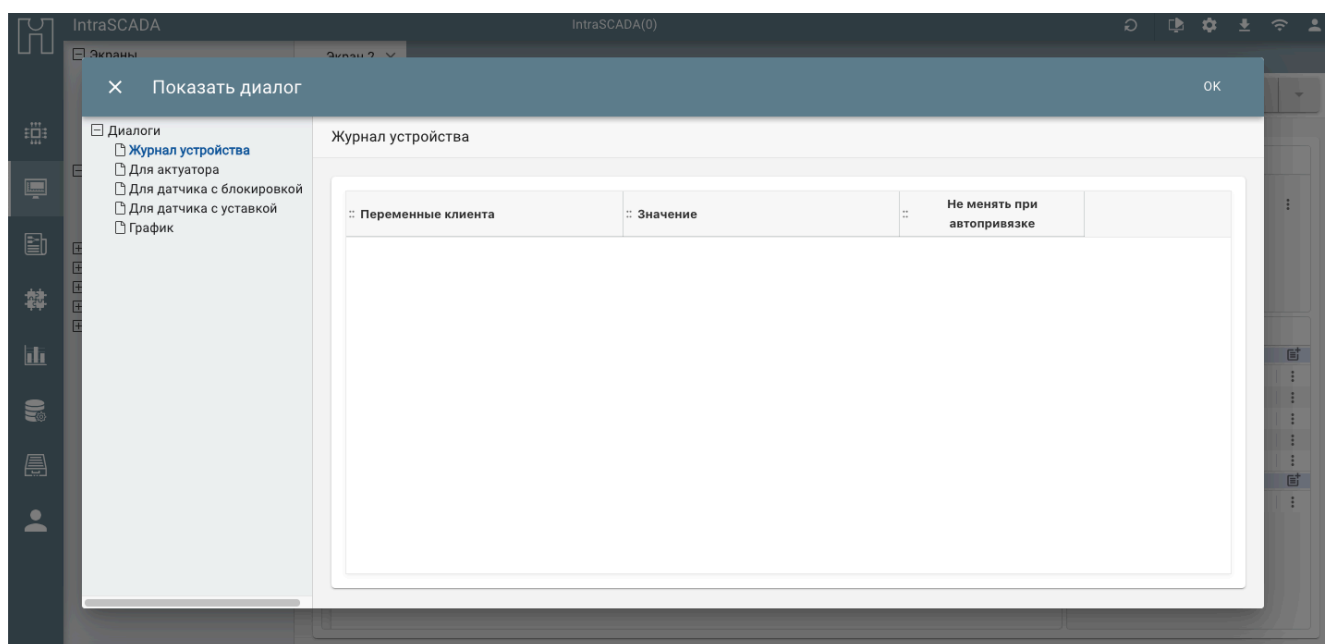
Если свойство представляет из себя список, то возвращать необходимо значения этого списка, например:

Наименование свойства	Список параметров	Возвращаемые значения
<b>Text</b>		
Стиль рамки	Dotted, Dashed, Solid, Groove, Ridge, Inset, Outset	dotted, dashed, solid, groove, ridge, inset, outset
Выравнивание по горизонтали	Слева, Центр, Справа	left, center, right
Выравнивание по вертикали	Вверх, Центр, Вниз	up, center, bottom
Шрифт	Arial, Serif, Sans-serif, Monospace	Arial, Serif, Sans-serif, Monospace
<b>Button</b>		
Ориентация	Сверху, Снизу, Центр, Слева, Справа	up, bottom, center, left, right
<b>Input</b>		
Тип значения	Строка, Число, Дата	string, number, date
Режим сохранения	Постоянный, Вне фокуса, Кнопка подтверждения	permanent, outfocus, button
<b>Checkbox</b>		
Размещение текста	-, Сверху, Слева, Справа, Снизу	none, top, left, right, bottom
<b>Autocomplete</b>		
Вариант	Minimal, Standart, Filled, Outlined	minimal, standard, filled, outlined
<b>Chart Line, Chart Multiline, Chart Timeline, Chart Columns, Chart Pie</b>		
Период	-, Минута, Час, День, Неделя, Месяц, Год, Пользовательский, Выберите переменную	-, minute, hour, day, week, month, custom, locals

Реалтайм	Не активно, Из БД, С Устройства, Интервал	disabled, db, raw, interval
Тип линии	Liner, Cubic, Cubic Monotone, Step	liner, cubic, cubic_monotone, step
Режим заливки	None, First up, First down, All up, All down	none, first_up, first_down, all_up, all_down
Легенда положение	-, Слева, Справа, Сверху, Снизу	none, left, right, up, down
Ось X положение	-, Снизу, Сверху	none, up, bottom

## Настройка локальных переменных

При вызове диалога есть возможность настройки локальных переменных.



**Переменная** выбирается из списка, **значение** можно выбрать из списка или ввести вручную.

*Добавлено:* v5.17.64

Добавлен флаг **Не менять при автопривязке**

# Rectangle

У элемента Rectangle есть только [общие свойства](#).






# Circle

Элемент Circle - это Rectangle с свойством Border->Radius равным 100.

У элемента Circle есть только [общие свойства](#).

# Text

Кроме [общих свойств](#) у элемента Text есть индивидуальные свойства:

   		
Text		
Value	Text 1	⋮
Color		⋮
Size	◀ 14 ▶	⋮
Bold	<input type="checkbox"/>	⋮
Italic	<input type="checkbox"/>	⋮
Font	Arial ▼	⋮
Horizontal	Center ▼	⋮
Vertical	Center ▼	⋮
Rotate	◀ 0 ▶	⋮

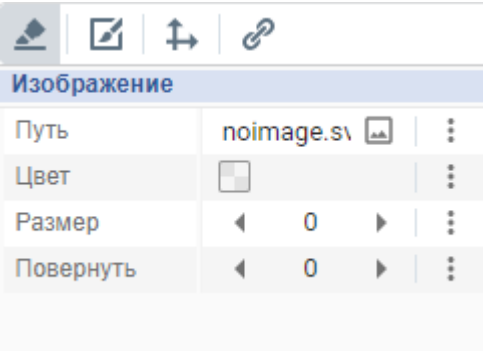
## Свойства элемента Text

Наименование	Описание	Примечание
<b>Text</b>		
Значение	Значение	Слово, строка или число
Цвет текста	Цвет текста	
Размер текста	Размер текста	
Высота строки	Высота строки	
Насыщенность	Стиль "Жирный"	
Курсив	Стиль "Наклонный"	
Шрифт	Шрифт текста	
Горизонталь	Выравнивание по горизонтали	Слева, Центр, Справа
Вертикаль	Выравнивание по вертикали	Вверх, Центр, Вниз
Повернуть	Поворот текста	



# Image

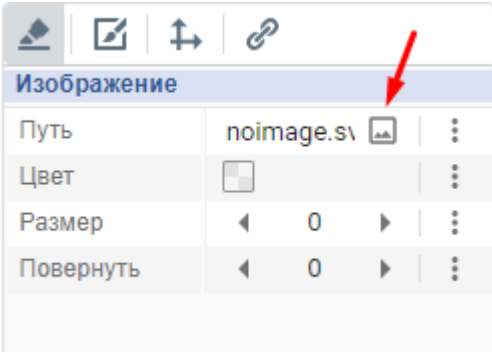
Кроме [общих свойств](#) у элемента Image есть индивидуальные свойства:



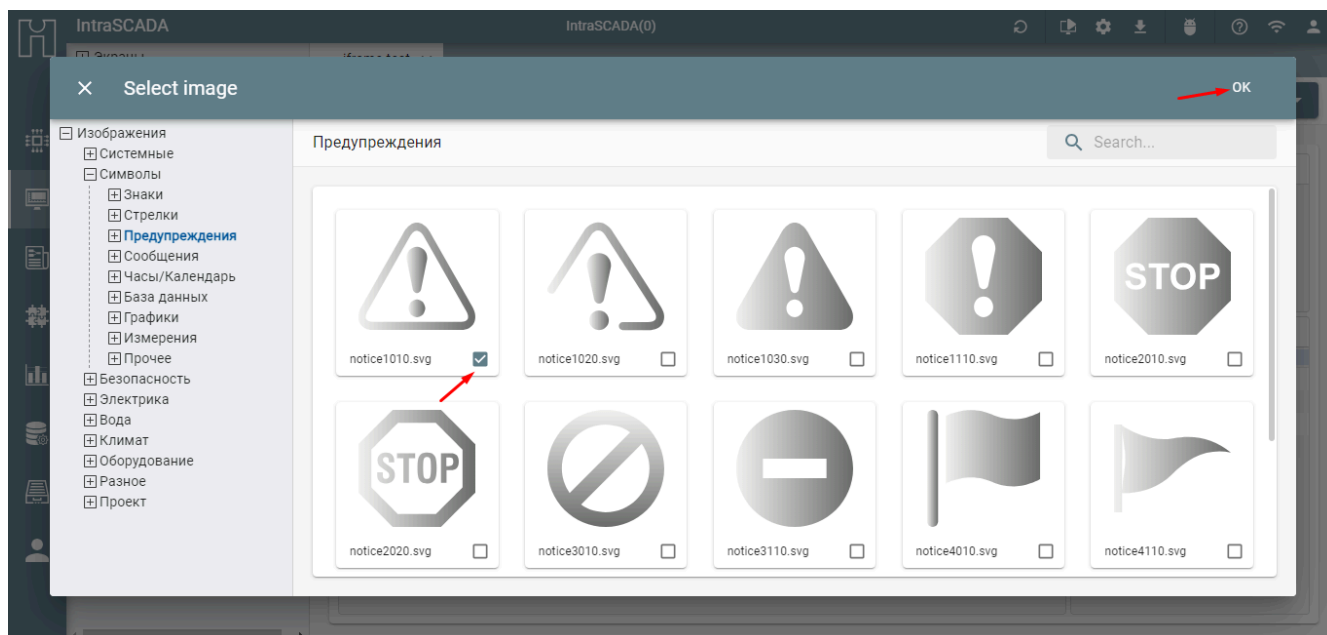
## Свойства элемента Image

Наименование	Описание
<b>Image</b>	
Путь	источник изображения
Цвет	Цвет изображения
Размер	Размер изображения
Повернуть	Поворот изображения

По умолчанию изображение имеет имя noimage.svg



Если нажать на кнопку, показанную выше, открывается окно выбора картинки из библиотеки изображений:



Нужно выбрать картинку и нажать кнопку OK

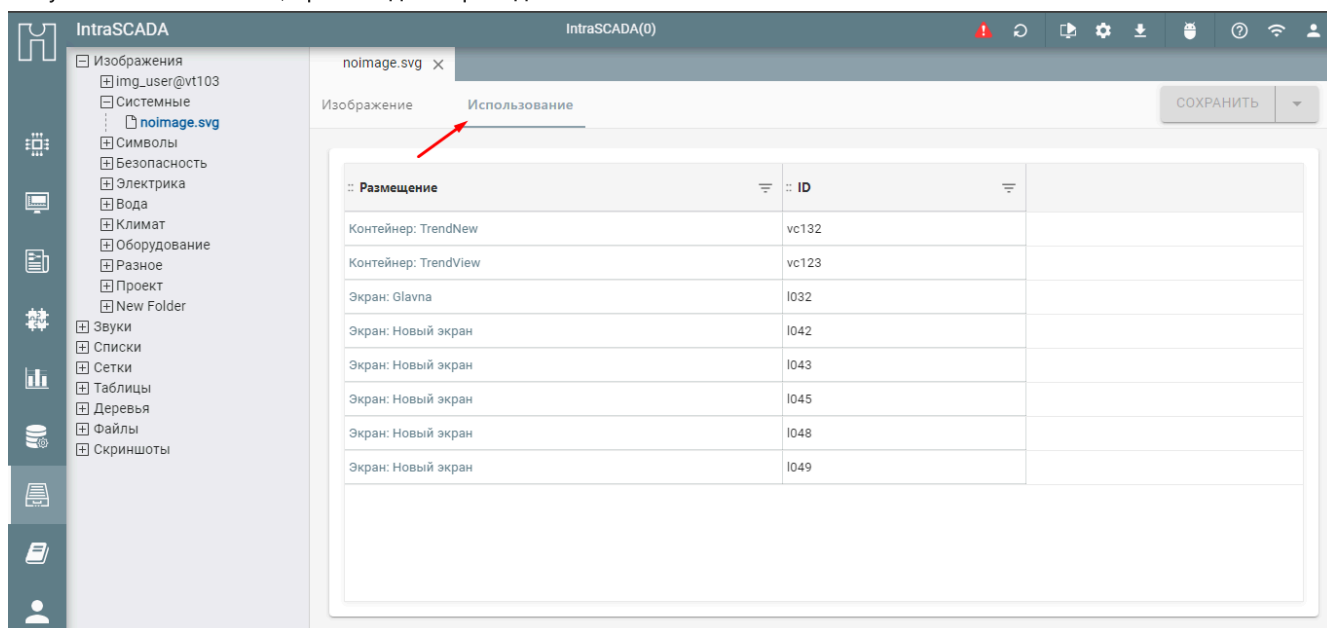
#### 📌 На заметку

В поле image вместо имени `noimage.svg` можно ввести адрес изображения из сети интернет.

Это дает возможность попробовать это изображение. Если оно вас устроит, тогда вы можете скачать его себе на сервер.

Добавлено v5.16.25

Добавлена вкладка **Использование** для Изображения. В таблице выводятся ссылки на визуальные компоненты, где это изображение используется (Диалоги, Шаблоны, Контейнеры, Экраны). Кликом мыши по ссылке на визуальный компонент, произойдет переход на этот компонент.



# Button

## Свойства элемента Button

Свойства элемента Button аналогичны свойствам элементов [Text](#) и [Image](#). Так же есть индивидуальные свойства

Наименование	Описание	Примечание
<b>Кнопка</b>		
Не активно	Кнопка станет некликабельной	
Ориентация	Ориентация расположения	Центр, Сверху, Снизу, Слева, Справа
Соотношение	Соотношение текста к изображению	
<b>Подтверждение</b>		
Текст	Текст подтверждения	
Цвет заголовка	Цвет заголовка подтверждения	
Цвет текста	Цвет текста	
Цвет кнопки	Цвет кнопки	
Цвет фона	Цвет фона	

## Команды элемента Button

Элемент Button может выполнять команды при нажатии кнопками мыши (касания тачскрина).

### Команды

Все действия мыши, такие как одинарный клик, двойной клик, долгий клик, нажатие клавиши, отпускание клавиши или клик правой клавишей мыши могут быть привязаны к следующим командам:

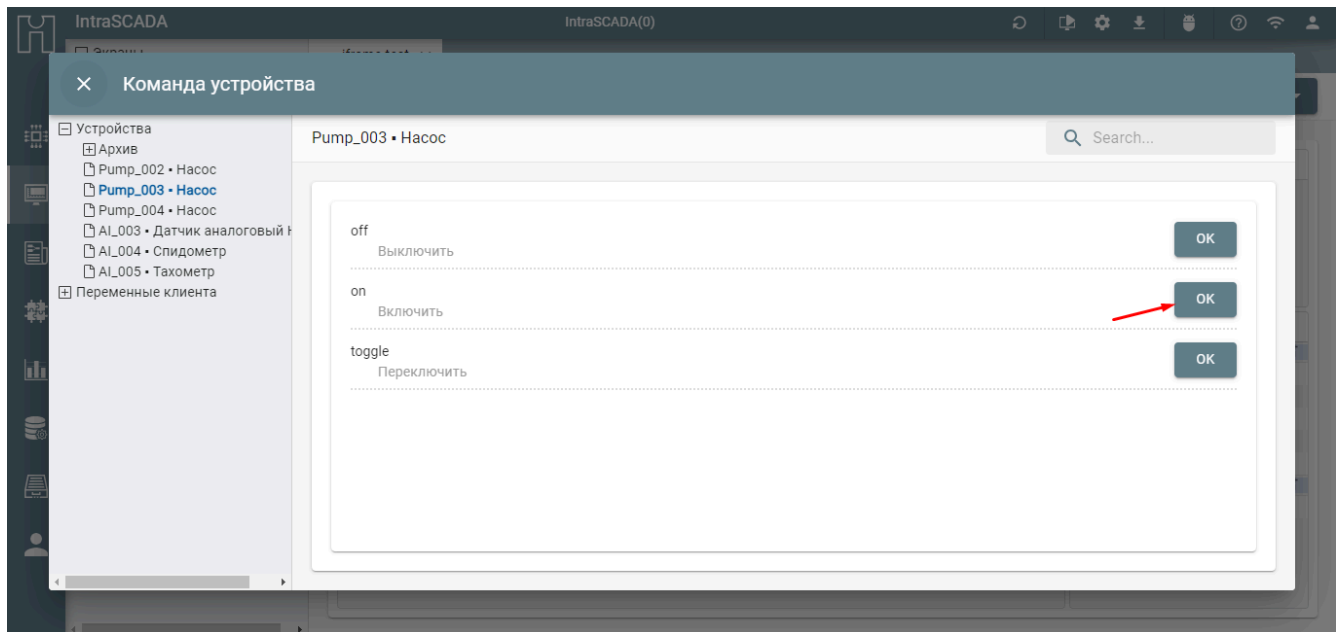
Наименование	Описание
Команда устройства	Выбрать команду устройства
Команда плагина	Отправить команду плагину
Команда элемента	Отправить команду графическому элементу
Запуск сценария	Выбрать сценарий для запуска
Пауза сценария	Поставить на паузу шаговый сценарий
Продолжение сценария после паузы	Возобновить работу сценария после паузы с ближайшей точки останова
Останов сценария	Выбрать сценарий для остановки
Переход на экран	Выбрать экран для перехода с возможностью присвоения значений клиентским переменным
Установить значение	Записать значение в свойство устройства
Показать диалог	Выбрать диалог с возможностью присвоения значений клиентским переменным
Закрыть диалоги	Закрыть все диалоги на экране
Скрипт визуализации	Выбрать скрипт визуализации для запуска
Запуск программы	Выбрать программу для запуска на стороне клиента в десктопном приложении (В браузере не работает)
Обновить экран	Команда Обновить экран
Переход на предыдущий экран	Команда Переход на предыдущий экран
Переход на главный экран	Команда Переход на главный экран
Полный экран	Команда открыть в полноэкранный режим
Снимок экрана	Вызвать окно для снимка экрана

Печать экрана      Вызвать окно для печати снимка экрана

Изменить пароль      Вызвать форму для изменения пароля пользователя

Выход      Команда Выход из сессии

Например, при нажатии кнопки нужно включить насос. Выбираем Команда Устройства. Открывается окно:



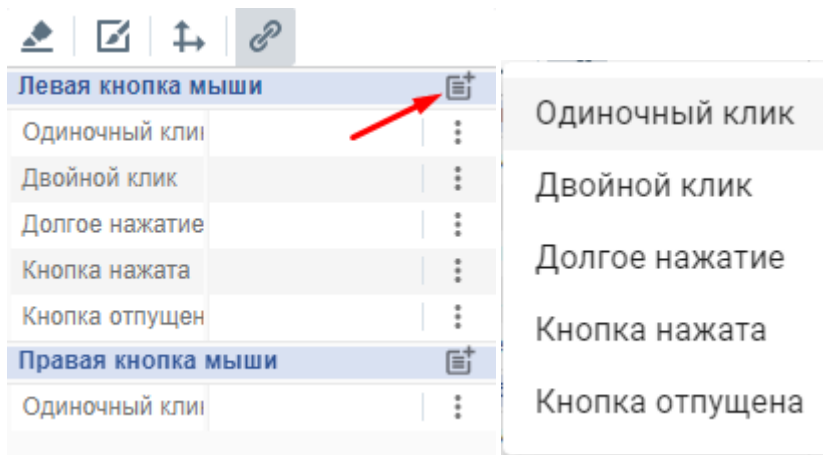
Выбираем команду и нажимаем ОК.

## Привязки

Для привязки команд можно задействовать любое действие левой кнопки мыши и одиночный клик правой кнопки мыши:

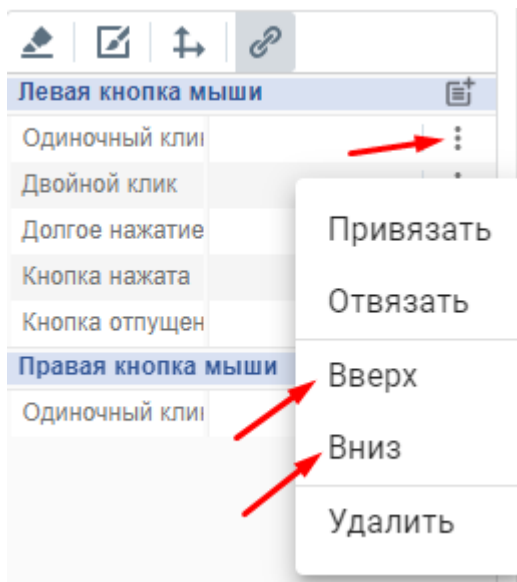
<b>Левая кнопка мыши</b>			
Одиночный кли			⋮
Двойной клик			⋮
Долгое нажатие			⋮
Кнопка нажата			⋮
Кнопка отпущен			⋮
<b>Правая кнопка мыши</b>			
Одиночный кли			⋮

При необходимости можно добавить действие:

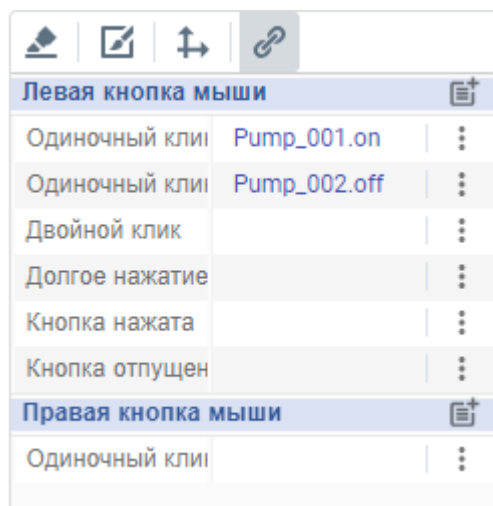


Например, нужно выполнить включение одного устройства и выключение другого одним нажатием кнопки мыши. Для этого достаточно добавить еще один Однократный клик и привязать к нему выполнение еще одной команды.

Очередность выполнения команд можно изменить, переместив строку с действием мыши. Вверх или вниз.



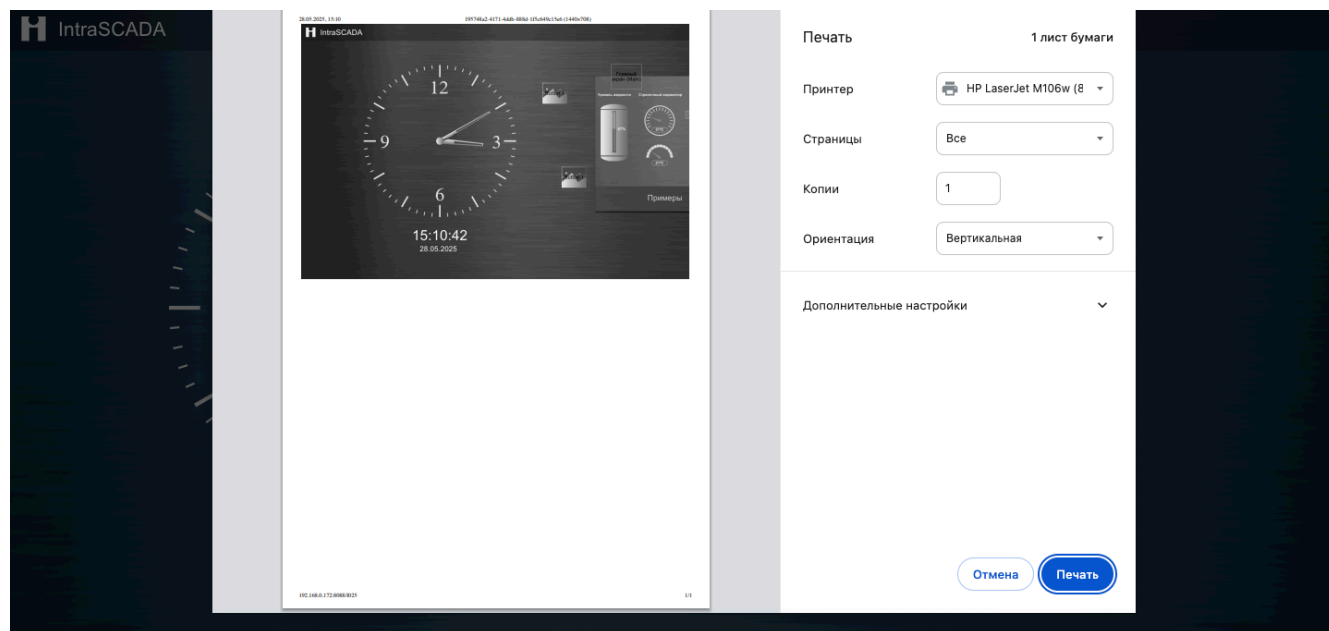
На рисунке ниже показано выполнение двух команд по одному клику левой кнопки мыши. Включить насос Pump\_001 и выключить насос Pump\_002



## Печать экрана

Добавлено v5.18.14

Начиная с версии 5.18.14 добавлена возможность печати снимка экрана



# Input

Input - активный элемент для ввода данных с пользовательского интерфейса, например:

- задания различных уставок (температура, давление, влажность, ...)
- выбора даты

Кроме [общих свойств](#), у элемента Input есть индивидуальные свойства:

## Свойства элемента Input



Наименование	Описание	Примечание
<b>Input</b>		
Не активно	Элемент Input становится некликабельным	
Полная ширина	Растянуть по ширине	
Полная высота	Растянуть по высоте	
Тип значения	Тип вводимых данных	Строка, Число, Дата
Режим сохранения	Условия передачи данных на сервер	Постоянный, Вне фокуса, Кнопка подтверждения
Цифр после запятой	Количество цифр после запятой	Используется в разных случаях, например для указания дробных чисел
Префикс	Текст перед значением	
Суффикс	Текст после значения	
Подсказка	Текст в области ввода значения	
Размер Текста	Размер текста	
Горизонталь	Расположение текста по горизонтали	Слева, Центр, Справа
Цвет текста	Цвет текста	
Min(Число)	Минимальное значение	Появляется при выборе типа значения : Число.
Max(Число)	Максимальное значение	Появляется при выборе типа значения : Число.
<b>Календарь</b>		
Цвет месяца	Цвет месяца	

Цвет дня	Цвет дня	
Цвет кнопок	Цвет кнопок	
Цвет выделения	Цвет выделения	
Цвет основы	Цвет основы	
<b>Подтверждение</b>		
Текст	Текст подтверждения	При задании числа, либо строки, открывается всплывающее окно с подтверждением действия : "Вы уверены?", можно написать любую фразу в разделе "Текст подтверждения".
Цвет заголовка	Цвет заголовка подтверждения	
Цвет текста	Цвет текста	
Цвет кнопки	Цвет кнопки	
Цвет фона	Цвет фона	
<b>Предупреждение Min-Max</b>		
Текст	Текст предупреждения	Текст будет появляться при выборе числа вне диапазона Min-Max в виде всплывающего окна
Цвет заголовка	Цвет заголовка предупреждения	
Цвет текста	Цвет текста	
Цвет кнопки	Цвет кнопки	
Цвет фона	Цвет фона	

Примеры :

Обратите внимание! Всплывающие окна **Подтверждение** и **Предупреждение Min-Max**, работают только если есть привязка

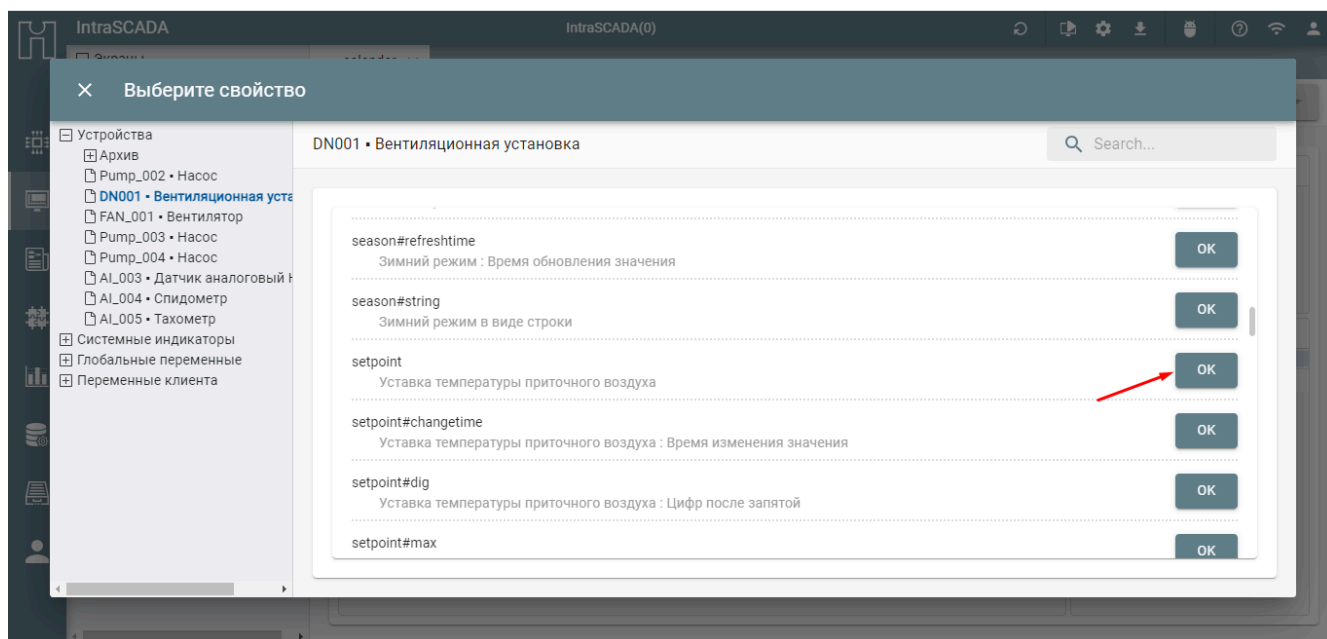
## Input Число

- Тип значения выбираем **Число**
- Указываем Min-Max значения (В нашем случае от 1 до 50)
- По желанию редактируем цвет, размер текста, суффиксы, префиксы и т.д
- Вводим Текст подтверждения и текст предупреждения Min-Max
- По желанию также можно выбрать разные цвета

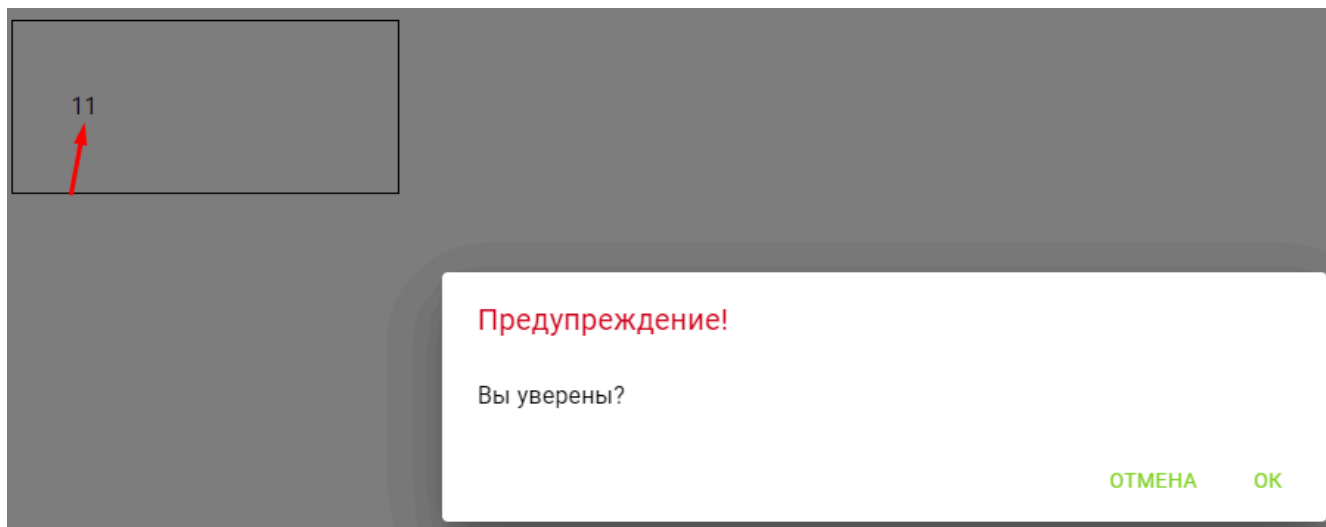
Получился элемент Input с числом :



- Привяжем его к setpoint Вентиляционной установки :

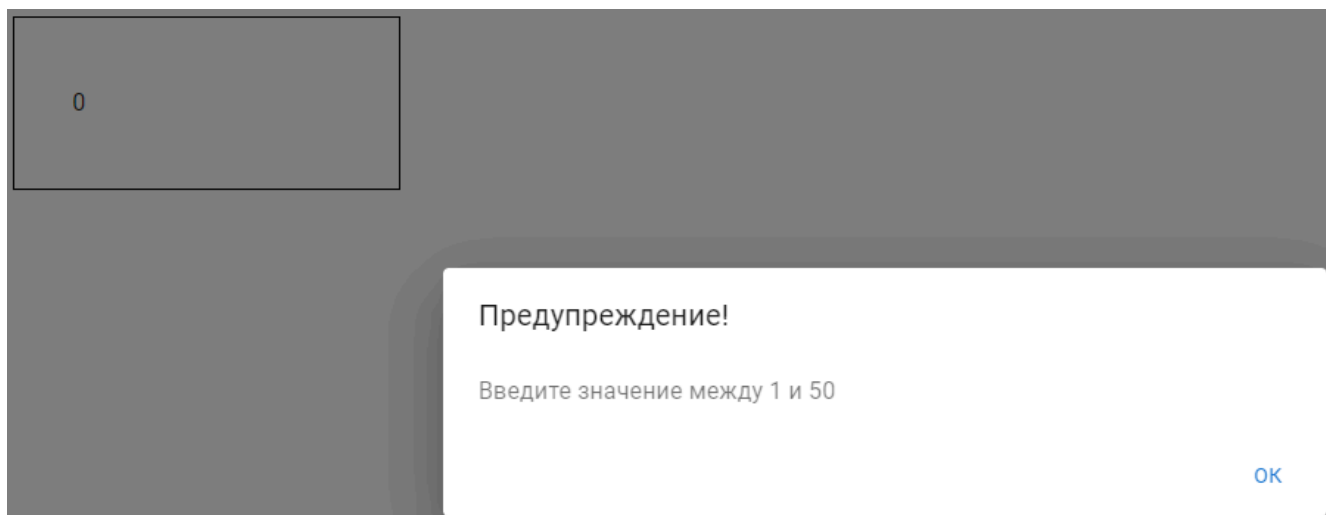


- Выводим на экран, и указываем значение от 1 до 50 :



В результате, при указании значения, у нас вывелось всплывающее окно с текстом, указанным в поле Подтверждение - Текст. Жмём ОК, если хотим задать указанное значение, Отмена - если нет.

- Пробуем указать значение вне диапазона Min-Max(1-50) :



В результате выводится окно с предупреждением : "Введите число от Min до Max значения", указанного нами

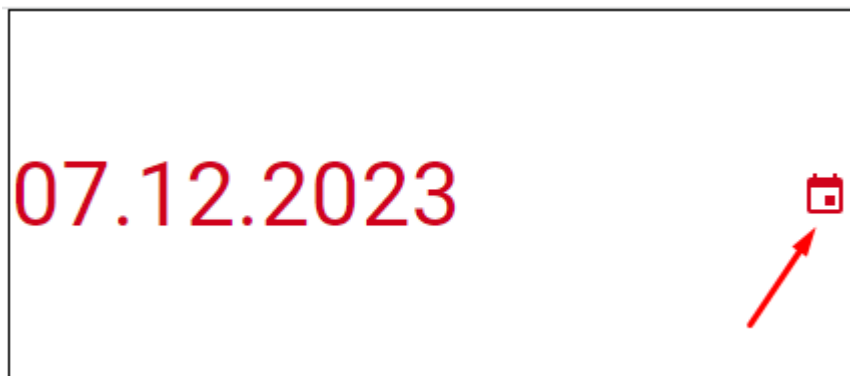
Данный Input позволяет нам задавать уставки.

## Input Строка

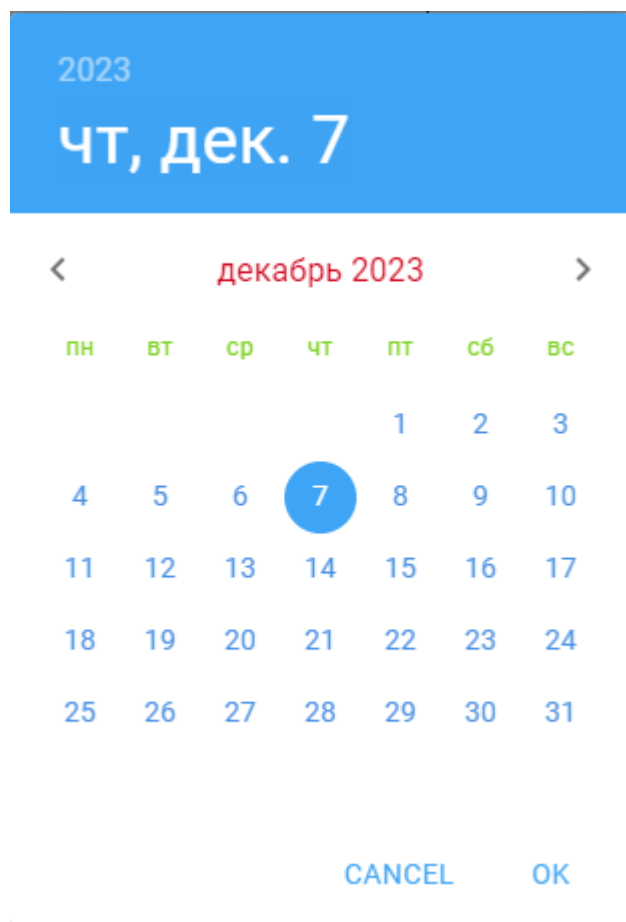
Настройка Input Тип значения - **Строка** идентична настройке Типа значения - **Число**, за исключением указания Min-Max.

## Input Календарь

- Тип значения выбираем **Дата**
- Настраиваем оформление(Цвет, размер)
- Выводим на экран



При нажатии на кнопку выводится окно с выбором даты



*Добавлено v5.17.2*

Добавлена виртуальная клавиатура для элемента Input, улучшающая удобство при вводе данных с пользовательского интерфейса.

Данная функция описана [по ссылке](#).

# Slider

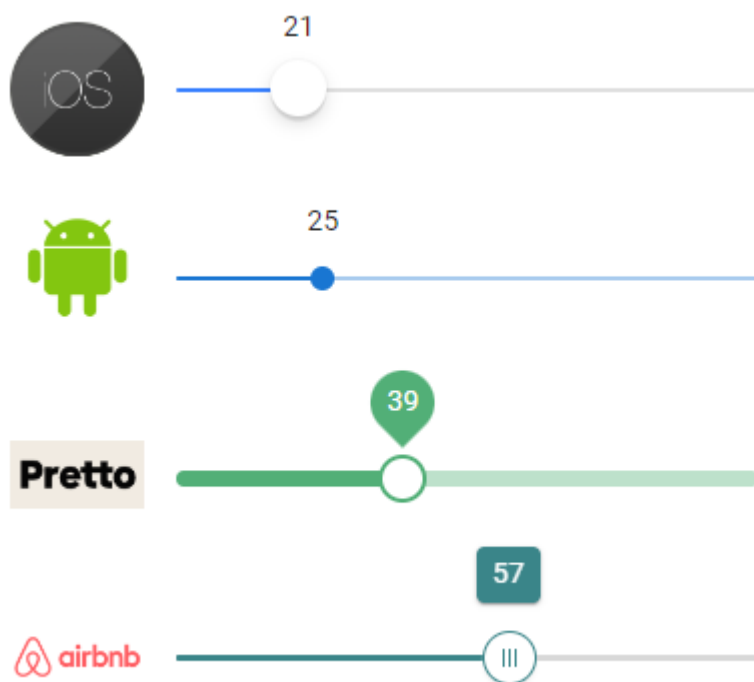
Слайдер - активный элемент для:

- управления аналоговыми актуаторами (диммируемые светильники, плавная регулировка оборотов двигателей ...)
- задания различных уставок (температура, влажность ...)

В системе IntraSCADA присутствуют как готовые (Android, iOS, Pretto, Airbnb) слайдеры, так и полностью настраиваемые (Custom) слайдеры. С помощью элемента Custom Slider можно создать любой слайдер, который вам нужен.

## Готовые слайдеры

- Slider Android
- Slider iOS
- Slider Pretto
- Slider Airbnb



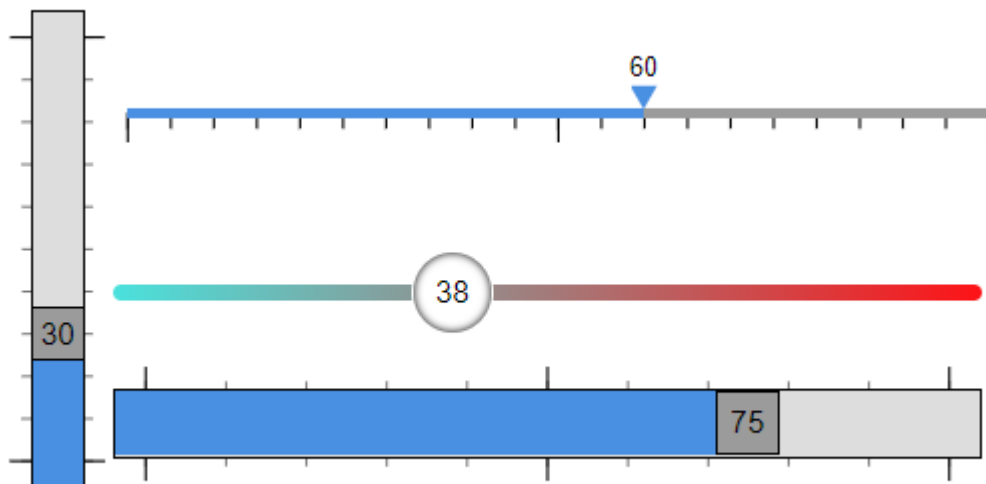
Индивидуальные свойства готовых слайдеров

Слайдер			
Min	◀ 0 ▶	⋮	
Max	◀ 100 ▶	⋮	
Скрыть значения	<input type="checkbox"/>	⋮	
Размер текста	◀ 14 ▶	⋮	
Цвет текста	<input type="color"/>	⋮	
Цвет линии сле	<input type="color"/>	⋮	
Цвет линии спр	<input type="color"/>	⋮	
Цвет ползунка	<input type="color"/>	⋮	
Подтверждение			
Текст			
Цвет заголовка	<input type="color"/>	⋮	
Цвет текста	<input type="color"/>	⋮	
Цвет кнопки	<input type="color"/>	⋮	
Цвет фона	<input type="color"/>	⋮	

## Custom Slider

Custom Slider - Полностью настраиваемый слайдер. Он бывает Вертикальным и Горизонтальным.

### Примеры Custom Slider



Кроме [общих свойств](#), у элемента Custom Slider есть индивидуальные свойства:

### Свойства элемента Custom Slider

Наименование	Описание	Примечания
<b>Слайдер</b>		
Без управления	Отключает управление слайдером	
Ориентация	Вертикальная или горизонтальная	
Min	Минимальное значение	
Max	Максимальное значение	
<b>Дорожка</b>		
Режим	С отступом/Заполнить	Заполняет дорожку слайдера по высоте, либо делает её с отступом
Смещение	Смещение по оси X/Y	Горизонтальный смещается по оси Y(Вверх и Вниз), Вертикальный - по оси X(Влево и Вправо)
Размер	Устанавливает размер шкалы слайдера	
Step	Устанавливает шаговое значение ползунка	При установке значения 2, шкала будет идти через 2 числа: 0,2,4,6,8..., при установке 5 - 0,5,10,15,20..., и так далее
Цвет фона 1	Цвет фона при заполненной шкале	
Цвет фона 2	Цвет фона при пустой шкале	
<b>Граница дорожки</b>		
Цвет	Цвет границы дорожки	
Размер	Размер границы дорожки	



100 - Закруглённая.

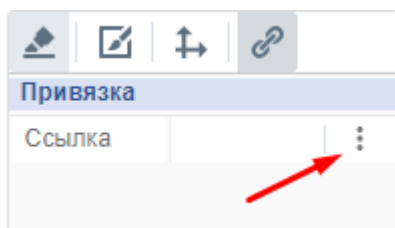
Стиль	Стиль границы дорожки	
Ползунок		
Смещение	Смещение по оси X/Y	Горизонтальный смещается по оси Y(Вверх и Вниз), Вертикальный - по оси X(Влево и Вправо)
Размер	Устанавливает размер ползунка	
Цвет	Устанавливает цвет ползунка	
Тень	Вкл/Выкл тень	Так же тень можно настроить через редактор слева от привязки
Граница ползунка		
Цвет	Устанавливает цвет границы(рамки) ползунка	
Размер	Устанавливает размер границы(рамки) ползунка	
Радиус	Устанавливает радиус границы(рамки) ползунка	При значении 0, граница будет квадратная. При значении 50 - Полукруглая. При значении 100 - Закруглённая.
Стиль	Устанавливает стиль границы(рамки) ползунка	
Изображение ползунка		
Ориентация	Изменяет ориентацию изображения ползунка	Сверху, Снизу, Центр, Слева, Справа
Путь	Путь изображения	Выбрать изображение ползунка можно либо из уже имеющихся, либо загрузить своё

Цвет	Устанавливает цвет изображения ползунка	
Размер	Устанавливает размер изображения ползунка	
Повернуть	Поворачивает изображение ползунка под разными углами	
Текст		
Смещение	Смещает текст ползунка по оси Y	
Цвет текста	Изменяет цвет текста ползунка	
Размер текста	Изменяет размер текста ползунка	
Насыщенность	Текст становится "Жирным"	
Курсив	Текст становится наклонным	
Шрифт	Изменяет шрифт текста	
Повернуть	Поворачивает текст под разными углами	
Шкала		
Отображение	Изменяет отображение шкалы	<b>1 вариант</b> - Шкала снизу(Для горизонтального слайдера), Шкала справа(Для вертикального слайдера). <b>2 вариант</b> - Шкала сверху(Для горизонтального слайдера), Шкала слева(Для вертикального слайдера). <b>3 вариант</b> - Шкала и снизу, и сверху(Для горизонтального слайдера), Шкала и слева, и справа(Для вертикального слайдера).
Step	Изменяет шаговое значение шкалы	При установке значения 2, шкала будет идти через 2 числа: 0,2,4,6,8..., при установке 5 - 0,5,10,15,20..., и так далее

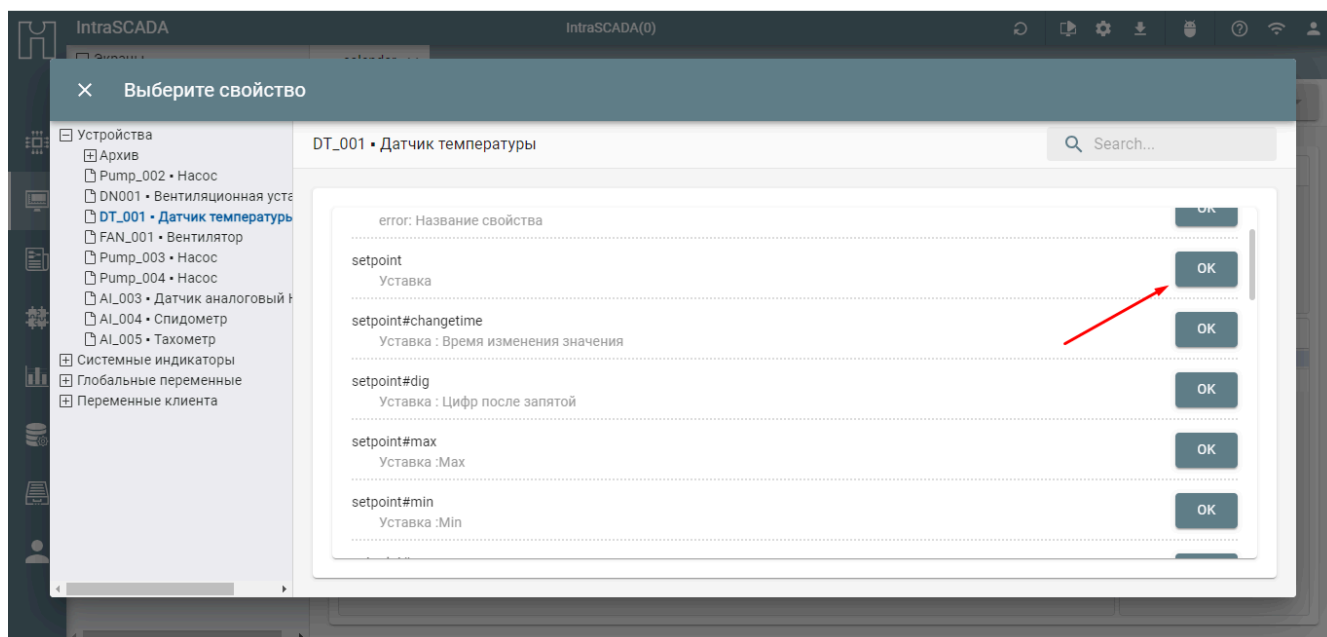
Цвет	Изменяет цвет шкалы	
Позиция	Изменяет позицию шкалы	От центра - шкала будет идти от центра слайдера, От дорожки - шкала будет идти от границы слайдера.
Смещение	Смещает шкалу в зависимости от <b>Отображения</b>	
Толщина	Изменяет толщину шкалы	
Длина	Изменяет длину шкалы в зависимости от <b>Позиции</b>	
<b>Изображение шкалы</b>		
Путь	Путь к изображению шкалы	Можно выбрать своё изображение шкалы вместо стандартного
Цвет	Цвет изображения	
Размер	Размер изображения	

## Привязки

Привязка слайдера к устройству выполняется при нажатии кнопки Привязки:



Например, для уставки температуры, привязываем слайдер к свойству **setpoint** Датчика температуры:



# Checkbox

Checkbox - элемент для выбора бинарного состояния 0 (false) или 1 (true)

Кроме [общих свойств](#) у элемента Checkbox есть индивидуальные свойства:

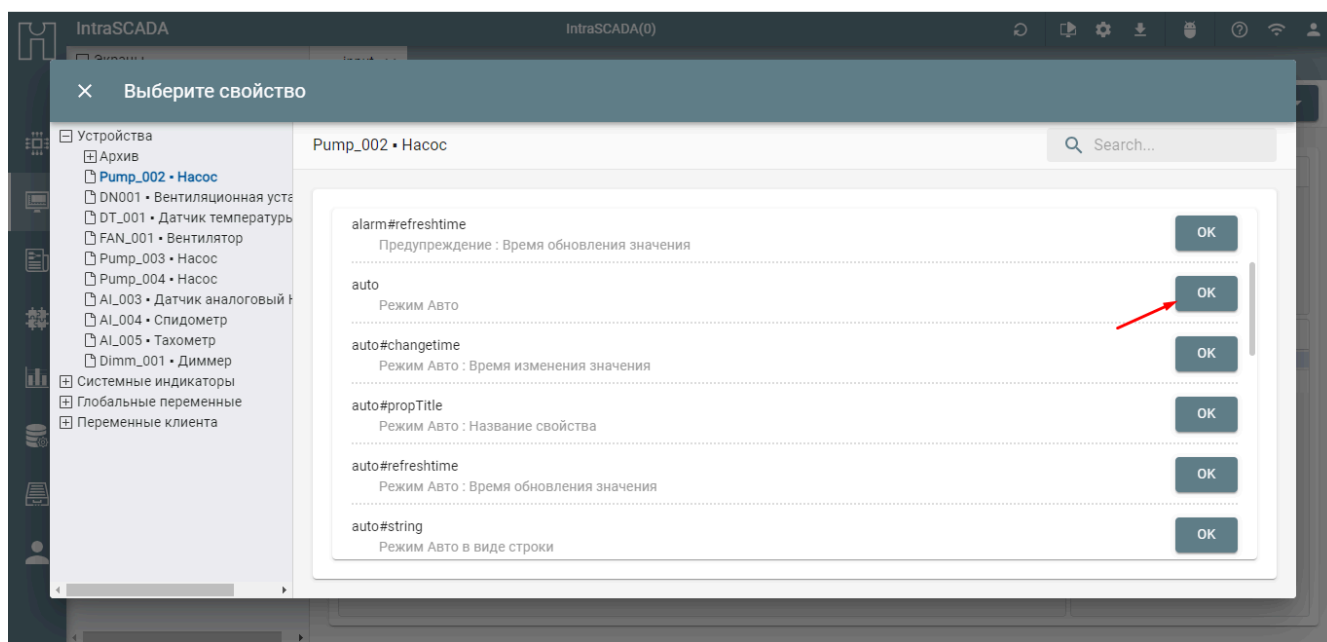
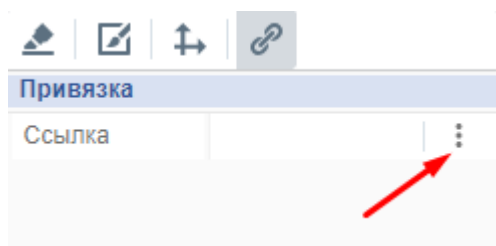
## Свойства элемента Checkbox

Наименование	Описание	Примечание
<b>Checkbox</b>		
Заголовок	Текст заголовка	
Размер текста	Размер текста заголовка	
Размещение текста	Расположение текста заголовка	None, Сверху, Справа, Слева, Снизу
Цвет заголовка	Цвет заголовка	
Цвет в неактивном состоянии	Цвет в состоянии 0 (false)	
Цвет в активном состоянии	Цвет в состоянии 1 (true)	
<b>Подтверждение</b>		
Текст	Текст подтверждения	Например : "Вы уверены?"
Цвет заголовка	Цвет заголовка подтверждения	
Цвет текста	Цвет текста подтверждения	
Цвет кнопки	Цвет кнопки подтверждения	
Цвет фона	Цвет фона подтверждения	

## Примеры использования

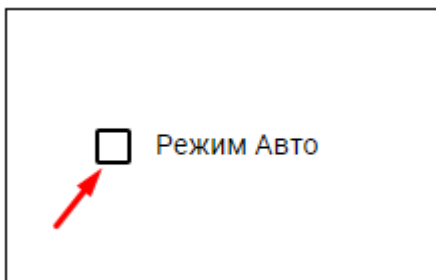
Пример : Создадим Checkbox для Включения/Выключения режима Авто устройства.

- Создадим элемент Checkbox (Добавить элемент - Interactive - Checkbox)
- Пишем в заголовке "Режим Авто"
- Выбираем цвет заголовка, цвет в активном и неактивном состоянии.(В примере цвет активного состояния выбран зелёным, неактивного - чёрным)
- В тексте подтверждения пишем "Вы уверены?"
- Настраиваем Фон, Рамку, размеры элемента.
- Привязываем Checkbox к Режиму Авто (auto) насоса.

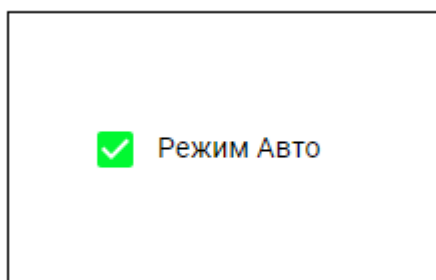


- Выводим элемент на Экран

При нажатии на Checkbox, появится всплывающее окно с подтверждением действия :



При нажатии ОК, включится Режим Авто насоса.



При нажатии на Checkbox еще раз, режим авто будет отключен.

# Date Range

Date Range - это компонент для выбора периода времени с помощью календаря.

Кроме [общих свойств](#) у элемента Date Range есть индивидуальные свойства:

## Свойства элемента Date Range

Наименование	Описание
<b>Date Range</b>	
Цвет текста	Цвет текста
Размер текста	Размер текста
Цвет кнопки	Цвет кнопки
Размер кнопки	Размер кнопки

## Привязки

У компонента Date Range существует две привязки:

- start - начальная дата в формате timestamp
- end - конечная дата в формате timestamp

Для привязки доступны только клиентские переменные.

Привязка	
Переменная start	⋮
Переменная end	⋮

Один из сценариев использования данного компонента - это отображение данных на графике за определенный период, через присваивание начала и конца периода клиентским переменным



# Calendar

Calendar - это компонент для выбора даты из календаря.

Кроме [общих свойств](#) у элемента Calendar есть индивидуальные свойства:

## Свойства элемента Calendar

Наименование	Описание
<b>Calendar</b>	
Цвет месяца	Цвет месяца
Цвет недели	Цвет недели
Цвет дня	Цвет дня
Цвет кнопок	Цвет кнопок
Цвет выделения	Цвет выделения

## Привязки

У компонента Calendar существует две привязки:

- Ссылка - выбор переменной клиента для записи даты в формате timestamp
- Скрипт при выборе - выбор скрипта для запуска после выбора даты

Привязка		
Ссылка		⋮
Скрипт при выборе		⋮
Выбор по умолчанию	Сегодня	▼

Для привязки доступны только клиентские переменные и скрипты визуализации.

Так же возможно настроить **Выбор по умолчанию** компонента Calendar при загрузке экрана:

- Сегодня

- Начало недели
- Начало месяца

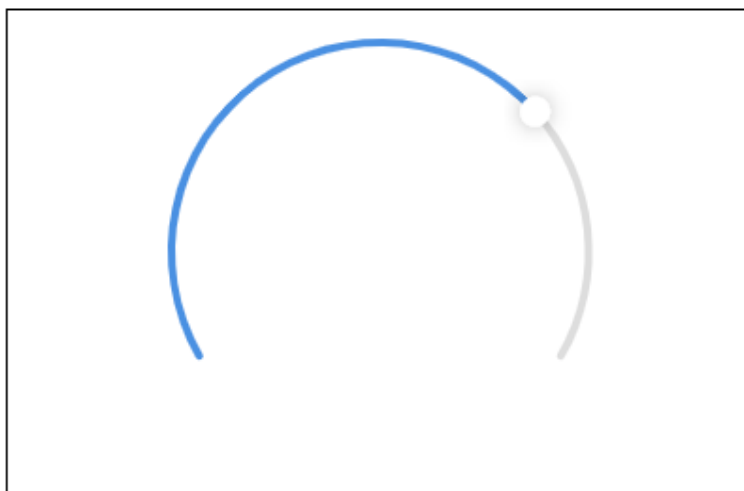
Один из сценариев использования данного компонента - это отображение данных в списке от даты выбранной в календаре

# Sector

Сектор - активный элемент для:

- управления аналоговыми актуаторами (диммируемые светильники, плавная регулировка оборотов двигателей ...)
- задания различных уставок (температура, влажность ...)
- отображения аналогового значения

## Примеры элемента Sector



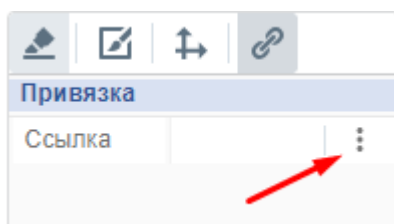
Кроме [общих свойств](#), у элемента Sector есть индивидуальные свойства:

## Свойства элемента Sector

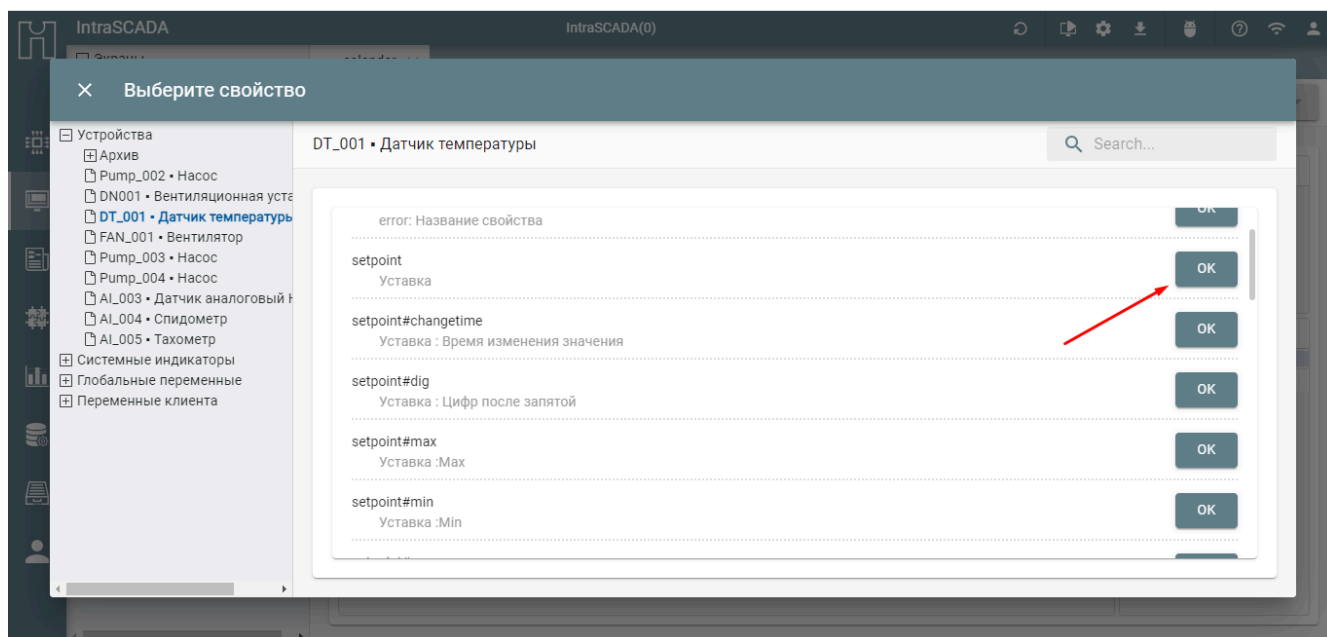
Наименование	Описание	Примечания
<b>Сектор</b>		
Не активно	Отключает управление сектором	
Min	Минимальное значение	
Max	Максимальное значение	
Начальный угол	Значение начального угла	
Конечный угол	Значение конечного угла	
Размер	Размер элемента	
Направление	Изменяет направление элемента	
Скругление	Скругление элемента	
Размер ползунка	Размер ползунка	
Цвет ползунка	Цвет ползунка	
Ширина трека	Ширина дорожки сектора	
Цвет 1	Основной цвет элемента	
Цвет 2	Дополнительный цвет элемента	

## Привязки

Привязка сектора к устройству выполняется при нажатии кнопки Привязки:



Например, для уставки температуры, привязываем сектор к свойству **setpoint** Датчика температуры:



# List

List - данный визуальный компонент выводит информацию в виде списка, созданного с помощью списков в [Ресурсах](#)

Кроме [общих свойств](#) у элемента List есть индивидуальные свойства:

## Свойства элемента List

Наименование	Описание
<b>Список</b>	
Горизонталь	Слева, справа, по центру
Размер шрифта	Размер шрифта
Цвет текста элемента	Цвет текста элемента
Цвет разделителя	Цвет разделителя
Цвет фона элемента	Цвет фона элемента
Цвет при наведении	Цвет при наведении
Цвет при выборе	Цвет при выборе

## Привязки

У компонента List существует три привязки:

- Ссылка - выбор списка из **Ресурсов**
- Скрипт при выборе - выбор скрипта для запуска после выбора элемента списка
- Переменная результата - выбор клиентской переменной для записи результата выбора элемента списка. Присваивается id элемента списка.

Привязка		
Ссылка		⋮
Скрипт при выборе		⋮
Переменная результата		⋮
Выбор по умолчанию	Первый элемент	▼

Для привязки доступны только клиентские переменные и скрипты визуализации.

Также возможно настроить **Выбор по умолчанию** компонента List при загрузке экрана:

- Первый элемент
- Последний элемент

## Пример создания

### Ручное заполнение

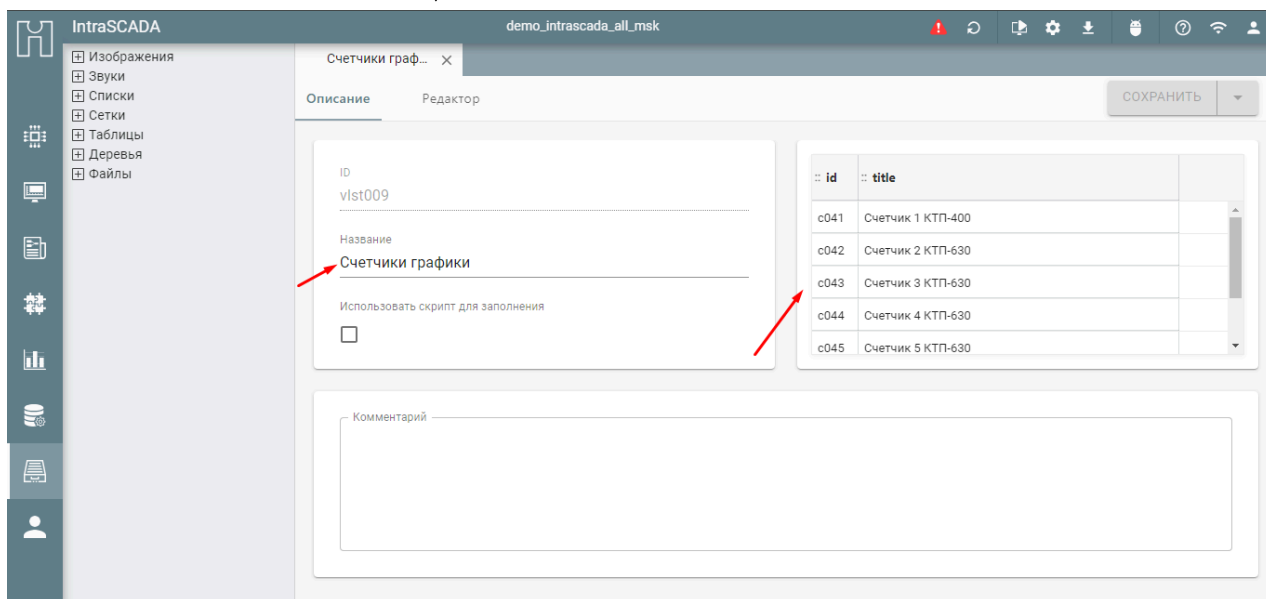
- Создаём новый список в разделе Ресурсы - Списки

The screenshot shows the IntraSCADA application window. On the left, a sidebar contains a tree view with categories like 'Изображения', 'Звуки', 'Списки', 'Сети', 'Таблицы', 'Деревья', and 'Файлы'. Under 'Списки', there is a 'Новый список' button highlighted with a red arrow. The main area shows a table titled 'Списки' with columns 'ID', 'Название', and 'Комментарий'. The table contains 12 rows of data, including 'Шкафы IT', 'ДГУ', 'ИБП', 'ГЗУ', 'Счетчики электричества', 'Счетчики графики', 'Снимки с камеры', 'Новый список', 'Список шаговых сценариев', 'Test', and 'Новый список'.

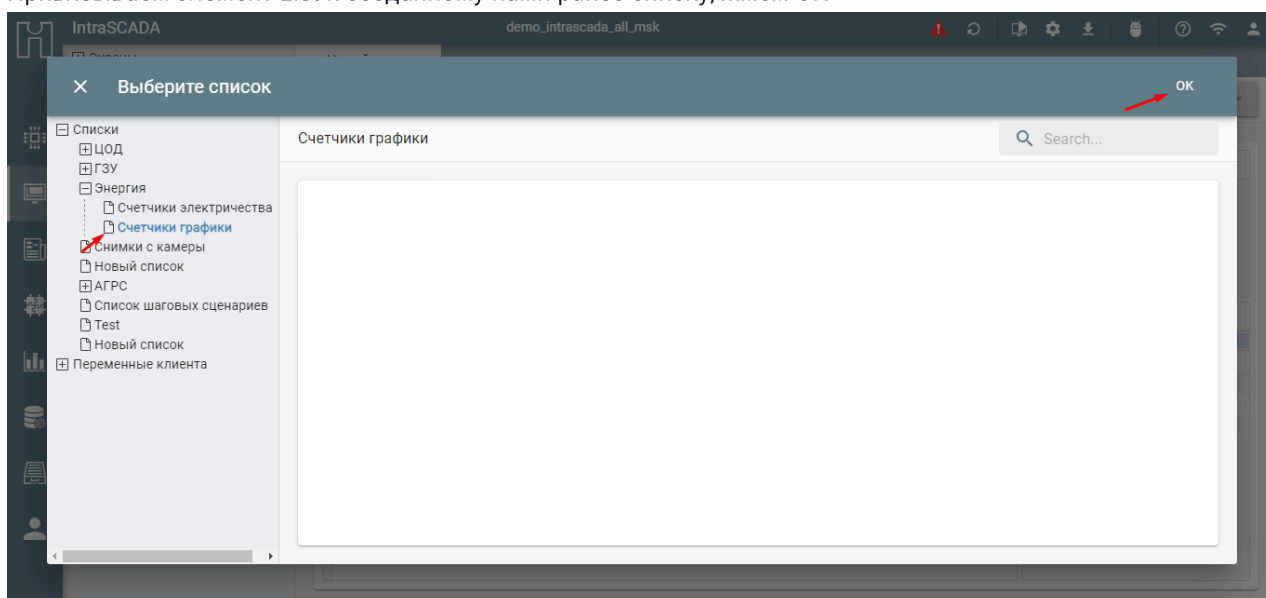
ID	Название	Комментарий
vlist002	Шкафы IT	
vlist003	ДГУ	
vlist004	ИБП	
vlist005	ГЗУ	
vlist007	Счетчики электричества	
vlist009	Счетчики графики	
vlist006	Снимки с камеры	
vlist008	Новый список	
vlist010	Список шаговых сценариев	
vlist011	Test	
vlist012	Новый список	

- Пишем название списка
- Скрипт для заполнения не используем

- В поле 'id' пишем id элемента списка, в поле 'title' пишем название элемента списка



- Добавляем на экран элемент List(Добавить элемент - View - List)
- Привязываем элемент List к созданному нами ранее списку, жмём OK



В результате на экране отобразится созданный нами список

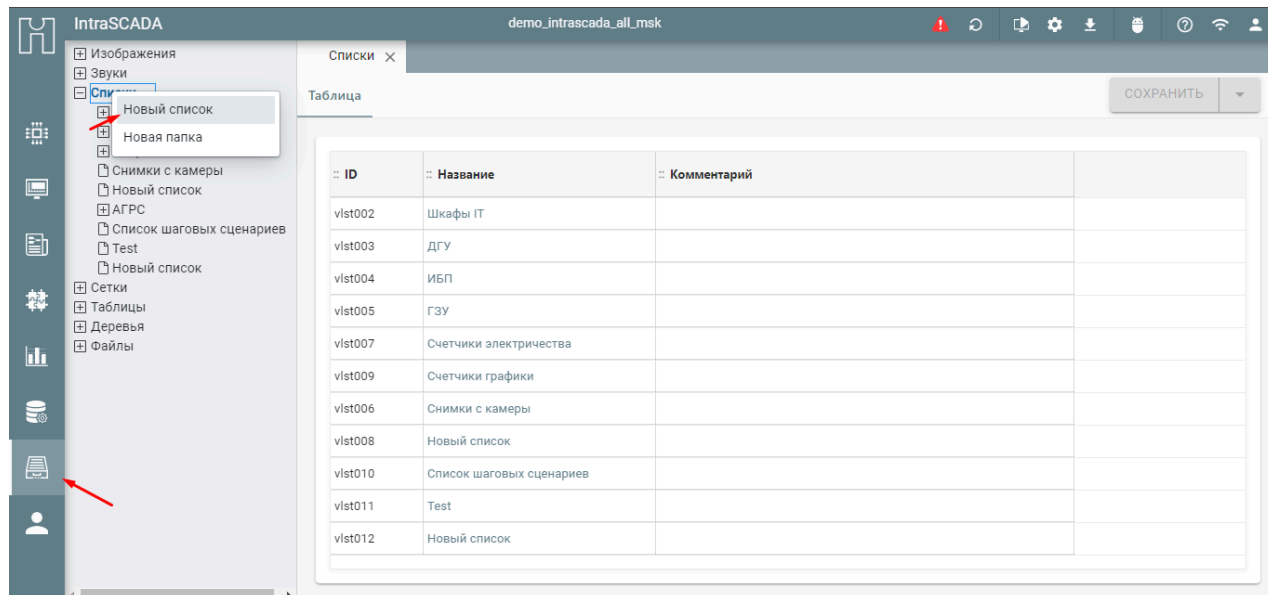
Счетчик 1 КТП-400
Счетчик 2 КТП-630
Счетчик 3 КТП-630
Счетчик 4 КТП-630
Счетчик 5 КТП-630
Счетчик 6 КТП-2500
Счетчик 7 КТП-630



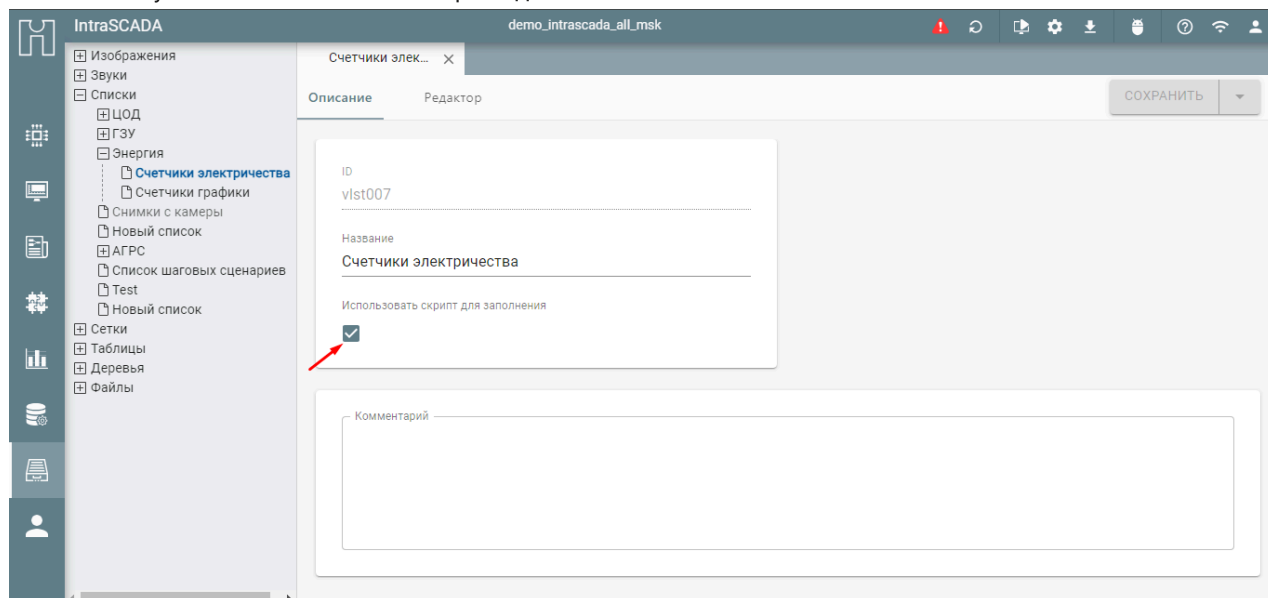
В привязках также доступны 'Скрипт при выборе' и 'Переменная результата'. Более подробно про них можно прочитать в разделе [Ресурсы](#)

## Заполнение с помощью скрипта

- Создаём новый список в разделе Ресурсы - Списки



- Пишем название списка
- Ставим галку в поле 'Использовать скрипт для заполнения'



- Переходим на вкладку Редактор

Стандартный скрипт выглядит так :

```
/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function({ local, context }, core, debug) {
  const data = [];
  data.push({id:'t1', title:'Test 1' });
  data.push({id:'t2', title:'Test 2' });
  return { data };
};
```

При его использовании на экране отобразится 2 элемента списка : Test 1 и Test 2

Test 1
Test 2

Скрипт можно изменять в зависимости от ваших нужд

#### Примеры использования других скриптов

Скрипт для отображения списка устройств

```
/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context, source}, core, debug) {
  const data = await core.getListFromTree('devdevices', 'dg076');
  data.forEach(item => {
    let title = item.title;
    let j = title.indexOf('■');
    if (j>0) item.title = title.substr(j+2);
  });
  return {data};
}
```

```
data.forEach(item => {
  let title = item.title;
  let j = title.indexOf('■');
  if (j>0) item.title = title.substr(j+2)
```

Данная команда убирает из названий устройств всё, что идет до элемента '■' Так выглядит список с использованием данного скрипта :

КТП-630 (2)
КТП-630 (3)
КТП-630 (4)
КТП-630 (5)
КТП-2500 (6)

Если мы уберем из скрипта команду

```
data.forEach(item => {
  let title = item.title;
  let j = title.indexOf('■');
  if (j>0) item.title = title.substr(j+2)
```

Список будет выглядеть так :

Meter_002 ■ КТП-630 (2)
Meter_003 ■ КТП-630 (3)
Meter_004 ■ КТП-630 (4)
Meter_005 ■ КТП-630 (5)
Meter_006 ■ КТП-2500 (6)

# Autocomplete

Autocomplete - данный визуальный компонент выводит информацию в виде всплывающего списка, созданного с помощью списков в [Ресурсах](#)

Кроме [общих свойств](#) у элемента Autocomplete есть индивидуальные свойства:

## Свойства элемента Autocomplete

Наименование	Описание	Примечание
<b>Список с автодополнением</b>		
Вариант	Вариант отображения	Minimal, Standart, Filled, Outlined
Заголовок	Текст заголовка	
Подсказка	Текст в области ввода значения	
Размер Текста	Размер текста	
Цвет Текста	Цвет текста	
Основной цвет	Базовый цвет	
Цвет при наведении	Цвет при наведении	
Активный цвет	Активный цвет	
Цвен подложки	Цвет подложки	
<b>Стили элементов списка</b>		
Цвет списка	Цвет списка	
Цвет текста	Цвет текста	
Цвет фона	Цвет фона	
Цвет при выборе	Цвет при выборе	
Цвет при наведении	Цвет при наведении	

## Привязки

У компонента Autocomplete существует три привязки:

- Ссылка - выбор списка из **Ресурсов**

- Скрипт при выборе - выбор скрипта для запуска после выбора элемента списка
- Переменная результата - выбор клиентской переменной для записи результата выбора элемента списка. Присваивается id элемента списка.

Привязка		
Ссылка		⋮
Скрипт при выборе		⋮
Переменная результата		⋮
Выбор по умолчанию	Первый элемент	▼

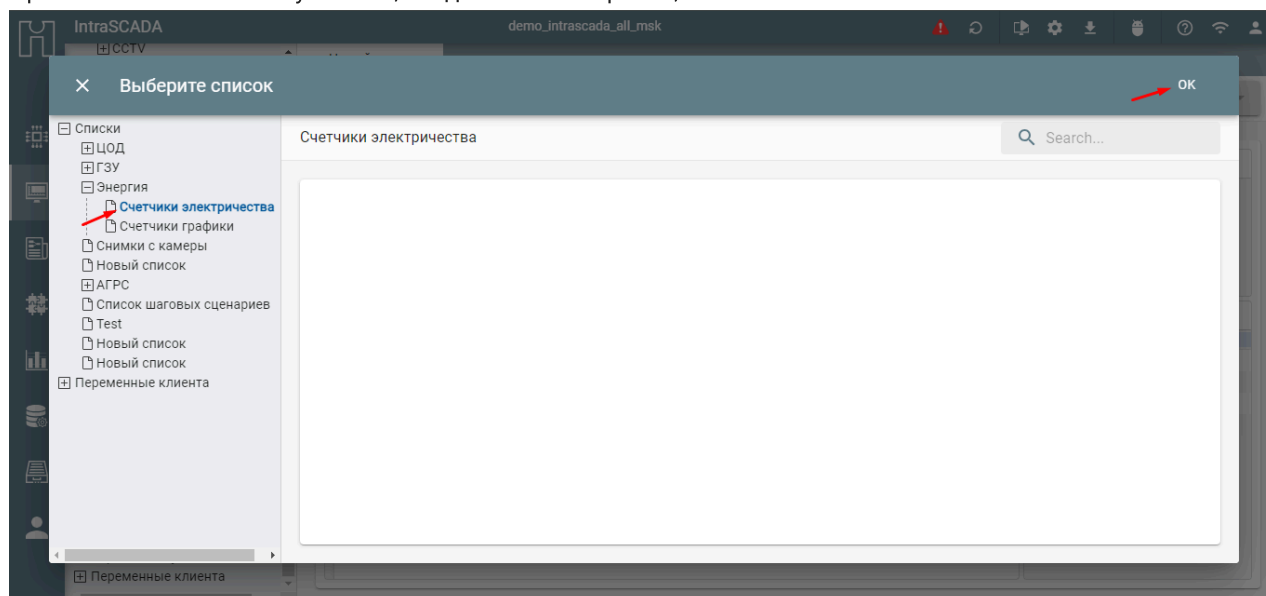
Также возможно настроить **Выбор по умолчанию** компонента Autocomplete при загрузке экрана:

- Первый элемент
- Последний элемент

## Пример использования

В примере будет использоваться список Счетчиков Электричества, созданный нами ранее в разделе [View -> List](#)

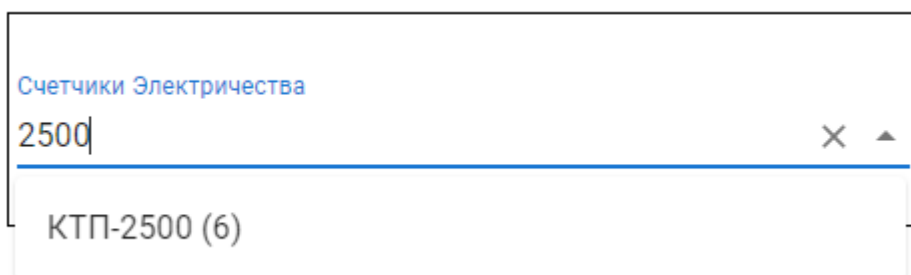
- Добавляем элемент Autocomplete на экран(Добавить Элемент - View - Autocomplete)
- Пишем Заголовок, меняем Цвет, Размер и т.д.
- Привязываем к элементу список, созданный нами ранее, жмём ОК



В результате у нас на экране отобразится всплывающий список с нашими данными:



В поле можно вписать название интересующего нас устройства, и список отфильтруется



# Table

Table - данный визуальный компонент выводит информацию в виде таблицы, созданного с помощью таблицы в [Ресурсах](#)

Кроме [общих свойств](#) у элемента Table есть индивидуальные свойства:

## Свойства элемента Table



Наименование	Описание
<b>Таблица</b>	
Цвет фона шапки	Цвет фона шапки
Цвет текста шапки	Цвет текста шапки
Цвет иконок шапки	Цвет иконок шапки
Цвет сетки	Цвет сетки
Цвет выбранной строки	Цвет выбранной строки
<b>Стандартное сообщение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Предупреждение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Аварийное сообщение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Панель фильтра</b>	
Цвет панелей	Цвет панелей
Цвет фона	Цвет фона
Цвет текста	Цвет текста

## Привязки

У компонента Table существует три привязки:

- Ссылка - выбор таблицы из **Ресурсов**
- Скрипт при выборе - выбор скрипта для запуска после выбора элемента таблицы
- Переменная результата - выбор клиентской переменной для записи результата выбора элемента таблицы. Присваивается id элемента таблицы.

Привязка		
Ссылка		⋮
Скрипт при выборе		⋮
Переменная результата		⋮
Выбор по умолчанию	Первый элемент	▼

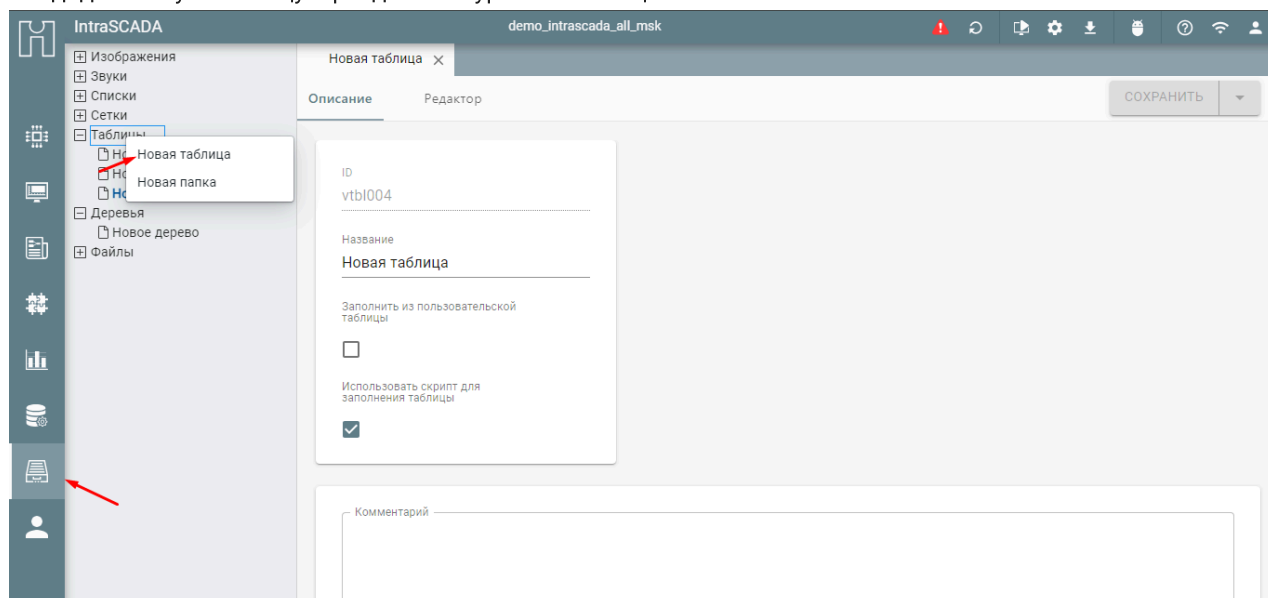
Также возможно настроить **Выбор по умолчанию** компонента Table при загрузке экрана:

- Первый элемент
- Последний элемент

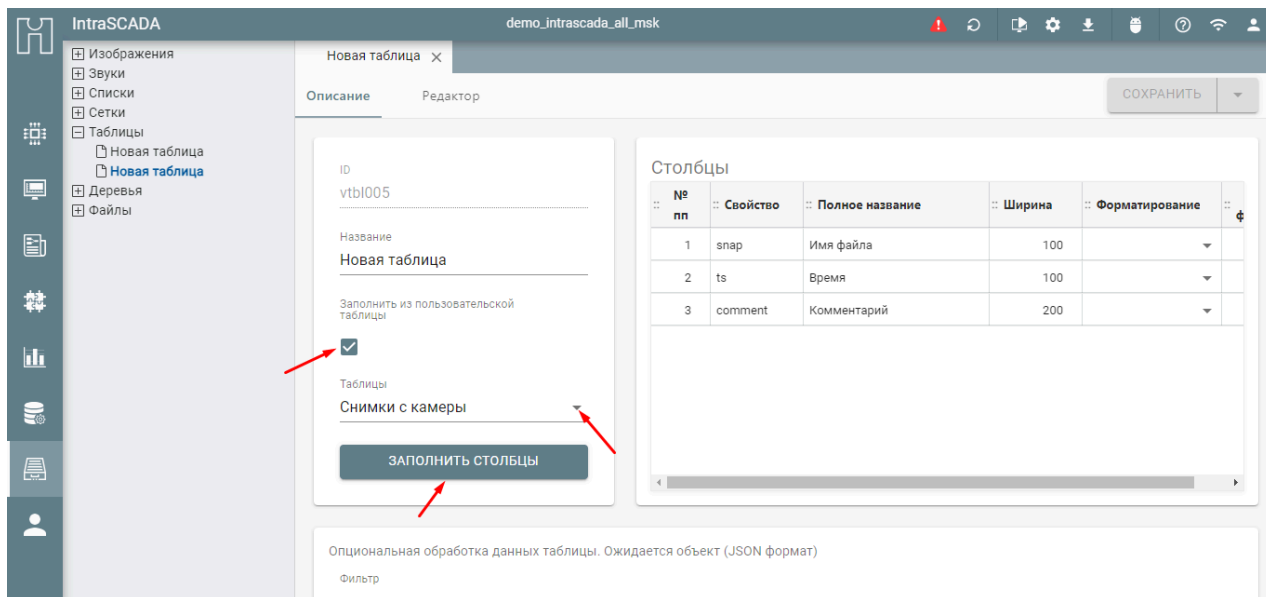
## Пример использования

### Заполнение с помощью пользовательской таблицы

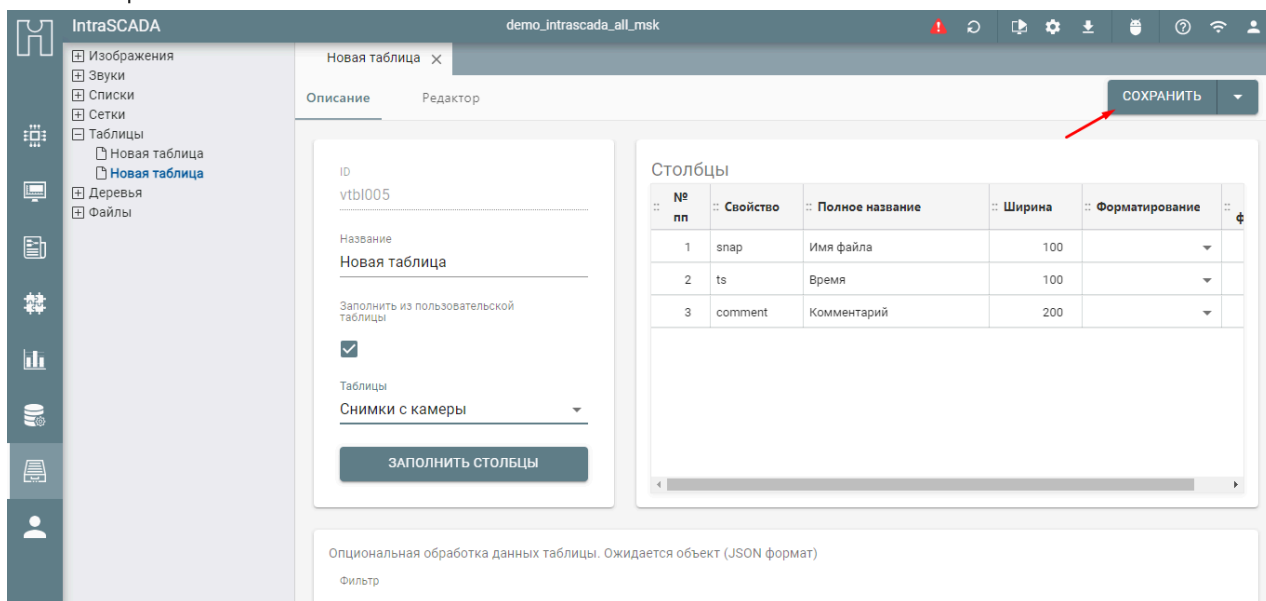
- Для начала нужно создать пользовательскую таблицу в разделе БД -> Пользовательские таблицы, более подробную информацию про создание пользовательских таблиц можно получить по [ссылке](#)
- Создадим новую таблицу в разделе Ресурсы -> Таблицы



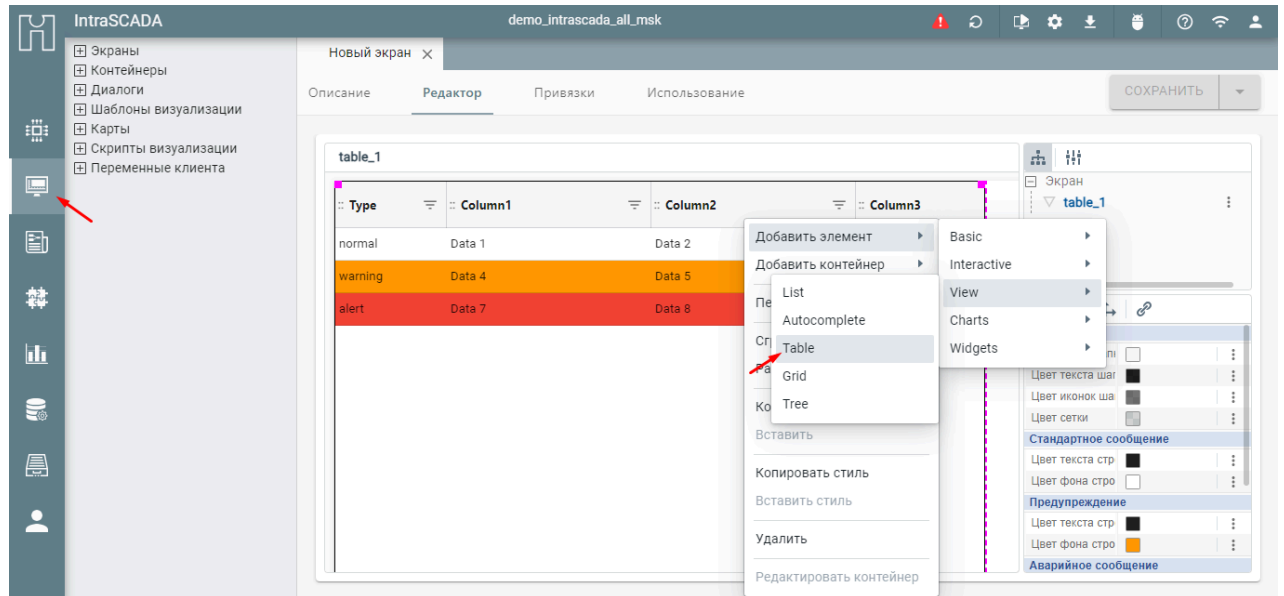
- Ставим галку в поле 'Заполнить из пользовательской таблицы'
- Выбираем созданную нами ранее таблицу из раздела БД -> Пользовательские таблицы
- Жмём Заполнить столбцы
- В результате таблица будет заполнена в соответствии с таблицей, созданной нами ранее в разделе БД -> Пользовательские таблицы



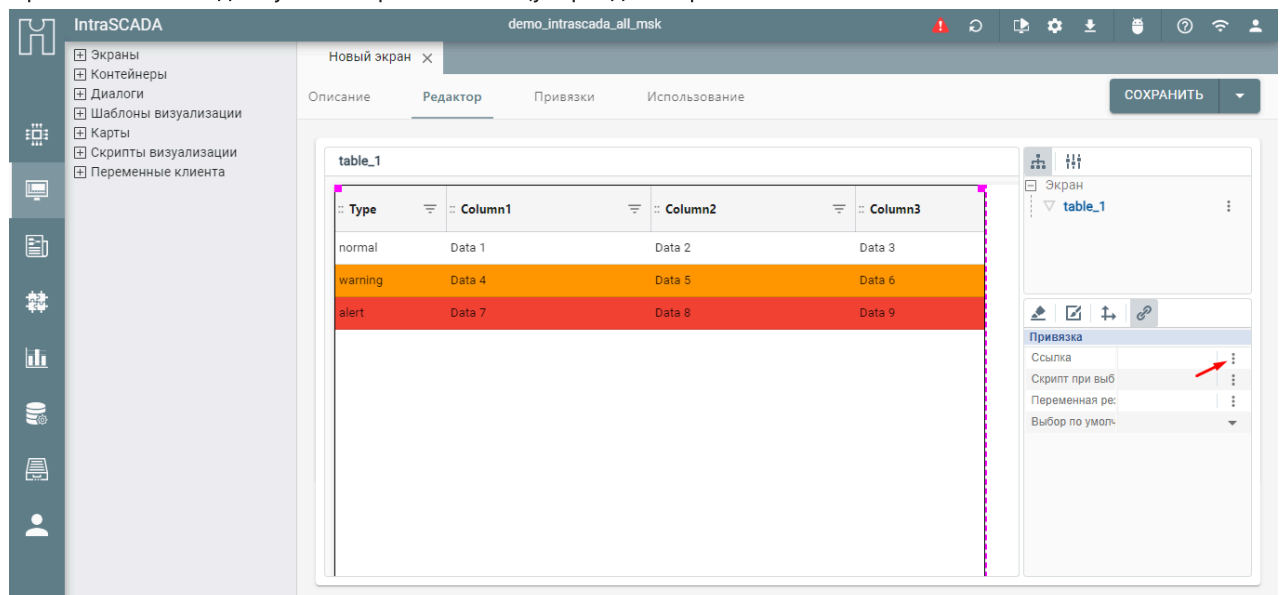
- Жмём Сохранить



- Добавляем элемент Table на экран в разделе Визуализация(Добавить элемент - View - Table)



- При необходимости настраиваем цвет таблицы, цвет рамки, размеры таблицы и т.д.
- Привязываем созданную нами ранее таблицу в разделе Привязка

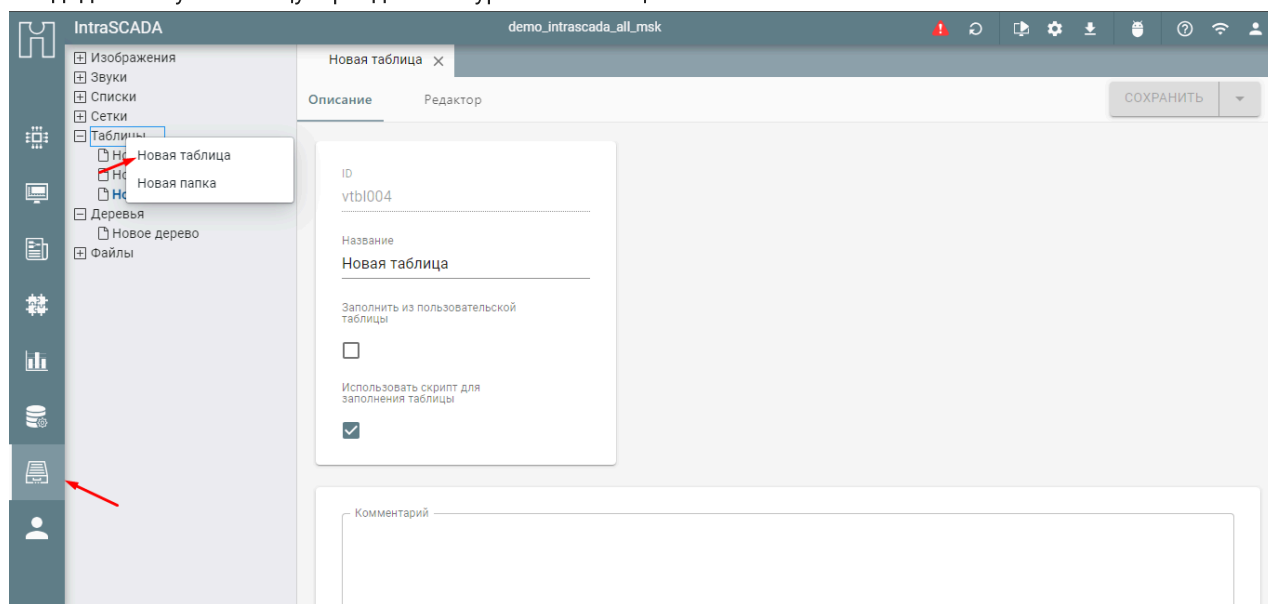


В результате на экране отобразится таблица с заданными нами ранее параметрами

Имя файла	Время	Комментарий
snar_168258...	16825853551...	Снапшот по датчику движен...
snar_168251...	16825126971...	Снапшот по датчику движен...
snar_168258...	16825840893...	Снапшот по датчику движен...
snar_168251...	16825121791...	Снапшот по датчику движен...
snar_168156...	16815648949...	Снапшот по датчику движен...
snar_168259...	16825970738...	Снапшот по датчику движен...
snar_168252...	16825203656...	Снапшот по датчику движен...
snar_168259...	16825937805...	Снапшот по датчику движен...
snar_168156...	16815636900...	Снапшот по датчику движен...
snar_168258...	16825807940...	Снапшот по датчику движен...
snar_168155...	16815552417...	Снапшот по датчику движен...
snar_168156...	16815609456...	Снапшот по датчику движен...
snar_168252...	16825207975...	Снапшот по датчику движен...
snar_168156...	16815660454...	Снапшот по датчику движен...

## Заполнение с помощью скрипта

- Создадим новую таблицу в разделе Ресурсы -> Таблицы



- Ставим галку в поле "Использовать скрипт для заполнения таблицы"
- Жмём сохранить
- Переходим во вкладку "Редактор"

Перед нами стандартный(дефолтный) скрипт для заполнения таблиц

```

/**
 * Скрипт для заполнения таблицы
 * Должен вернуть объект, содержащий
 *   columns: массив объектов с описанием столбцов {title, prop, width,
 *   data: массив объектов, каждый элемент – это строка таблицы
 */
module.exports = async function({ local, context }, core, debug) {
  const columns = [
    { title: 'Column1', prop: 'c1', width: 280, filter: true },
    { title: 'Column2', prop: 'c2', width: 280, filter: true },
    { title: 'Column3', prop: 'c3', width: 280, filter: true }
  ];

  const data = [
    { id:1, c1: 'Data 1', c2: 'Data 2', c3: 'Data 3', level: 0 },
    { id:2, c1: 'Data 4', c2: 'Data 5', c3: 'Data 6', level: 1 }
  ];
  return { columns, data };
};

```

Для столбцов заполним:

- prop - имя свойства, которое будет выводиться в столбце
- title - шапка столбца
- width - ширина столбца
- filter: true - включить фильтр(появится кнопка фильтрации справа от столбца),

Массив data содержит данные и параметры id для выбора строки в таблице, level для выделения цветом строки в таблице

При его использовании на экране отобразятся 3 столбца(Column1,2,3) и 2 строки с данными

Column1	Column2	Column3
Data 1	Data 2	Data 3
Data 4	Data 5	Data 6

Скрипт можно изменять в зависимости от ваших нужд

# Tree

Tree - данный визуальный компонент выводит информацию в виде дерева, созданного с помощью дерева в [Ресурсах](#)

Кроме [общих свойств](#) у элемента Tree есть индивидуальные свойства:

## Свойства элемента Tree

Наименование	Описание
<b>Tree</b>	
Цвет узла	Цвет узла
Цвет выбранного узла	Цвет выбранного узла

## Привязки

У компонента Tree существует три привязки:

- Ссылка - выбор дерева из **Ресурсов**
- Скрипт при выборе - выбор скрипта для запуска после выбора элемента дерева
- Переменная результата - выбор клиентской переменной для записи результата выбора элемента дерева. Присваивается id элемента таблицы.

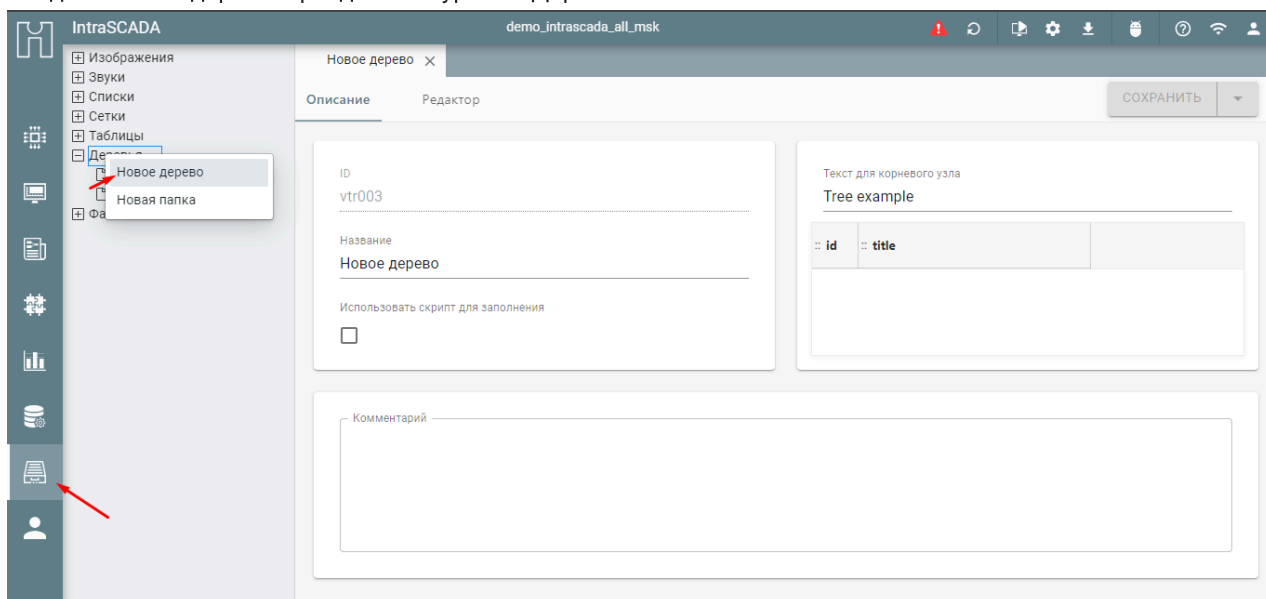
Привязка		
Ссылка		⋮
Скрипт при выборе		⋮
Переменная результата		⋮
Выбор по умолчанию	Первый элемент	▼

Также возможно настроить **Выбор по умолчанию** компонента Tree при загрузке экрана:

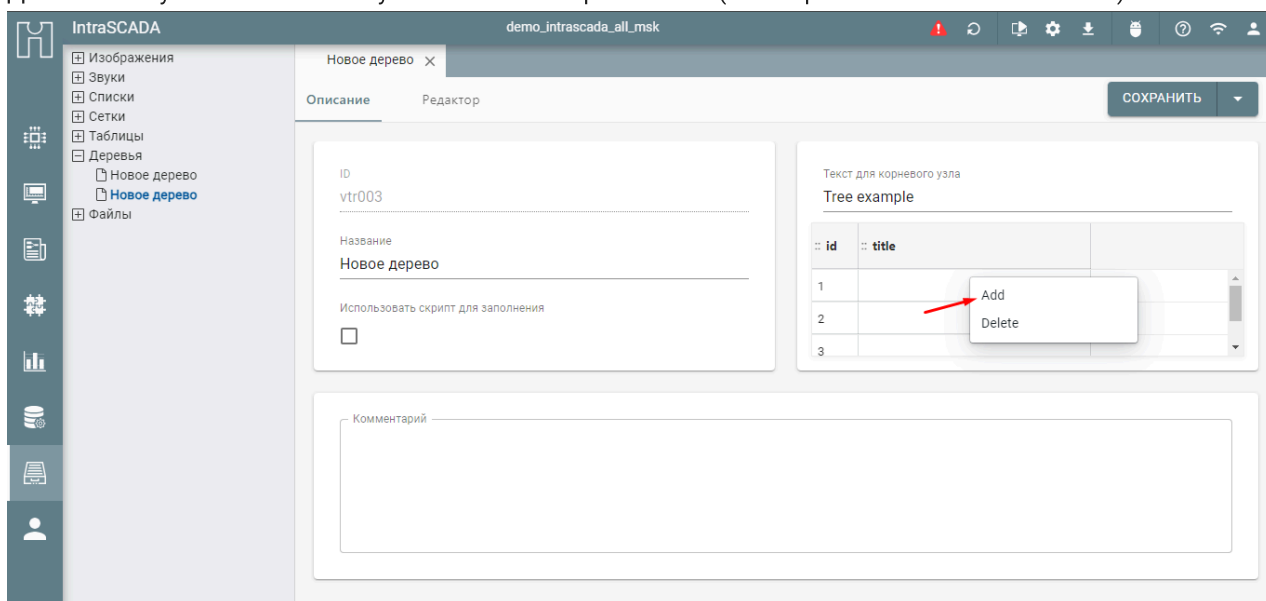
- Первый элемент
- Последний элемент

## Ручное Заполнение

- Создаём новое дерево в разделе Ресурсы -> Деревья



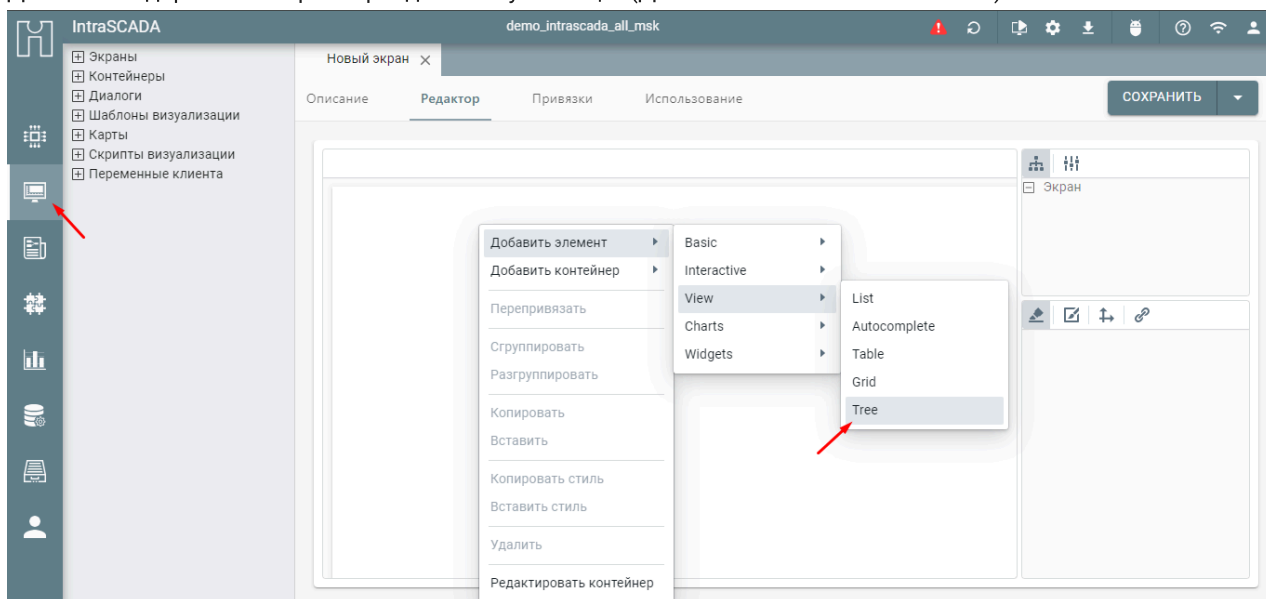
- Пишем название, галку в поле "Использовать скрипт для заполнения" не ставим
- Пишем название корневого узла дерева
- Добавляем нужное количество узлов в нижнем правом поле (Клик правой кнопкой мыши - Add)



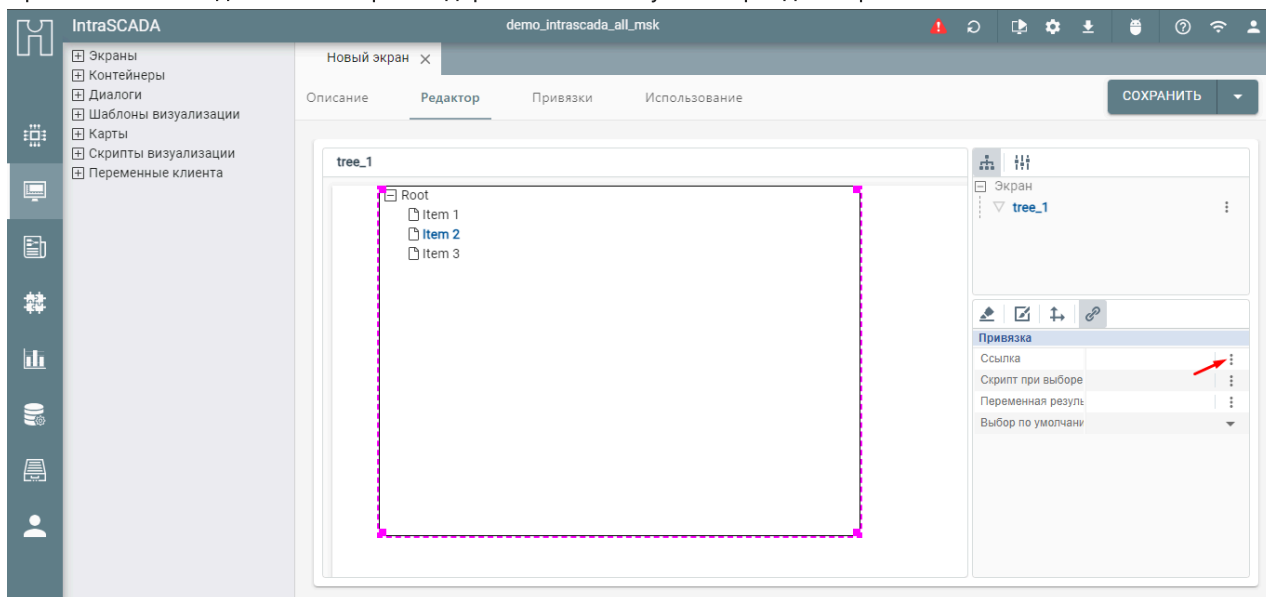
- Вводим данные каждого узла (id - id узла, title - название узла)
- Сохраняем созданное нами дерево



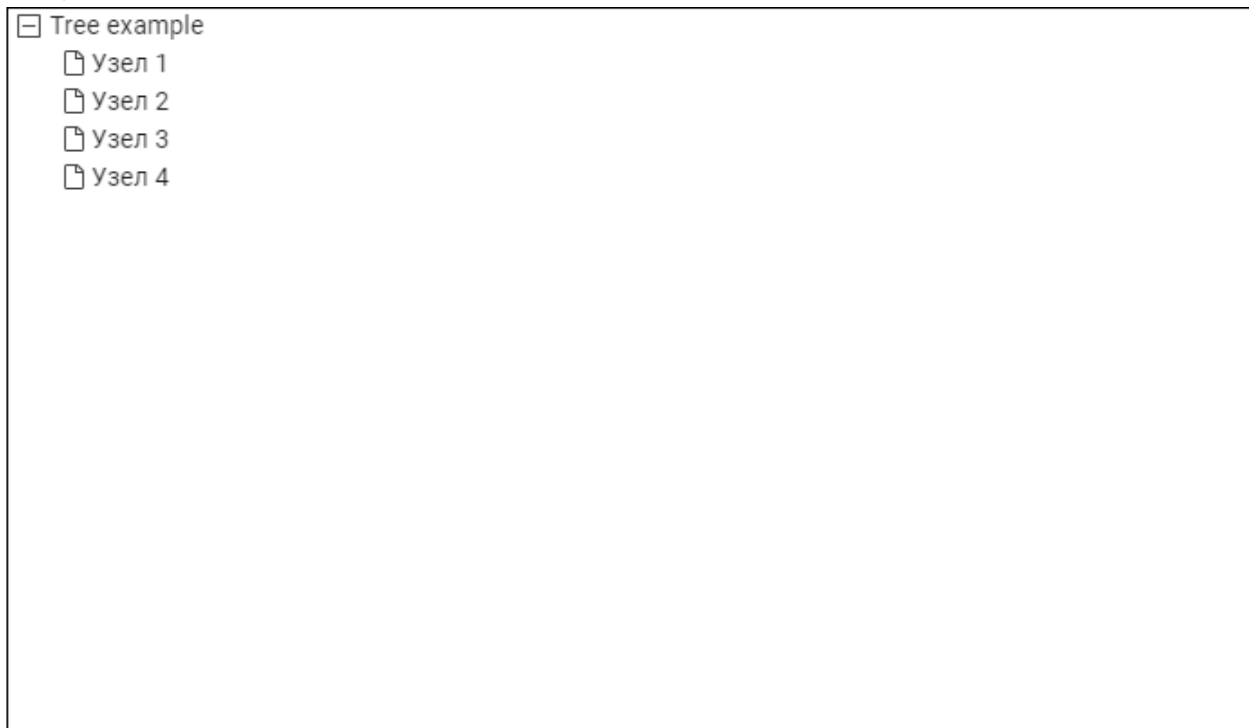
- Добавляем дерево на Экран в разделе Визуализация(Добавить элемент - View - Tree)



- Настраиваем цвета узлов, рамку, размеры элемента и т.д.
- Привязываем созданное нами ранее дерево к элементу Tree в разделе привязка



- В результате на экране отобразится дерево с заданными нами ранее параметрами



## Заполнение с помощью скрипта

Пример скрипта для заполнения дерева

```

/**
 * Скрипт для заполнения дерева
 * Должен вернуть объект, содержащий
 *   data: массив узлов дерева
 */
module.exports = async function ({local, context, source}, core, debug) {
  const data = [
    {
      id: 'root',
      title: 'Главный узел',
      children: [
        {
          id: 'lev1',
          title: 'Узел вложенный',
          children: [
            { id: '1', title: 'Раз' },
            { id: '2', title: 'Два' },
            { id: '3', title: 'Три' }
          ]
        }
      ]
    }
  ]
};
return { data };
}

```

Пример скрипта для заполнения дерева устройствами из дерева устройств, начиная с конкретного узла дерева:  
 'dg061' - расположения узла в дереве (location)

```
module.exports = async function ({local, context, source}, core, debug) {
  const data1 = await core.getTree('devices', 'dg061');
  const data2 = await core.getTree('devices', 'dg083');
  const data3 = await core.getTree('devices', 'dg085');

  return {
    data: [
      ...expandFirst(data1),
      ...expandFirst(data2),
      ...expandFirst(data3)
    ]
  };
}

function expandFirst(data) {
  data[0].expanded = true;
  return data;
}
```

# Grid

Grid - данный визуальный компонент выводит информацию в виде сетки, созданной с помощью сетки в [Ресурсах](#)

Кроме [общих свойств](#) у элемента Grid есть индивидуальные свойства:

## Свойства элемента Grid

Наименование	Описание
<b>Grid</b>	
<i>Заголовок</i>	
Фиксированных столбцов	Количество фиксированных столбцов
Высота заголовка	Высота заголовка
Размер текста	Размер текста
Цвет текста	Цвет текста
Цвет выделения	Цвет выделения
Цвет фона	Цвет фона
Цвет рамки	Цвет рамки
Цвет подчеркивания	Цвет подчеркивания
Цвет в фокусе	Цвет в фокусе
Цвет при наведении	Цвет при наведении
<i>Ячейки</i>	
Высота	Высота ячейки
Размер текста	Размер текста
Цвет текста	Цвет текста
Цвет фона	Цвет фона
Цвет рамки	Цвет рамки
<i>Выделение</i>	

Подсветка      Подсветка

*Маркеры*

Разрешены      true/false

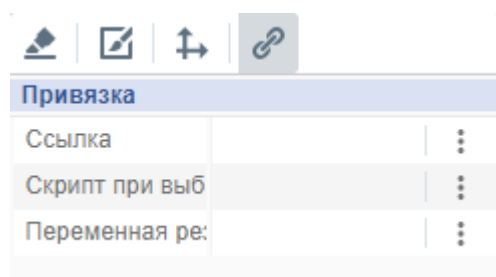
Цвет маркера      Цвет маркера

Цвет текста      Цвет текста

## Привязки

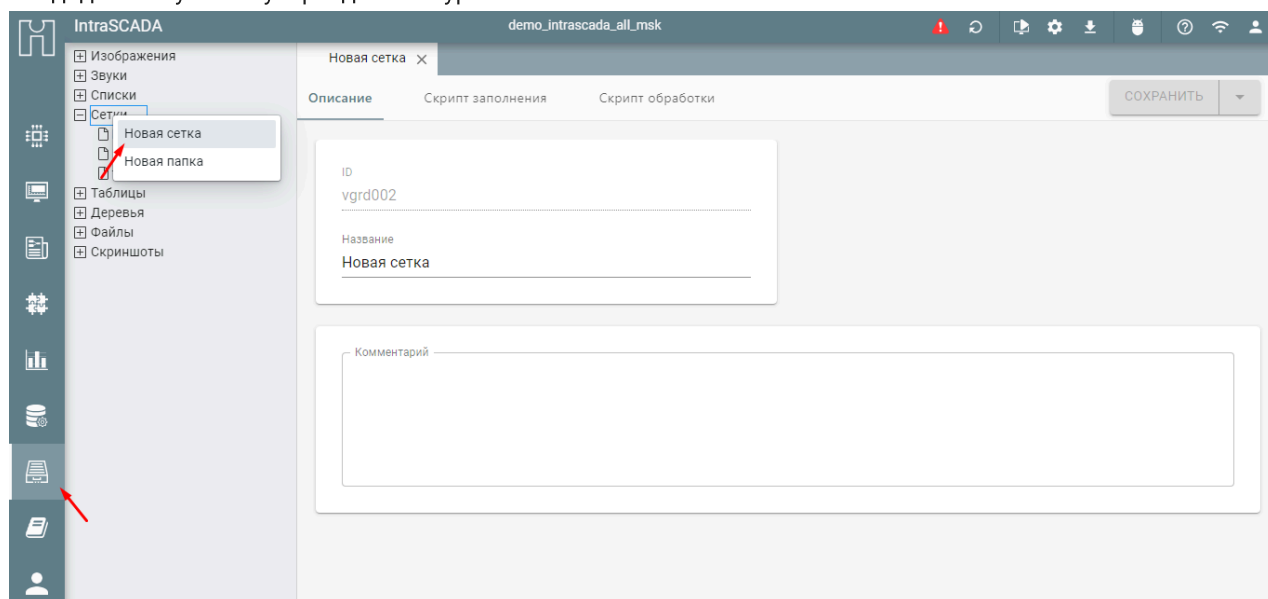
У компонента Grid существует три привязки:

- Ссылка - выбор сетки из **Ресурсов**
- Скрипт при выборе - выбор скрипта для запуска после выбора элемента сетки
- Переменная результата - выбор клиентской переменной для записи результата выбора элемента сетки. Присваивается id элемента таблицы.



## Заполнение с помощью скрипта

- Создадим новую сетку в разделе Ресурсы -> Сетки



- Переходим на вкладку Скрипт заполнения

Скрипт заполнения будем использовать из примера в разделе Ресурсы -> Сетки

```
/**
 * Скрипт заполнения сетки
 */
module.exports = async function({ local, context }, core, debug) {

  const columns = [
    { title: 'Техпроцесс', prop: 'main', width: 128, readonly: true },
    { title: 'Фаза 1', prop: 'step1', width: 100 },
    { title: 'Фаза 2', prop: 'step2', width: 100 },
    { title: 'Фаза 3', prop: 'step3', width: 100 }
  ];

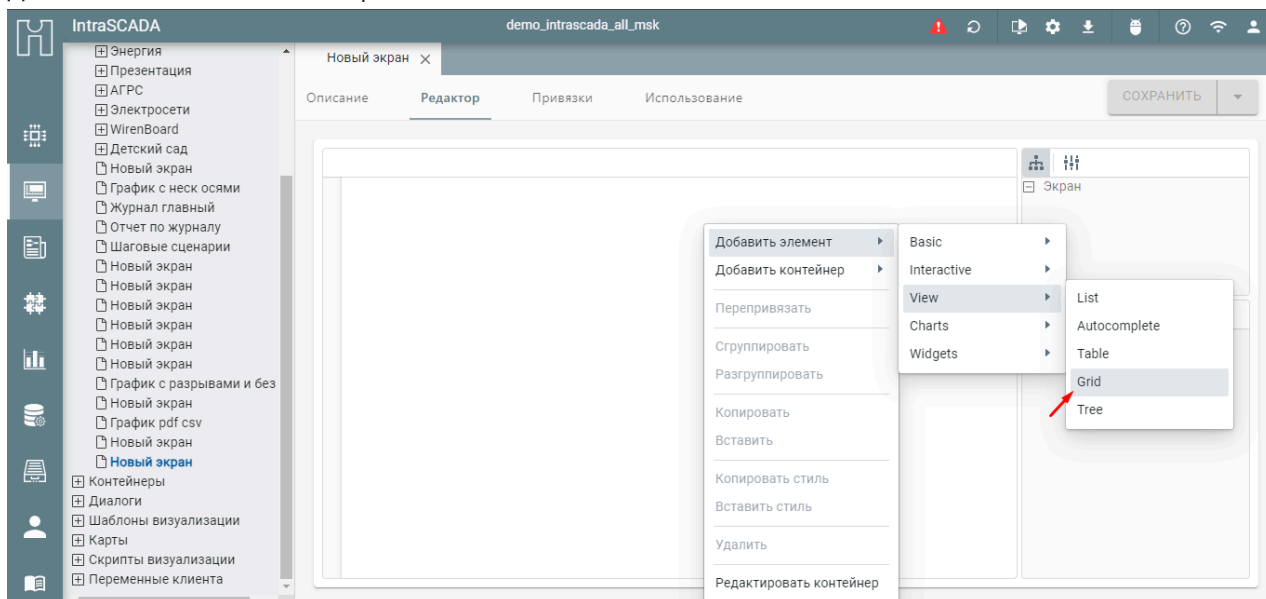
  const data = [
    {
      main: { value: 'Температура' },
      step1: { value: 1, type: 'number' },
      step2: { value: 2, type: 'number' },
      step3: { value: 3, type: 'number' }
    },
    {
      main: { value: 'Режим' },
      step1: { value: 'Dry', type: 'droplist', data: 'mode' },
      step2: { value: 'Heating', type: 'droplist', data: 'mode' },
      step3: { value: 'Off', type: 'droplist', data: 'mode' }
    },
    {
      main: { value: 'Цвет' },
      step1: { value: 'red', type: 'string', background: '#FF0000' },
      step2: { value: 'green', type: 'string', background: '#00FF00' },
      step3: { value: 'blue', type: 'string', background: '#0000FF' }
    }
  ];

  return {
    data,
    columns,
    droplists: {
      mode: ['Off', 'Heating', 'Dry']
    }
  };
};
```

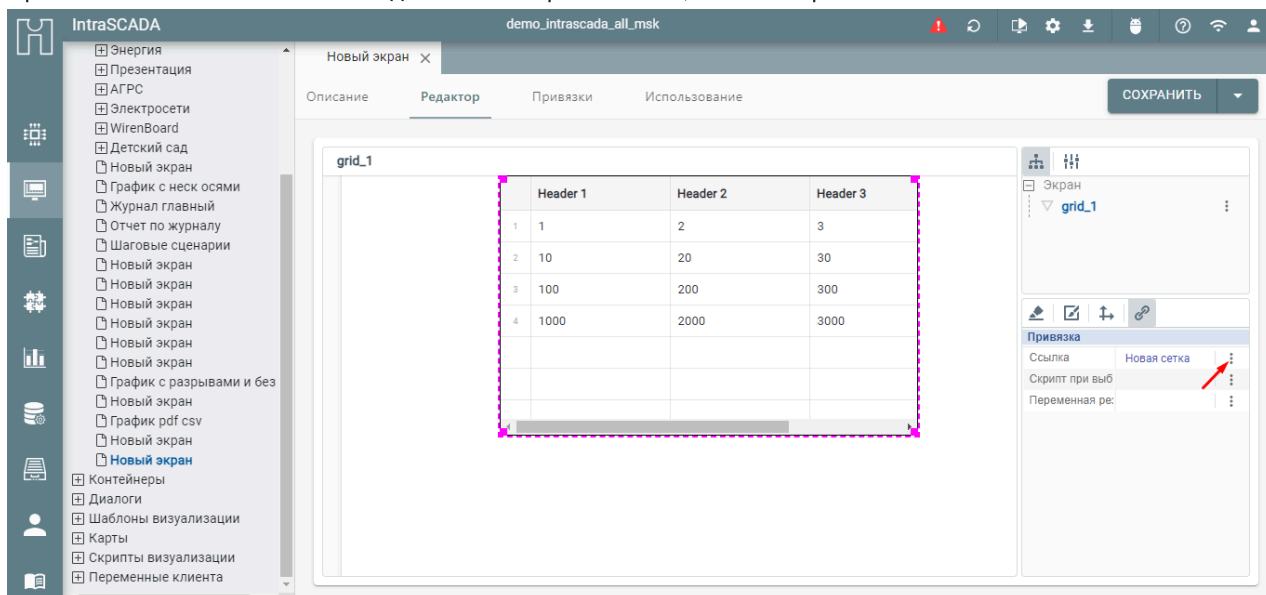


- Скрипт обработки не меняем, жмём Сохранить

- Добавляем элемент Grid на экран



- Привязываем элемент Grid к созданной нами ранее сетке, жмём Сохранить



- В результате на экране отобразится сетка, с заданными нами в скрипте данными

	Техпроцесс	Фаза 1	Фаза 2	Фаза 3	
1	Температура	1	2	3	
2	Режим	Dry	Heating	Off	
3	Цвет	red	green	blue	

## Элементы для вывода графиков

Элементы для вывода графиков доступны в меню графического редактора: Добавить элемент -> Charts.

Графики можно размещать на экране, в контейнере, диалоге и даже в шаблоне. В системе есть несколько видов графиков.

- [Chart Line](#)
  - простейший вариант графика для вывода одного свойства с привязкой напрямую к свойству устройства

Остальные графики позволяют выводить как несколько свойств одного устройства, так и данные разных устройств. Настройка правил формирования данных выполняется в разделе [Аналитика -> Графики](#). Затем созданная сущность **График** привязывается к визуальному элементу на вкладке Привязка.

- [Chart MultiLine](#)
  - линейный многоперьевой график
- [Chart MultiLine GL](#)
  - линейный многоперьевой график с технологией отрисовки WebGL
- [Chart Timeline](#)
  - график для отображения состояний на временной шкале
- [Chart Columns](#)
  - столбчатый график
- [Chart Pie](#)
  - круговой график (диаграмма) без временной шкалы

# Chart Line

Это простейший линейный одноперьевого график. Выводит данные одного свойства одного устройства.

Главное **отличие** этого графика в том, что правила формирования данных настраиваются прямо в визуальном элементе, отдельная сущность **График** не требуется.

Достаточно на вкладке **Привязка** выбрать нужное свойство устройства.

Для того, чтобы информация появилась на графике, необходимо [сохранять в БД](#) выбранное свойство.

## Свойства элемента Chart Line

Кроме [общих свойств](#) у элемента Chart Line есть индивидуальные свойства:

Наименование	Описание	Примечание
<b>Chart Line</b>		
Период	Период отображения данных	Минута, Час, День, Неделя, Месяц, Пользовательский
Реалтайм	Если поставить галочку, данные будут поступать в режиме реального времени	
Цвет сетки	Цвет сетки	
<b>Зумирование</b>		
По оси X	Увеличение/уменьшение масштаба графика по оси X	
По оси Y	Увеличение/уменьшение масштаба графика по оси Y	
<b>Перемещение</b>		
По оси X	Перемещение по графику по оси X	
По оси Y	Перемещение по графику по оси Y	
<b>Line</b>		
Ширина линии	Ширина линии	
Цвет	Цвет линии	
Тип	Интерполяция	Liner, Cubic, Cubic monotone, Step
<b>Заполнение</b>		
Цвет заполнения	Цвет заполнения	
Режим заполнения	Режим заполнения	None, First up, First down, All up, All down

Заголовок	Текст заголовка	
Положение	Положение текста	None, Слева Справа, Сверху, Снизу
Цвет текста	Цвет текста	
Размер текста	Размер текста	
<b>Точка</b>		
Радиус	Радиус точки на графике	
Радиус при наведении	Радиус точки на графике при наведении	
<b>Всплывающая подсказка (тултип)</b>		
Показывать	Показывать подсказку	
Цвет фона	Цвет фона	
Цвет даты	Цвет даты	
Цвет значения	Цвет значения	
<b>Ось X</b>		
Положение	Положение Оси X	None, Сверху, Снизу
Цвет	Цвет текста Оси X	
Размер	Размер текста Оси X	
<b>Ось Y</b>		
Положение	Положение Оси Y	None, Сверху, Снизу
Цвет	Цвет текста Оси Y	
Размер	Размер текста Оси Y	

Масштаб	Масштабирование	Статическое, Динамическое
Min	Минимальное значение по Оси Y	
Max	Максимальное значение по Оси Y	
<b>Кнопка Сброс/Обновить</b>		
Показывать	Отображает кнопку	В левом верхнем углу появляется кнопка Сброса/Обновления
Цвет	Изменяет цвет кнопки	
<b>Кнопка Синхронизации</b>		
Показывать	Отображает кнопку	В левом верхнем углу появляется кнопка Синхронизации
Цвет	Изменяет цвет кнопки	

## Примечание к свойству Масштаб

- Если выбрать Статическое масштабирование, то необходимо задать Min и Max значения по оси Y. Шкала по оси Y будет фиксирована
- Динамическое масштабирование будет подстраивать границы Min-Max под отображаемые в текущий момент значения

## Кнопка Синхронизации

Эта кнопка используется, если в контейнере есть несколько графиков с возможностью зуммирования по оси X. При нажатии на Кнопку Синхронизации все графики синхронизируют шкалу X с графиком, на котором нажата кнопка.

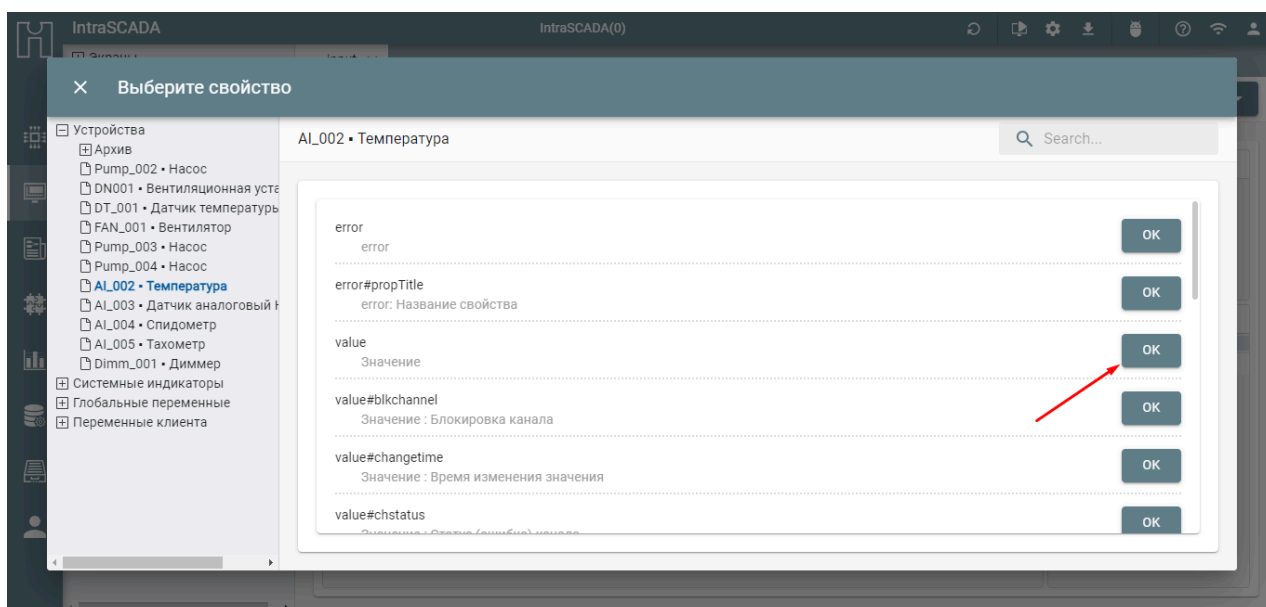
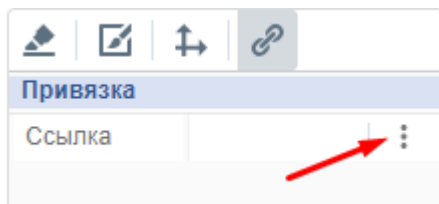
## Привязка

Привязка графика к свойству устройства выполняется на вкладке **Привязка**

## Пример создания линейного графика

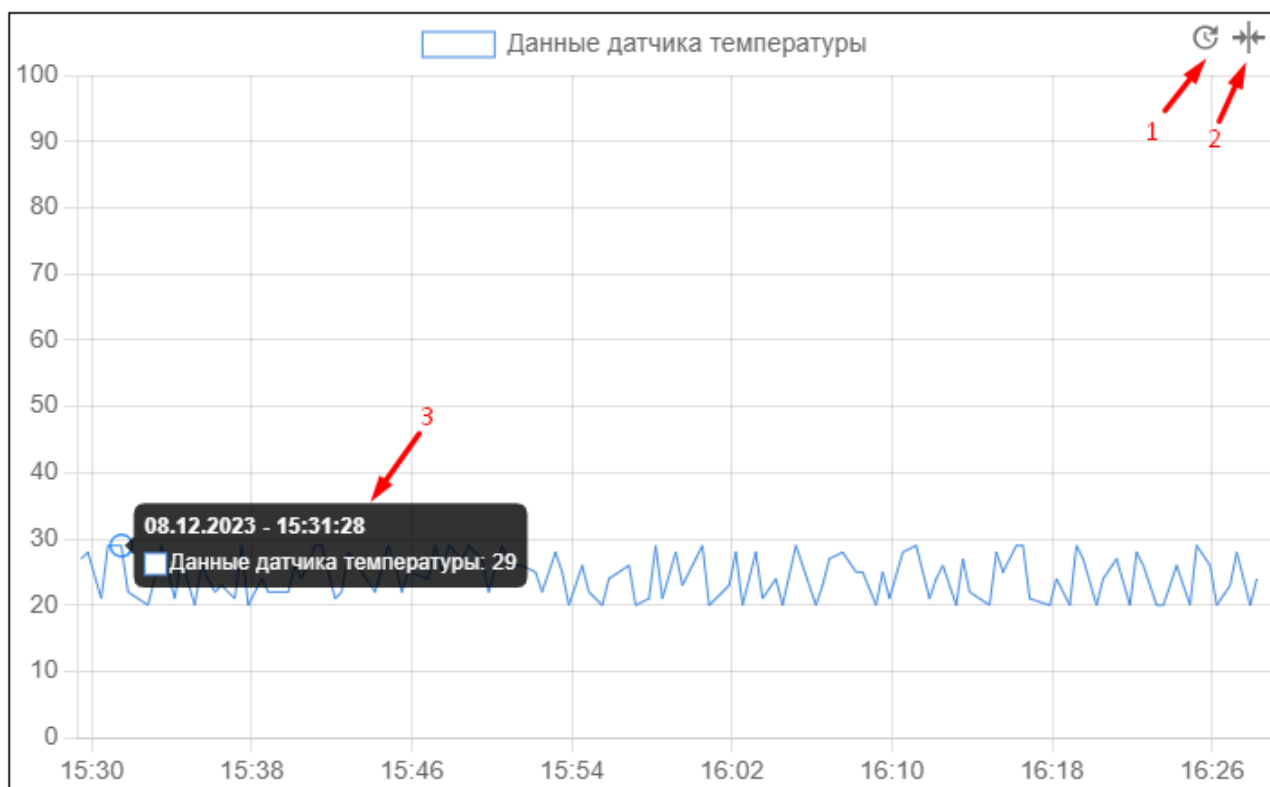
- Создаём элемент Chart Line (Добавить элемент -> Charts -> Chart Line)
- Настраиваем период отображения данных (в примере "Час")

- Настраиваем Цвет, Зумирование, Перемещение и т.д.
- Вводим текст легенды (в примере "Данные датчика температуры")
- Выбираем отображение всплывающей подсказки (При наведении на график появится всплывающее окно с информацией)
- Выбираем положение осей X/Y, Цвет, Размер
- Выбираем отображение кнопок Сброс/Обновить и Синхронизации
- Настраиваем Рамку, Размеры элемента
- Привязываем График к свойству **value** датчика температуры



В результате у нас получился линейный график, показывающий изменения температуры за выбранный период времени.





- 1 - Кнопка Сброс/Обновить
- 2 - Кнопка Синхронизации
- 3 - Всплывающее окно с подсказкой

# Chart Multiline

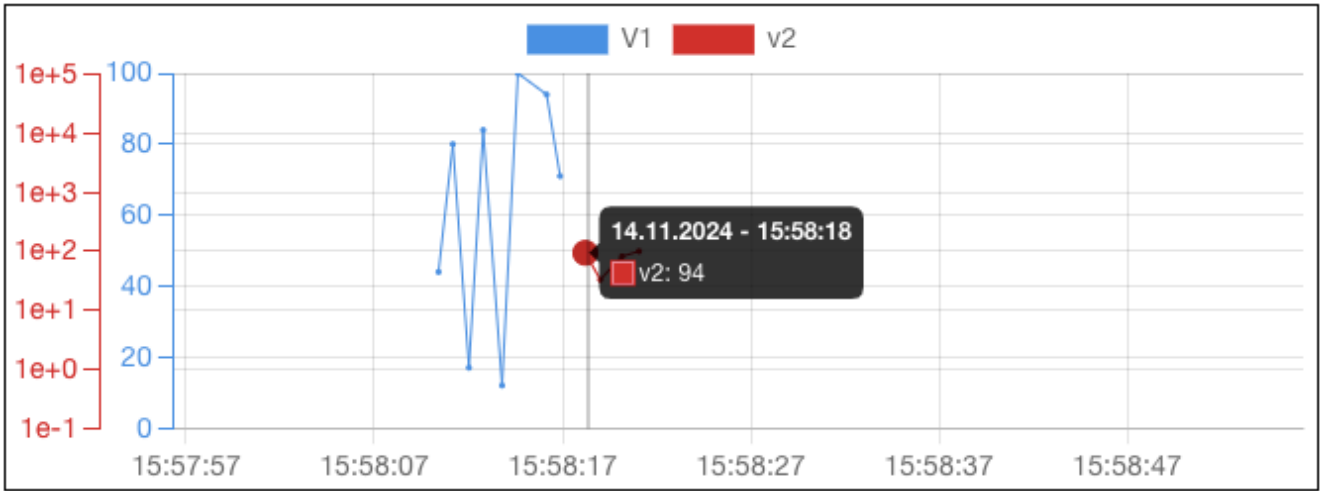
Визуальные компоненты **Chart Multiline** и **Chart Multiline GL** выводят линейный многоперьеовой график. Отличие компонентов в различных методах отрисовки, **Chart Multiline GL** использует технологию WebGL.

## Свойства элемента Chart Multiline

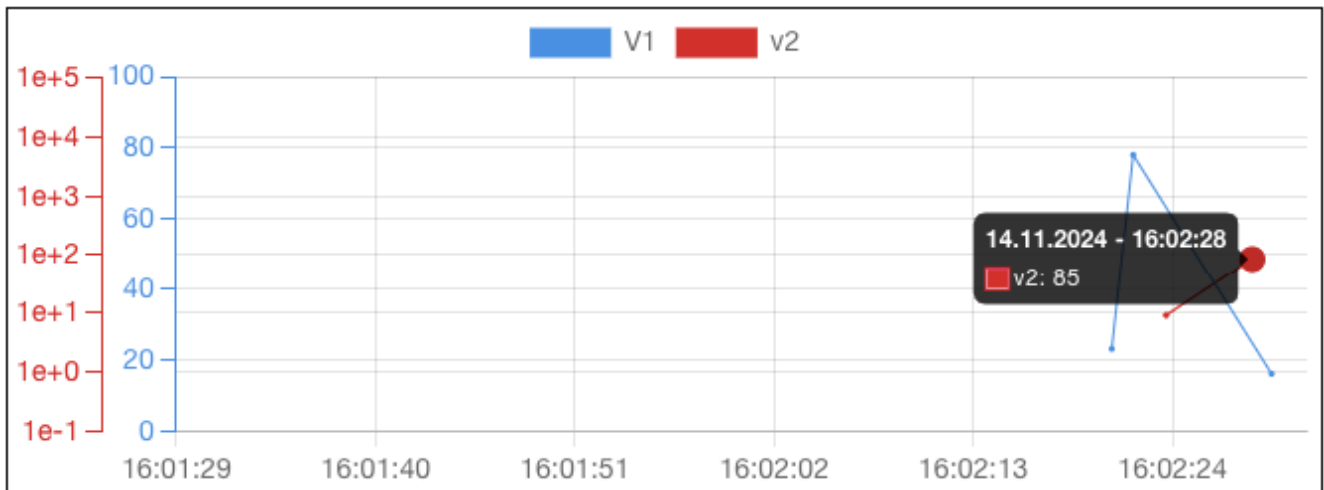
Индивидуальные свойства Chart Multiline идентичны [Chart Line](#), за исключением всплывающей подсказки (тултипа). Всплывающая подсказка (тултип), в отличии от [Chart Line](#), имеет два дополнительных свойства:

Наименование	Описание	Примечание
<b>Всплывающая подсказка (тултип)</b>		
Отображать	Отображать всегда или при наведении	
Перекрестье	Включить/Выключить перекрестье	

Тултип с перекрестьем:



Тултип без перекрестья:



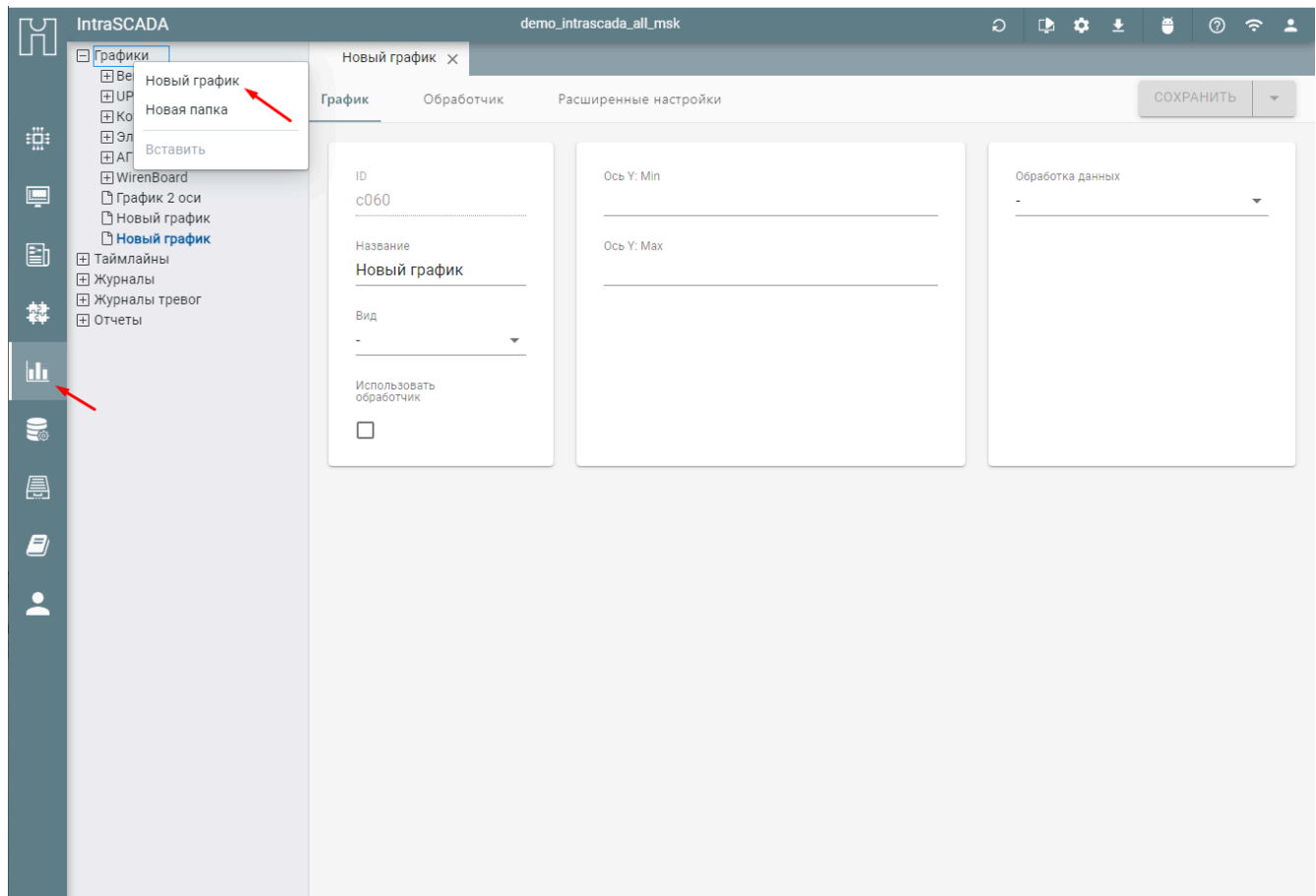
## Привязка

Привязка графика выполняется на вкладке **Привязка**. Для привязки доступны [Графики](#) и [Переменные клиента](#).

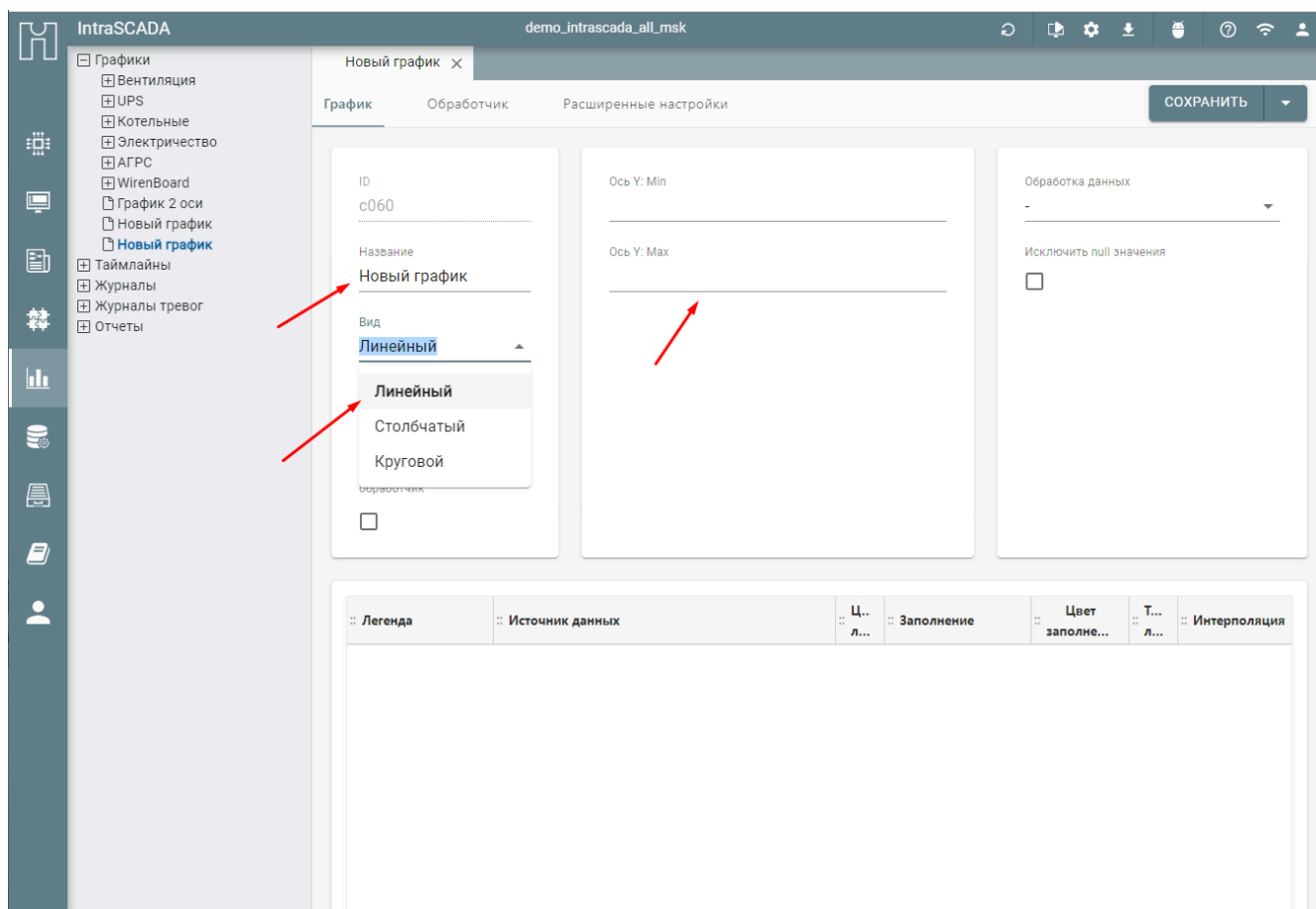
## Пример создания многоперьевого графика

Создадим график, содержащий значения температуры трех котлов: Boiler\_001, Boiler\_002, Boiler\_003

Добавим новый график в разделе [Аналитика->Графики](#)



- Вводим название графика (в примере "graphtest")
- Вид графика выбираем "Линейный"



В нижней таблице добавляем три строки для формирования перьев:

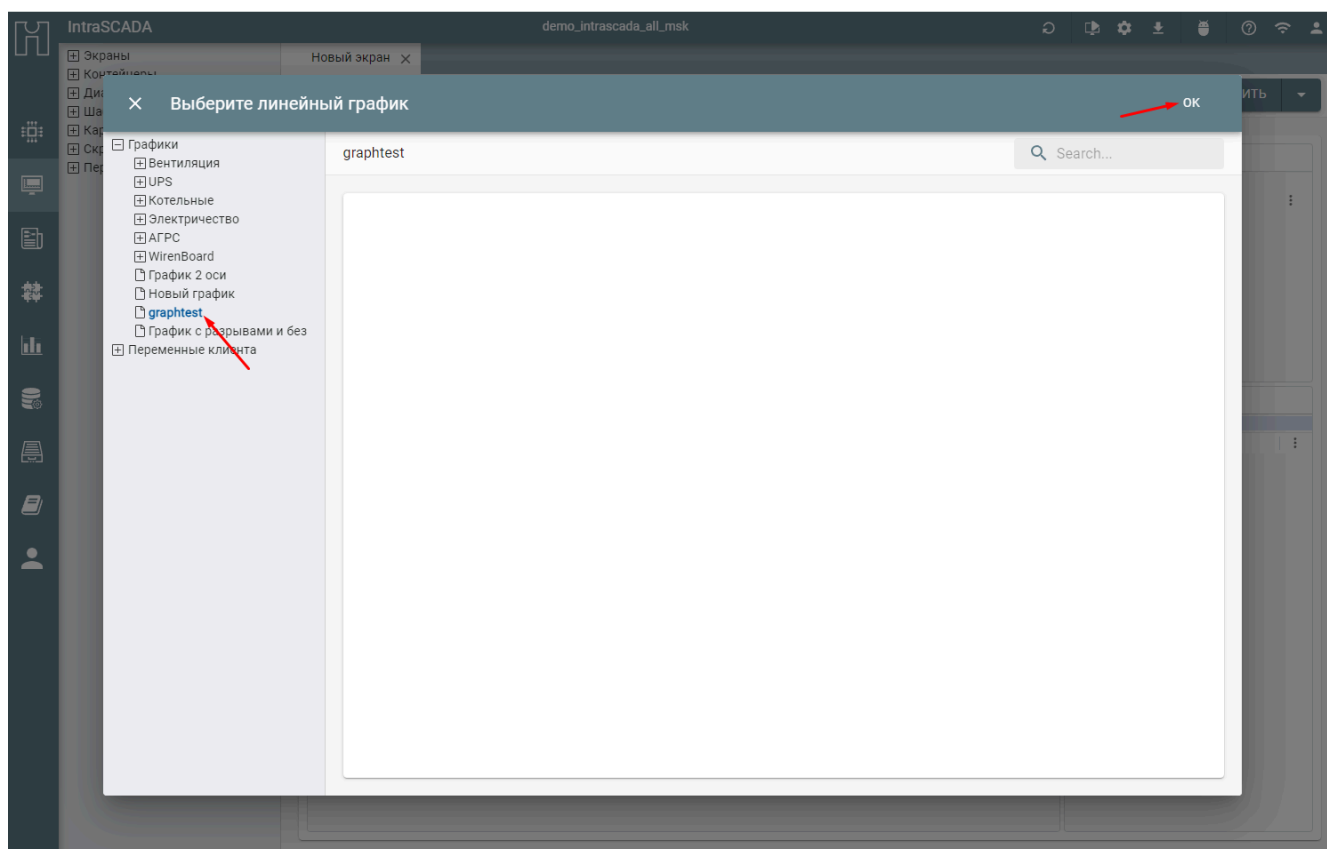
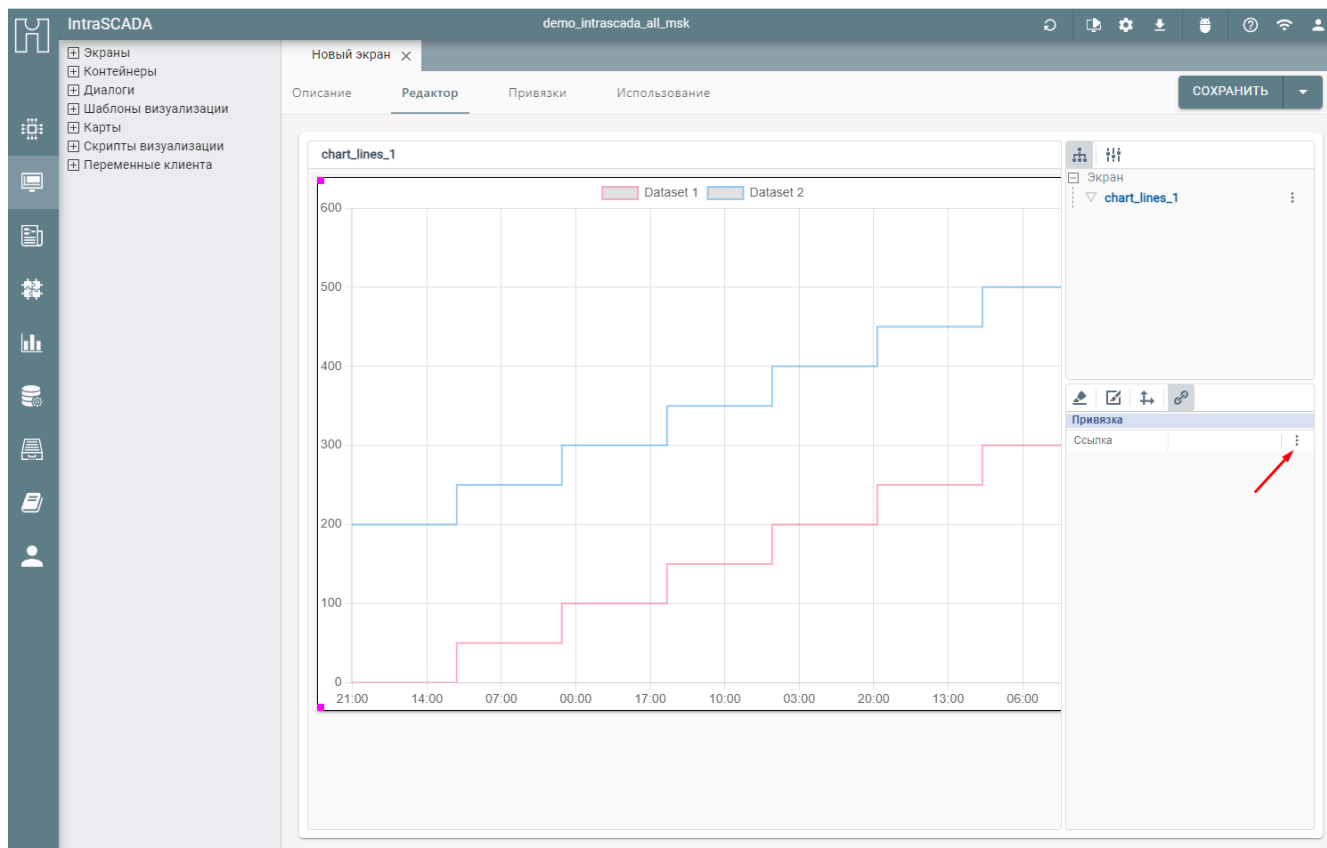
- Легенда - Температура Котла 1,...
- Источник данных - Привязка к свойству устройства (Boiler\_001.TEMP...)
- Цвет линии - Разные цвета для разных графиков
- Заполнение - None - без закрашки
- Толщина линии - 1
- Цвет заполнения - нет
- Интерполяция - Default(самый быстрый способ отрисовки)
- Цифр после запятой - 1
- N шкалы по оси Y - не заполняем.

В примере используем одну шкалу Y, так как у нас три графика с температурой

Сохраняем созданные правила формирования графика и переходим на экран.

На экране добавляем элемент Chart Multiline (Добавить Элемент - Charts - Chart Multiline).

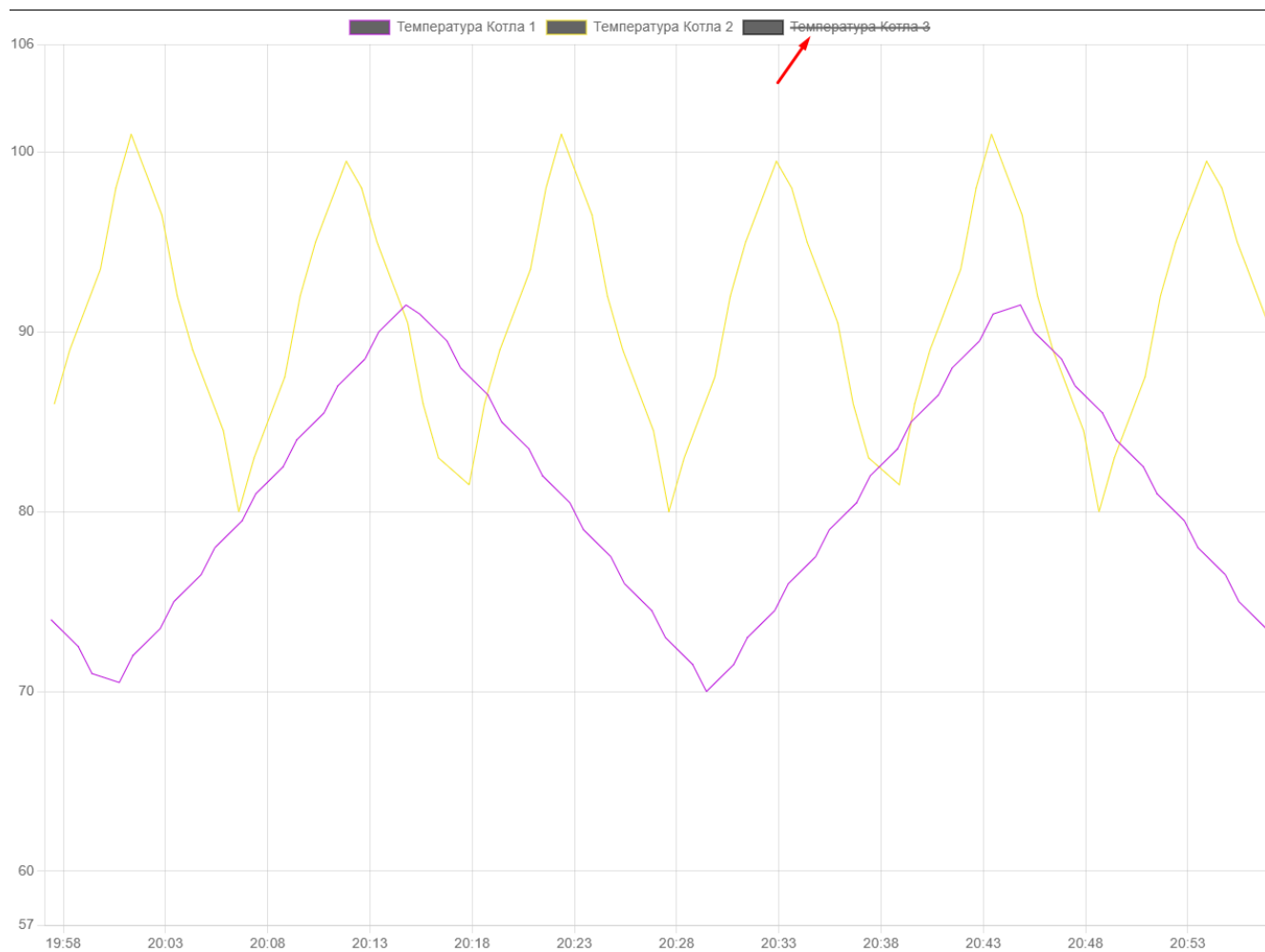
- Настраиваем период отображения данных (в примере "Час")
- Привязываем элемент Chart Multiline к созданному нами ранее графику("graphtest"), жмём OK.



В результате на экране отобразится график с тремя перьями: Температура Котла 1, Температура Котла 2, Температура Котла 3



Отображение любого пера можно убрать, нажав на легенду



# Chart Timeline

Chart Timeline - этот визуальный компонент выводит информацию в виде диаграммы Ганта.

Для того, чтобы информация появилась на таймлайне, необходимо настроить [сохранение в БД](#) для свойства устройства, выбрав метод сохранения Таймлайн. Настройка правил формирования данных выполняется в разделе [Аналитика -> Таймлайны](#).

## Свойства элемента Chart Timeline

Кроме [общих свойств](#) у элемента Chart Timeline есть индивидуальные свойства:

Наименование	Описание	Примечание
<b>Chart Timeline</b>		
Период	Период отображения данных	Минута, Час, День, Неделя, Месяц, Пользовательский, Выберите переменную
Подвижный	Возможность перемещения	
Цвет сетки	Цвет сетки	
Цвет текста	Цвет текста	
<b>Легенда</b>		
Ширина	Ширина легенды	
<b>Axis Bottom</b>		
Показать Дату	Показать Дату	
Показать Время	Показать Время	

## Привязка

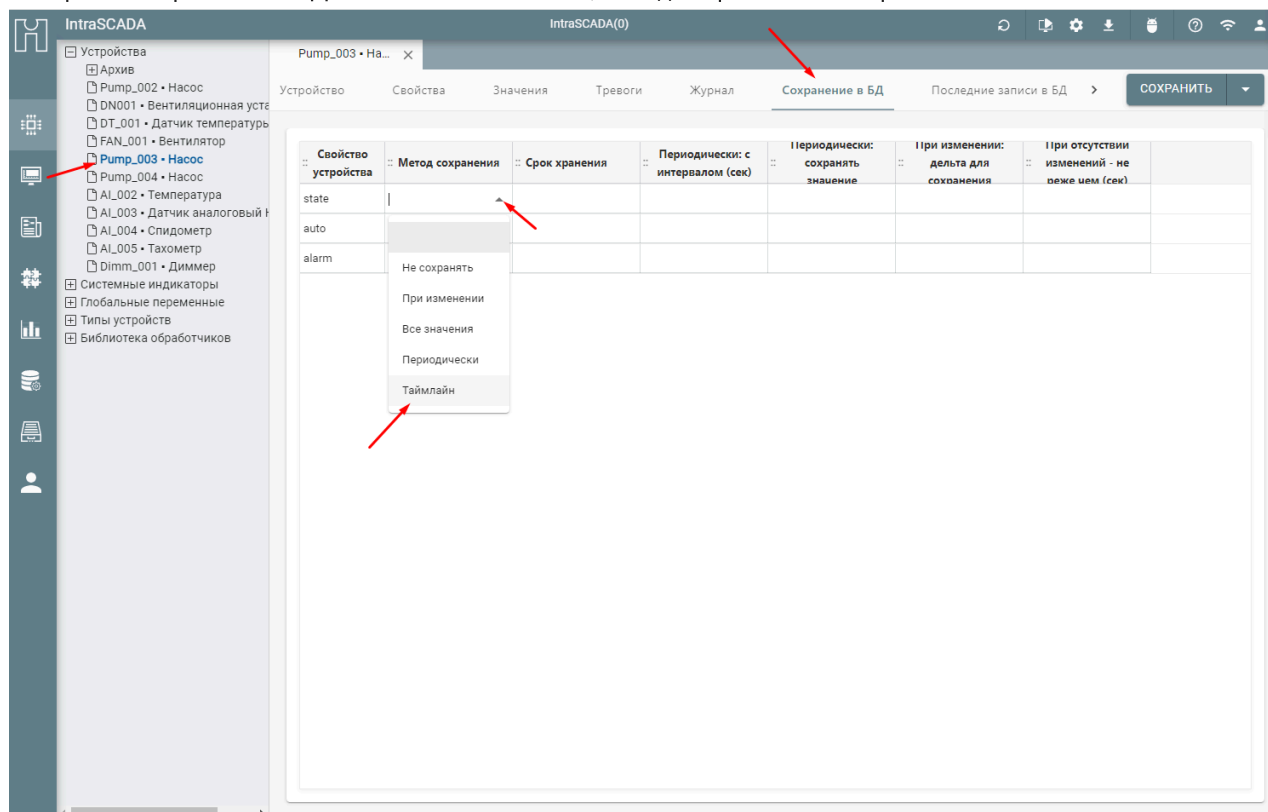
Привязка выполняется на вкладке **Привязка**. Для привязки доступны [Таймлайны](#) и [Переменные клиента](#).

## Пример создания Графика Таймлайн

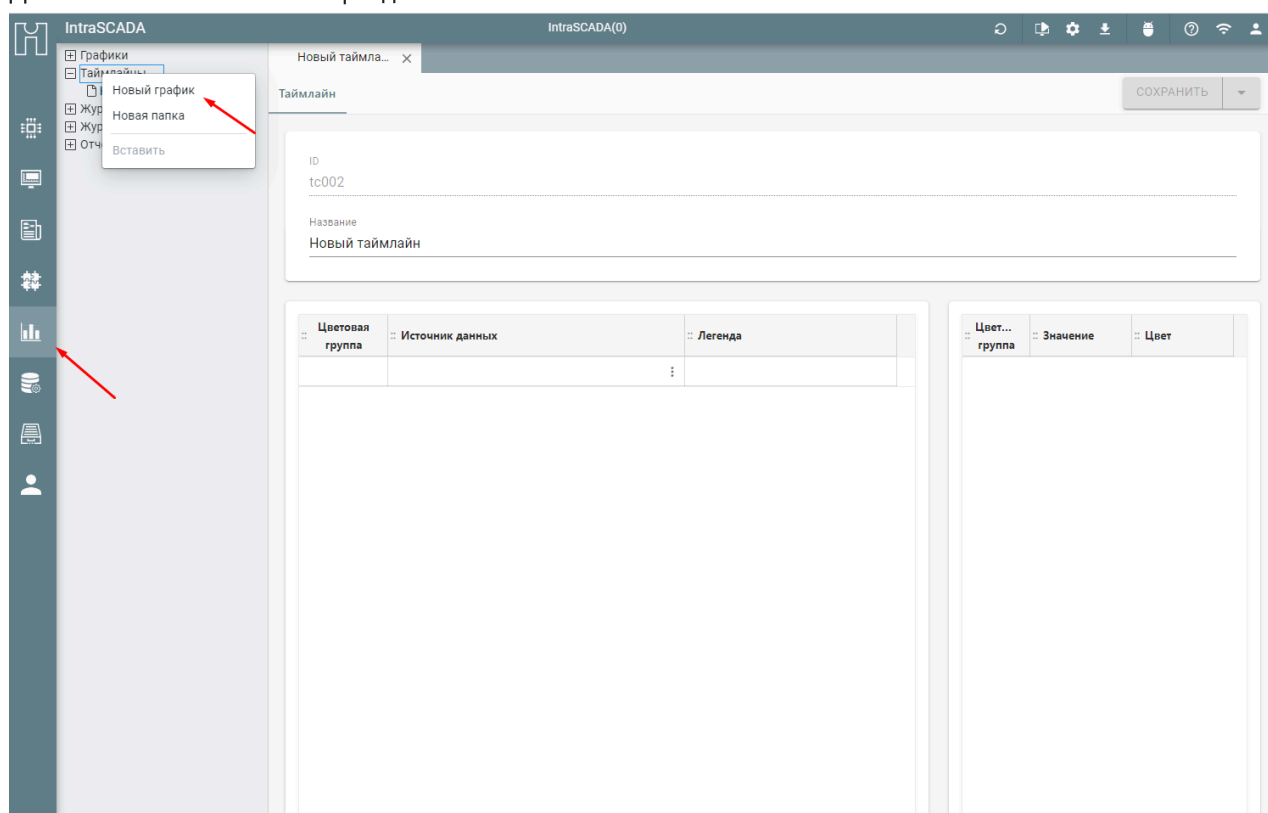


Создадим таймлайн для отображения свойства **state** Насоса


- Настроим Сохранение в БД свойства state насоса, метод сохранения выбираем Таймлайн



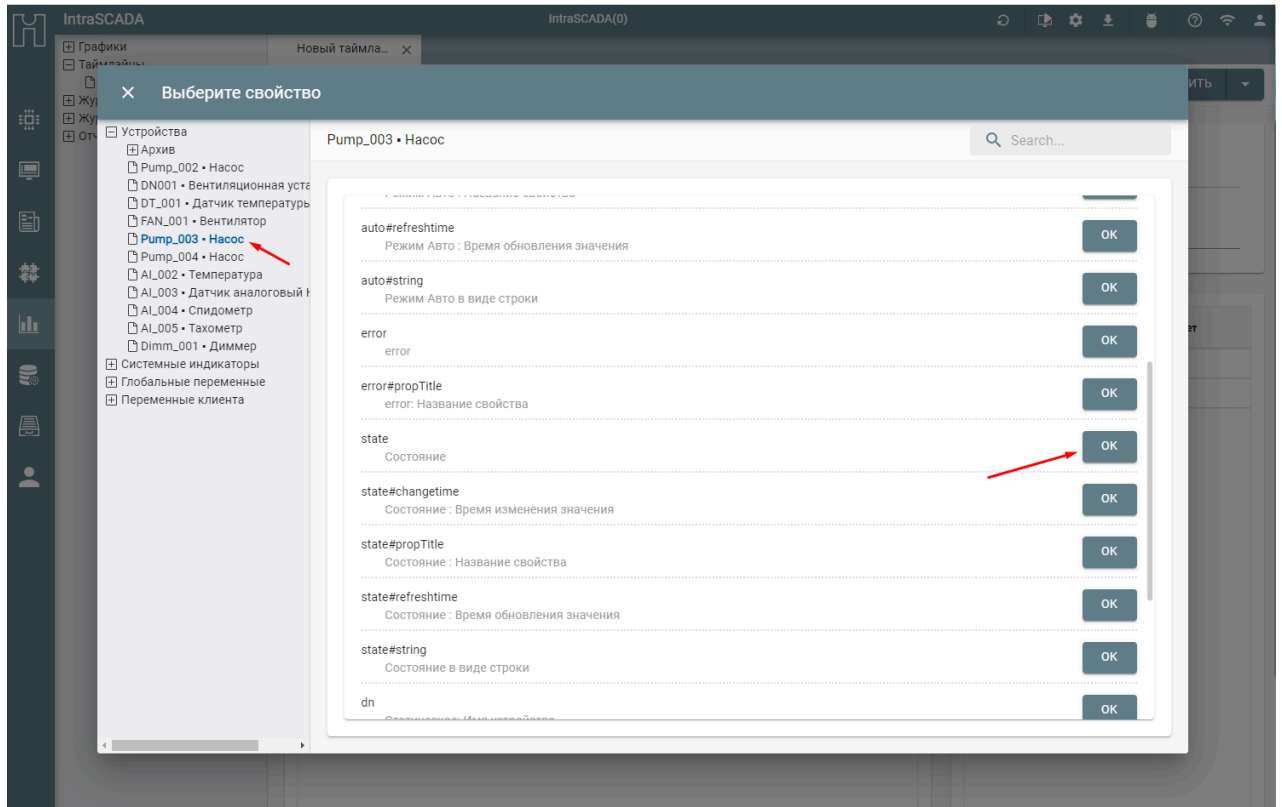
- Добавим новый таймлайн в разделе [Аналитика->Таймлайны](#)



- В правом окне настроим цвет для состояния 1 - Зелёный

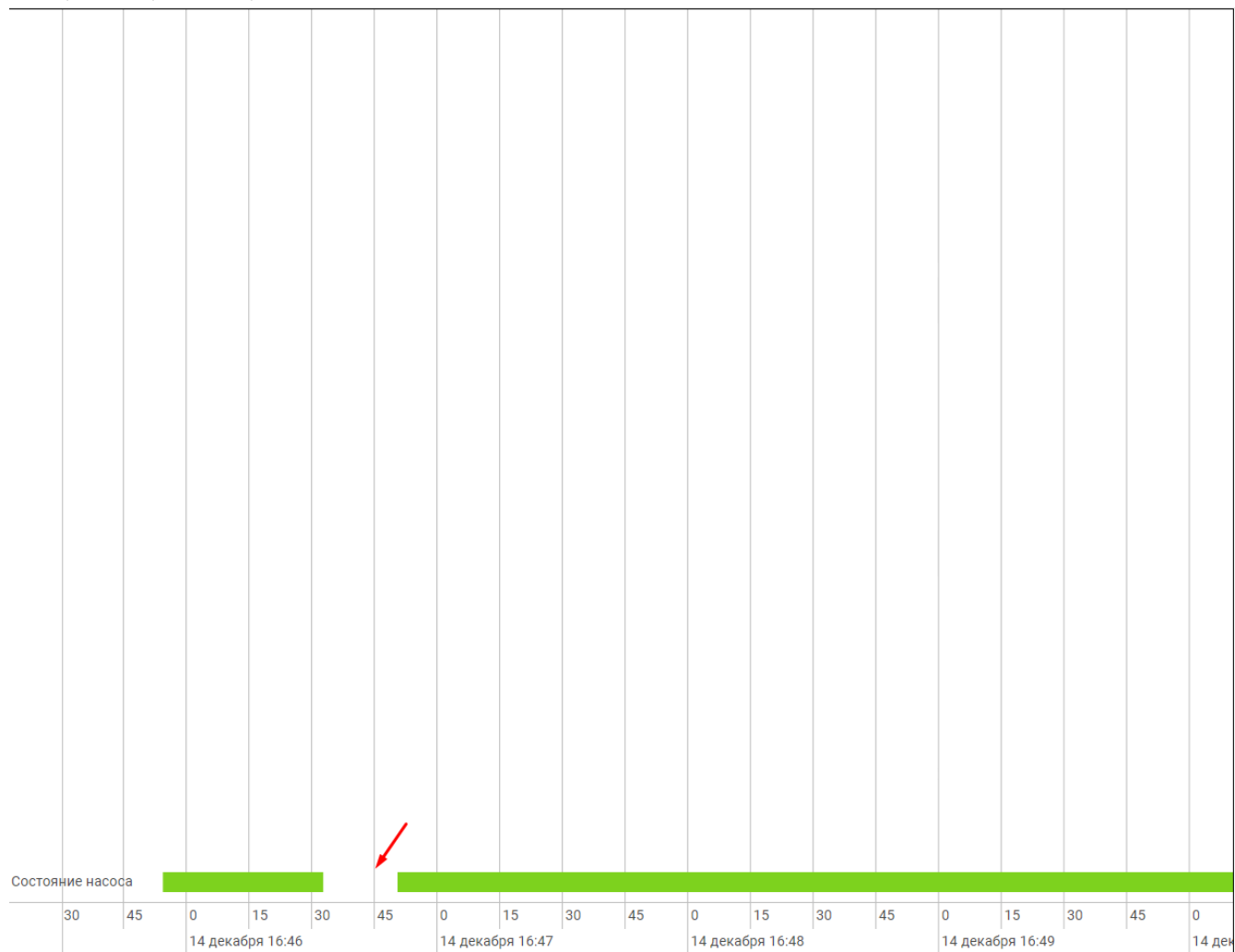
Цветовая группа	Значение	Цвет
1	1	

- В левом окне в поле Источник Данных привяжем свойство state Насоса



- В поле Легенда пишем "Состояние насоса"
- Добавляем на экран элемент Chart Timeline
- Настраиваем период отображения, отображение даты/времени, ширину, высоту и т.д.

В результате у нас получился график, отображающий свойство state насоса



Зелёная полоска - Насос работает, Полоски нет - Насос выключен

# Chart Columns

Chart Columns - это визуальный компонент выводит информацию в виде столбчатого графика, созданного с помощью графиков из [Аналитики](#). Для того, чтобы информация появилась на графике необходимо настроить [сохранение в БД](#) для свойства устройства

Кроме [общих свойств](#) у элемента Chart Columns есть индивидуальные свойства:

## Свойства элемента Chart Columns

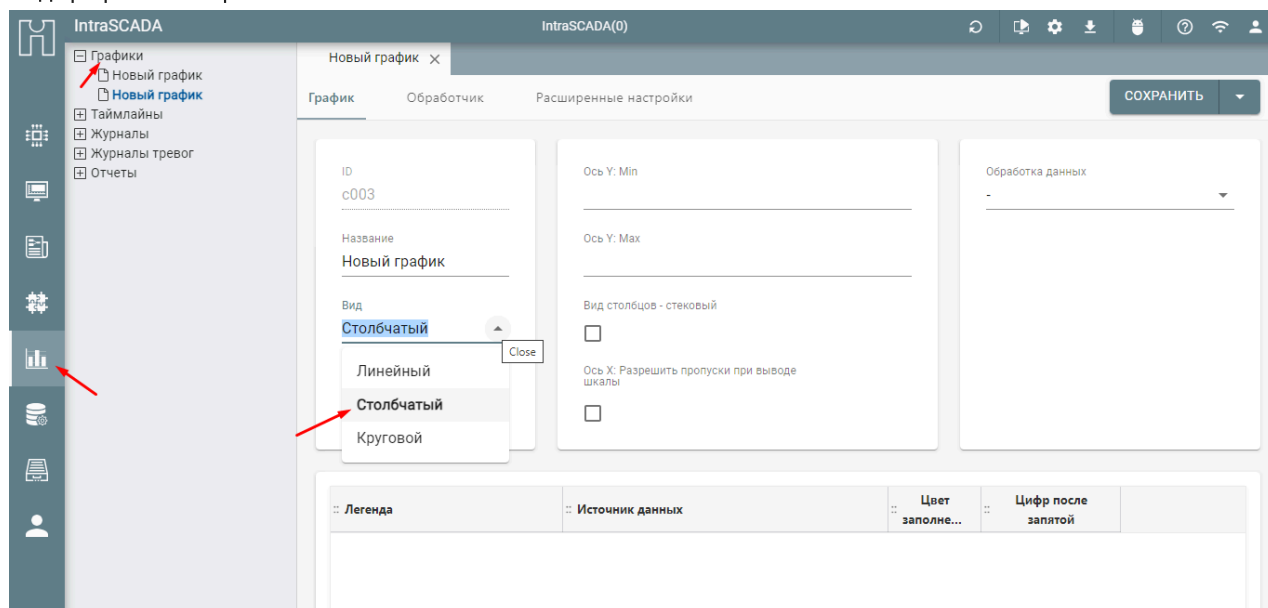
Наименование	Описание	Примечание
<b>Chart Columns</b>		
Период	Период отображения данных	Минута, Час, День, Неделя, Месяц, Пользовательский, Выберите переменную
Реалтайм	Отрисовка данных в режиме реального времени	
Цвет сетки	Цвет сетки	
<b>Зуммирование</b>		
По оси X	Увеличение/уменьшение масштаба графика по оси X	
По оси Y	Увеличение/уменьшение масштаба графика по оси Y	
<b>Перемещение</b>		
По оси X	Перемещение по графику по оси X	
По оси Y	Перемещение по графику по оси Y	
<b>Легенда</b>		
Положение	Положение текста	None, Слева Справа, Сверху, Снизу
Цвет текста	Цвет текста	
Размер текста	Размер текста	
<b>Точка</b>		
Радиус	Величина радиус точки	
Радиус при		

выделении	Величина радиус точки при выделении	
Всплывающая подсказка		
Показывать	Показывать подсказку	
Цвет фона	Цвет фона	
Цвет даты	Цвет даты	
Цвет значения	Цвет значения	
Ось X		
Положение	Положение Оси X	None, Сверху, Снизу
Цвет	Цвет текста Оси X	
Размер	Размер текста Оси X	
Ось Y		
Положение	Положение Оси Y	None, Сверху, Снизу
Цвет	Цвет текста Оси Y	

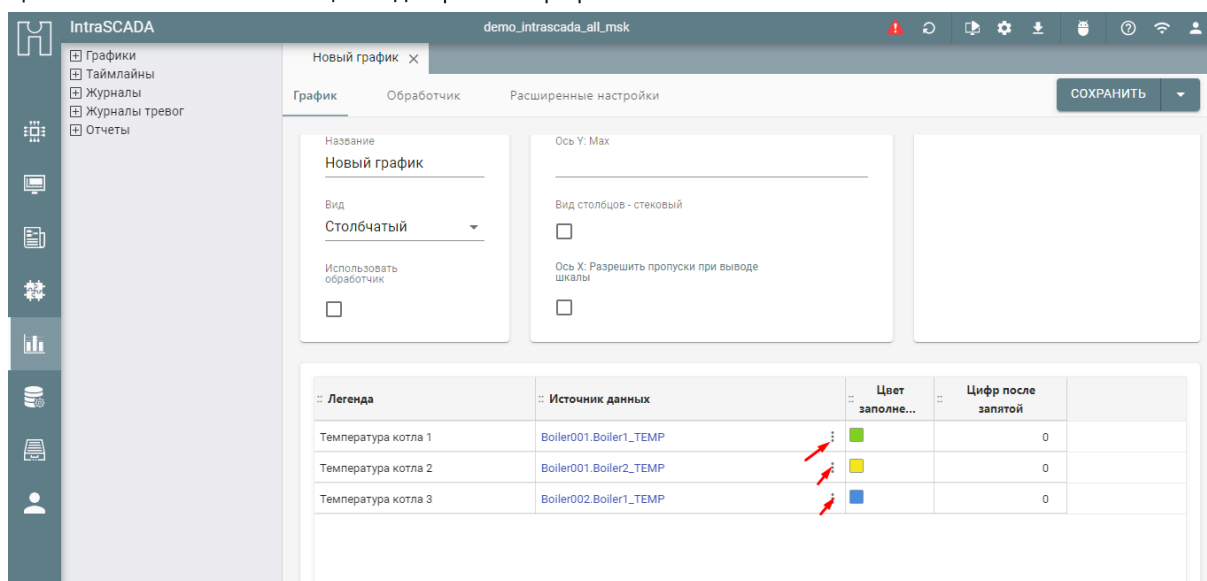
## Пример создания столбчатого графика

- Создаём новый график в разделе Аналитика - Графики

- Вид графика выбираем Столбчатый



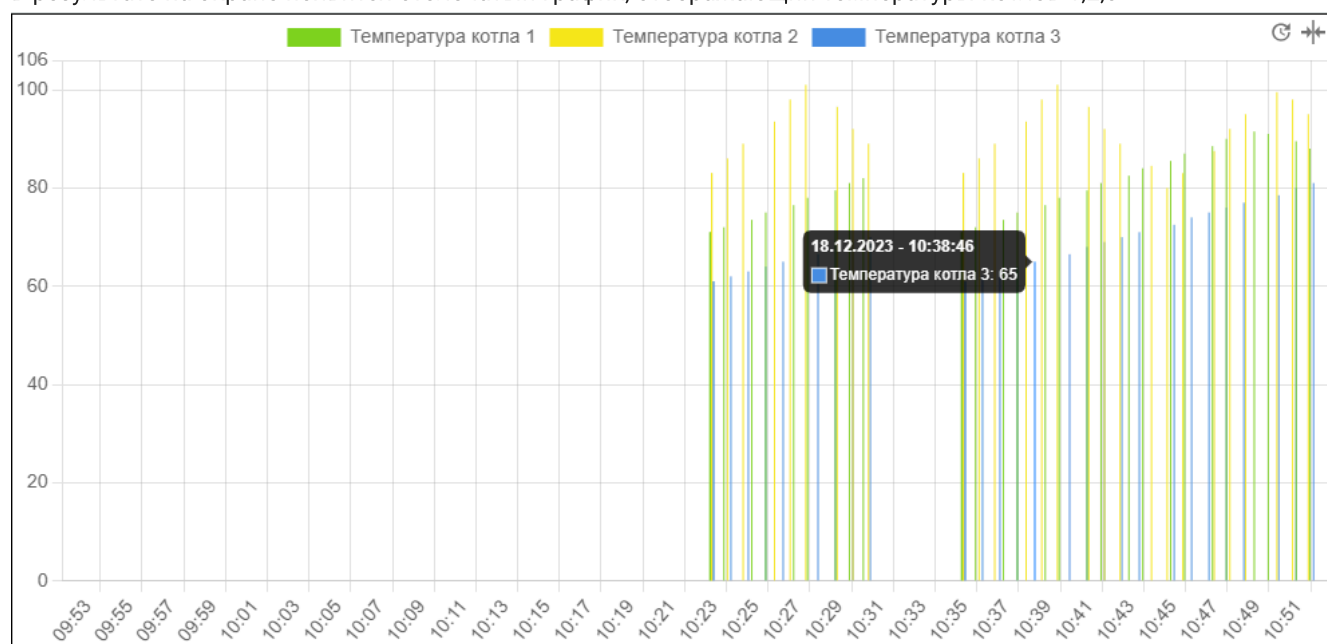
- При необходимости, настраиваем Min-Max значения, в данном случае график будет динамическим
- В нижней таблице добавляем три строки для формирования столбцов:
  - Легенда - Температура Котла 1,..
  - Источник данных - Привязка к свойству устройства (Boiler\_001.TEMP,..)
  - Цвет заполнения - Разные цвета для разных графиков



На экране добавляем элемент Chart Columns (Добавить Элемент - Charts - Chart Columns).

- Настраиваем период отображения данных (в примере "Час")
- Привязываем элемент Chart Columns к созданному нами ранее графику, жмём ОК.

В результате на экране появится столбчатый график, отображающий температуры котлов 1,2,3





# Chart Pie

Chart Pie - это визуальный компонент выводит информацию в виде круговой диаграммы, созданного с помощью графиков из [Аналитики](#). Для того, чтобы информация появилась на графике необходимо настроить [сохранение в БД](#) для свойства устройства

Кроме [общих свойств](#) у элемента Chart Pie есть индивидуальные свойства:

## Свойства элемента Chart Pie

Наименование	Описание	Примечание
<b>Chart Pie</b>		
Период	Период отображения данных	Минута, Час, День, Неделя, Месяц, Пользовательский, Выберите переменную
Вариант	Вариант отображения диаграммы	Classic, Classic padded, Classic outer, Modern, Modern strong, Donut, Donut rounded, Donut padded, Partial 90, Partial 180, Single
Скругление	Скругление элементов диаграммы	
textMode	Отображение текста	None, Значения, Проценты
Размер Текста	Размер Текста	
Цвет Текста	Цвет Текста	
Позиция метки	Позиция метки	
Начальный угол	Начальный угол отображения диаграммы	
Суммарный угол	Суммарный угол отображения диаграммы	
Ширина линии	Ширина линии диаграммы	
Угол отступа	Угол отступа каждого элемента диаграммы	
Радиус	Радиус диаграммы	
Сдвиг	Сдвиг от выбранного радиуса диаграммы	
Итоговое значение	Итоговое значение	

по X

Центр Y	Смещение диаграммы относительно центра по Y
View Box Size W	Ширина отображения диаграммы
View Box Size H	Высота отображения диаграммы

## Привязки

Привязка круговых диаграмм выполняется при нажатии кнопки Привязки. Для привязки доступны переменные клиента и круговые диаграммы.

# Iframe

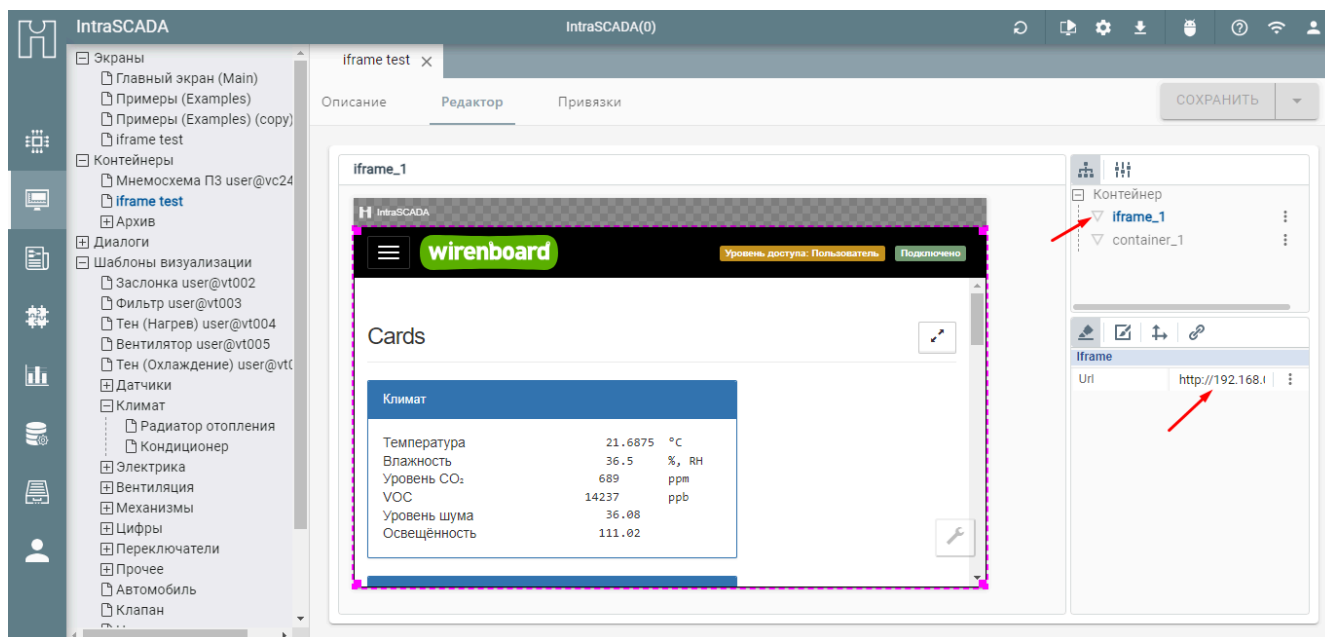
Iframe - элемент для отображения следующего веб контента:

1. Web страниц по прямому адресу URL
2. Файлов, которые поддерживает браузер : pdf, txt, csv и т.д.

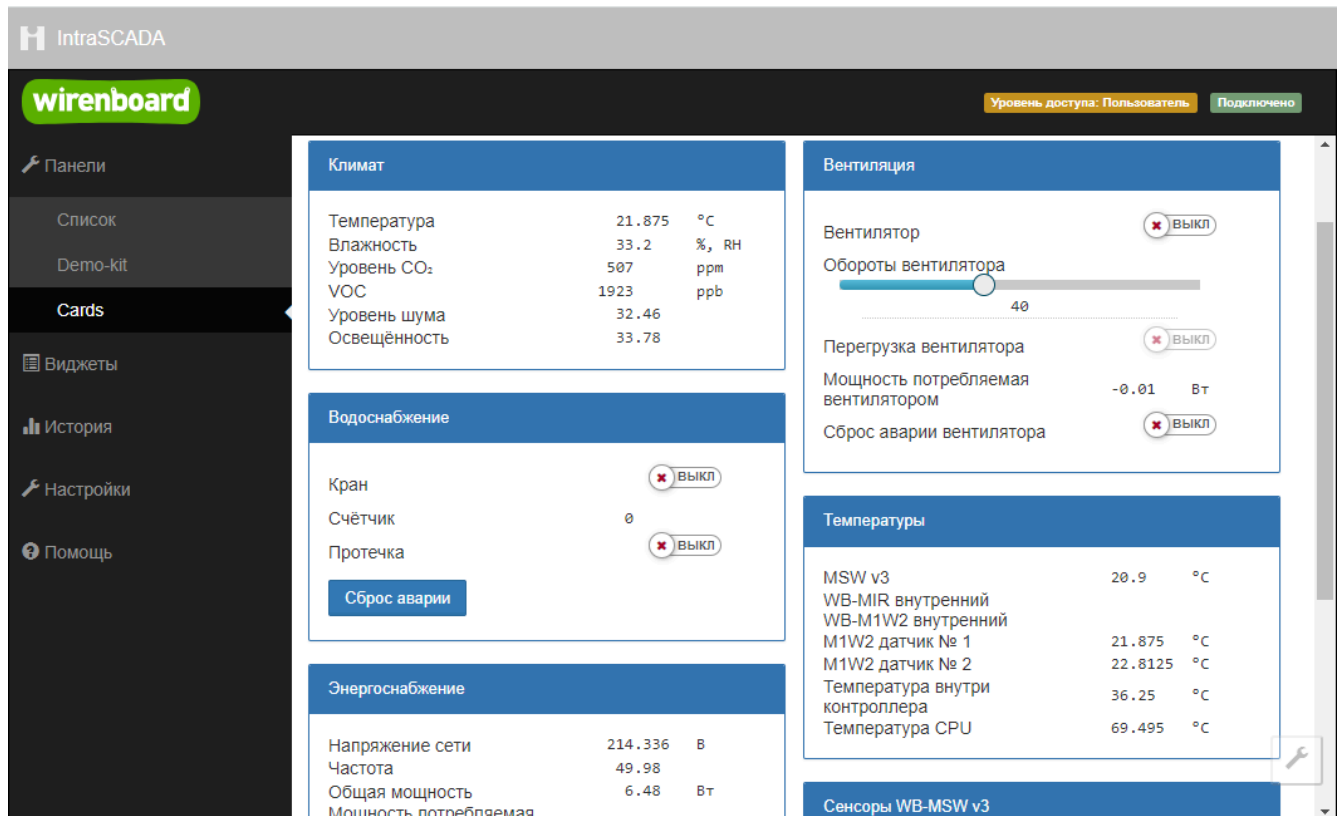
Любое использование ссылок на локальные url или пути файлов не будет работать через механизм p2p

Пример:

- Создаём новый контейнер, либо выбираем из уже имеющихся
- Добавляем в контейнер элемент Iframe(Правый клик - Добавить элемент - Widgets - Iframe)
- В редакторе вводим нужный нам URL адрес, либо адрес файла
- Редактируем ширину, высоту и т.д.
- Добавляем контейнер на экран



В результате на экране отобразится сайт либо файл, который мы настроили в виджете Iframe



Кроме [общих свойств](#), у элемента Iframe есть индивидуальные свойства:

## Свойства элемента Iframe

Наименование	Описание	Примечание
<b>Iframe</b>		
Url	Прямая ссылка на web страницу либо файл	

Пример URL:

- <https://intrascada.com> - прямая ссылка на ресурс
- /files/test.pdf - отображение файлов из ресурсов системы. Файлы должны находится в папке anyfiles в папке проекта
- /images/test.png - отображение изображений из ресурсов системы. Файлы должны находится в папках images или screenshots в папке проекта

# HTML

HTML - элемент для отображения HTML страницы, полностью созданной пользователем.

Любое использование ссылок на локальные js, css, html не будет работать через механизм p2p

Кроме [общих свойств](#), у элемента HTML есть индивидуальные свойства:

## Свойства элемента HTML

Наименование	Описание	Примечание
<b>HTML</b>		
Код	Редактор кода HTML	

Пример HTML кода:

```
<style type="text/css">

</style>

<script type="text/javascript">

</script>

<div>Hello World!</div>
```

*Добавлено v5.18.4*

Для HTML виджета добавлено API с набором методов для получения данных, подписки и управления

## Пояснения

**window.ihapi** - API с набором методов для управления

**uuid** - уникальный идентификатор виджета (строка), задан автоматически

## Сетевые запросы

Сетевые запросы (работает через P2P) - возвращают Promise:

- **window.ihapi.fetch('/images/...')** - аналог fetch, следует спецификации браузера
- **window.ihapi.fetch('/restapi/myapi').then(res => res.json()).then(json => console.log(json))**

## Управление устройствами и сценой

Управление устройствами и сценой:

- **window.ihapi.deviceList()** - массив имен устройств/свойств сцены (локально)
- **window.ihapi.deviceList(true)** - включает клиентские переменные (localXXX\_XX)
- **window.ihapi.deviceValue('d0282', 'value')** - текущее значение (строка/число, локально), использовать для переменных или без подписки
- **window.ihapi.deviceCommand('d0282', 'value', 100)** - отправка команды на сервер

## Команды сцены

Команды сцены:

- **window.ihapi.command({ command: 'gotolayout', id: 'l051' })** - переход на экран
- **window.ihapi.command({ command: 'setval', data: { d0282\_value: 100, local013\_v1: 55 } })** - локальное обновление сцены на клиенте в рамках сессии без привязки к серверу
- **window.ihapi.command({ command: 'showdialog', id: 'di0034', src: uuid })** - показать диалог
- **window.ihapi.command({ command: 'closedialog', id: 'di0034' })** - закрыть диалог
- **window.ihapi.command({ command: 'closealldialogs' })** - закрыть все диалоги
- **window.ihapi.command({ command: 'startvisscript', id: 'vs0003' })** - запустить скрипт визуализации
- **window.ihapi.command({ command: 'playsound', files: ['tada.wav'] })** - проиграть звук, требуется один клик пользователя после загрузки страницы (ограничение браузера)
- **window.ihapi.command({ command: 'repeatsound', files: ['tada.wav'] })** - проиграть звук с повтором
- **window.ihapi.command({ command: 'stopsound' })** - остановить все звуки
- **window.ihapi.command({ command: 'refresh' })** - обновить экран, синхронизировать с сервером

## Подписка на данные

Подписка на данные:

- **window.ihapi.addListener(uuid, 'data', callback)** - основной способ получения данных устройств и клиентских переменных, callback - функция слушатель. При создании виджета вызывается один раз со всеми значениями сцены, далее срабатывает при изменениях
- **window.ihapi.deviceSub(uuid, ['d0282\_value', 'local013\_v1'])** - подписка на данные с сервера, начальное значение приходит через 'data', клиентские переменные (localXXX\_XX) не поддерживают начальное значение через 'data', используйте deviceValue
- **window.ihapi.deviceUnsub(uuid, ['d0282\_value', 'local013\_v1'])** - отписка (обычно используется в destroy)

Где **localXXX\_XXXX** для переменной клиента берем из адресной строки браузера, когда выбираем переменную клиента в дереве проекта, а **XXXX** - это имя переменной клиента

## Уничтожение виджета

Уничтожение виджета:

- **window.ihapi.addEventListener(uuid, 'destroy', callback)** - реагирует на удаление (уничтожение) виджета, callback - функция слушатель

## Расширенные команды

Расширенные команды:

- **window.ihapi.templateMeta(uuid)** - если виджет HTML размещен внутри шаблона, то можно получить метаданные самого шаблона (список переменных, привязки и т.д.)
- **window.ihapi.templateValues(uuid)** - если виджет HTML размещен внутри шаблона, то можно получить значения привязок, может работать в паре с addEventListener - 'data' как триггер обновления данных

## Хранение данных пользователя

Хранение данных пользователя:

- **window.ihapi.getClientData('data1')** - получает данные текущего пользователя по указанному ключу. Возвращает Promise, с данными типа объект
- **window.ihapi.setClientData('data1', { a: 1, b: 2, c: 2 })** - сохраняет данные текущего пользователя по указанному ключу с данными типа объект.

## Пример для HTML виджета в контейнере

Типовой пример подписки на конкретное свойство устройство по id и вывод значения в log браузера :



```

<style type="text/css">

</style>

<script type="text/javascript">
    function destroy() {
        window.ihapi.deviceUnsub(uuid, ['d0282_value'])
    }

    function update(data) {
        console.log(data)
    }

    function init() {
        window.ihapi.addEventListener(uuid, 'destroy', destroy)
        window.ihapi.addEventListener(uuid, 'data', update)

        window.ihapi.deviceSub(uuid, ['d0282_value'])
    }

    init()
</script>

<div id="${uuid}">Hello World!</div>

```

## Пример для HTML виджета в шаблоне визуализации

Шаблон визуализации получает ссылки на необходимые переменные через привязку или мастер привязку, поэтому с помощью функции **const meta = window.ihapi.templateMeta(uuid)** необходимо получить подписки (subs) и их использовать для подписки на необходимые переменные **window.ihapi.deviceSub(uuid, meta.subs)**, так же использовать их при уничтожении компонента **window.ihapi.deviceUnsub(uuid, meta.subs)** для того, чтобы отписаться от устройств на сервере.

Для обновления значений по **window.ihapi.addEventListener(uuid, 'data', update)** воспользуемся функцией помощника **window.ihapi.templateValues(uuid)**, которая вернет объект со значениями переменных шаблона.

Для шаблона визуализации очень важно иметь ввиду, что таких шаблонов на одном контейнере может быть больше чем один, поэтому необходимо в названии id элемента в html использовать уникальный **uuid**

```

<style type="text/css">
  .div-text {
    width: 100%;
    height: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
  }
</style>

<script type="text/javascript">
  function update() {
    const data = window.ihapi.templateValues(uuid)

    document.getElementById(uuid).innerHTML = data.value
  }

  function destroy() {
    const meta = window.ihapi.templateMeta(uuid)

    window.ihapi.deviceUnsub(uuid, meta.subs)
  }

  function init() {
    const meta = window.ihapi.templateMeta(uuid)

    window.ihapi.addEventListener(uuid, 'destroy', destroy)
    window.ihapi.addEventListener(uuid, 'data', update)

    window.ihapi.deviceSub(uuid, meta.subs)
  }

  init();
</script>

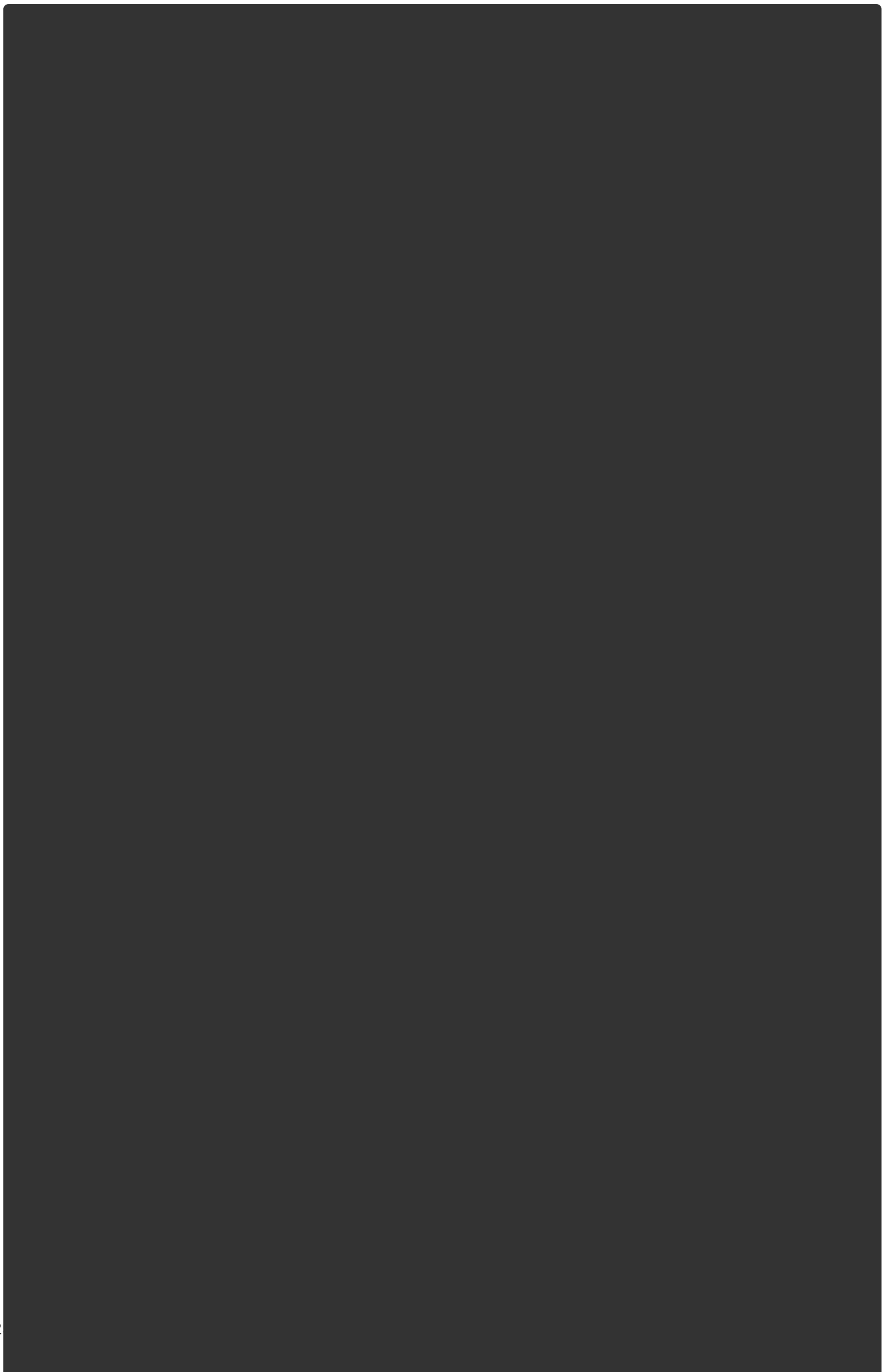
<div id="${uuid}" class="div-text">
  Loading...
</div>

```

Пример для HTML виджета с получением библиотеки three.mini.js с сервера

Для того, чтобы получить three.mini.js с сервера, ее необходимо разместить в папке проекта **anyfiles**

Данный пример работает только при прямом подключении, через р2р работать не будет



# CCTV

CCTV - элемент для отображения веб камер, которые работают через плагин CCTV

Всю доступную информацию можно найти [по ссылке](#)

## Привязки

Для привязки доступны только камеры из плагина CCTV

# Map

Map - Элемент для отображения интерактивной карты

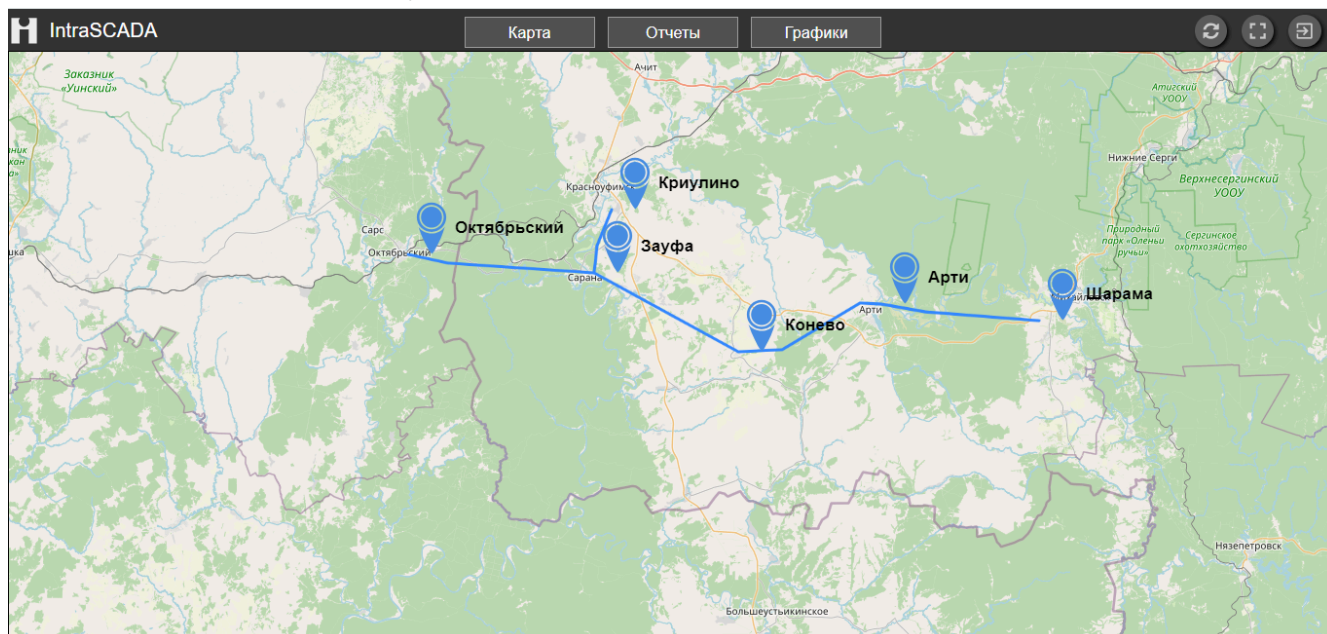
Кроме [общих свойств](#), у элемента Map есть индивидуальные свойства:

## Свойства элемента Map

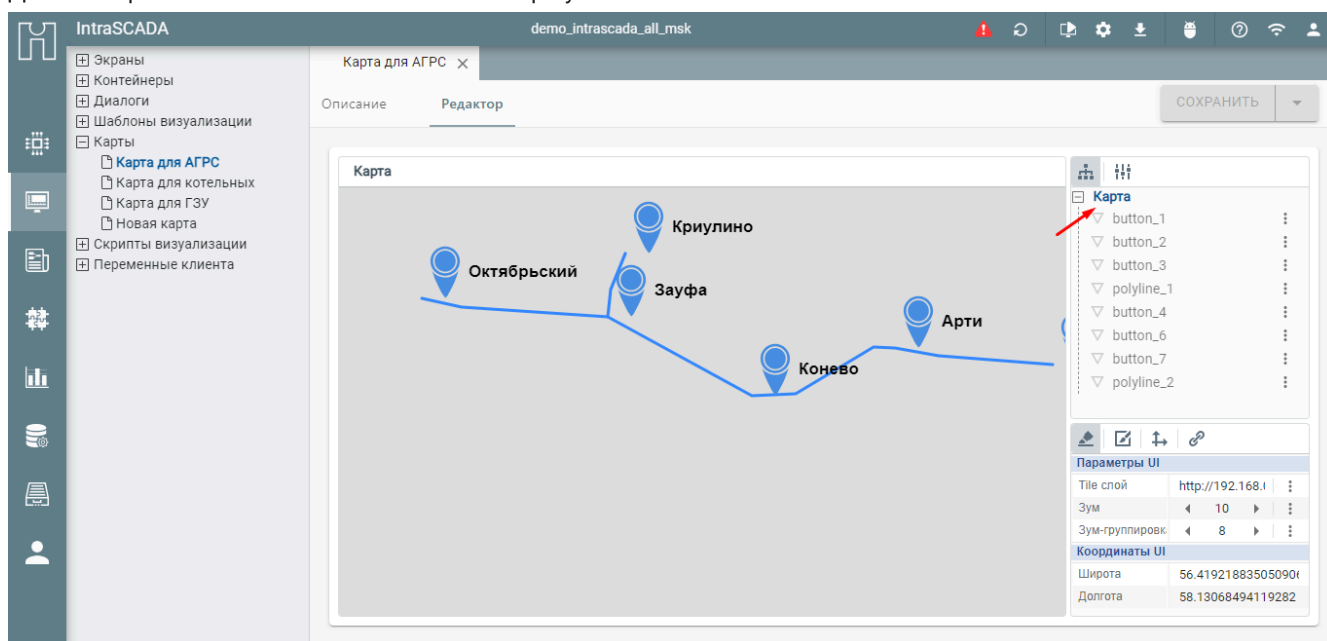
Наименование	Описание	Примечание
Мар		
Параметры UI		
Tile слой		
Зум	Приближение/Отдаление карты	
Зум-группировка		
Координаты UI		
Широта	Изменение Широты	
Долгота	Изменение Долготы	

## Примеры

В качестве примера возьмём карту для АГРС из демо-версии Пользовательского Интерфейса :

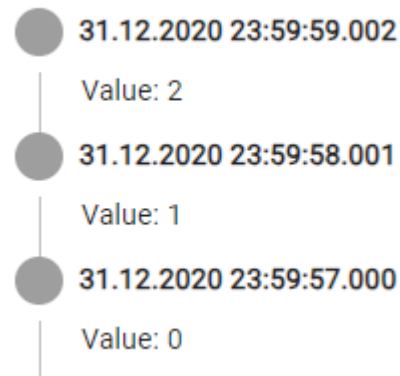


Данная карта состоит из элементов button и polyline



- Элементы button выступают как кнопки для перехода на соответствующий экран
- button\_1 привязан для перехода на Мнемосхему для АГРС Шарама, button\_2 для перехода на АГРС Арти и так далее.
- Элементы polyline выступают для визуализации путевых линий между АГРС
- Для каждого элемента указаны свои широта и долгота

# Device log



Device log - элемент для отображения журнала устройства:

## Пример использования

- Выбираем устройство, свойство которого нам нужно отобразить в Журнале Устройства(Device log)
- Заходим во вкладку Свойства, и выбираем свойства, которые нужно отобразить в Журнале устройства

**IntraSCADA** demo\_intrascada\_all\_msk

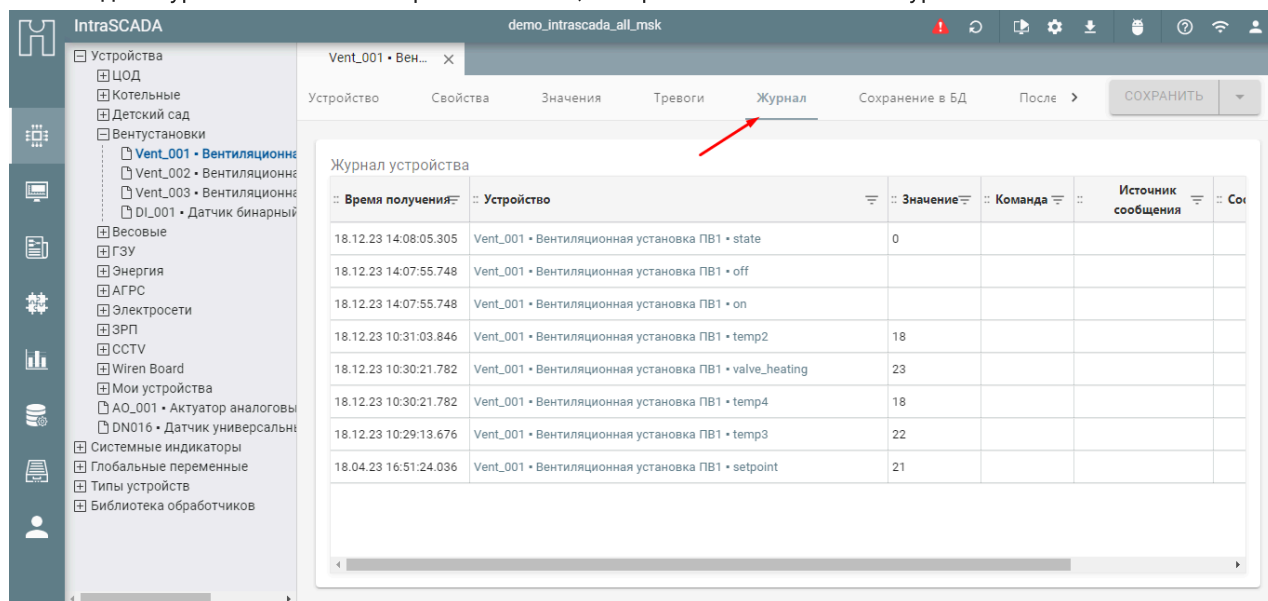
Устройство: Vent\_001 • Вен...

Вкладки: Устройство | **Свойства** | Значения | Тревоги | Журнал | Сохранение в БД | После > | СОХРАНИТЬ

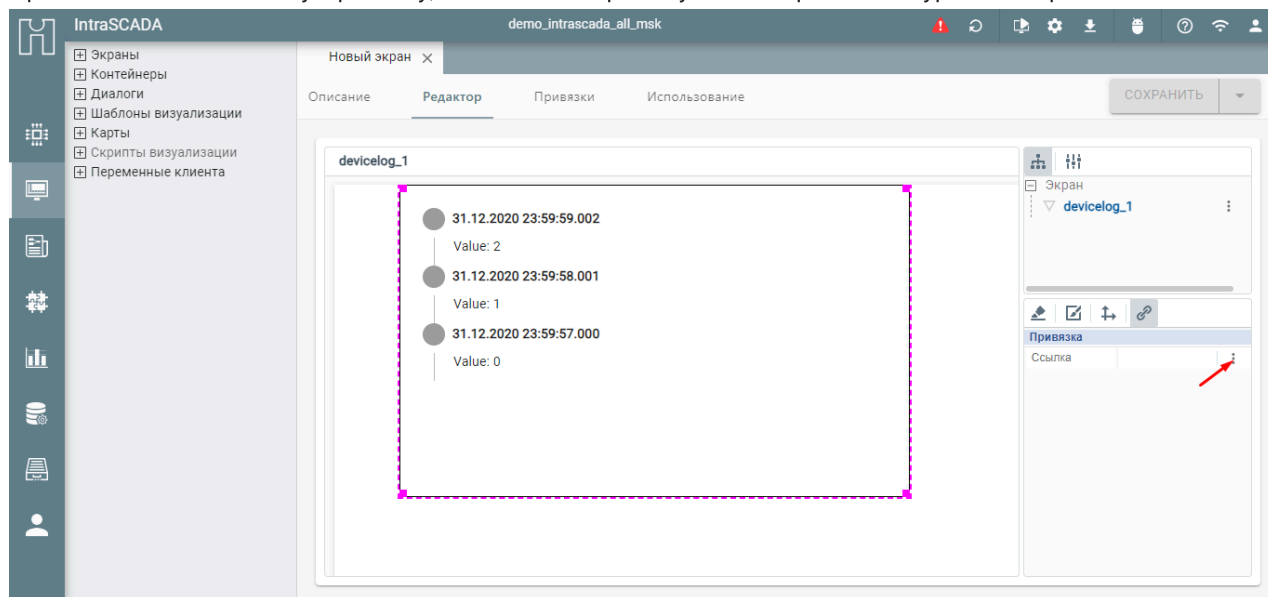
ID свойства	Название	Значение	Тип свойства	Сохранять в журнал устройства	Сохранять в главный журнал	Цифр после запятой	Ед.изм.
state	Состояние установки	0	Data Bool	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
temp1	Температура наружн...	0	Event Number	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	°C
temp2	Температура приточ...	22	Data Number	<input type="checkbox"/>	<input type="checkbox"/>	0	°C
temp3	Температура обратн...	18	Data Number	<input type="checkbox"/>	<input type="checkbox"/>	0	°C
temp4	Температура вытяжки	20	Data Number	<input type="checkbox"/>	<input type="checkbox"/>	0	°C
temp5	Температура обратн...	0	Data Number	<input type="checkbox"/>	<input type="checkbox"/>	0	°C
setpoint	Уставка температур...	21	Parameter Nu...	<input type="checkbox"/>	<input type="checkbox"/>	0	°C
fan1	Вентилятор приточки	0	Data Bool	<input type="checkbox"/>	<input type="checkbox"/>		
fan2	Вентилятор вытяжки	0	Data Bool	<input type="checkbox"/>	<input type="checkbox"/>		
filter1	Фильтр приточки	0	Data Bool	<input type="checkbox"/>	<input type="checkbox"/>		
filter2	Фильтр вытяжки	0	Data Bool	<input type="checkbox"/>	<input type="checkbox"/>		



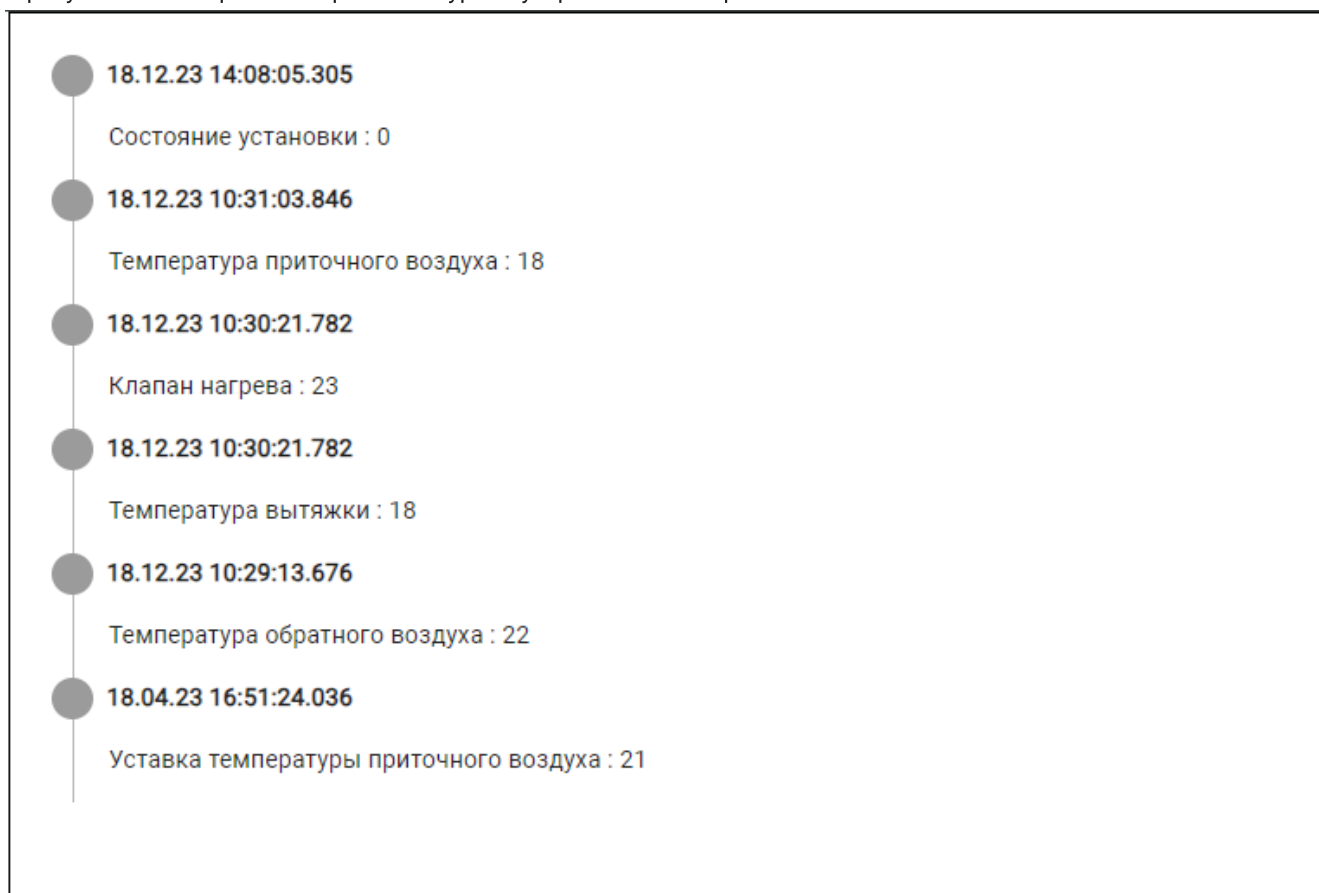
- На вкладке Журнал можно посмотреть свойства, которые записываются в журнал



- Добавляем элемент Device log на экран(Добавить элемент - Widgets - Device Log)
- Привязываем элемент к устройству, свойства которого нужно отобразить в Журнале Устройства



В результате на экране отобразится журнал устройства с выбранными свойствами



# Journal

Journal - данный визуальный компонент выводит информацию в виде таблицы о событиях в системе, созданной с помощью журналов в [Аналитике](#)

Кроме [общих свойств](#) у элемента Journal есть индивидуальные свойства:

## Свойства элемента Journal

Наименование	Описание
<b>Таблица</b>	
Цвет фона шапки	Цвет фона шапки
Цвет текста шапки	Цвет текста шапки
Цвет иконок шапки	Цвет иконок шапки
Цвет сетки	Цвет сетки
<b>Стандартное сообщение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Предупреждение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Аварийное сообщение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Панель фильтра</b>	
Цвет фона	Цвет фона
Цвет текста	Цвет текста
<b>Поля ввода</b>	
Цвет текста	Цвет текста

Цвет при наведении	Цвет при наведении
Цвет при выборе	Цвет при выборе
<b>Список выбора</b>	
Цвет списка	Цвет списка
Цвет текста	Цвет текста
Цвет фона	Цвет фона
Цвет при выборе	Цвет при выборе
Цвет при наведении	Цвет при наведении
<b>Календарь</b>	
Цвет фона	Цвет фона
Цвет текста	Цвет текста
Цвет рамки	Цвет рамки
Цвет кнопки	Цвет кнопки
Цвет кнопки при наведении	Цвет кнопки при наведении
День, цвет текста	
День, цвет фона	
День н/а, цвет текста	
День н/а, цвет фона	
Выбранный день, цвет фона	

Выбранный диапазон, цвет фона

---

Выбранный диапазон, цвет текста

---

Диапазон при выборе, цвет фона

---

Диапазон при выборе, цвет текста

---

Диапазон при наведении, цвет фона

---

Диапазон при наведении, цвет текста

---

День при наведении, цвет текста

---

День при наведении, цвет фона

---

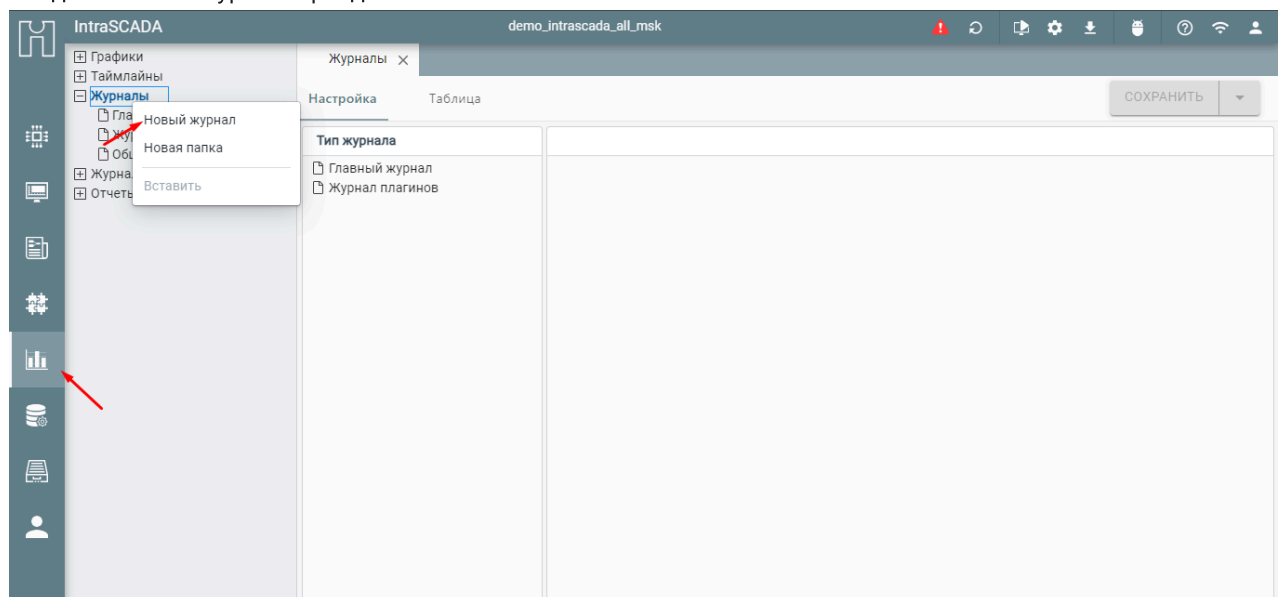
## Привязки

Привязка журнала выполняется при нажатии кнопки Привязки. Для привязки доступны переменные клиента и журналы.

## Пример создания журнала

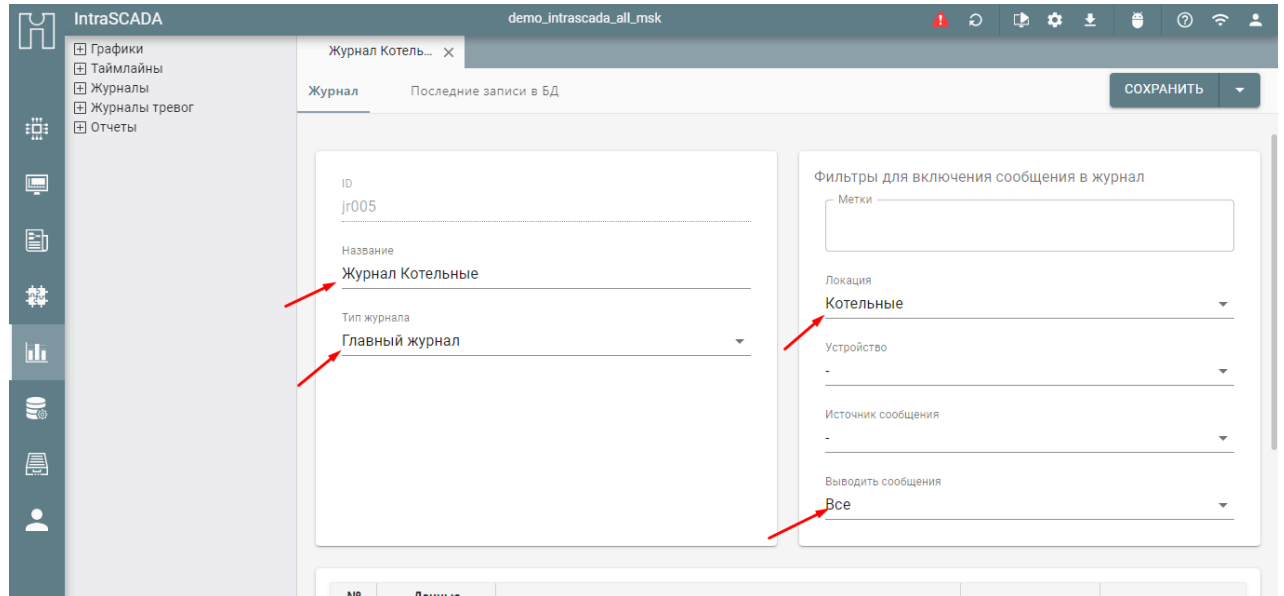
Создадим журнал мониторинга Котельных

- Создаём новый журнал в разделе Аналитика



- Пишем имя Журнала (В примере - Журнал Котельные)
- Тип журнала выбираем "Главный Журнал"

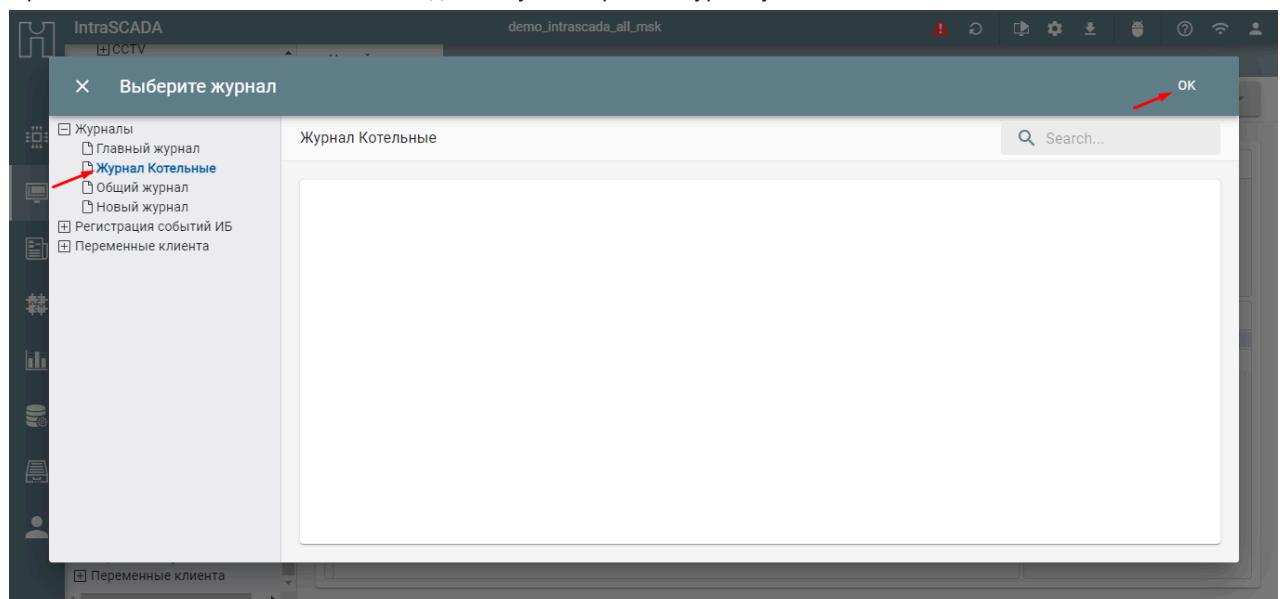
- Локацию выбираем "Котельные"
- Выводить сообщения Выбираем "Все"



- Добавляем столбцы : выбираем Данные столбца, заполняем Шапку столбца, указываем Ширину столбца

№ пп	Данные столбца	Шапка столбца	Ширина столбца
0	Уровень соо...	Уровень сообщения	100
1	Дата, время	Дата	200
2	Сообщение	Текст сообщения	300
3	Устройство	Устройство	0200
4	Локация	Локация	200

- Добавляем элемент Journal на экран(Добавить элемент - Widgets - Journal)
- Привязываем элемент Journal к созданному нами ранее журналу, жмём OK



В результате на экране отобразится журнал мониторинга Котельных

Метки	Урове... сооб...	Дата	Текст сообщения	Устройство	Локация
ВМК	Предупрежд...	21.12.2023 10:14:10	Давление газа : верхний предупредительны...	Boiler001 • ВМК 17	Котельные
ВМК	Предупрежд...	21.12.2023 10:14:00	Давление газа : верхний предупредительны...	Boiler002 • ВМК 21	Котельные
ВМК	Стандартное...	21.12.2023 10:13:55	Температура на выходе котла 2 : 89°C	Boiler001 • ВМК 17	Котельные
ВМК	Авария	21.12.2023 10:13:50	Давление газа : верхний аварийный уровень...	Boiler002 • ВМК 21	Котельные
ВМК	Стандартное...	21.12.2023 10:13:40	Температура на выходе котла 1 : 84°C	Boiler001 • ВМК 17	Котельные
ВМК	Предупрежд...	21.12.2023 10:13:30	Давление газа : верхний предупредительны...	Boiler002 • ВМК 21	Котельные
ВМК	Предупрежд...	21.12.2023 10:13:20	Температура на выходе котла 2 : верхний пр...	Boiler003 • ВМК 38	Котельные
ВМК	Предупрежд...	21.12.2023 10:13:19	Температура на выходе котла 1 : верхний пр...	Boiler002 • ВМК 21	Котельные
ВМК	Стандартное...	21.12.2023 10:13:10	Температура на выходе котла 2 : 92°C	Boiler001 • ВМК 17	Котельные
ВМК	Стандартное...	21.12.2023 10:13:00	Температура на выходе котла 1 : 83°C	Boiler001 • ВМК 17	Котельные
ВМК	Предупрежд...	21.12.2023 10:12:50	Давление газа : нижний предупредительный...	Boiler002 • ВМК 21	Котельные
ВМК	Предупрежд...	21.12.2023 10:12:41	Температура на выходе котла 2 : верхний пр...	Boiler002 • ВМК 21	Котельные
ВМК	Авария	21.12.2023 10:12:39	Давление газа : нижний аварийный уровень ...	Boiler002 • ВМК 21	Котельные
ВМК	Предупрежд...	21.12.2023 10:12:34	Температура на выходе котла 2 : верхний пр...	Boiler001 • ВМК 17	Котельные

## Фильтры журнала

Нужные сообщения в журнале можно выбирать с помощью **фильтров**

Метки	Уровень сообщения	Дата	Текст сообщения	Устройство	Локация
ВМК	Авария	21.12.2023 11:2			Котельные
ВМК	Предупреждение	21.12.2023 11:2			Котельные
ВМК	Авария	21.12.2023 11:2			Котельные
ВМК	Предупреждение	21.12.2023 11:2			Котельные
ВМК	Стандартное сообщение	21.12.2023 11:2			Котельные
ВМК	Авария	21.12.2023 11:2			Котельные
ВМК	Предупреждение	21.12.2023 11:2			Котельные
ВМК	Стандартное сообщение	21.12.2023 11:2			Котельные
ВМК	Предупреждение	21.12.2023 11:2			Котельные
ВМК	Стандартное сообщение	21.12.2023 11:2			Котельные
ВМК	Предупреждение	21.12.2023 11:2			Котельные
ВМК	Предупреждение	21.12.2023 11:2			Котельные
ВМК	Стандартное сообщение	21.12.2023 11:2			Котельные

Настройки фильтра

Метки

Уровень сообщения

Дата

Текст сообщения

Устройство

Локация

УДАЛИТЬ ФИЛЬТР

Сообщения можно фильтровать по :

- Меткам
- Уровням сообщения(Стандартное сообщение, Предупреждение, Авария)
- Дате
- Тексту сообщения
- Устройству
- Локации



Применённые фильтры отображаются на иконках фильтров

Уровень сообщения	Дата	Текст сообщения	Устройство	Локация
Предупреждение	21.12.2023 11:31:47	Давление газа : нижний предупредительный...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:31:07	Температура на выходе котла 2 : верхний пр...	Boiler003 • БМК 38	Котельные
Предупреждение	21.12.2023 11:31:07	Давление газа : верхний предупредительны...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:30:57	Давление газа : верхний предупредительны...	Boiler003 • БМК 38	Котельные
Предупреждение	21.12.2023 11:30:37	Давление газа : верхний предупредительны...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:30:02	Давление газа : нижний предупредительный...	Boiler001 • БМК 17	Котельные
Предупреждение	21.12.2023 11:29:57	Давление газа : нижний предупредительный...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:29:42	Давление газа : верхний предупредительны...	Boiler003 • БМК 38	Котельные
Предупреждение	21.12.2023 11:29:27	Давление газа : нижний предупредительный...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:28:47	Давление газа : верхний предупредительны...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:28:37	Авария котельной	Boiler003 • БМК 38	Котельные
Предупреждение	21.12.2023 11:28:32	Температура на выходе котла 2 : верхний пр...	Boiler002 • БМК 21	Котельные
Предупреждение	21.12.2023 11:28:32	Давление газа : нижний предупредительный...	Boiler001 • БМК 17	Котельные
Предупреждение	21.12.2023 11:28:17	Давление газа : верхний предупредительны...	Boiler002 • БМК 21	Котельные

# Alert Journal

Alert Journal - данный визуальный компонент выводит информацию в виде таблицы о текущих аварийных событиях, созданной с помощью журналов тревог в [Аналитике](#)

Кроме [общих свойств](#) у элемента Alert Journal есть индивидуальные свойства:

## Свойства элемента Alert Journal

Наименование	Описание
<b>Таблица</b>	
Цвет фона шапки	Цвет фона шапки
Цвет текста шапки	Цвет текста шапки
Цвет иконок шапки	Цвет иконок шапки
Цвет сетки	Цвет сетки
<b>Стандартное сообщение</b>	
Цвет текста строки	Цвет текста строки
Цвет фона строки	Цвет фона строки
<b>Предупреждение</b>	
Цвет текста - неактивное сообщение	Цвет текста строки
Цвет фона - неактивное сообщение	Цвет фона строки
Цвет текста - активное сообщение	Цвет текста строки
Цвет фона - активное сообщение	Цвет фона строки
<b>Аварийное сообщение</b>	
Цвет текста - неактивное сообщение	Цвет текста строки
Цвет фона - неактивное сообщение	Цвет фона строки
Цвет текста - активное сообщение	Цвет текста строки
Цвет фона - активное сообщение	Цвет фона строки
<b>Кнопка подтверждения для предупреждений</b>	

Цвет фона кнопки

Цвет фона кнопки

**Кнопка подтверждения для аварийных сообщений**

Цвет текста кнопки

Цвет текста кнопки

Цвет фона кнопки

Цвет фона кнопки

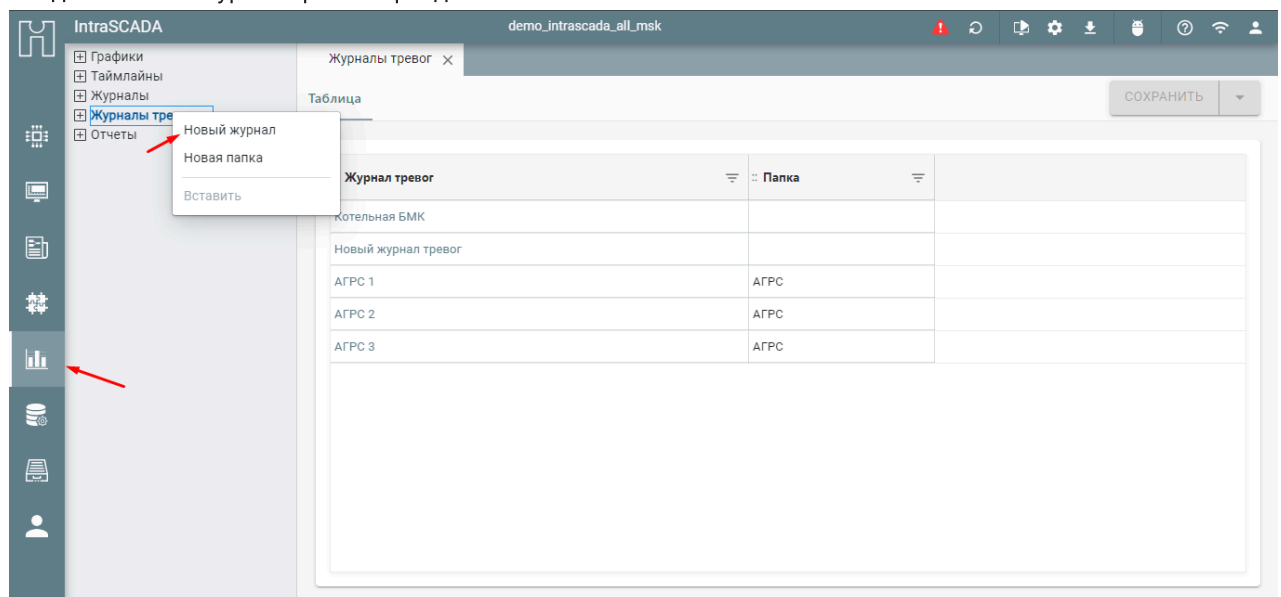
## Привязки

Привязка журнала тревог выполняется при нажатии кнопки Привязки. Для привязки доступны переменные клиента и журналы тревог.

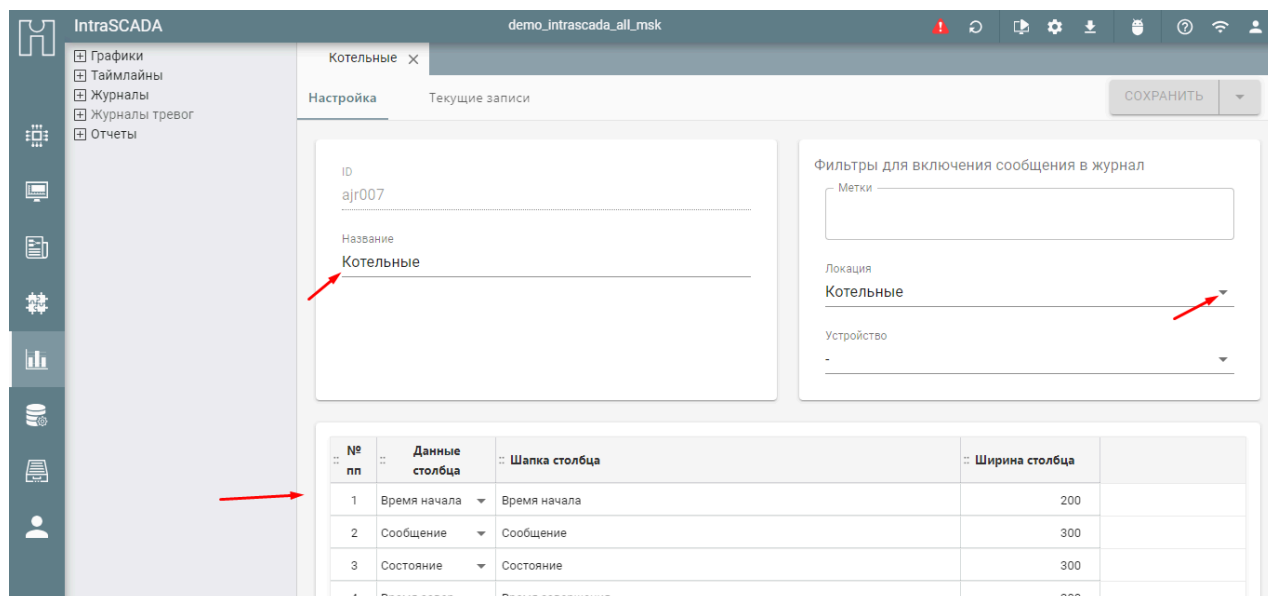
## Пример создания журнала тревог

Создадим журнал мониторинга тревог котельных

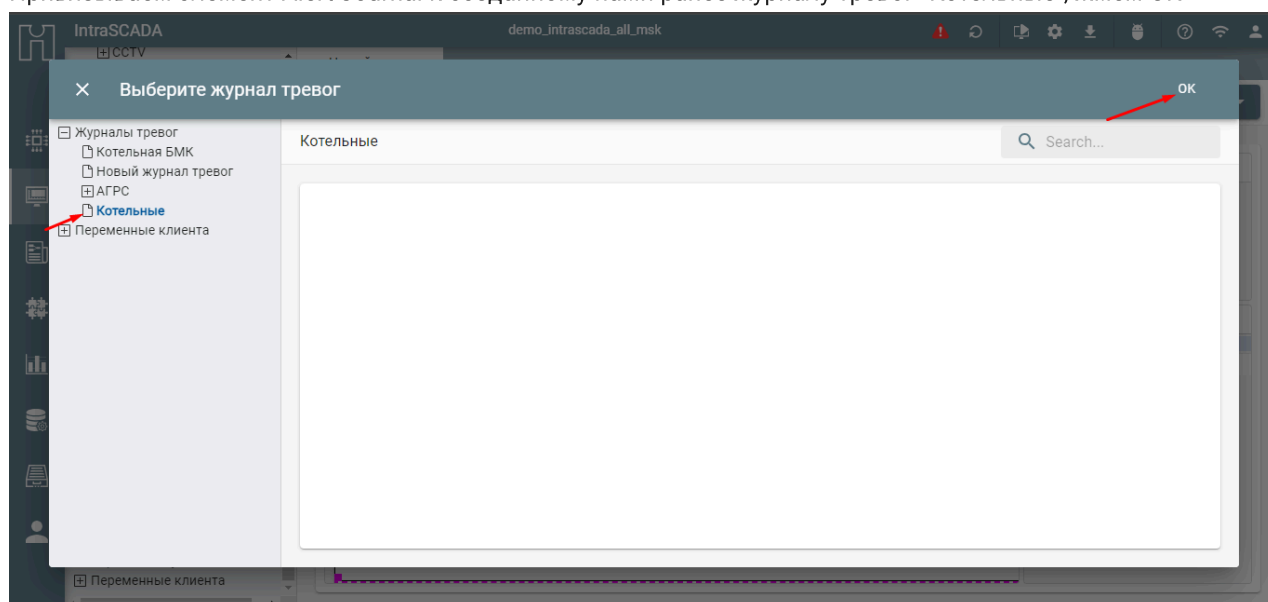
- Создаём новый журнал тревог в разделе Аналитика



- Пишем Название "Котельные"
- Выбираем Локацию "Котельные"
- Заполняем столбцы (по умолчанию отображаются 8 столбцов, при желании ненужные можно убрать, либо добавить свои)



- Добавляем элемент Alert Journal на экран( Добавить элемент - Widgets - Alert Journal)
- Привязываем элемент Alert Journal к созданному нами ранее журналу тревог "Котельные", жмём OK



В результате на экране отобразится журнал мониторинга тревог котельных

⌵ Время начала	⌵ Сообщение	⌵ Состояние	⌵ Время завершения
21.12.23 11:19:16.501	Давление газа : верхний аварийный уровень...	Активно	
21.12.23 11:19:16.501	Давление газа : нижний аварийный уровень ...	Активно	
21.12.23 11:19:16.501	Давление газа : верхний предупредительны...	Активно	
21.12.23 11:19:14.240	Температура выше нормы	Активно	
21.12.23 11:18:47.648	Температура на выходе котла 2 : верхний пр...	Активно	
21.12.23 11:18:46.445	Температура на выходе котла 2 : верхний пр...	Активно	
21.12.23 11:16:26.211	Температура на выходе котла 1 : верхний пр...	Активно	

# Report

Report - данный визуальный компонент выводит информацию в виде отчета в формате pdf, созданного с помощью Отчетов в [Аналитике](#). На компоненте доступен выбор отчетов, периода запроса данных, выгрузка PDF, CSV.

Кроме [общих свойств](#) у элемента Report есть индивидуальные свойства:

## Свойства элемента Report

Наименование	Описание
<b>Report</b>	
Цвет фона панели	Цвет фона панели
Цвет текста панели	Цвет текста панели
Цвет иконок панели	Цвет иконок панели

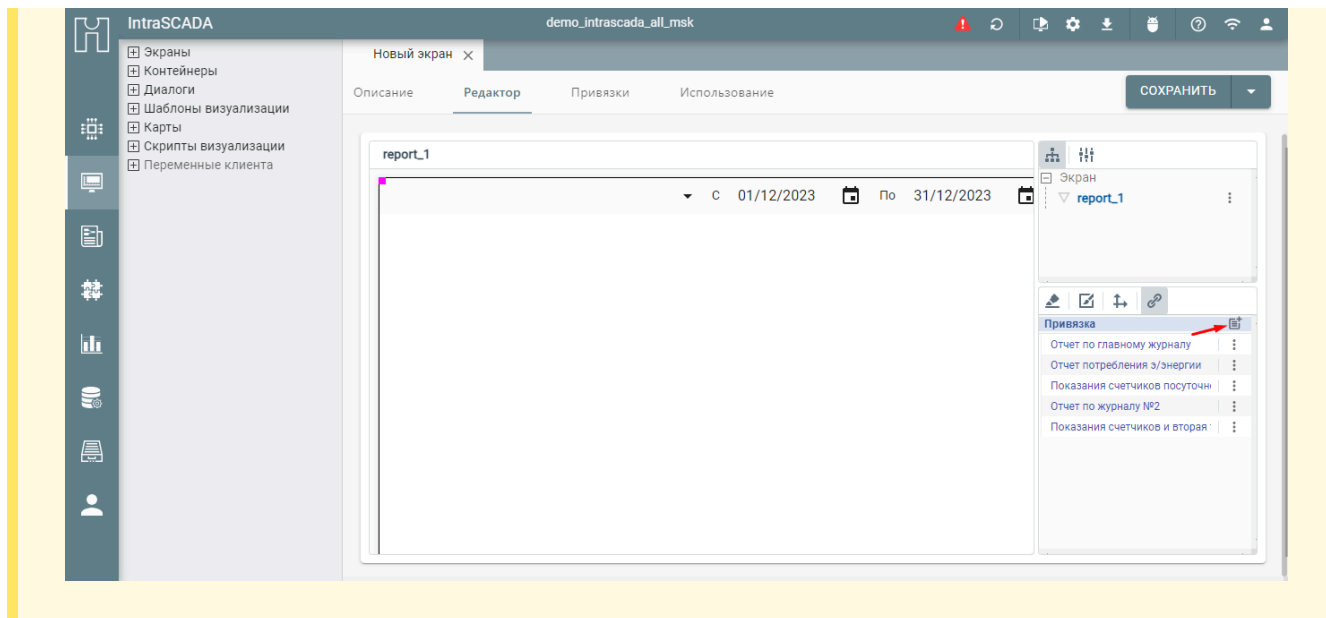
## Привязки

Привязка отчёта выполняется при нажатии кнопки Привязки. Для привязки доступны отчёты, созданные в [Аналитике](#).

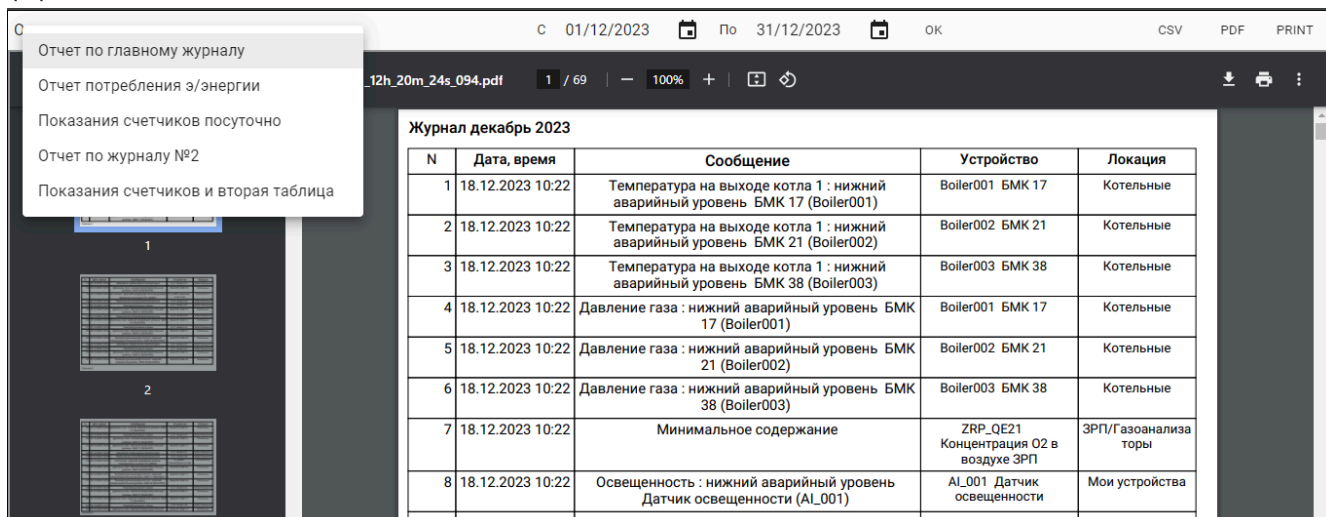
## Пример отчёта

- Создаём отчёт в [Аналитике](#)
- Добавляем на экран элемент Report (Добавить элемент - Widgets - Report)
- Привязываем элемент Report к отчётам, созданным в [Аналитике](#)

Элемент Report можно привязать сразу к нескольким отчётам, пользователь сможет выбрать нужный отчет из выпадающего списка



В результате на экране отобразятся выбранные нами отчёты, которые можно распечатать или сохранить в форматах CSV,PDF





# Скрипты визуализации

Скрипты визуализации позволяют добавить логику и интерактивность вашим интерфейсам.

## Для чего используются

Рассмотрим примеры задач, которые можно решить с помощью скрипта:

### 1. Навигация

- о переключение контейнеров
- о переходы по экранам
- о показ диалогов

Все эти задачи можно решить и без скрипта, просто выбирая действия по кликам.

Но если переход должен быть условный и/или нужно учитывать несколько параметров, без скрипта не обойтись.

```
module.exports = async function ({local, context, source}, core) {  
  if (local.level == 1) {  
    core.gotoLayout('l022');  
  } else {  
    // переход на стартовый экран пользователя  
    core.gotoLayout(context.start_layoutid)  
  }  
}
```

Также скрипт может обновить экран и заполнить элементы в зависимости от текущего контекста (пользователя, введенных данных)

### 2. Валидация пользовательского ввода.

Более простой способ решить эту задачу - сделать невозможным ввод неверных данных за счет выбора правильного элемента и его настройки (тип, диапазон, шаг).

Если этого недостаточно - поможет скрипт, в котором можно вывести всплывающее окно с предупреждением о неверном вводе. Можно также выдать звуковое сообщение.

```
module.exports = async function ({local, context, source}, core) {
  if (!local.carnumber) {
    core.playSound('beep.wav');
    return {err: 'Введите номер машины!'};
  }
}
```

### 3. Использование в UI интерфейсе списков (List, Autocomplete).

Если список фиксирован, заполнить его можно интерактивно.

[Визуальный элемент Список](#)

Для динамического формирования списка нужен скрипт.

Другой скрипт должен отработать выбор элемента списка.

### 4. Использование в UI интерфейсе таблиц (Table).

Для таблицы нужен скрипт, который определит столбцы таблицы и заполнит строки.

Более подробную информацию можно найти по [ссылке](#).

### 5. Использование в UI интерфейсе деревьев (Tree).

Одноуровневое дерево можно заполнить интерактивно.

С помощью скрипта можно построить любое дерево.

Более подробную информацию можно найти по [ссылке](#).

## Доступный функционал

- Навигация и обновление данных экрана в зависимости от контекста
- Работа с устройствами системы
- Работа с пользовательскими таблицами
- Получение снимков с камер
- Запись сообщений в журнал
- Запуск сценариев с передачей аргументов
- Вывод звуковых сообщений вызвавшему клиенту
- Отправка команд информирования

[Описание API скрипта визуализации](#)

## Библиотека функций

*Добавлено v5.17.80*

Для скриптов визуализации добавлен раздел *Библиотека функций*

[Более подробно про библиотеку функций](#)

## Отличие от сценариев

На первый взгляд скрипт визуализации выглядит как разновидность сценария. Однако между ними есть существенные отличия, которые нужно учитывать при написании скриптов:

### Контекст

- Сценарий работает в контексте устройств и системы в целом. В любой момент этот контекст един для всех пользователей системы.
- Скрипт визуализации работает в контексте индивидуального пользовательского экрана, действия выполняются от имени пользователя.

### Цикл жизни экземпляра

- Для сценария существует понятие **активное состояние**. Система отслеживает и фиксирует начало и завершение работы.  
Известно, в каком состоянии находится сценарий в данный момент.  
Если экземпляр сценария запущен, повторно он не запустится.
- Скрипт визуализации, напротив, может быть запущен многократно с различных клиентских терминалов  
Для каждого клиента скрипт должен отработать переданный контекст.

### Асинхронность

- Каждая функция сценария работает синхронно, асинхронность контролируется движком. При завершении сценария происходит сброс ресурсов этого сценария. Таким образом гарантируется, что не произойдет накопления небработоранных таймеров, подвисших обработчиков и т.д.
- Скрипт визуализации допускает асинхронные операции, можно использовать `await`, `setTimeout` напрямую.

### Действия с устройствами

- Если сценарий выполняет действия с устройствами, отправителем команды считается сценарий.
- Для скрипта визуализации действия выполняются от имени пользователя.

### Трассировка и отладка

- Для сценария выполняется полная трассировка всех действий.
- Для скрипта визуализации такая трассировка не выполняется. Для отслеживания выполнения в отладчике можно использовать функцию `debug`.

### Рекомендации по созданию скрипта

Из всего изложенного следуют рекомендации по созданию скрипта визуализации:

- скрипт визуализации не должен содержать сложный, разветвленный код, потенциально ведущий к утечкам памяти.
- цель скрипта - отработать переданный клиентом контекст и вернуть результат.
- асинхронность используется, но она должна быть контролируема.
- если нужен долгоиграющий процесс, лучше использовать сценарий (его можно запустить из скрипта)

# Структура скрипта визуализации

В отличие от сценария, который является объектом, скрипт визуализации - это функция.

В скрипте можно использовать любые возможности JavaScript и Node.js, в том числе [Асинхронные функции](#)

Для вызова встроенных функций используется специальный объект **core**

Описание функций представлено в разделе [API скрипта визуализации](#)

Команда на запуск скрипта выполняется на клиенте и обычно привязывается к элементам визуализации.

Пример: Будем выдавать звуковое сообщение при клике на кнопке.

1. Создадим скрипт, назовем его "Дин-дон" ````js /**`

- Скрипт визуализации
- Выдает звук ding на клиент
- В окно отладчика выводит 'Now play ding'
- `/ module.exports = async function({ local, context, source }, core, debug) { core.playSound('ding.wav'); debug('Now play ding'); }; ````

2. Создадим кнопку **button\_1** на экране или в контейнере

3. Привяжем запуск скрипта к кнопке **button\_1**:

Вкладка Привязки элемента - Левая кнопка мыши - Одиночный клик - Привязать - Скрипт визуализации  
В дереве скриптов выберем наш "Дин-дон".

Рассмотрим скрипт более подробно.

## Входные аргументы

На входе скрипт получает 3 аргумента:

- объект **{local, context, source}** - данные от клиента
  - **local** - объект, содержащий локальные переменные и их значения
  - **context** - объект, содержащий данные о пользователе
    - **userid**: Идентификатор пользователя
    - **username**: Имя пользователя
    - **usergroups**: Группы, в которые входит пользователь
    - **layoutid**: ID текущего экрана
    - **start\_layoutid**: ID стартового экрана пользователя
  - **source** - объект, содержащий данные об элементе, запустившем скрипт
    - **uuid**: Идентификатор элемента ('button\_1')
    - **type**: Тип элемента ('button')
- **core** - объект, реализующий API
- **debug** - функция, позволяющая выводить отладочные сообщения в отладчике

Наш простейший скрипт пока не использует **local**, **context** и **source**

Мы могли бы, например, выдавать различные звуки в зависимости от нажатой кнопки, пользователя и/или экрана.

## Отладчик

В консоли **Редактора** можно отслеживать запуск и выполнение скрипта. Чтобы отладить скрипт, откроем в отдельной вкладке окно **Редактор** скрипта

В другой вкладке вызовем UI и нажмем кнопку.

При запуске скрипта в консоли Отладчика увидим такую информацию:

```
02.05 14:00:57.198 client: dd36NYdiXRyU8KbC2a1mZQ== Start
source={
  // Это ID элемента, который запустил скрипт
  id: 'button_1',
  value: { active: true },
  type: 'button',
  // Это ID экрана
  layoutid: 'l025'
}
// Здесь будут значения всех переменных текущего сеанса
local = { level: 1 }.
context = { username: 'Admin', start_layoutid: 'l025', layoutid: 'l025' }

02.05 14:00:57.200 client: dd36NYdiXRyU8KbC2a1mZQ== Now play ding
```

Поскольку скрипт может быть запущен с нескольких клиентов, и (при выполнении асинхронных операций) одновременно может работать несколько экземпляров, после времени выводится текущий ID клиента (динамически генерируемое значение). При успешном запуске скрипта выводится строка Start, затем входные данные (source, local, context)

Если в скрипте есть ошибка, в пользовательском интерфейсе будет выведено сообщение **Ошибка скрипта!** без подробностей.

В консоли **Редактора** этого скрипта будет выведена информация об ошибке.

```
01.06 20:29:17.596 client: HbMwqmHy4TepDAvx1Vueug== Ошибка скрипта!
/private/var/lib/ih-v5/projects/demo_2/visscripts/vs0004.js:2
  core.playSound('ding.wav');
      ^^^^^^^^^^^

SyntaxError: Invalid or unexpected token
```

Используется для вывода отладочной информации.

Функция имеет один аргумент. Это может быть строка или объект.

Если нужно вывести объект, сериализацию можно не выполнять.

**debug** выведет строковое представление объекта (допускается 6 уровней вложенности)

Пример:

```
module.exports = async function ({local, context, source}, core, debug) {
  const field1 = {x:1, y:42, z: 'test debug', q: true};
  const field2 = [{id:1, title:'Раз'}, {id:2, title:'Два'}];

  const myObj = { field1, field2};
  debug('Now myObj:')
  debug(myObj)
}
```

Результат:

```
01.05 18:24:57.446 client: LRLQFBm/5mq4HaStg7H28g== Now myObj:
01.05 18:24:57.447 client: LRLQFBm/5mq4HaStg7H28g== {
  field1: { x: 1, y: 42, z: 'test debug', q: true },
  field2: [ { id: 1, title: 'Раз' }, { id: 2, title: 'Два' } ]
}
```

## Выходное значение

Обычный скрипт визуализации может ничего не возвращать.

**return** с результатом можно использовать:

- если нужно сообщить, что скрипт закончил работу. В этом случае можно вернуть объект со строкой info. В нижней части экрана клиента отобразится всплывающее окно с сообщением.

```
....
return {info:'Все запланированное выполнено!'}
```

- чтобы показать ошибку пользователю, нужно вернуть объект со строкой err. В этом случае всплывающее окно будет красным

```
if (!local.carnumber) return {err:'Введите номер машины!'}
```

- если это не обычный скрипт, а [Скрипт заполнения](#) (Списка, Таблицы, Древа).  
Такие скрипты пишутся в Редакторе соответствующей сущности, и дефолтный скрипт показывает, какой результат ожидается.

## Обработка ошибок

Ошибки при работе скрипта могут возникать по разным причинам:

- синтаксическая ошибка в скрипте
- ошибка при вызове функции
- ошибка при выполнении (не определено значение, не найден файл, пользователь вводит данные, которые не ожидалось,... )

Скрипт визуализации (в отличие от сценариев и обработчиков) при ошибке не блокируется.

По умолчанию при любой ошибке в пользовательском интерфейсе будет выведено сообщение **Ошибка скрипта!** без подробностей.

В консоли **Редактора** этого скрипта будет выведена более подробная информация об ошибке.

Для ошибок времени выполнения можно перехватить ошибку и попытаться выдать пользователю более информативное сообщение.

Для этого используется конструкция try - catch.

```
module.exports = async function ({local, context, source}, core, debug) {
  // Обернем код в блок try
  try {
    // .....
    // Что-то делаем...

  } catch (e) {
    // Если в процессе произошла ошибка – будет выполнен этот код
    debug(e); // Для вывода в консоль скрипта, если нужна диагностика
    return {err: 'Не удалось выполнить эту операцию!'};
  }
}
```

При выполнении функций API объект ошибки **e** может содержать описание ошибки **e.message**. Иногда бывает полезно показать это сообщение пользователю



```
module.exports = async function ({local, context, source}, core, debug) {  
  // Оборнем код в блок try  
  try {  
    // .....  
    // Что-то делаем...  
  } catch (e) {  
    // Если в процессе произошла ошибка – будет выполнен этот код  
    const errStr = e && e.message ? e.message : 'Что-то пошло не так!';  
    return {err:errStr };  
  }  
}
```

## Асинхронные функции

Часто код выполняется не мгновенно, например, операции ввода/вывода, доступ к БД, сетевое взаимодействие, ...

JavaScript работает на однопоточном цикле событий, следующая команда выполняется только после того, как закончится предыдущая.

Если код синхронный, на время выполнения длительной операции основной поток выполнения будет заблокирован.

Платформа Node.js из коробки использует демultipлексирование событий для организации неблокирующего ввода/вывода. Тяжелые операции могут выполняться асинхронно и не блокировать основной поток.

Для организации асинхронной работы на уровне кода могут использоваться функции обратного вызова (callback), промисы и асинхронные функции (async).

Использование функции обратного вызова заключается в передаче callback-а, который вызывается при завершении операции.

Этот подход предполагает вызов цепочки функций и вложенную структуру кода. По такой модели у нас построены [сценарии автоматизации](#).

## Механизм async-await

Последние версии JavaScript имеют удобный встроенный механизм асинхронных функций. Хотя действия выполняются асинхронно, внешне код выглядит синхронным и более простым.

Асинхронная функция объявляется с ключевым словом **async** и возвращает так называемый **Promise** (обещание).

**Promise** - это объект-обёртка для асинхронного кода. Он содержит в себе состояние:

- вначале pending («ожидание»),
- затем — одно из: fulfilled («выполнено успешно») или rejected («выполнено с ошибкой»).

Ключевое слово **await** позволяет дождаться выполнения обещания.

*Более подробно см. документацию JavaScript*

## Использование асинхронных функций внутри скрипта

В простейшем случае для применения асинхронной функции внутри скрипта визуализации нужно помнить, что:

- чтобы дождаться результата выполнения асинхронной функции, нужно вызывать ее с ключевым словом **await**.
- **await** можно использовать только внутри функции, объявленной как **async**

Функция модуля со скриптом визуализации всегда объявлена как **async**, поэтому внутри можно использовать **await**

## Синхронные и асинхронные варианты функций Node.js

Функции библиотеки Node.js для операций ввода-вывода обычно имеют синхронный и асинхронный варианты, например:

- `fs.readFileSync()` - синхронный вариант. Код останавливается и ожидает, пока файл не считывается
- `fs.promises.readFile()` - асинхронный вариант.

По возможности всегда нужно выбирать асинхронные варианты, особенно для операций ввода-вывода.

Следует помнить, что ваш скрипт запускается в общем потоке выполнения системы

## Обработка ошибки асинхронной функции

Если асинхронная функция завершилась с ошибкой (rejected), то генерируется исключение.

Рекомендуется оборачивать асинхронные вызовы в блоки `try - catch`. В этом случае можно дать пользователю информацию об ошибке в контексте выполняемых действий

```
const filename = '.....';
try {
  // Асинхронная операция чтения файла
  const data = await fs.promises.readFile(filename, 'utf8');
  // выполнено успешно – обрабатываем считанный файл
  // ....
} catch (e) {
  // выполнена с ошибкой (rejected)
  debug(e);
  return {err: 'Не удалось считать файл '+filename}
}
```

Если `try-catch` опущены, то ошибка будет перехвачена системой, и сообщение пользователю будет обычно выглядеть как 'Ошибка скрипта'.

## Асинхронные функции API

Среди функций API скрипта визуализации довольно много асинхронных функций. В их описании всегда есть **`await`**.

Например, получение снимка с камеры:

- **`await core.snap(<ID камеры - строка>, <Таймаут в сек = 2>)`**

```
const camera = 'cam_5';
// snapRes = {filename:'/var/lib/..../snap_123456789000.jpg'}
const snapRes = await core.snap(cam);
// Здесь операция уже выполнена, имеем имя файла,
// дальше отправляем файл в телеграм, по почте ...
```

## Вызов асинхронной функции без await

Рассмотрим скрипт, в котором асинхронная функция вызвана без await.

```
const snapRes = core.snap('cam_5');
debug(snapRes.filename);
```

В этом случае snapRes будет содержать тот самый **Promise**. Здесь **await** был опущен случайно, и мы столкнулись с ошибкой - snapRes.filename будет undefined

Однако, это можно использовать, если понимать, как дальше работать с объектом **Promise**. Например, запустить асинхронную операцию, а дождаться результат позже:

```
const snap1 = core.snap('cam_1');
// .....
const snapRes1 = await snap1;
debug(snapRes1.filename);
```

Или даже запустить несколько асинхронных операций параллельно.

Например, чтение снимков с нескольких камер:

```
const snap1 = core.snap('cam_1');
const snap2 = core.snap('cam_2');
const snap3 = core.snap('cam_3');
const [snapRes1, snapRes2, snapRes3] = await Promise.all([
  snap1,
  snap2,
  snap3
]);
```

Операция завершится, когда без ошибок завершатся все переданные промисы, и результатом будет массив результатов. Порядок элементов массива результата в точности соответствует порядку исходных промисов. Даже если первый промис будет выполняться дольше всех, его результат всё равно будет первым в массиве.

Кроме Promise.all, есть Promise.allSettled и Promise.race - все это штатные возможности, описанные в любой документации по JavaScript.

## Можно ли вызвать асинхронную функцию совсем без await

В этом случае момент завершения операции ("разрешения" промиса) будет для скрипта потерян.

Простейшая асинхронная функция - это временная задержка (фактически это обернутый в промис таймер)

- **await core.sleep(<Время в мс - число>)**

```
module.exports = async function ({local, context, source}, core, debug) {
  await core.sleep(3000);
  // Дальнейшие действия будут выполнены через 3 сек
  return {info: 'Прошло 3 секунды'}
```

Если await не использовать, задержки не будет.

```
module.exports = async function ({local, context, source}, core, debug) {
  core.sleep(3000);
  // Выполнение продолжается без задержки
  return {info: 'Прошло 3 секунды?'}
```

Без await таймер запустился, отработал, но скрипт об этом никогда не узнает, так как уже завершился. В данном случае команда **core.sleep** без await была бесполезна, но при этом безвредна, так как она никогда не завершается с ошибкой.

Рассмотрим асинхронную функцию для отправки команды плагину:

- **await core.pluginCommand(<Объект-команда>, <Таймаут в сек = 2>)**

Функция завершается с ошибкой, если плагин не работает или команда не поддерживается.

Попробуем вызвать эту функцию без **await**.

```
module.exports = async function ({local, context, source}, core, debug)
  try {
    core.pluginCommand({unit:'myplug1', command:'test', arg:42});
  } catch (e) {
    // Без await это не сработает
    return {err:'Ошибка отправки команды плагину'} // Без await это не с
  }
}
```

Команда будет отправлена на плагин и скрипт будет сразу завершен.  
Если возникнет ошибка, исключение будет сгенерировано, но перехвачено на другом уровне как `unhandledRejection`.

Таким образом, функцию, потенциально способную завершиться с ошибкой, запускать таким образом не рекомендуется.

## Примеры

Пример 1. Получение списка снимков с камер

Используем асинхронную функцию Node.js `fs.promises.readdir()` для получения списка файлов в папке.

```
const fs = require('fs');
module.exports = async function ({local, context, source}, core, debug)
  let folder;
  let files;
  try {
    // Функция API для получения пути к папке со снимками
    folder = core.getSnapFolder();
    files = await fs.promises.readdir(folder);
  } catch (e) {
    debug(e);
    return {err:'Произошла ошибка при получении снимков из папки '+folder}
  }

  // Получили список файлов files, дальше их обрабатываем .
}
```

## Пример 2. Чтение файла с диска

В примере показано, как можно написать свою асинхронную функцию внутри основной функции скрипта:

```
const fs = require('fs');
module.exports = async function ({local, context, source}, core, debug) {
  const myfile = '....'; // путь к файлу, который содержит json
  const myObj = await getDataObj(myfile);

  async function getDataObj(filename) {
    try {
      const data = await fs.promises.readFile(filename, 'utf8');
      return data ? JSON.parse(data) : {};
    } catch (e) {
      debug(e);
      return {};
    }
  }
}
```

## Переход на экран

Отправляет команду текущему клиенту "Переход на экран".

- **core.gotoLayout**(<ID экрана>)  
Возвращаемое значение - нет.

```
core.gotoLayout('l002');  
// Скрипт может продолжиться  
const vent1 = core.getDevice('VENT1');
```

## Обновление значений переменных клиента

Отправляет текущему клиенту значения переменных

- **core.updateLocals**({<имя>:<значение>, ...})  
Возвращаемое значение - нет.

```
core.updateLocals({ dev_id:'d042', level:3, list_id:'vlst003' });
```

Обратите внимание, эта команда также используется, чтобы визуальный компонент (список, таблица) обновили свое содержимое.



## Завершение сеанса

*Добавлено v5.18.0*

Завершает сеанс (выход из системы) пользователя с указанным идентификатором

- **core.logout**(<ID пользователя>)  
Возвращаемое значение - нет.

Пример. Выход из системы пользователя, вызвавшего скрипт. Выполняется логирование в главный журнал.

```
module.exports = async function({ local, context, source }, core, debug) {  
  core.logout(context.userid);  
  core.mainlog('Выполнен выход из системы. Пользователь '+context.username);  
};
```

## Доступ к устройствам, типам и глобальными переменными

### Доступ к устройству

- **core.getDevice**(<ID устройства | DN устройства>)

Можно передать *ID устройства* ('d0042') или *Имя устройства* ('VENT1')

Возвращаемое значение - объект устройства.

Далее с устройством можно работать, используя свойства и методы этого устройства.

Если устройство не существует, возвращается undefined

```
const vent1 = core.getDevice('VENT1');
if (vent1 && !vent1.state && vent1.auto) {
  vent1.on();
}
```

В отличие от сценария, где **Отправителем** команды выступает сценарий, здесь **Отправителем** считается пользователь, вызвавший скрипт.

Если действия с устройством фиксируются в журнале, при выполнении команды из скрипта в журнале будет такая же запись, как при обычной интерактивной команде: *'Вентилятор 1', 'Команда: Включить', 'Оператор: Петров'*

Здесь объект устройства - это так называемый 'обернутый' объект.

При попытке сериализации будет возвращаться пустой объект, это не должно вводить в заблуждение.

Если устройство существует, все его свойства и методы доступны.

```
const dev = core.getDevice('METER_1');
if (!dev) return {err: 'Устройство METER_1 не найдено!'};

// => ID d0065 Счетчик METER_1, показание: 24367.34
debug('ID='+dev.id+' Счетчик '+dev.dn+', показание: '+ dev.value);

// НО
debug('Счетчик '+ JSON.stringify(dev)); // => Счетчик {}

const util = require('util');
debug('Счетчик '+ util.inspect(dev)); // => Счетчик {}
```

После получения устройства есть возможность обратиться к внутренним атрибутам свойств таким как #min, #max, #string и т.д. с помощью функции dev.getPropValue("value#min")

## Доступ к группе устройств

- **core.getDevices("Vent\*");** // фильтр по dn устройства с использованием wildcard char (\*)
- **core.getDevices(/^Vent/);** // фильтр по dn устройства с использованием регулярного выражения
- **core.getDevices({tag:'климат'});** // фильтр по статическому свойству
- **core.getDevices({place:'dg062'});** // фильтр по узлу дерева
- **core.getDevices({tag:'климат', place:'dg062'});** // фильтры можно скомбинировать

```
const devs = core.getDevices({place: 'dg005'});
if (!devs) return {err: 'Устройства не найдены!'};

devs.forEach(item => {
  debug('ID='+item.id+ ' Устройство '+item.dn+', название: '+ item.name);
})
```

## Список типов устройств

Добавлено: v5.17.58

- **core.getTypes();**
- **core.getTypes({tag:'климат'});** // фильтр по метке

Возвращает массив типов устройств, элемент массива:

{id:<идентификатор типа>, title:<название типа>, tags:<массив меток>}

```
const releTypes = core.getTypes({tag: 'Реле'});
if (!releTypes) return {err: 'Типы с меткой Реле не найдены!'};

releTypes.forEach(item => {
  debug(item.id + ' ' + item.title);
  if (item.tags && item.tags.length) debug('Метки: ' + item.tags);
});
```

## Доступ к глобальными переменными

- **core.globals.<Название переменной>**

```
const global_var = core.globals.var1
```

Из скрипта визуализации для глобальных переменных доступно только чтение

## Запись в главный журнал системы

- `core.mainlog(<Сообщение>, <Уровень>)`
- `core.mainlog(<Сообщение>, {level:<Уровень>, tags:<Тэг>, location:<Расположение>, sender:<Оператор>})`
- `core.mainlog({txt:<Сообщение>, level:<Уровень>, tags:<Тэг>, location:<Расположение>, sender:<Оператор>})`

```
core.mainlog('Нажата кнопка СТОП', 2);
core.mainlog('Нажата кнопка СТОП', {level:2, tags:"ГРЩ"});
core.mainlog({txt:'Нажата кнопка СТОП', level:2, tags:"ГРЩ"});
```

При записи в главный журнал **Отправителем** автоматически считается пользователь, вызвавший скрипт.

### Пример.

Допустим, необходимо фиксировать нажатия кнопок. Можно создать единый скрипт **Запись кликов в журнал**.

При запуске скрипта ему передаются [Входные аргументы context и source](#), которые содержат нужную информацию.

```
module.exports = async function({ local, context, source }, core, debug) {

  if (source.uuid == 'stopButton') {
    core.mainlog(
      'Нажата кнопка СТОП на экране ' + context.layoutid +
      ' Оператор: ' + context.username,
      2
    );
  } else {
    // ...
    core.mainlog(
      'Нажата кнопка ' + source.uuid +
      ' на экране ' + context.layoutid
    );
  }

};
```

Далее к основному действию кнопки (например, по Одиночному клику) можно добавить еще один Одиночный клик, привязать к нему запуск этого **Скрипта визуализации**, и информация о нажатиях будет фиксироваться в журнале

## Вызов диалога

Вызов диалога на том терминале, с которого запустили скрипт.

- `core.showDialog(<ID диалога>);`

```
core.showDialog('di002');
```

## Заккрытие диалога

Заккрытие диалога на том терминале, с которого запустили скрипт.

- `core.closeDialog(<ID диалога>);`

```
core.closeDialog('di002');
```

## Звуковое оповещение

### playSound

Воспроизвести звук **на том терминале, с которого запустили скрипт**

- `core.playSound(<Список звуковых файлов через запятую>);`

```
core.playSound('tada.wav, внимание.wav, прекращеноЭлектроснабжение.wav');
```

### repeatSound

*Добавлено в v5.17.17*

Воспроизводить звук циклически **на том терминале, с которого запустили скрипт**

- `core.repeatSound(<Список звуковых файлов через запятую>);`

```
core.repeatSound('tada.wav, внимание.wav, прекращеноЭлектроснабжение.wav');
```

### stopSound

*Добавлено в v5.17.17*

Остановить воспроизведение звука **на том терминале, с которого запустили скрипт**

- `core.stopSound();`

```
core.stopSound();
```

## Информирование

Отправка сообщения для пользователя или группы:

- **core.info**(<ID плагина>, <ID пользователя>, <Текст сообщения>);
- **core.info**(<ID плагина>, <ID группы>, <Текст сообщения>);

Аналогично сценарию, можно отправить сообщение через плагины информирования (**telegram**, **email**).

Операция будет выполнена, если соответствующий плагин установлен.

```
core.info('telegram', 'admgrp', 'Сообщение группе администраторов...');
```

Также можно отправить сообщение **sound** (проиграть звук на активных терминалах, запущенных для конкретного пользователя или группы )

- **core.info**('sound', <ID пользователя>, <Список звуковых файлов через запятую>);

```
core.info('sound', 'grp003', 'ding.wav, внимание.wav, авария.wav');
```



## Действия со сценариями

### Запустить сценарий

Из скрипта можно запустить сценарий автоматизации. Если сценарий уже запущен и находится в активном состоянии - команда будет проигнорирована.

- **core.startScene**(<ID сценария>, <Параметры, опционально - число|строка|объект>);

```
core.startScene('scen001', 'myparam');
core.startScene('scen007', {'myparam1':1, 'myparam2':2});
```

Параметры будут переданы функции **start** сценария.

Например, будем передавать число звончков для звукового воспроизведения.

```
core.startScene('scen003', 4);
```

В самом сценарии функция **start** должна принять и обработать входной аргумент.

```
/**
 * @desc scen003
 * @version 5
 */
const script = {
  start(param) {
    if (typeof param == 'number' && param > 0 && param < 10) {
      const arr = Array(param).fill( 'ding.wav');
      this.info('sound', 'u0003', arr.join(','));
    }
  }
}
```

### Остановить сценарий

Из скрипта можно остановить сценарий. Если сценарий не запущен - команда будет проигнорирована.

```
core.stopScene('scen003');
```

## Действия с шаговыми сценариями

Методы startScene и stopScene работают и для шаговых сценариев. Кроме того, из скрипта можно приостановить и возобновить выполнение шагового сценария.

### Запустить шаговый сценарий

```
core.startScene('trscen003');
```

### Остановить шаговый сценарий

```
core.stopScene('trscen003');
```

### Приостановить шаговый сценарий

```
core.suspendScene('trscen003');
```

### Возобновить шаговый сценарий

```
core.resumeScene('trscen003');
```

## Временная задержка

Временная задержка - это [асинхронная функция](#), всегда вызывается с ключевым словом **await**.

- **await core.sleep(<Время в мс - число>)**
- **await core.sleepSec(<Время в сек - число>)**

Возвращаемое значение - нет.

```
await core.sleep(2000);  
// Дальнейшие действия будут выполнены через 2 сек
```

Этот скрипт выведет два звуковых файла с интервалом 2 сек

```
core.playSound('ding.wav');  
await core.sleep(2000);  
core.playSound('tada.wav');
```

Если await не использовать, задержки не будет, воспроизведение наложится друг на друга

```
core.playSound('ding.wav');  
core.sleep(2000);  
core.playSound('tada.wav');
```

## Отправка команды плагину

Можно отправить прямую команду плагину из скрипта.

Штатно команды плагину отправляются через механизм каналов.

Не все плагины принимают прямые команды. Формат команды зависит от плагина.

- **await core.pluginCommand**(<Объект-команда>, <Таймаут в сек = 2>);

Объект-команда должен содержать:

- **unit** - ID экземпляра плагина
- **command** - команда плагина - строка или объект.

Также в объект можно включить дополнительные параметры

Опционально можно задать таймаут - время ожидания ответа от плагина, по умолчанию 2 сек.

```
const result = await core.pluginCommand({
  unit: 'myplug1',
  command: 'test',
  arg: 42
});
debug(result);
```

Если плагин не запущен или не отвечает, то скрипт выйдет по таймауту.

Клиент получит сообщение 'Ошибка скрипта', а саму ошибку можно увидеть в консоли скрипта.

Можно перехватить ошибку в скрипте и обработать как нужно. Здесь таймаут установлен 5 сек

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const result = await core.pluginCommand(
      {
        unit: 'myplug1',
        command: 'test',
        arg: 42
      },
      5
    );

    debug(result);

  } catch (e) {
    return {
      err: e.message || 'Ошибка при выполнении команды плагина!'
    };
  }
};
```

Для отправки команды плагину иногда бывает полезно получить список каналов для устройства.

Команда `core.getDeviceChannels` возвращает массив привязанных каналов.

## Снимок с видеокамеры

Получить снимок с видеокамеры можно, если

- установлен плагин [CCTV](#)
- Для камеры введен параметр **Snapshot Url** для получения снимка.

- **await core.snap**(<ID камеры - строка>, <Таймаут в сек = 2>);

Опционально можно задать таймаут - время ожидания снимка, по умолчанию 2 сек.

Результат придет после выполнения снимка, но не позже, чем через <Таймаут сек>.

Возвращаемое значение - объект, содержащий полное имя файла.

Снимки сохраняются в папке проекта. Можно, например, отправить снимок в Telegram.

```
const camera = 'cam_5';

const snapRes = await core.snap(cam);
// => { filename: '/var/lib/.../snap_123456789000.jpg' }

core.info(
  "telegram",
  "admin",
  {
    img: snapRes.filename,
    txt: "Камера " + camera
  }
);
```

Обратите внимание, что в отличие от сценария, здесь можно не анализировать результат.

Если снимок по какой-то причине не получен (не доступен плагин cctv, не отвечает камера, ..), эту ситуацию обрабатывает **core**. Выполнение скрипта будет прервано.

Клиенту будет отправлен ответ: "Снимок не получен".

## Генерация отчёта

Добавлено: v5.16.21

Скрипт может запросить генерацию отчета.

- **await core.getReport**({<объект, содержащий параметры отчета>})
  - **id** - ID отчета
  - **start** - начало периода (timestamp), обязательный параметр
  - **end** - конец периода (timestamp), если не указан, будет равен Date.now()
  - **content** - 'pdf' || 'csv'
  - **local** - опционально - объект, содержащий локальные переменные. Нужно добавить, если обработчик для подготовки отчета использует локальные переменные

Плагин генерации отчета подготовит отчет, запишет его в файл и вернет имя файла в результате. Вы можете отправить этот файл в телеграм или по e-mail

```

module.exports = async function({ local, context, source }, core, debug) {

  const start = Date.now() - 3600000;

  const result = await core.getReport({
    id: 'r011',
    start,
    content: 'pdf',
    local
  });

  if (result && result.filename) {

    core.info(
      "telegram",
      "admin",
      {
        txt: 'Отчет за последний час',
        pdf: result.filename
      }
    );

    core.info(
      "email",
      "u0003",
      {
        txt: 'Отчет за последний час',
        img: result.filename
      }
    );

    return {
      info: 'Отчет за последний час подготовлен и отправлен'
    };

    return {
      err: 'При подготовке отчета произошла ошибка!'
    };
  };
};

```

## Определение папки для отчета



По умолчанию отчет сохраняется в папку проекта **temp**.

Можно сохранить отчет в другую папку, указав параметр **folder**.

```
const folder = core.getProjectPath('anyfiles');  
// Папка anyfiles внутри проекта  
  
const result = await core.getReport({  
  id: 'r011',  
  start,  
  folder,  
  content: 'pdf',  
  local  
});
```

## Квитирование тревог

Из скрипта можно выполнить операцию **Квитировать все** для конкретного журнала.

- **await core.ackAllAlerts**(*<Идентификатор журнала тревог - строка>*)

```
/**
 * Квитировать все алерты, ожидающие квитирования, в журнале тревог
 * Скрипт вызывается с кнопки
 * id журнала передается в локальной переменной ajr_id
 */
module.exports = async function ({local, context, source}, core, debug) {
  try {
    await core.ackAllAlerts(local.ajr_id);
  } catch (e) {
    debug(e);
    return {err: 'Не удалось квитировать все!'}
  }
}
```

## Получение доступа к журналу тревог

Добавлено v5.18.13

Из скрипта можно получить текущие записи журнала тревог.

- **await core.getAlerts()** - все записи (журнал без статических фильтров)
- **await core.getAlerts**(*<Идентификатор журнала - строка>*) - записи конкретного журнала в учетом статических фильтров
- **await core.getAlerts**(*<Фильтр - объект>*) - отфильтрованные записи журнала

Пример 1.

```
/**
 * Получить все записи журнала тревог с идентификатором ajr003
 * Учитываются статические фильтры
 */
module.exports = async function ({local, context, source}, core, debug) {
  const recs = await core.getAlerts('ajr003');
  return {info: 'Количество записей в журнале ajr003: ' + recs.length}
}
```

Пример 2.

```
/**
 * Получить все записи журнала тревог по заданному устройству, требующие к
 *
 */
module.exports = async function({ local, context, source }, core, debug)

const recs = await core.getAlerts({
  did: 'd0204',
  notacked: { $gt: 0 }
});

let x = recs.reduce(
  (accumulator, current) => accumulator + current,
  0
);

return {
  info:
    'По устройству ожидает квитирования ' +
    recs.length +
    ' записей, всего неквитированных событий: ' +
    x
};
};
```

## Получение доступа к данным строки в журнале тревог

*Добавлено v5.17.85*

Из скрипта можно получить доступ к данным выбранной строки в журнале тревог (например, для перехода на экран/мнемосхему или для вызова диалога).

- **await core.getAlertObj(<Идентификатор строки>)**

Возвращаемое значение - объект строки журнала тревог:

```
{
  "did": "d0141",
  "prop": "state",
  "level": 2,
  "aruleId": "Alert",
  "propTitle": "Состояние ",
  "devTitle": "Датчик CO (E_DP_001) ",
  "location": "/place/dg047/dg048",
  "tags": "#Вентиляция#",
  "txt": "Повышен уровень CO! Датчик CO (E_DP_001) ",
  "tsStart": 1679386867706,
  "tsAck": 1679386883062
}
```

Если строка не существует, возвращается undefined

```
/**
 * Скрипт вызывается при выборе строки в журнале тревог (Скрипт при выборе
 * id строки передается в локальной переменной aline_id (Переменная резул
 */
module.exports = async function ({local, context, source}, core, debug) {

  const aline_id = local.aline_id;
  if (!aline_id) return {info: 'Строка не выбрана'};

  const aleObj = await core.getAlertObj(aline_id);
  if (!aleObj) return {info: 'Строка не найдена'};

  if (aleObj.did === 'd002') {
    core.gotoLayout('l002');
  } else {
    return {info: aleObj.txt};
  }
}
```

## Функции работы с пользовательскими файлами

В скриптах визуализации доступны операции с пользовательскими файлами:

- **await core.getUserfiles**(*<фильтр - объект>*, *<опции - объект>*)  
получить массив списка файлов с учетом фильтра
- **await core.copyUserfile**(*<имя файла - строка>*, *<новое имя файла - строка>*)  
создать копию файла
- **await core.renameUserfile**(*<имя файла - строка>*, *<новое имя файла - строка>*)  
переименовать файл
- **await core.removeUserfiles**(*<фильтр - объект>*)  
удалить пользовательские файлы с учетом фильтра
- **existsUserfile**(*<имя файла - строка>*) проверить, что пользовательский файл существует

### getUserfiles

**await core.getUserfiles(filter, opt)**

Возвращает массив, каждый элемент содержит объект для одного файла:

[{file:<имя файла>, ts:<время записи файла - timestamp>, parent:<id папки - родителя>, id:<совпадает с file>}, ...]

Идентификатором является имя файла включая расширение.

Если вызвать функцию без параметров, будет возвращен весь список пользовательских файлов проекта, упорядоченный по возрастанию ts:

```
const files = await core.getUserfiles();
debug(files); // -> [{file:'test1.pdf', ts:1665222571808, id:'test.pdf'},
```

Для фильтрации списка используется первый аргумент **filter**

Объект **filter** может иметь следующие свойства:

- start - время записи файла - timestamp (больше или равно)
- end - время записи файла - timestamp (меньше или равно)
- file - имя файла
- parent - id папки - родителя

Для сортировки списка используется второй аргумент **opt**

Пример: получить список пользовательских файлов из папки с ID=flgr003 за последний час, упорядоченный по убыванию ts:

```
const files = await core.getUserfiles(  
  {  
    start: Date.now() - 3600000,  
    parent: 'flgr003'  
  },  
  {  
    sort: { ts: -1 }  
  }  
);
```

## removeUserfiles

**await core.removeUserfiles(filter)**

Удаляет пользовательские файлы в соответствии с фильтром.

В отличие от getUserfiles, filter пустым быть не может.

Команда без указания фильтра будет проигнорирована.

(Если нужно удалить все пользовательские файлы, используйте функционал раздела **Удаление рабочих данных проекта**)

Пример: удалить все пользовательские файлы, кроме тех, что были загружены за последний час

```
await core.removeUserfiles({end:Date.now() - 3600000});
```

## copyUserfile, renameUserfile

**await core.copyUserfile(file, newfile)**

Создает копию файла. Если файл newfile уже есть - будет ошибка скрипта.

Можно до выполнения операции проверить, что файл уже есть и сообщить пользователю.

### Пример скрипта для копирования файла

```

/**
 * Скрипт вызывается по кнопке Копировать
 *
 * На экране размещается:
 *  - таблица со списком файлов
 *  - Переменная результата привязана к Переменной fileTableRow
 *  - элемент Input, привязанный к Переменной fileNameInput
 */
module.exports = async function({ local, context, source }, core, debug) {
  const file = local.fileTableRow;
  if (!file) return {err: 'Файл не выбран!'}

  const newfile = local.fileNameInput;

  if (!newfile) return {err: 'Введите имя файла!'}
  if (core.existsUserfile(newfile)) {
    return {
      err: newfile + ' – ошибка, такой файл уже существует!'
    };
  }

  await core.copyUserfile(file, newfile)
};

```

**await core.renameUserfile(file, newfile)**

Переименовывает файл. Работает аналогично copyUserfile.

## Функции работы с клиентскими данными

Добавлено v5.18.7

Иногда возникает необходимость сохранять данные каждого клиента (пользователя) отдельно.

Например, это могут быть настройки, специфичные для пользователя.

Из скрипта визуализации можно записать и считать такие данные.

Кроме **ID пользователя** требуется идентификатор, определяющий название контента. Это должна быть строка, соответствующая правилам построения идентификаторов (допустимы только латинские буквы, цифры и знак подчеркивания).

Сами данные представляют собой произвольный объект, который можно сериализовать.

- **await core.saveClientData**(<ID пользователя>, <название - строка>, <data - объект> )  
Сохранить данные пользователя
- **const data = await core.getClientData**(<ID пользователя>, <название - строка>)  
Считать данные пользователя

### saveClientData

Сохраняет данные пользователя, вызвавшего скрипт, тема - 'test'.

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const data = {mydata:{txt:'раз два', val:42}, arr:[1,2,3]};
    await core.saveClientData(context.userid, 'test', data);
  } catch (e) {
    debug(e);
    return {err:'Не удалось сохранить данные пользователя!'}
  }
}
```

### getClientData

Возвращает ранее сохраненные данные пользователя.



```
module.exports = async function({ local, context, source }, core, debug) {  
  try {  
    const data = await core.getClientData(context.userid, 'test');  
    const str = data  
    ? 'Сохраненные данные: ' + JSON.stringify(data)  
    : 'Нет сохраненных данных';  
    return {info: str}  
  } catch (e) {  
    debug(e);  
    return {err: 'Не удалось считать данные пользователя!'}  
  }  
}
```

## Функции работы с деревьями проекта

Скрипт визуализации может получить данные практически любого дерева (поддерева) проекта. Обычно эти функции используются в [Скриптах заполнения](#) для деревьев, списков и таблиц пользовательского интерфейса.

Нужно задать *Имя дерева* и, опционально, *ID узла*, с которого нужно начать. Если *ID узла* не задано, возвращается все дерево, начиная с корня.

Данные можно получить в виде древовидной структуры либо в виде плоского списка:

- **await core.getTree(<Имя дерева - строка>[, <ID узла - строка>])**  
возвращает массив, представляющий древовидную структуру

Описание структуры см [Скрипты заполнения](#) -> Скрипт заполнения дерева или далее в Примере.

- **await core.getListFromTree(<Имя дерева - строка>[, <ID узла - строка>])**  
возвращает плоский массив всех **листьев** дерева, начиная с заданного узла в формате: [{id, title}, ...].

Имена часто используемых деревьев даны ниже:

devdevices, devices	Устройства
users	Пользователи
agroups	Группы пользователей
layout	Экраны
viscont	Контейнеры
dialog	Диалоги
scenes	Сценарии
charts	Графики
dbtargets	Устройства со свойствами, сохраняемыми в БД
customtables	Пользовательские таблицы

Полный список деревьев см Приложение -> Список деревьев системы

Например, получим все устройства, находящиеся в папке 'dg061' (это идентификатор **Place ID** папки), и разместим эти данные в дереве и в списке.

## Пример заполнения компонента Дерево из дерева устройств

```
/**
 * Скрипт для заполнения дерева
 * Должен вернуть объект, содержащий
 *   data: массив узлов дерева
 */
module.exports = async function ({local, context, source}, core, debug) {
  const data = await core.getTree('devices', 'dg061');
  debug(data);
  return {data};
}
```

В отладчике можно увидеть содержимое массива data:

```
{
  title: 'Серверная 104',
  id: 'dg061',
  children: [
    {
      title: 'Ряд A',
      id: 'dg089',
      children: [
        {
          id: 'd0270',
          title: 'A-01 ■ Шкаф A-01'
        },
        {
          id: 'd0361',
          title: 'A-02 ■ Шкаф A-02'
        },
        {
          id: 'd0362',
          title: 'A-03 ■ Шкаф A-03'
        }
      ]
    }
  ]
}
```

## Пример заполнения компонента Список из дерева устройств

```
/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context, source}, core, debug) {
  const data = await core.getListFromTree('devices', 'dg061');
  debug(data);
  return {data};
}
```

В отладчике можно увидеть содержимое массива data:

```
{ id: 'd0270', title: 'A-01 ■ Шкаф A-01' },
{ id: 'd0361', title: 'A-02 ■ Шкаф A-02' },
{ id: 'd0362', title: 'A-03 ■ Шкаф A-03' },
{ id: 'd0363', title: 'A-04 ■ Шкаф A-43' }
]
```

## Функции работы с пользовательскими таблицами

Полное описание работы с пользовательскими таблицами: [API для пользовательских таблиц](#)

В скриптах визуализации доступны функции чтения и записи пользовательских таблиц:

- **await core.getRecords**(*<Имя таблицы - строка>*, *<фильтр - объект>*, *<Опции - объект>*)  
получить массив записей таблицы с учетом фильтра и сортировки
- **await core.findOneRecord**(*<Имя таблицы - строка>*, *<фильтр - объект>*, *<Опции - объект>*)  
получить одну (первую) запись таблицы с учетом фильтра и сортировки
- **await core.insertRecords**(*<Имя таблицы - строка>*, *<массив записей>*)  
добавить записи в таблицу
- **await core.updateRecords**(*<Имя таблицы - строка>*, *<фильтр - объект>*, *<Изменяемые поля>*)  
изменить записи таблицы (множественное изменение)
- **await core.updateRecord**(*<Имя таблицы - строка>*, *<фильтр - объект>*, *<Изменяемые поля>*)  
изменить одну запись таблицы
- **await core.removeRecords**(*<Имя таблицы - строка>*, *<фильтр - объект>*, *<Опции - объект>*)  
удалить записи из таблицы

Добавлено: v5.17.55

- **await core.getSQLRecords**(*<SQL запрос - строка>*)  
получить массив записей таблицы по прямому SQL запросу, только для таблиц в СУБД

## Чтение исторических данных

Можно читать исторические данные устройств, используя функцию **core.getRecords**, которая используется для чтения пользовательских таблиц.

Исторические данные хранятся в таблице 'records'.

Даже для небольших проектов это обычно миллионы записей.

Для такого массива информации **строго не рекомендуется** использовать запросы типа *SELECT \* FROM records*, а потом разбираться.

Необходимо использовать фильтры, API предоставляет для этого 2 варианта:

- Параметрическая фильтрация - передаются параметры, SQL запрос система подготовит сама
- Сформированный SQL запрос

### Параметрическая фильтрация

Рекомендуется использовать этот вариант, если не нужна специфическая обработка. SQL запрос будет подготовлен с учетом подключенной СУБД.

В первом аргументе передается имя таблицы - 'records', второй аргумент содержит фильтр.

- **await core.getRecords('records', {start, end, dn\_prop })**
  - start - timestamp - начало периода
  - end - timestamp - конец периода
  - dn\_prop - строка, содержащая через запятую метрики, которые нужно получить.

Метрика состоит из имени устройства и свойства через точку: 'Meter1.P1'

Можно не указывать свойство, тогда будут считаны значения всех записываемых свойств для устройства

Например,

dn\_prop:'Meter1.P1' - значения P1 для счетчика Meter1

dn\_prop:'Meter1' - значения всех записываемых свойств для счетчика Meter1

dn\_prop:'Meter1.P1,Meter2.P1,Meter1.P2,Meter2.P2' - значения P1, P2 для счетчиков Meter1, Meter2

Возвращается массив элементов {dn, prop, ts, val}, упорядоченный по времени (ts).

Пример 1. Получить массив значений температуры за период:

```
const recs = await core.getRecords(  
  'records',  
  {  
    start,  
    end,  
    dn_prop: 'TEMP1.value'  
  }  
);
```

Пример 2. Построить таблицу, содержащую расчетные значения по показаниям нескольких датчиков за последний час.

(для простоты будем считать min и max значения)

```
/**
 * Скрипт для заполнения таблицы
 * Должен вернуть объект, содержащий
 *   columns: массив объектов с описанием столбцов {title, prop, width, fi
 *   data: массив объектов, каждый элемент – это строка таблицы
 */
module.exports = async function({ local, context, source }, core, debug) {
  const columns = [
    { title: 'Устройство', prop: 'dn', width: 400 },
    { title: 'Min за час', prop: 'min', width: 200 },
    { title: 'Max за час', prop: 'max', width: 200 }
  ];

  const end = Date.now();
  const start = end - 3600 * 1000;
  const dn_prop = 'TEMP1.value,TEMP2.value,TEMP3.value';
  const recs = await core.getRecords('records', { start, end, dn_prop });

  const data = [];
  if (recs.length) {
    const tempObj = {};
    recs.forEach(rec => {
      if (!tempObj[rec.dn]) {
        tempObj[rec.dn] = { dn: rec.dn, min: rec.val, max: rec.val };
      } else {
        if (rec.val < tempObj[rec.dn].min) tempObj[rec.dn].min = rec.val;
        if (rec.val > tempObj[rec.dn].max) tempObj[rec.dn].max = rec.val;
      }
    });

    Object.keys(tempObj).forEach(dn => {
      data.push(tempObj[dn]);
    });
  }
  return { columns, data };
};
```

Пример 3. Построить таблицу, содержащую расчетные значения по нескольким свойствам устройства. Устройство будет выбрано динамически и придет как `local.dev_id`. Выведем все свойства, которые записываются в БД.



```
/**
 * Скрипт для заполнения таблицы
 * Должен вернуть объект, содержащий
 *   columns: массив объектов с описанием столбцов {title, prop, width, filter},
 *   data: массив объектов, каждый элемент – это строка таблицы
 */
module.exports = async function ({local, context, source}, core, debug) {
  const columns = [
    { title: 'Устройство', prop: 'name', width: 200},
    { title: 'Свойство', prop: 'devprop', width: 200},
    { title: 'Min за период', prop: 'min', width: 200},
    { title: 'Max за период', prop: 'max', width: 200}
  ];

  const did = local.dev_id; // Выбрал пользователь в списке или дереве ...
  const dobj = core.getDevice(did);
  const name = dobj.name;
  // Свойства не указываем – будут считаны все свойства
  const dn_prop = dobj.dn;
  const end = Date.now();
  const start = end-1000000;
  const recs = await core.getRecords('records', {start, end, dn_prop });

  const data = [];
  if (recs.length) {
    const tempObj = {};
    recs.forEach(rec => {
      if (!tempObj[rec.prop]) {
        tempObj[rec.prop] = { name, devprop: rec.prop, min: rec.val, max: rec.val };
      } else {
        if (rec.val < tempObj[rec.prop].min) tempObj[rec.prop].min = rec.val;
        if (rec.val > tempObj[rec.prop].max) tempObj[rec.prop].max = rec.val;
      }
    });

    Object.keys(tempObj).forEach(prop => {
      data.push(tempObj[prop]);
    });
  }
  return {columns, data};
}
```

Другой вариант - можно передать функции уже сформированный sql запрос.

При формировании запроса следует учесть, что структура записей для разных СУБД различна:

#### Схема для SQLite:

- dn - String - имя устройства
- prop - String - свойство
- ts - Number - timestamp
- val - Number - значение

#### Схема для других СУБД:

- id - Number - идентификатор метрики (имя устройства + свойства)  
Формируется при создании правила для записи в БД
- ts - Number - timestamp
- val - Number - значение
- **await core.getRecords('records', {sql })**
  - sql - строка, содержащая запрос

Обратите внимание, хотя в запросе имеется имя таблицы, первый аргумент по прежнему должен это имя содержать, так как функция getRecords работает с любыми таблицами системы и по имени определяет, куда делать запрос.

Пример 1. Получить записи, где значения были null (не поступали) для **СУБД SQLite**

```
const sqlStr =  
'SELECT * FROM records ' +  
'WHERE dn="TEMP1" ' +  
'AND prop="value" ' +  
'AND val IS NULL';  
const recs = await core.getRecords('records', {sql:sqlStr});
```

Пример 2. Получить записи, где значения были null для PostgreSQL

```
const sqlStr = 'SELECT * FROM records WHERE id=5 AND val is NULL';  
const recs = await core.getRecords('records', {sql:sqlStr});
```

Как видно в примере, id - это числовой идентификатор метрики, его можно увидеть в РМ:

База данных -> вкладка Правила записи в БД -> столбец Db id

В скрипте для получения идентификаторов метрик можно использовать специальную функцию **getIds**

#### Получение идентификаторов метрик

- **await core.getIds(['dn1.prop1','dn1.prop2'])**

Принимает массив строк dn.prop.

Возвращает массив идентификаторов, которые можно использовать при подготовке запроса.

```
const ids = await core.getIds(['TEMP1.value']);
if (!ids[0]) return {err: 'Ошибка при подготовке запроса!'}

const sqlStr = 'SELECT * FROM records WHERE id='+ids[0];
const recs = await core.getRecords('records', {sql:sqlStr});
```

## Список метрик

Можно получить список всех метрик, сохраняемых в исторической БД.

- `await core.getDbTargets()`

Возвращает массив строк dn.prop.

```
const targets = await core.getDbTargets();
debug(targets);
```

В результате получим массив: ['TEMP1.value','TEMP1.value2','METER102.value',...]

## Чтение данных из таблиц рецептов

Рецепты в системе хранятся в таблице **formulas** в следующем виде:

- id - уникальный идентификатор формулы - число (PRIMARY KEY NOT NUL)
- rid - идентификатор рецепта (схемы) - строка (NOT NULL)
- title - название формулы - строка (NOT NULL)
- description - описание формулы - строка
- comments - комментарий - строка
- active - флаг активности - число (1/0)
- ts - таймштамп - момент последней модификации
- jhead - JSON поле, содержит значения для **Общих параметры формулы**
- jrows - JSON поле, содержит массив значений ингредиентов из **Схемы** по каждой фазе.

Пример скрипта визуализации для получения данных с таблицы рецептов

```
const records = await core.getRecords(  
  'formulas',  
  {  
    sql: 'SELECT * FROM formulas WHERE active=1 ORDER BY id'  
  }  
);
```

## Сервисные функции

### Вывод даты, времени в виде строки

Для работы с временем в javascript удобно использовать таймстемп (Timestamp).

Он однозначно определяет время с точностью до миллисекунд.

**Timestamp** - это целое число, представляющее собой количество миллисекунд, прошедших с начала 1970 года в UTC.

В исторических таблицах системы используется именно timestamp (ts).

Работать с timestamp в javascript легко:

```
// Получить текущий таймстемп – 13-значное число
const ts = Date.now();
// 1662155144643
console.log(ts);

// Преобразовать ts во встроенный js объект типа Date
const date = new Date(ts);
// 2022-09-02T21:45:44.643Z – это стандартный формат вывода даты в UTC
console.log(date);

// Можно вывести дату в локальных форматах, зависит от настроек ОС
console.log(date.toString());
// Sat Sep 03 2022 00:45:44 GMT+0300 (Москва, стандартное время)

console.log(date.toLocaleString()). // 03.09.2022, 00:45:44
```

Хотя javascript имеет разные, в том числе локальные форматы вывода даты, для показа пользователю часто удобно использовать predefined форматы.

Можно, например, не выводить год и/или вывести миллисекунды. Или вывести только дату, или только время и т д

В API для этого предусмотрена функция вывода даты с использованием различных форматов. В любом случае используется локальное время сервера (с учетом timezone, установленной на уровне ОС).

- **core.getDateTimeStr(ts <, format >)**
  - ts - timestamp
  - format - опциональная форматная строка: если format не задан, дата будет выведена в формате DD.MM.YYYY HH:MM:SS

Значение параметра format	Формат вывода	Пример
'dtms'	DD.MM.YY HH:MM:SS.mmm	03.09.2022 00:45:44.643
'shortdtms'	DD.MM HH:MM:SS.mmm	03.09 00:45:44.643
'shortdt'	DD.MM HH:MM:SS	03.09 00:45:44
'reportdt'	DD.MM.YYYY HH:MM	03.09.2022 00:45
'reportd'	DD.MM.YYYY	03.09.2022
'onlytime'	HH.MM.SS	00:45:44
'id'	YYMMDDHHMMSSMMMM	2209030045440643
не задан	DD.MM.YYYY HH:MM:SS	03.09.2022 00:45:44

```
const str1 = core.getDateTimeStr(ts);
const str2 = core.getDateTimeStr(ts, 'dtms');
```

Конечно, всегда можно вручную отформатировать дату, используя функции объекта Date. Например, вывести время:

```
const ts = Date.now() - 3600000;
const dt = new Date(ts);
const str = dt.getHours() + ':' + dt.getMinutes() + ':' + dt.getSeconds();
```

## Операции CRUD в проекте

Операции CRUD (Create, Read, Update, Delete) - это четыре основных типа функциональности при работе с сущностями в ИТ.

В рамках нашей системы для такой работы с сущностями проекта предназначен Project Manager. Все операции по созданию, изменению, удалению сущностей проекта выполняются интегратором интерактивно.

Чтобы сделать доступным этот функционал из UI, начиная с версии v5.17.55 в API скриптов добавлены функции CRUD для работы с устройствами, каналами и плагинами.

Также постепенно добавляются функции для работы с сущностями раздела Аналитика (графики,...), содержащие правила формирования аналитических объектов.

Теперь интегратор может разработать в рамках UI экраны и скрипты, которые позволят специалистам конечных заказчиков расширять и дорабатывать проект без использования Project Manager.

## Обработка ошибок в операциях CRUD

Большинство функций, реализующих операции CRUD, являются **асинхронными**.

Асинхронные функции:

- вызываются с ключевым словом **await**
- в случае удачного завершения скрипт переходит к следующей строке
- в случае ошибки - генерируется исключение, выполнение скрипта прерывается.

Ошибка при выполнении может возникнуть, если невозможно выполнить операцию, например:

- отсутствует устройство, свойство которого хотим изменить
- отсутствует плагин, для которого хотим создать экземпляр
- дублируется имя, которое должно быть уникальным
- ...

Если ошибку не обрабатывать, скрипт будет завершен с сообщением пользователю **Ошибка скрипта**. Очевидно, нужно дать пользователю более адекватное сообщение.

Для этого рекомендуется всегда оборачивать вызов асинхронных функций в try - catch. В случае ошибки управление передается блоку catch с объектом, описывающим ошибку. Объект содержит код и текстовое описание ошибки:

```
{message:<Текстовое сообщение>, code:<код ошибки>}
```

В описании функции обычно есть таблица ошибок, генерируемых конкретной функцией.

### Пример

Создадим новый бинарный актуатор (RELE\_1) с помощью функции createDevice. В описании функции createDevice находим таблицу ошибок:

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
В проекте нет такого типа	Тип t020 не найден	TypeNotFound
В data передан dn, а устройство с таким dn уже существует	Устройство DT_101 уже существует	DnExists

Вы можете использовать текст ошибки (e.message) напрямую, или ориентироваться на e.code и выводить свои сообщения, например:



```
module.exports = async function({ local, context, source }, core, debug)
const dn = 'RELE_1'; // В реальном скрипте получаем через local
try {
  const rele = await core.createDevice('t018',{dn:dn});
  // ....
} catch (e) {
  debug(e); // В консоль можно вывести объект ошибки

  let errStr;
  if (e.code == 'DnExists') {
    // Можно сформировать свое сообщение
    errStr = dn+' уже есть, выберите другое имя для реле.';
  } else if (e.code == 'TypeNotFound') {
    // Можно вывести то, которое передает функция
    errStr = e.message;
  } else {
    // Во всех других случаях, кроме предусмотренных
    errStr = 'Не удалось создать устройство!';
  }
  return {err: errStr}
}
};
```

Последний вариант используется, чтобы пользователь гарантированно увидел, что что-то пошло не так, но при этом скрыть детали непредвиденной ошибки.

## Работа с устройствами

Добавлено: v5.17.57

Из скрипта визуализации можно не только получить [доступ к существующим устройствам](#), но и создать, изменить, удалить устройство, а также выполнить привязку к каналам.

### createDevice: создать новое устройство

Асинхронная функция **createDevice** позволяет создать новое устройство.

- **const dev = await core.createDevice**(<typeld - тип>, <data - опционально>);
  - **typeld** - идентификатор типа - обязательный параметр
  - **data** - объект, может содержать следующие статические свойства:
    - **name** - текстовое название устройства.  
Если не задано - берется из типа
    - **dn** - уникальное имя устройства, допустимы только символы azAZ09\_  
Должно быть уникально для проекта.  
Если не задано - генерируется автоматически с префиксом из типа.
    - **parent** - идентификатор узла дерева устройств.  
Определяет папку, в которую нужно поместить устройство.  
Новое устройство всегда помещается в конец списка внутри папки  
Если не задано или не существует - помещается в корневой узел.
    - **tags** - теги, может быть строкой или массивом строк.

**Результат** - объект устройства, аналогичный функции [getDevice](#)

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не задан тип (первый аргумент пустой)	Не задан тип	TypeMissing
В проекте нет такого типа	Тип t020 не найден	TypeNotFound
В data передан dn, а устройство с таким dn уже существует	Устройство DT_101 уже существует	DnExists
В data передан пустой dn (или содержащий только пробелы)	dn: значение не может быть пустым	EmptyDn
В data передан dn, строка содержит недопустимые символы для идентификатора	Мой_клапан недопустимое значение	InvalidDn
В data передан пустой parent (или содержащий только пробелы)	parent: значение не может быть пустым	EmptyParent
В data передан parent, папка с таким id не существует	Папка dg004 не существует	ParentNotFound

Подходы к обработке ошибок описаны на странице [Обработка ошибок](#)

## Примеры

Пример 1. Создание устройства без задания опциональных свойств. Устройство будет размещено в корневой папке.

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const dev = await core.createDevice('t020');
    return {info: 'Создано новое устройство '+dev.dn}
  } catch (e) {
    return {err: e.message || 'Не удалось создать устройство!'}
  }
};
```

Пример 2. Создание устройства с заданием текстового названия и папки. В реальном применении значения свойств, конечно, будут приходить от пользователя и извлекаться из local.

```
module.exports = async function({ local, context, source }, core, debug) {
  const parent = 'dg042';
  const name = 'Новый датчик';
  try {
    const dev = await core.createDevice('t020',{name, parent});
    return {info: 'Создано новое устройство '+dev.dn}
  } catch (e) {
    debug(e);
    return {err: e.message || 'Не удалось создать устройство!'}
  }
};
```

Пример 3. Создание устройства с заданием уникального имени, папки и тегов. Если устройство с таким dn уже существует, будет сгенерирована ошибка с сообщением 'Устройство DT\_101 уже существует'.

```
module.exports = async function({ local, context, source }, core, debug) {
  const dn = 'DT_101';
  const parent = 'dg042';
  try {
    const dev = await core.createDevice('t020', {dn, parent, tags:'Вентиляция'});
    return {info: 'Создано новое устройство '+dev.dn}
  } catch (e) {
    return {err: e.message || 'Не удалось создать устройство! '}
  }
};
```

## updateDevice: изменить статические свойства устройства

Асинхронная функция **updateDevice** позволяет изменить статические свойства устройства.

- **const dev = await core.updateDevice** (<did - id устройства >, <data - объект>);\*\*
  - **did** - строка, идентификатор устройства
  - **data** - объект, содержит изменяемые статические свойства: name, dn, parent, tags

**Результат** - объект устройства, аналогичный функции [getDevice](#)

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найдено устройство, передаваемое в первом аргументе	Устройство d0234 не найдено	DeviceNotFound
В data передан dn, а устройство с таким dn уже существует	Устройство DT_101 уже существует	DnExists
В data передан пустой dn (или содержащий только пробелы)	dn: значение не может быть пустым	EmptyDn
В data передан dn, строка содержит недопустимые символы для идентификатора	Мой_клапан недопустимое значение	InvalidDn
В data передан пустой parent (или содержащий только пробелы)	parent: значение не может быть пустым	EmptyParent
В data передан parent, папка не существует	Папка dg004 не существует	ParentNotFound

## Примеры

Пример 1. Изменение папки и тегов.

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const dev = await core.updateDevice('d0234',{parent:'dg018', tags:'01
    return {info: 'Изменены свойства устройства '+dev.dn}
  } catch (e) {
    return {err: e.message || 'Не удалось изменить свойства устройства! '}
  }
};
```

Пример 2. Изменение уникального имени.

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const dev = await core.updateDevice('d0734',{dn:'DT_122'});
    return {info: 'Имя устройства изменено на '+dev.dn}
  } catch (e) {
    return {err: e.message || 'Не удалось изменить имя устройства!'}
  }
};
```

### removeDevice: удалить устройство

Асинхронная функция **removeDevice** позволяет удалить устройство. После удаления существующие привязки каналов сбрасываются. Удаление работает аналогично интерактивному - если устройство используется в проекте, оно не может быть удалено.

- `await core.removeDevice(<ID устройства>);`

**Результат** - нет

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найдено устройство	Устройство d0234 не найдено	DeviceNotFound
Устройство используется в проекте	Устройство d0234 используется и не может быть удалено	DeviceIsUsed

Пример 1.

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const dev = await core.removeDevice('d0734');
    return {info: 'Устройство успешно удалено'}
  } catch (e) {
    return {err: e.message || 'Не удалось удалить устройство!'}
  }
};
```

Асинхронная функция **getDeviceChannels** возвращает массив каналов, привязанных к свойствам указанного устройства.

Если указать ID экземпляра плагина, будут возвращены каналы, связанные с конкретным плагином.

- **const arr = await core.getDeviceChannels({did:<ID устройства>, unit:<ID плагина>});**

Входной объект должен содержать:

- **did** - ID устройства
- **unit** - ID экземпляра плагина (опционально)

**Результат** - массив, содержащий каналы

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найдено устройство	Устройство d0234 не найдено	DeviceNotFound
В проекте нет указанного экземпляра плагина	Плагин mqttclient99 не найден	PluginNotFound

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const arr = await core.getDeviceChannels({did:'d0002', unit:'modbus1'
  } catch (e) {
    return {err:e.message}
  }
};
```

## removeDeviceAndChannels: удалить устройство и связанные с ним каналы

Асинхронная функция **removeDeviceAndChannels** удаляет устройство и связанные с ним каналы.

- **await core.removeDeviceAndChannels(<ID устройства>);**

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найдено устройство	Устройство d0234 не найдено	DeviceNotFound
Устройство используется в проекте	Устройство d0234 используется и не может быть удалено	DevicesUsed

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const arr = await core.removeDeviceAndChannels('d0002');
  } catch (e) {
    return {err:e.message}
  }
};
```

## createDevicesFolder: создать папку для устройств

Асинхронная функция **createDevicesFolder** позволяет создать папку для устройств.

- **const dev = await core.createDevicesFolder(<data - опционально>);**
  - **data** - объект, может содержать следующие свойства
    - **name** - название папки  
Если не задано - создается с названием *Новая папка*
    - **parent** - идентификатор узла дерева устройств.  
Определяет папку, в которую нужно поместить новую папку.  
Если не задано - папка добавляется в корневую папку.  
Если заданная папка не найдена, возникает ошибка

**Результат** - id новой папки

Пример 1.

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const id = await core.createDevicesFolder();
    return {info: 'Создана папка для устройств с названием Новая папка, ID: ' + id};
  } catch (e) {
    return {err: e.message || 'Не удалось создать папку для устройств! '};
  }
};
```

Пример 2.



```
module.exports = async function({ local, context, source }, core, debug) {  
  try {  
    const folderId = await core.createDevicesFolder({name:'Test', parent:'dg0  
    return {info: 'Создана папка для устройств с названием Test, Place ID '+i  
  } catch (e) {  
    return {err: e.message || 'Не удалось создать папку для устройств! '}  
  }  
};
```

## Работа с плагинами

Из скрипта визуализации можно получить информацию о плагинах, а также создать, изменить, удалить экземпляры плагина.

### getPlugins: получить список плагинов

Функция **getPlugins** возвращает массив всех плагинов проекта.

- `const plugins = await core.getPlugins()`

Каждый объект результирующего массива содержит информацию об одном плагине:

```
[{id: 'cctv', title: 'cctv', single: 1, state: 1}, ...]
```

Свойство state - это [текущее состояние плагина](#)

Если это мультиплагин (single: 0), свойство units содержит массив экземпляров плагина.

```
[
  {id: 'modbus', title: 'MODBUS', single: 0, units: [
    { id: 'modbus1', title: 'Первый контроллер', state: 1 },
    { id: 'modbus2', title: 'Второй контроллер', state: 3 }
  ]
},
...
]
```

Если ни один плагин не используется в проекте, будет возвращен пустой массив.

```
module.exports = async function({ local, context, source }, core, debug) {
  const arr = await core.getPlugins();
  if (!arr.length) return {info: 'В проекте нет ни одного плагина!'}
  // ...
};
```

### getPlugin: получить данные плагина

Функция **getPlugin** возвращает объект, содержащий расширенную информацию об одном плагине, включая данные экземпляров.

- `const onePlugin = await core.getPlugin(<ID плагина>)`

Кроме данных, возвращаемых `getPlugins` {id, title, single, state}, объект содержит вложенный объект `params` - параметры плагина.

```
{
  id: 'cctv',
  title: 'cctv',
  single: 1,
  state: 1,
  params: {
    wsport: 8099,
    restarttime: 1,
    // ...
  }
}
```

Если это мультиплагин, свойство `units` содержит массив экземпляров плагина и параметры передаются для экземпляров.

```
{
  id: 'modbus',
  title: 'MODBUS',
  single: 0,
  units: [
    {
      id: 'modbus1',
      title: 'Первый контроллер',
      state: 1,
      params: {
        transport: 'tcp',
        host: '192.168.0.250',
        // ...
      }
    },
    {
      id: 'modbus2',
      title: 'Второй контроллер',
      state: 3,
      params: {
        transport: 'tcp',
        host: '192.168.0.251',
        // ...
      }
    }
  ]
}
```

Если плагин не используется в проекте, будет возвращено значение **undefined**.

```
module.exports = async function({ local, context, source }, core, debug) {
  const modbusPlugin = await core.getPlugin('modbus');
  if (!modbusPlugin) return {info: 'В проекте не используется плагин modbus!'};
  // ...
};
```

## getPluginUnit: получить данные экземпляра плагина

Функция **getPluginUnit** возвращает объект, содержащий информацию только об одном экземпляре плагина

- **const unit = await core.getPluginUnit(<ID экземпляра плагина>)**

- Для мультиплагина будет возвращен элемент массива units.

Если экземпляр плагина не найден, будет возвращено значение **undefined**.

```
module.exports = async function({ local, context, source }, core, debug) {
  const modbusUnit1 = await core.getPluginUnit('modbus1');
  if (!modbusUnit1) return {info: 'Экземпляр modbus1 не найден!'}

  // ...
};
```

## startPluginUnit: запустить экземпляр плагина

Функция **startPluginUnit** позволяет отправить команду Start для экземпляра плагина

- **core.startPluginUnit(<ID экземпляра плагина>)**

Функция вызывается без await и не возвращает результат.

Если экземпляр не найден, возникнет ошибка скрипта.

Если экземпляр уже запущен, команда будет проигнорирована.

```
module.exports = async function({ local, context, source }, core, debug) {
  const onePlugin = await core.getPluginUnit('modbus1');
  if (!onePlugin) return {info: 'Экземпляр modbus1 не найден!'}
  if (onePlugin.state == 1) return {info: 'Экземпляр modbus1 уже запущен'}
  core.startPluginUnit('modbus1');
  return {
    info:
      'Запускается экземпляр плагина ' +
      onePlugin.id +
      ': ' +
      onePlugin.title
  };
};
```

## stopPluginUnit: остановить экземпляр плагина

Функция **stopPluginUnit** позволяет отправить команду Stop для экземпляра плагина

- **core.stopPluginUnit(<ID экземпляра плагина>)**

Функция вызывается без `await` и не возвращает результат.

Если экземпляр не найден, возникнет ошибка скрипта.

Если экземпляр уже приостановлен (находится в состоянии `suspend`), команда будет проигнорирована.

```
module.exports = async function({ local, context, source }, core, debug) {
  const onePlugin = await core.getPluginUnit('modbus1');
  if (!onePlugin) return {info: 'Экземпляр modbus1 не найден!'}

  if (onePlugin.state == 2) return {info: 'Экземпляр modbus1 уже остановлен'}
  core.stopPluginUnit('modbus1');
  return {
    info:
      'Для экземпляра плагина ' +
      onePlugin.id +
      ': ' +
      onePlugin.title +
      ' отправлена команда STOP'
  };
};
```

## createPluginUnit: создать экземпляр плагина

Функция **createPluginUnit** создает новый экземпляр плагина.

Очевидно, что этот функционал доступен только для мультиплагинов.

- `const unit = await core.createPluginUnit(<ID плагина>, <параметры плагина - опционально>)`

Если второй аргумент не определен, всем параметрам будут присвоены дефолтные значения.

```
module.exports = async function({ local, context, source }, core, debug) {
  const emulUnit = await core.createPluginUnit('emuls');
};
```

Обычно имеет смысл передать основные параметры, остальным будут присвоены дефолтные значения.

Также можно дать название экземпляру, добавив свойство **name**.

```
module.exports = async function({ local, context, source }, core, debug) {
  const modbusUnit = await core.createPluginUnit(
    'modbus',
    {
      name: 'Контроллер цех 1',
      transport: 'tcp',
      host: '192.168.0.250'
    }
  );
};
```

Функция вернет значение в формате getPluginUnit, после создания state=2 (Приостановлен).

```
{
  id: 'modbus3',
  title: 'Контроллер цех 1',
  state: 2,
  params: {
    transport: 'tcp',
    host: '192.168.0.250',
    port: 502,
    // ...
  }
}
```

Если экземпляр создать не удалось, возникнет ошибка скрипта. Ошибку, как обычно, можно перехватить, обернув вызов в try-catch

```
module.exports = async function({ local, context }, core, debug) {
  try {
    const onePlugin = await core.createPluginUnit(
      'modbus',
      {
        name: 'Суперновый экземпляр',
        host: '172.16.21.22'
      }
    );

    return {
      info:
        'Создан экземпляр плагина ' +
        onePlugin.id +
        ': ' +
        onePlugin.title
    };
  } catch (e) {
    let errStr = e.message || '';
    return {err: 'Не удалось создать экземпляр плагина! ' + errStr}
  }
};
```

## updatePluginUnit: изменить экземпляр плагина

Функция **updatePluginUnit** позволяет изменить наименование и параметры экземпляра плагина.

- `const unit = await core.updatePluginUnit(<ID экземпляра плагина>, <изменяемые параметры>)`

Функция вернет значение в формате getPluginUnit.



```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const onePlugin = await core.updatePluginUnit(
      'modbus3',
      {
        name: 'Контроллер 42',
        host: '192.160.0.42'
      }
    );

    return {
      info:
        'Изменен экземпляр плагина ' +
        onePlugin.id +
        ': ' +
        onePlugin.title
    };
  } catch (e) {
    let errStr = e.message || '';
    return {err: 'Не удалось изменить экземпляр плагина modbus3! ' + errStr}
  }
};
```

Измененные параметры будут переданы плагину при следующем запуске. Если нужно, чтобы изменения вступили в силу сразу, необходимо остановить и запустить экземпляр плагина.

```
module.exports = async function({ local, context, source }, core, debug) {
  const onePlugin = await core.getPluginUnit('modbus3');
  if (!onePlugin) return {info: 'Экземпляр modbus3 не найден!'}

  if (onePlugin.state == 1) {
    debug('Остановить modbus3')
    core.stopPluginUnit('modbus3');
  }

  const onePlugin = await core.updatePluginUnit('modbus3', {host: '192.160.0.42'});
  core.startPluginUnit('modbus3');
  debug('Запустить modbus3')
  return {info: onePlugin.id+ ' запущен с новым адресом'}
};
```

## deletePluginUnit: удалить экземпляр плагина

Функция **deletePluginUnit** позволяет удалить экземпляр плагина, включая каналы. После выполнения операции привязки к свойствам устройств будут сброшены.

- `const unit = await core.deletePluginUnit(<ID экземпляра плагина>)`

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    await core.deletePluginUnit('modbus3');
    return {info: 'Удален экземпляр плагина modbus3.'}
  } catch (e) {
    let errStr = e.message || '';
    return {err: 'Не удалось удалить экземпляр плагина modbus3! ' + errStr}
  }
};
```

## Работа с каналами

Из скрипта визуализации можно получать информацию о каналах, а также создавать, копировать, удалять каналы и выполнять привязку к устройству.

### getChannels: получить каналы плагина

Добавлено: v5.17.57

Асинхронная функция **getChannels** возвращает дерево всех каналов экземпляра плагина, включая папки/узлы.

- **const arr = await core.getChannels(<ID экземпляра плагина>);**

**Результат** - массив, содержащий дерево экземпляра плагина

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
В проекте нет указанного экземпляра плагина	Плагин modbus1 не найден	PluginNotFound

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const arr = await core.getChannels('modbus1');
    debug(arr);
  } catch (e) {
    return {err:e.message}
  }
};
```

Результирующий массив содержит дерево, состоящее из папок (папка имеет массив children).  
Внутри массива children могут быть как папки (узлы), так и непосредственно каналы.

```
[
  {id: 'XInx5fATg', title: 'ALL', component: 'channelfolder.modbus1', children: [
    {id: 'Fw-5xrjHiM', title: 'DD1', component: 'channelview.modbus1'},
    {id: 'KN6GGNpnBE', title: 'DD2', component: 'channelview.modbus1'}
  ]
}]
```

Каналы содержат только id и title. Чтобы получить данные конкретного канала, используйте функцию `getChannel`.

## getChannel: получить данные канала

Добавлено: v5.17.63

Асинхронная функция `getChannel` возвращает данные конкретного канала.

- `const arr = await core.getChannel(<ID канала>);`
  - **ID канала** - уникальный идентификатор канала - обязательный параметр

**Результат** - объект, содержащий данные канала

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
В проекте нет канала с указанным ID	Канал не найден	ChannelNotFound

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const chanId = local.channelId;
    const chanData = await core.getChannel(chanId);
    debug(chanData);
  } catch (e) {
    return {err:e.message}
  }
};
```

Объект содержит данные канала.

Если канал входит в узел, данные также включают свойства родительского узла.

```
{
  parent_title: 'Energy Meter', // Свойство узла
  unitid: '1', // Свойство узла
  chan: 'v1',
  r: 1,
  w: 0,
  req: 0,
  address: '5136',
  vartype: 'uint16',
  manbo: 0,
  fcr: '3',
  _id: 'Fw-5xrjHiM',
  unit: 'modbus1'
}
```

## createChannel: создать канал

*Добавлено: v5.17.57*

Асинхронная функция **createChannel** позволяет создать новый канал для экземпляра плагина. На этом этапе можно сразу привязать канал к свойству устройства.

- **const chanFolderId = await core.createChannel**(<unit - ID плагина>, <data - данные для канала>, <link - данные для привязки>)
  - **unit** - ID экземпляра плагина
  - **data** - объект, содержащий свойства нового канала:
    - chan - новое название канала, если нет - формируется автоматически
    - r:1/0 - флаг операции чтения
    - w:1/0 - флаг операции записи
    - parent - ID папки или узла, по умолчанию корневая папка
    - другие свойства канала, которые зависят от конкретного плагина
  - **link** - опционально: объект, содержащий данные для привязки нового канала к устройству
    - did - ID устройства
    - prop - ID свойства

Возвращает объект вновь созданного канала.

Пример 1. Создать новый канал для плагина mqttclient1

```
try {
  const oneChannel = await core.createChannel(
    'mqttclient1',
    {
      chan: 'ch_42',
      r: 1,
      topic: '/devices/dev/42'
    }
  );
  debug(oneChannel);
  return {info: 'Создан канал ' + oneChannel.chan}
} catch (e) {
  let errStr = e.message || '';
  return {err: 'Не удалось создать канал! ' + errStr}
}
```

Пример 2. Создать новый канал для плагина mqttclient1 и привязать его к устройству

```
try {
  const oneChannel = await core.createChannel(
    'mqttclient1',
    {
      chan: 'ch_44',
      r: 1,
      topic: '/devices/dev/42'
    },
    {
      did: 'd0306',
      prop: 'state'
    }
  );
  debug(oneChannel);
  return {info: 'Создан канал ' + oneChannel.chan}
} catch (e) {
  let errStr = e.message || '';
  return {err: 'Не удалось создать канал! ' + errStr}
}
```

## updateChannel: изменить канал

Добавлено: v5.17.63

Асинхронная функция **updateChannel** позволяет изменить данные канала.

- **const chanFolderId = await core.updateChannel(<ID канала>, <data - изменяемые данные для канала>)**

**Результат** - объект, содержащий обновленные данные канала

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
В проекте нет канала с указанным ID	Канал не найден	ChannelNotFound

Пример 1. Изменить топик канала для плагина mqttclient1

```
try {
  const chanId = local.channelId;
  const oneChannel = await core.updateChannel(
    chanId,
    {
      topic: '/devices/dev/set/status'
    }
  );
  debug(oneChannel);
  return {info: 'Данные канала ' + oneChannel.chan+ ' изменены'}
} catch (e) {
  return {err: e.message}
}
```

## linkDevicePropToChannel: привязать канал к свойству устройства

Добавлено: v5.17.63

Асинхронная функция **linkDevicePropToChannel** позволяет привязать канал к свойству устройства

- **await core.linkDevicePropToChannel(<did - id устройства>, , <chanId - ID канала>)**
- **linkDevicePropToChannel(did, prop, chanId)**
  - **did** - ID устройства
  - **prop** - свойство устройства
  - **chanId** - ID канала (уникальный внутренний идентификатор канала)

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
В проекте нет устройства с указанным ID	Устройство d0234 не найдено	DeviceNotFound
В проекте нет канала с указанным ID	Канал не найден	ChannelNotFound

Пример. Привязать свойство value выбранного устройства к выбранному каналу

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const devId = local.devId;
    const chanId = local.channelId;
    await core.linkDevicePropToChannel(devId, 'value', chanId);
    return {info: 'Выполнена привязка канала к свойству value' }
  } catch (e) {
    return {err:e.message}
  }
}
```

## removeChannel: удалить канал

Добавлено: v5.17.63

Асинхронная функция **removeChannel** позволяет удалить канал.

- **await core.removeChannel**(<ID канала>)

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
В проекте нет канала с указанным ID	Канал не найден	ChannelNotFound

Пример 1. Удалить канал

```
try {
  const chanId = local.channelId;
  await core.removeChannel(chanId);
  return {info: 'Удаление канала успешно'}
} catch (e) {
  return {err: e.message}
}
```



## Работа с папками/узлами каналов

Каналы можно группировать в **папки**. Например, можно объединить в одной папке каналы, связанные с одним устройством.

В этом случае привязку всех свойств можно выполнить с помощью одной операции - **Привязать к устройству**.

Также можно создать специальную папку - **узел**. Узел содержит дополнительную информацию, которая передается во все вложенные каналы.

Например, для Modbus-RTU узлами могут выступать отдельные устройства со своим unitid.

Из скрипта визуализации можно создавать, копировать, удалять папки/узлы.

### createChannelsFolder: создать папку или узел

Добавлено: v5.17.72

Асинхронная функция **createChannelsFolder** позволяет создать новую папку/узел в каналах

- **await core.createChannelsFolder**(<unit - ID плагина>, <data - свойства новой папки>)
  - **unit** - ID экземпляра плагина
  - **data** - объект, содержащий свойства новой папки/узла, все свойства опциональны
    - **chan** - новое название папки/узла, по умолчанию Новая папка/Новый узел
    - **parentid** - ID родительской папки, по умолчанию размещается в корневой папке
    - **foldertype** - 'node', если нужно создать узел
    - другие свойства узла, зависящие от плагина, например, unitid, offset

**Результат** - Возвращает уникальный внутренний идентификатор новой папки, который затем можно использовать при привязке к устройству.

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найден экземпляр плагина unit	Плагин не существует!	PluginNotFound

Пример 1. Создать новый узел в дереве плагина modbus1.

```
module.exports = async function({ local, context, source }, core, debug)
  try {
    const chanFolderId = await core.createChannelsFolder(
      'modbus1',
      {
        chan: 'myPLC_15',
        unitid: 15,
        parentoffset: 1500
      }
    );
    return {info: 'Новый узел создан'}
  } catch (e) {
    return {err: e.message}
  }
}
```

## copyChannelsFolder: копировать папку/узел

Добавлено: v5.17.57

Асинхронная функция **copyChannelsFolder** позволяет создать новую папку/узел каналов, скопировав существующую.

- **await core.copyChannelsFolder**(<unit - ID плагина>, <chanFolderId - id папки>, <data - свойства новой папки>)
  - **unit** - ID экземпляра плагина
  - **chanFolderId** - ID папки (уникальный внутренний идентификатор папки/узла каналов)
  - **data** - объект, содержащий свойства новой папки:
    - **chan** - новое название папки/узла, если нет - формируется из названия копируемой папки
    - другие свойства узла, например, unitId, offset

**Результат** - Возвращает уникальный внутренний идентификатор новой папки, который затем можно использовать при привязке к устройству.

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найден экземпляр плагина unit	Плагин не существует!	PluginNotFound
Не найдена папка/узел с указанным ID	Исходная папка не существует!	SourceFolderNotFound
Указанный ID - канал, а не узел	Выберите папку!	ExpectFolder

```
module.exports = async function({ local, context, source }, core, debug)
// id выбранного в дереве каналов узла
const selected_chanfolder_id = local.selected_chanfolder_id;
if (!selected_chanfolder_id) return {err: 'Узел для копирования не выбран'}

try {
    const chanFolderId = await core.copyChannelsFolder(
    'modbus1',
    selected_chanfolder_id,
    {
        chan: 'myPLC_15',
        unitid: 15,
        parentoffset: 1500
    }
    );
    return {info: 'Узел скопирован'}
} catch (e) {
    return {err: e.message}
}
```

Пример 2. Узел для копирования выбрать в скрипте по названию папки.  
Используется функция **getChanFolderId**, описанная ниже.

## getChanFolderId: получить идентификатор папки/узла

Добавлено: v5.17.57

Асинхронная функция **getChanFolderId** возвращает уникальный внутренний идентификатор папки/узла каналов.

Применяется при использовании функций **copyChannelsFolder**, **linkDeviceToChannels**, если нужно выбрать папку внутри скрипта (не интерактивно).

- **const chanFolderId = await core.getChanFolderId(<unit -ID плагина>, filter);**
  - **unit** - ID экземпляра плагина
  - **filter** - объект для поиска узла Должен содержать свойства, однозначно идентифицирующие папку/узел.  
Если фильтру удовлетворяет несколько записей, будет взята одна из них
    - **chan** - название папки/узла с каналами
    - другие свойства узла, специфичные для плагина

Пример 1. Получить идентификатор по названию папки

```
module.exports = async function({ local, context, source }, core, debug) {  
  const chanFolderId1 = await core.getChanFolderId('modbus1', {chan:'PLC_1'  
};
```

Пример 2. Получить идентификатор по свойству узла для Modbus RTU

```
module.exports = async function({ local, context, source }, core, debug)  
  const chanFolderId1 = await core.getChanFolderId('modbus1', {unitid:7 }  
};
```

## linkDeviceToChannels: привязать устройство к каналам папки/узла

Добавлено: v5.17.57

Асинхронная функция **linkDeviceToChannels** позволяет привязать свойства устройства к каналам, находящимся в папке/узле.

Привязка выполняется при совпадении поля канала **Свойство для привязки** с именем свойства устройства. Если привязка канала уже выполнена к другому устройству, привязка не выполняется.

- **await core.linkDeviceToChannels(<did - id устройства>, <chanFolderId - ID папки>)**

- **chanFolderId** - ID папки (уникальный внутренний идентификатор папки/узла каналов) Возвращается в дереве каналов или может быть получен функцией **getChanFolderId**

Пример. Создать устройство и привязать его свойства к каналам нового узла плагина modbus1. Узел создать копированием существующего узла.

```
module.exports = async function({ local, context, source }, core, debug) {
  try {
    const dev = await core.createDevice('t020');
    const sourceId = await core.getChanFolderId('modbus1', {chan:'PLC_1'});
    const newId = await core.copyChannelsFolder(
      'modbus1',
      sourceId,
      {
        chan: 'myPLC_16',
        unitid: 16
      }
    );

    await core.linkDeviceToChannels(dev.id, newId);

    return {
      info:
        'Создано новое устройство ' +
        dev.id +
        ', выполнена привязка к каналам узла myPLC_16'
    };
  } catch (e) {
    return {err:e.message}
  }
}
```

## updateChannelsFolder: изменить параметры папки/узла

Добавлено: v5.17.63

Асинхронная функция **updateChannelsFolder** позволяет изменить данные узла.

- **const data = await core.updateChannelsFolder(<ID папки>, <data - изменяемые данные>)**

**Результат** - объект, содержащий обновленные данные

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найдена папка/узел с указанным ID	Исходная папка не существует!	SourceFolderNotFound
Указанный ID - канал, а не узел	Выберите папку!	ExpectFolder

Пример 1. Изменить адрес устройства в узле PLC\_8 плагина modbus1

```
try {
  const chanFolderId = await core.getChanFolderId('modbus1', {chan: 'PLC_8'})
  const data = await core.updateChannelsFolder(chanFolderId, {unitid:16})
  debug(data);
  return {info: 'Данные канала '+ data.chan+' изменены'}
} catch (e) {
  return {err: e.message}
}
```

## removeChannelsFolder: удалить папку/узел

Добавлено: v5.17.63

Асинхронная функция **removeChannelsFolder** позволяет удалить папку/узел, включая все вложенные каналы и узлы

- **await core.removeChannelsFolder(<ID папки>)**

**Ошибка** генерируется в следующих случаях:

Причина ошибки	e.message	e.code
Не найдена папка/узел с указанным ID	Исходная папка не существует!	SourceFolderNotFound
Указанный ID - канал, а не узел	Выберите папку!	ExpectFolder

Пример 1. Удалить узел PLC\_8 плагина modbus1

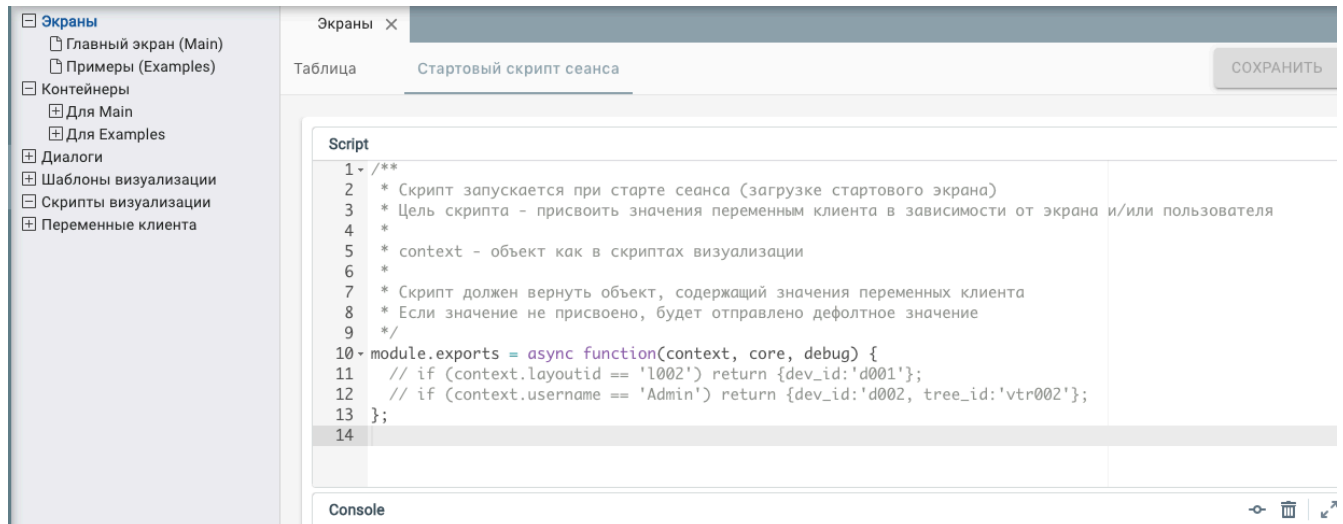
```
try {  
  const chanFolderId = await core.getChanFolderId('modbus1', {chan: 'PLC_8  
  await core.removeChannelsFolder(chanFolderId);  
  debug(oneChannel);  
  return {info: 'Узел с каналами успешно удален'}  
} catch (e) {  
  return {err: e.message}  
}
```

## Стартовый скрипт сеанса

Есть возможность создать скрипт, который запускается при старте сеанса пользователя (загрузке стартового экрана UI).

Основная цель скрипта - присвоить значения **переменным клиента** в зависимости от экрана и/или пользователя.

Редактор скрипта доступен при клике на узел "Экраны".



## Активация скрипта

По умолчанию в редакторе выводится заготовка для скрипта. Чтобы скрипт начал запускаться, нужно внести любое изменение и нажать **Сохранить**. Теперь при вызове UI в новой вкладке скрипт всегда будет вызываться.

## Входные аргументы

На входе скрипт получает 3 аргумента:

- **context** - объект, содержащий данные о пользователе
- **core** - объект, реализующий API
- **debug** - функция, позволяющая выводить отладочные сообщения в отладчике

Отличие от обычного скрипта визуализации в том, что первый аргумент содержит только **context**, тогда как обычный скрипт визуализации содержит **{local, context, source}**

При старте переменные клиента (local) еще не определены.

Как сказано выше, основной целью скрипта является инициализация **переменных клиента** для сеанса. Если скрипт не используется или ничего не возвращает, всем переменным клиента присваиваются дефолтные значения.

Объект **context** включает:

- **userid** - ID пользователя
- **username** - Имя пользователя
- **start\_layoutid** - Стартовый экран пользователя
- **layoutid** - Текущий экран (если пользователь зашел по ссылке, они могут отличаться)



Добавлено в 5.11.47

- user\_agent - информация из заголовка запроса

Также есть ограничения на использование функций **core** - можно использовать функции, выполняемые на стороне сервера: getDevice, getRecords, ... Но интерактивные функции клиента, типа gotoLayout, playSound, showDialog - работать не будут.

## Отладчик

Как для любого скрипта, в консоли **Редактора** можно отслеживать запуск и выполнение. Чтобы отладить скрипт, откроем в отдельной вкладке окно **Редактор** скрипта.

В другой вкладке вызовем UI.

В консоли увидим, что скрипт получил контекст:

```
23.12 10:51:25.391 layoutgroup
23.12 10:54:44.896 Session Start Script started
context = {
  userid: 'admin',
  username: 'Admin',
  start_layoutid: 'l025',
  layoutid: 'l025',
  user_agent:
'Mozilla/5.0 AppleWebKit/537.36 ' +
'(KHTML, like Gecko) ' +
'Chrome/108.0.0.0 ' +
'Safari/537.36'
}

23.12 10:54:44.904 Session Start Script result
{}
```

Если скрипт не содержит **return**, по умолчанию результат - это пустой объект.

## Выходные аргументы

Чтобы изменить дефолтные значения переменных для сеанса, скрипт должен вернуть объект, содержащий пары *имя переменной:значение*.

Пример 1.

В зависимости от пользователя нужно переключать контейнер и выбирать текущее устройство (например, вентустановку)

```
module.exports = async function(context, core, debug) {  
  switch (context.username) {  
    case 'Electric01': return {vc_id:'vc101', dev_id:'d0242'};  
    case 'Electric02': return {vc_id:'vc102', dev_id:'d0780'};  
    default: // Можно ничего не возвращать, будут использованы дефолтные значения  
  }  
};
```

Пример 2.

Можно в переменную **user\_mobile** записать 1/0 и в дальнейшем использовать для навигации в других скриптах.

```
module.exports = async function(context, core, debug) {  
  const ismobile = context.user_agent.indexOf('Mobile') > 0;  
  return {user_mobile: ismobile};  
};
```

# Скрипты заполнения

Для **Списка**, **Таблицы**, **Дерева** есть возможность создать скрипт заполнения.

Скрипт вызывается при отрисовке компонента.

По сути это скрипт визуализации, который должен вернуть объект специального вида.

Можно использовать все функции [API скрипта визуализации](#)

## Скрипт заполнения списка

Должен вернуть {data : [массив]}

Элементы массива - это объекты {id, title}.

Каждый элемент представляет строку списка : {id : <идентификатор строки>, title : <видимый текст>}

**Пример 1.** Список заполняется статическими данными.

```
/**
 * Скрипт заполнения
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context}, core, debug) {
  const lines = [
    {id: 'red', title: 'Красный'},
    {id: 'green', title: 'Зеленый'},
    {id: 'blue', title: 'Синий'}
  ];
  return {data: lines};
}
```

**Пример 2.** Список заполняется из пользовательской таблицы 'mytable'.

Данные фильтруются по значению поля 'myfield' и сортируются по полю 'ts' в порядке убывания

```

/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context}, core, debug) {
  const data = [];
  const docs = await core.getRecords('mytable', {myfield:'test'}, {sort:{ts:-
    if (docs) {
      docs.forEach(doc => data.push({id:doc._id, title: doc.name}));
    }
    return {data};
  }
}

```

**Пример 3.** Список заполняется из дерева пользователей.

```

/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context}, core, debug) {
  const data = await core.getListFromTree('users');
  return {data};
}

```

Обратите внимание, скрипт заполнения может не использовать никакие входные аргументы.

Его цель - просто вернуть массив, который используется для заполнения списка. Но, конечно, если нужно, то можно использовать входные аргументы local и/или context.

Подробнее про входные аргументы скрипта см [Структура скрипта визуализации](#)

**Пример 4.** Список заполняется из дерева пользователей с учетом context.

В зависимости от текущего экрана выбираются часть списка, выдается весь список или пустой массив.

```

/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context}, core, debug) {
  let data = [];
  if (context.layoutid == 'l001') { // Весь список
    data = await core.getListFromTree('users');
    // Только пользователи из папки 'ug002'
  } else if (context.layoutid == 'l021'){
    data = await core.getListFromTree('users', 'ug002');
  }

  return {data};
}

```

**Пример 5.** Иногда, обычно для организации навигации, нужно заполнить список папками из дерева на заданном уровне.

В примере выбираются папки из дерева устройств, вложенные в заданную папку (dg084).

В дереве узлы-папки имеют атрибут children (массив дочерних узлов)

```

/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function({ local, context, source }, core, debug) {
  const data = await core.getTree('devices', 'dg084');

  const folders = [];
  if (data && data.length && data[0].children) {
    data[0].children.forEach(item => {
      if (item.children) folders.push({id:item.id, title:item.title});
    })
  }
  debug(folders);
  return { data:folders };
};

```

## Скрипт заполнения таблицы

Должен вернуть { columns: [массив столбцов], data: [массив данных] }

Элементы массива columns - это объекты определения столбца:

- title: Шапка столбца
- prop: Название поля в массиве data
- width: Ширина столбца
- filter: Возможность фильтровать данные по столбцу <true|false>

Элементы массива data - это объекты, содержащие данные строки: {<название поля>:<значение>,...}

**Пример 1.** Таблица заполняется статическими данными.

```
/**
 * Скрипт для заполнения таблицы
 * Должен вернуть объект, содержащий
 *   columns: массив объектов с описанием столбцов {title, prop, width, filter},
 *   data: массив объектов, каждый элемент – это строка таблицы
 */
module.exports = async function ({local, context}, core, debug) {
  const columns = [
    { title: 'Столбец 1', prop: 'c1', width: 180, filter: true },
    { title: 'Столбец 2', prop: 'c2', width: 280, filter: true },
    { title: 'Столбец 3', prop: 'c3', width: 50, filter: true }
  ];

  const data = [
    { c1: 'Data 1', c2: 'Data 2', c3: 'Data 3' },
    { c1: 'Data 4', c2: 'Data 5', c3: 'Data 6' }
  ];
  return {columns, data};
}
```

**Пример 2.** Таблица заполняется из пользовательской таблицы 'weighing'. Таблица содержит данные взвешиваний машин по нескольким цехам. Номер цеха передается в локальной переменной cех\_number.

```

/**
 * Скрипт для заполнения таблицы
 * Должен вернуть объект, содержащий
 *   columns: массив объектов с описанием столбцов {title, prop, width, filter},
 *   data: массив объектов, каждый элемент – это строка таблицы
 */
module.exports = async function ({local, context}, core, debug) {
  const columns = [
    { title: 'Цех', prop: 'cex', width: 50, filter: true },
    { title: 'Номер машины', prop: 'car', width: 180, filter: false },
    { title: 'Вес, тн', prop: 'weight', width: 100, filter: true },
    { title: 'Время взвешивания', prop: 'datestr', width: 150, filter: true }
  ];

  const data = [];
  const filter = local.cex_number > 0 ? {cex: local.cex_number} : {};
  const docs = await core.getRecords('weighing', filter, {sort:{ts:-1}});
  if (docs) {
    docs.forEach(doc => {
      // Время хранится как timestamp, в таблицу выведем в читаемом виде
      const datestr = core.getDateTimeStr(doc.ts);
      data.push({
        id: doc._id,
        cex: doc.cex,
        car: doc.car,
        weight: doc.weight,
        datestr
      });
    });
  }
  return {columns, data};
}

```

## Скрипт заполнения дерева.

Должен вернуть { data: [массив]}

Элементы массива data - это объекты, представляющие узлы дерева:

- узел, имеющий потомков (в том числе корневой узел): {id, title, children:[]} }
- узел, не имеющий потомков (лист): {id, title}

Массив потомков children может включать вложенные узлы с потомками или листья:

children: [{id, title, children:[]} , {id, title}, ...]

**Пример 1.** Дерево заполняется статическими данными. Имеет один корневой узел root.

```
/**
 * Скрипт для заполнения дерева
 * Должен вернуть объект, содержащий
 *   data: массив узлов дерева
 */
module.exports = async function ({local, context}, core, debug) {
  const data = [
    {
      id: 'root',
      title: 'Главный узел',
      children: [
        {
          id: 'lev1',
          title: 'Узел вложенный',
          children: [
            { id: '1', title: 'Раз' },
            { id: '2', title: 'Два' },
            { id: '3', title: 'Три' }
          ]
        },
        {id: 'leaf1',title: 'Лист'}
      ]
    }
  ]
  return { data };
}
```

Есть более простой способ построения дерева - можно получить любое дерево проекта (дерево устройств, пользователей, сценариев, экранов, ...), используя функцию `getTree`, и просто вернуть результат.

**Пример 2.** Дерево устройств проекта

```
/**
 * Скрипт для заполнения дерева
 * Должен вернуть объект, содержащий
 *   data: массив узлов дерева
 */
module.exports = async function ({local, context}, core, debug) {
  const data = await core.getTree('devices');
  return { data };
}
```



Можно сразу развернуть узел дерева, для этого у узла дерева есть атрибут 'expanded';

```
module.exports = async function ({local, context}, core, debug) {
  const data = await core.getTree('devices');
  data[0].expanded = true;
  return { data };
}
```

Можно взять не все дерево, а только часть.

```
module.exports = async function ({local, context}, core, debug) {
  // Берем только устройства из папки 'dg061'
  const data = await core.getTree('devices', 'dg061');
  return { data };
}
```

Дерево может иметь несколько корней (несколько элементов массива на 1 уровне). Можно взять, например, устройства из трех папок и объединить в дереве

```
module.exports = async function ({local, context}, core, debug) {
  const data1 = await core.getTree('devices', 'dg061');
  const data2 = await core.getTree('devices', 'dg083');
  const data3 = await core.getTree('devices', 'dg085');

  return {
    data: [
      ...expandFirst(data1),
      ...expandFirst(data2),
      ...expandFirst(data3)
    ]
  };

  function expandFirst(data) {
    data[0].expanded = true; // Каждое поддерево будет раскрыто
    return data;
  }
}
```

## Список снимков с ip камеры по датам

В данном примере мы рассмотрим решение задачи просмотра снимков с IP камеры, которые были сделаны по сценарию, например датчика движения и сохранены в памяти сервера.

### Что нам нужно для решения данной задачи?

1. [Настроить плагин CCTV на работу с ip камерой и получения снимков с нее](#)
2. [Написать сценарий для сохранения снимков на сервер](#)
3. Создать три переменные визуализации для динамического обновления списка снимков при изменении даты на календаре **snapListId** (в качестве значения по умолчанию указать Id списка из Аналитики - vlst008), **selectedSnap**, **calendar**.
4. Создать контейнер и добавить три элемента на него: **list**, **image**, **calendar**.
5. Создать список в разделе Аналитика и написать обработчик на получение списка снимков от выбранной даты

```
/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */

const fs = require('fs');

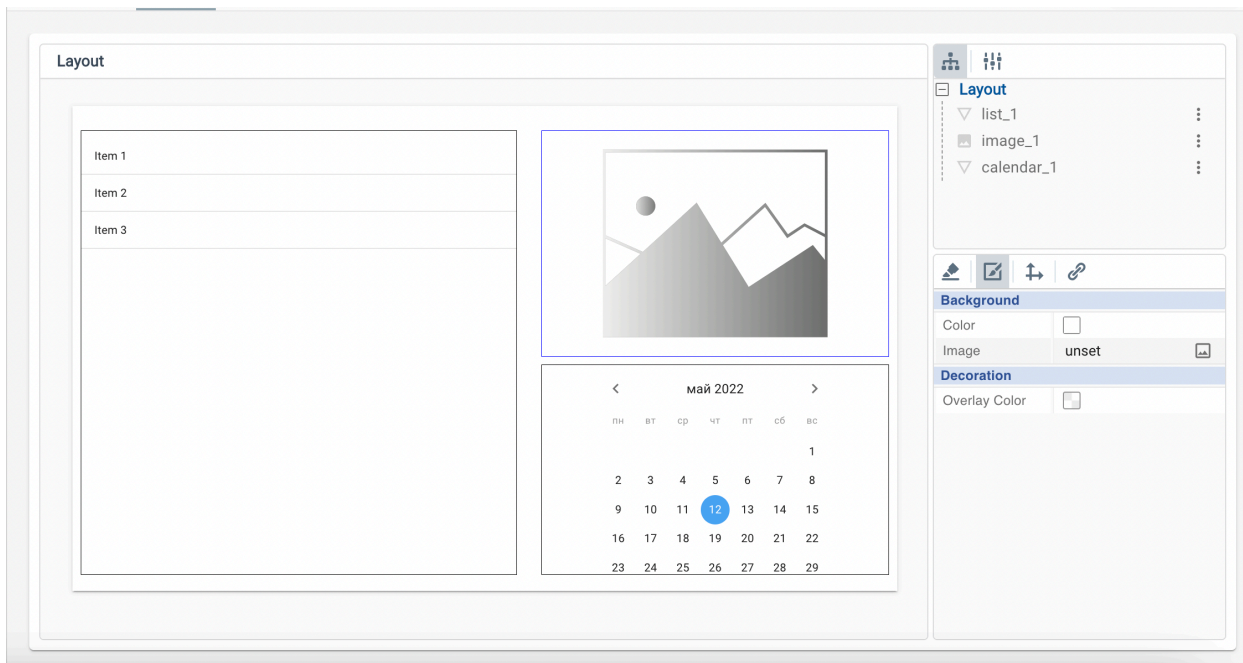
module.exports = async function ({local, context, source}, core, debug) {
  //здесь нужно указать свой путь в зависимости от вашего проекта
  const path = '/var/lib/intrahouse-d/projects/demo_1634279930745/temp/snapsh
  const data = [];
  let fileTs = '';
  //регулярное выражение для извлечения времени из названия файла,
  //например : snap_1645175892414.jpg
  let regexp = /^(?<=\_)(.*?)(?=\.)/g;
  try {
    //чтение папки с файлами, синхронная операция. Получаем список файлов из па
    const files = await fs.promises.readdir(path);
    for (const file of files)
      if (file) {
        fileTs = Number(file.match(regexp));
        let startDate = new Date(Number(local.calendar));
        startDate.setHours(0,0,0,0);
        let endDate = new Date(Number(local.calendar));
        endDate.setHours(23,59,59,999);
        //если дата сохранения файла из названия попадает в выбранный диапазон
        //то добавляем в список
        if (fileTs >= Date.parse(startDate) && fileTs <= Date.parse(endDate)) {
          //В качестве id мы здесь будем использовать путь к файлу,
          //это необходимо для передачи этого пути напрямую в элемент Изображение
          data.push({id: file, title: fileTs });
        }
      }
  } catch (err) {
    debug('ERR: '+err);
  }

  debug('data='+JSON.stringify(data));
  return {data};
}
```

```
module.exports = async function ({local, context, source}, core, debug) {  
  core.updateLocals({snapListId: 'vlst008'});  
}
```

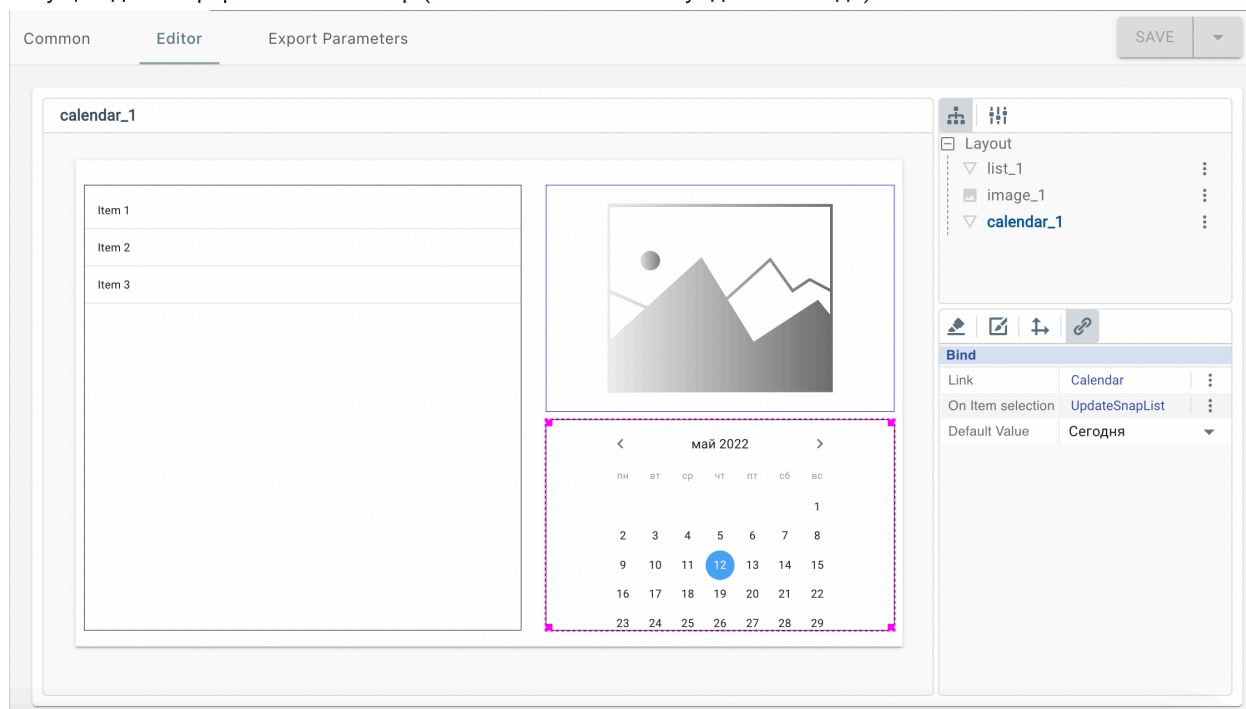
Теперь давайте разберем основную логику работы данного механизма.

1. Мы имеем в контейнере визуализации три наших основных элемента - это **Список**, **Изображение** и **Календарь**.



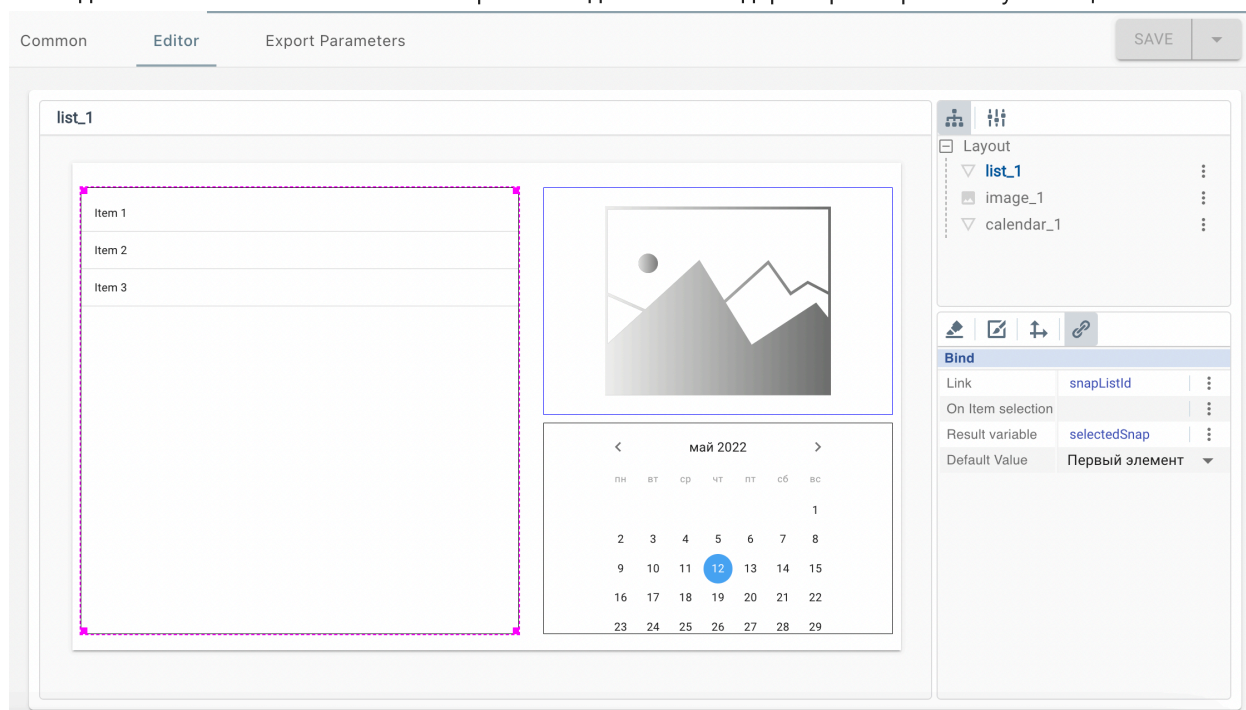
2. При нажатии на календарь мы должны записать в переменную визуализации **calendar** дату, которая была выбрана в календаре. Для этого нам достаточно у календаря во вкладке **Привязки** для **Ссылки** выбрать переменную **calendar**. После этого при выборе даты в переменную **calendar** будет записана

текущая дата в формате timestamp (количество миллисекунд с 1970 года)



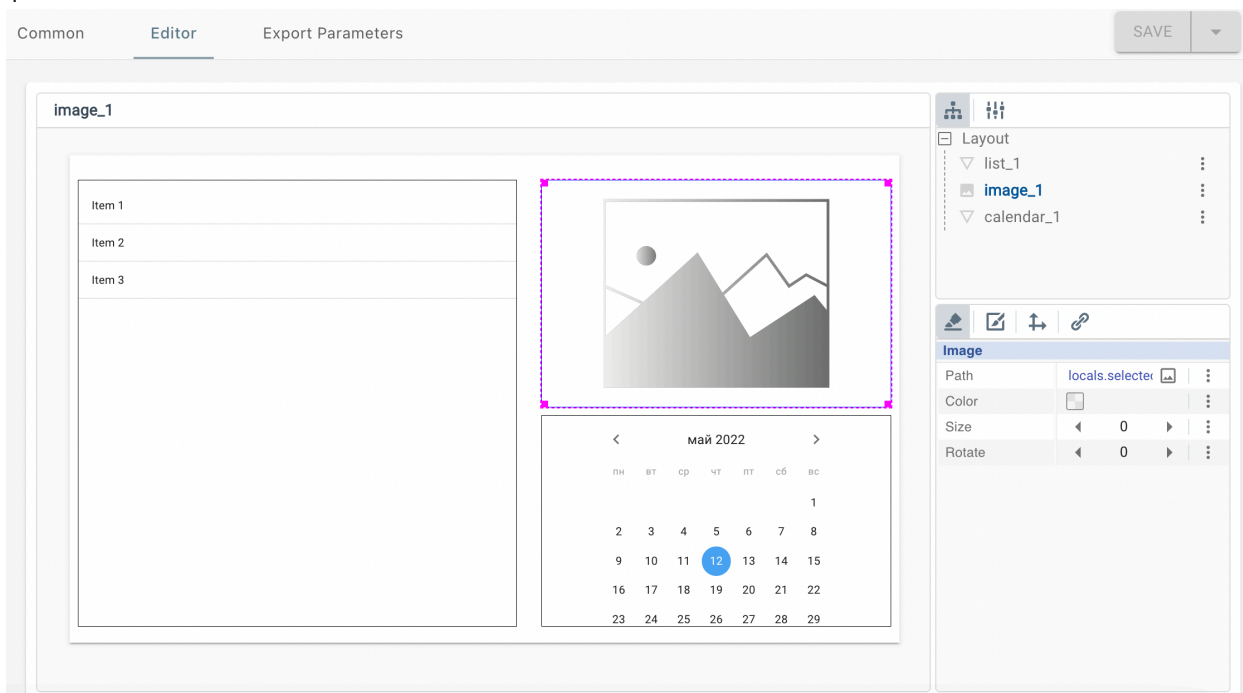
3. После того, как дата присвоена необходимо обновить список. Для этого у календаря во вкладке **Привязки** для **Скрипт при выборе** выбрать скрипт визуализации для обновления списка из пункта 6.

4. Переходим к настройке элемента **Список**. В данном элементе нам необходимо во вкладке **Привязки** для **Ссылки** выбрать переменную **snapListId**. Здесь можно выбрать и прямую ссылку на список из Аналитики, но тогда мы не сможем его обновлять при смене даты в календаре через скрипт визуализации.



5. После того, как список станет обновляться в зависимости от выбранной даты в календарь, нам нужно получить информацию о выбранном элементе в списке. Для этого у списка во вкладке **Привязки** для **Переменная результата** выбрать переменную визуализации **selectedSnap**. Теперь при нажатии на элементы списка мы получаем id этого элемента в переменной **selectedSnap**.

6. Теперь остается отобразить изображение выбранного снимота из списка. Для этого у изображения во вкладке **Изображение** для **Путь** выбрать переменную **selectedSnap**. Обратите внимание, что в качестве Id в списке, который мы сформировали с помощью обработчика в пункте 5, мы как раз использовали имя файла.



# Переменные клиента

**Переменные клиента** используются для построения пользовательских интерфейсов.

## Параметры для переменных клиента

При создании **Переменной клиента** вводятся:

- Имя переменной (допустимы латинские буквы, цифры и знак "\_")
- Название - текстовое описание
- Формула расчета - используется в случае необходимости инициализации переменной с текущими значениями, например времени **Date.now()**
- Значение по умолчанию.

**Переменные клиента** внутри сеанса каждого пользователя имеют свое значение. Это может быть индекс выбранной строки в списке, дата на календаре, ID выбранного устройства, графика и т.д.

**Переменные клиента** используются только для целей визуализации, и никогда - в сценариях и обработчиках устройств.

## Привязка к свойствам визуального элемента

Можно привязать переменную к любому свойству визуального элемента. Используя функции, можно изменять визуальные представления.

Пример 1. Подсветка кнопок в зависимости от текущего выбора.

- Поставить на экран или контейнер несколько кнопок
- Создать переменную "selected\_button"
- На кнопках (например при Одиночном клике) выполнить привязку:
  - **Установить значение**, выбрать переменную "selected\_button" и присвоить номер кнопки
- На кнопках для свойства **backgroundColor** выполнить привязку к той же переменной "selected\_button" В качестве функции прописать:
  - Для первой

```
return inData == 1 ? "rgba(255,255,255,0.2)" : 'TRANSPARENT';
```

- Для второй

```
return inData == 2 ? "rgba(255,255,255,0.2)" : 'TRANSPARENT';
```

## Использование в скриптах визуализации

При вызове скрипта визуализации ему передаются все **Переменные клиента** с текущими значениями.

## Пример 2. Создание меню навигации

- Поставить на экран или контейнер несколько кнопок
- Создать переменную "menu" (Выбранный элемент меню)
- На кнопках (например при Одиночном клике) выполнить привязки:
  1. **Установить значение**, выбрать переменную "menu" и присвоить номер кнопки
  2. Вызвать **Скрипт визуализации**, который в зависимости от значения "menu" выполнит нужные переходы или команды

```
module.exports = async function ({local, context, source}, core, debug)

  // по умолчанию – переход на стартовый экран пользователя
  let layoutid = context.start_layoutid);
  if (local.menu == 1) {
    layoutid = 'l021';
  } else if (local.menu == 2) {
    // Параллельно можно что-то сделать, например, выдать звуковой сигнал
    core.playSound('ding.wav');
    layoutid = 'l042';
  }
  core.gotoLayout(layoutid);
}
```

Скрипт в свою очередь может изменить любую локальную переменную. Для этого используется команда скрипта **updateLocals**

## Пример 3. Вывод данных на экран из скрипта

- Поставить на экран или контейнер несколько кнопок и элемент **Text**
- Создать переменную "fill\_text" и привязать ее к свойству **Значение** Элемента **Text**
- Создать переменную "select\_text"
- На кнопках (например при Одиночном клике) выполнить привязки:
  1. **Установить значение**, выбрать переменную "select\_text" и присвоить номер кнопки
  2. Вызвать **Скрипт визуализации**, который в зависимости от значения "select\_text" выведет в поле **Text** различные тексты



```

module.exports = async function ({local, context, source}, core, debug) {
  let text = '';
  if (local.selected_text == 1) {
    text = 'Раз';
  } else if (local.selected_text == 2) {
    text = 'Два';
  }
  core.updateLocals({fill_text: text});
  // Здесь для простоты просто выдаются константные строки в один элемент
  // Можно выбирать данные из таблиц или формировать их используя данные устр
  // Можно вернуть сразу несколько переменных:
  // core.updateLocals({text1: text, text2:'xxx', myvar1:42});
}

```

## Использование для динамических привязок

Начиная с версии 5.9 **Переменные клиента** используются для динамических привязок визуальных компонентов. То есть там, где ранее использовались привязки типа "Любое (устройство, график, ...)", нужно использовать переменную.

По умолчанию в проекте существует переменная "ID устройства" (dev\_id).

Стандартные диалоги используют эту переменную для ссылки на произвольное устройство.

При использовании (вызове) диалога нужно выбрать переменную "ID устройства" и присвоить ID реального устройства.

Переменную dev\_id можно использовать и в других случаях. Но, конечно, можно создавать свои переменные, если это удобно для проекта.

[Динамическая привязка устройств](#) позволяет динамически переключать контекст.

## Динамическая привязка устройств

Динамическая привязка устройств позволяет не привязывать жестко атрибут элемента визуализации или переменную шаблона к свойству конкретного устройства. Привязка делается к [Переменной клиента](#), которая должна содержать идентификатор устройства.

### Механизм переключения

Например, на экране размещены [Список](#) или [Дерево](#) устройств, содержащие id устройств

При клике нужно отобразить мнемосхему, содержащую информацию и элементы управления, относящиеся к выбранному устройству.

Не будем делать мнемосхему для каждого устройства.

В простейшем случае, если устройства однотипные, достаточно:

- создать переменную клиента, например, *mydev\_id* (можно использовать стандартную переменную *dev\_id*)
- сделать одну мнемосхему с привязками атрибутов элементов или переменных шаблонов к *dev\_id*
- указать эту же переменную *dev\_id* как **Переменную результата** для списка (дерева)

Что происходит в результате?

- При клике в списке ID устройства записывается в **Переменную результата** *dev\_id*
- При изменении значения *dev\_id* изменяются все элементы, связанные с переменной *dev\_id*

Для решения этой задачи нам даже не пришлось создавать скрипт визуализации, так как список сам меняет *dev\_id*.

Другой вариант, где также можно обойтись без скрипта - менять контекст при клике на кнопках:

- разместить несколько кнопок
- Для каждой кнопки, например, по Одиночному клику
  - выбрать действие **Установить значение**
  - в дереве выбрать переменную *dev\_id*
  - вернуть `return 'd0042'` (Указать разные ID устройств для разных кнопок)

Если же нужно отображать несколько устройств одновременно или выбор делается более сложным способом, можно написать скрипт визуализации, который будет изменять значения переменной (-ым).

Для переключения контекста достаточно изменять значения переменных (с сервера или прямо на клиенте)

Движок визуализации отслеживает изменения привязанных переменных и обновляет экран, запрашивая данные для нового контекста

### Механизм привязки

Когда идет привязка к конкретному устройству, выбирается устройство и свойство.

Пример: Вывести в текстовое поле значение температуры TEMP1 (свойство 'value').

Выбираем конкретное устройство TEMP1, а затем свойство 'value'. Привязка выглядит как: TEMP1.value.

В этом случае **return inData** вернет то что надо - значение свойства 'value'.

Попробуем привязать текстовое поле к переменной, содержащей ID устройства.

В текстовом поле увидим что-то типа 'd0212', это просто ID. Очевидно, это не то что нужно. Мы должны дополнительно указать, как по этому ID получать данные.

Для этого используется специальная функция **deviceValue**, которой передается ID устройства (inData) и имя свойства.

### **deviceValue(inData, <свойство устройства>)**

**return deviceValue(inData, 'value')** - будет отображать значение свойства 'value' выбранного устройства.

**deviceValue** применяется и в случае привязки к интерактивным элементам - Input, Slider,...

. В этом случае функция будет не только отображать, но и отправлять команду на запись этого свойства.

### **deviceCommand(inData, <свойство устройства>)**

**deviceCommand** с двумя аргументами используется для выполнения **Команды устройства**

Пример: `deviceCommand(inData, 'toggle')`

### **deviceCommand(inData, <свойство устройства>, <значение или выражение> )**

**deviceCommand** с тремя аргументами используется для выполнения операции **Установить значение**

При выполнении операции **Установить значение** для конкретного устройства привязка выглядит так:

`TEMP1.setpoint`

В функции прописывается значение, которое хотим присвоить, например:

- `return 22; // присваиваем константу`

При выполнении команды через переменную нужно использовать функцию:

- `deviceCommand(inData, 'setpoint', 22)`

Более сложный вариант - присвоить не константу, а рассчитать новое на базе текущего значения

- `return inData+1; // TEMP1.setpoint + 1`


В случае переменной мы не можем использовать inData в третьем аргументе.

Работающий вариант - строка с указанием операции и операнда:

- `deviceCommand(inData, 'setpoint', '+1')`

Можно использовать вычитание, деление, умножение, например:

- `deviceCommand(inData, 'setpoint', '-5')`
- `deviceCommand(inData, 'setpoint', '*1.5')`
- `deviceCommand(inData, 'setpoint', '/2')`

	Если выбрано устройство и свойство 'setpoint'	Если используется переменная
Присвоить константу (22)	<code>return 22;</code>	<code>deviceCommand(inData, 'setpoint', 22)</code>
Добавить 1 к текущему значению	<code>return inData+1;</code>	<code>deviceCommand(inData, 'setpoint', '+1')</code>
		

Обратите внимание, при использовании `deviceCommand` **return** не нужен. Нам не нужно менять ID устройства.

# Сценарии

## Для чего нужны сценарии

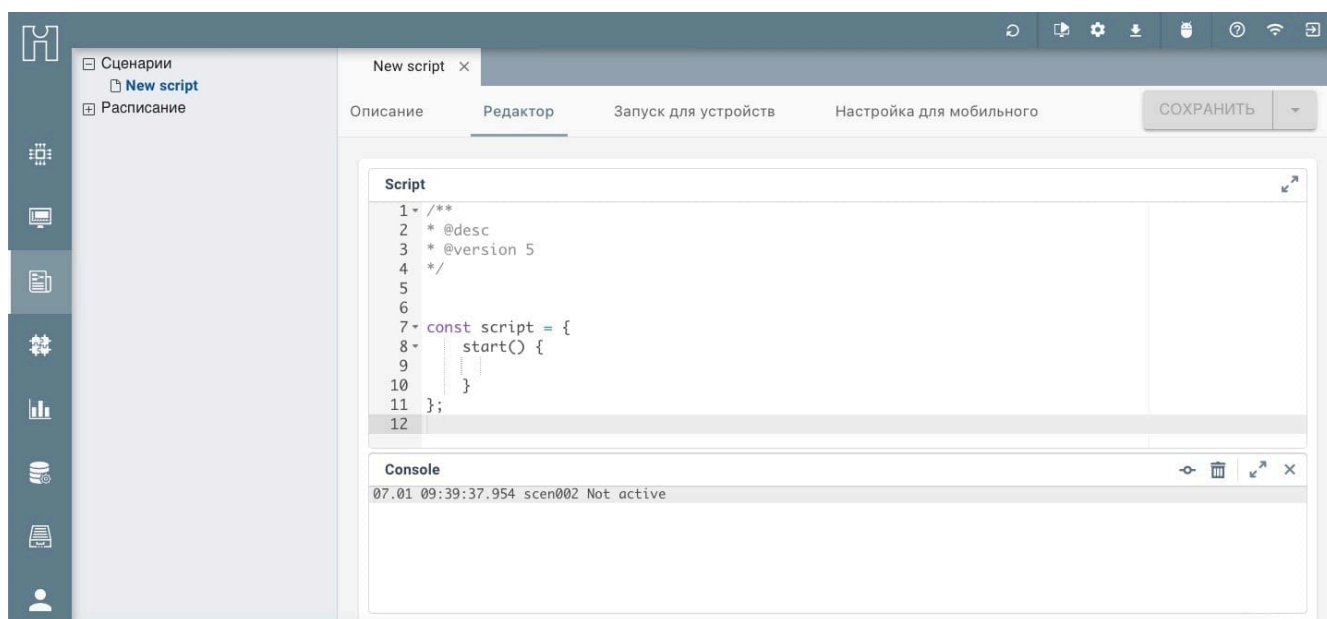
Сценарии системы позволяют анализировать состояние и выполнять действия с несколькими устройствами, учитывая множество критериев, работать с таймерами, отслеживать события, писать в журналы и БД, передавать сообщения по каналам связи.

Сценарии могут иметь богатую логику, это не просто When-Then, так как пишутся на полноценном JavaScript.

Для каждого проекта можно создать свои сценарии или загрузить уже имеющиеся.

## Как создать сценарий

Сценарий создается интерактивно в редакторе кода сценариев среды разработки Project Manager (PM). При редактировании в PM сценарий сразу обновляется в системе, никаких перезагрузок не требуется. Если сценарий имеет ошибки - он будет заблокирован. Если ошибок нет, сценарий переходит в состояние **Not active**. Это означает, что он готов к работе:



## Как запустить сценарий

Сценарий может быть запущен разными способами:

- по событиям устройств
- по расписанию
- интерактивно из пользовательского интерфейса (привязать запуск сценария к кнопке)
- командой API
- интерактивно из PM - при нажатии кнопки Start

Отладчик под редактором кода позволяет отслеживать все этапы выполнения сценария.

## Как хранятся сценарии.

Фактически каждый сценарий — это отдельный файл .js.

Сценарии хранятся в папке проекта: /projects/<имя проекта>/scenes/script.

При создании нового сценария генерируется его ID (например, scen002) — исходный сценарий будет храниться в файле scen002.js.

При необходимости можно напрямую загрузить или редактировать файл сценария в папке проекта scenes/script. Такие изменения будут доступны после перезагрузки сервера.

# Структура сценария

Начинается сценарий обычно с многострочного комментария.

В опциональной декларативной части объявляются переменные-устройства и условия запуска.

Далее идет обязательная часть сценария - скрипт выполнения.

Скрипт - это объект, который имеет методы (функции, которые будут выполняться "от имени" скрипта).

Обязательной является функция **start**. Она выполняется первой при запуске сценария.

Это может быть единственная функция скрипта, как в примере ниже.

```
/**
 * Это многострочный комментарий
 * @name Свет по датчику движения
 * @desc Очень примитивный сценарий
 * - свет включается, если есть движение; если прекратилось - сразу выключает
 *
 * @version 5
 */

// Объявление переменных-устройств
const lamp = Device("LAMP1");
const motion = Device("MOTION1");

// Условия запуска.
// Будет запускаться при каждом изменении состояния датчика движения
startOnChange(motion.state);

// Скрипт выполнения
const script = {

  start() {
    if (motion.state == 1) {
      lamp.on();
    } else {
      lamp.off();
    }
  }
}
```

В сценарии может и не быть описания переменных и/или условий запуска.

Пример такого сценария:

```

/**
 * @name Продувка
 * @desc Запускается по расписанию или с кнопки
 * – Включает вентиляторы продувки, присылает сообщение, через 10 мин выключ
 *
 * @version 5
 */

// Скрипт выполнения
const script = {

  start() {
    this.doAll({tag:"продувка"}, "on");
    this.info("telegram", "admin", "Включены вентиляторы продувки");
    this.startTimer("T1", 600, "doOff");
  },

  doOff() {
    this.doAll({tag:"продувка"}, "off");
    this.info("telegram", "admin", "Выключены вентиляторы продувки");
  }
}

```

Кроме функции **start** здесь есть функция **doOff**, которая вызывается по таймеру

## Объявление устройств

Для того, чтобы начать взаимодействовать с устройствами внутри сценария, необходимо их объявить.

Существует три типа объявления:

1. В декларативной части указываем Имя устройства

```
const boiler = Device("Boiler_001");
```

2. Для мультисценария в декларативной части можно написать любую строку, которая определяет это устройство

```
const boiler = Device("Бойлер");
```



```
const script = {
  start() {
    const pump = this.getDevice('PUMP_001');
    pump.on();
  }
}
```

## Условия запуска по событиям устройств (триггеры сценария)

Для определения условий запуска по событиям устройств используется функция **startOnChange()**.

Если сценарий планируется запускать интерактивно или по расписанию, функция **startOnChange()** не нужна.

Параметр функции определяет, события каких устройств должны запускать сценарий

Можно указать:

1. Имя переменной устройства. Сценарий будет запускаться при изменении любого свойства.

```
const motion = Device("MOTION1");
startOnChange(motion);
```

2. Имя свойства переменной устройства. Сценарий будет запускаться при изменении конкретного свойства.

```
const motion = Device("MOTION1");
startOnChange(motion.state);
```

3. Массив имен свойств и/или имен переменных для запуска по событиям нескольких устройств.

```
const lamp = Device("LAMP1");
const motion = Device("MOTION1");
startOnChange([motion.state, lamp]);
```

Сценарий будет запускаться при изменении свойства state датчика и при изменении любого свойства лампы (например, лампа имеет свойства state и auto)

4. Можно использовать не только переменные устройств, но и глобальные переменные (globals).

Глобальные переменные объявлять не надо

```
const motion = Device("MOTION1");
startOnChange([motion.state, globals.mode]);
```

Можно узнать, по какому триггеру запустился сценарий. Внутри скрипта эта информация доступна как массив **this.triggers**

```
const boiler = Device("sc_boiler");
const temp = Device("TEMP1");
startOnChange([temp, boiler, globals.mode]);

const script = {
  start() {
    // Проверим, что среди триггеров есть boiler.state
    if (this.triggers.includes('boiler.state')) {
      this.log(
        boiler.state
        ? 'Сейчас бойлер включился'
        : 'Сейчас бойлер выключился'
      );
    }
    // Выводим все триггеры – может быть не один элемент
    this.log('Triggers='+this.triggers.join(', '))
  }
}
```

## Скрипт выполнения

Скрипт выполнения представляет из себя объект `const script = {...}`

Он создается на базе встроенного объекта `script` и наследует встроенные методы, которые позволяют выполнять различные действия, работать с таймерами, отслеживать события устройств, писать в журнал и БД, передавать сообщения по каналам связи. Раздел [Встроенные методы](#) содержит описание этих методов и примеры использования.

## Условные функции **check** и **boot**

Каждый скрипт **должен** иметь функцию с именем **start()** - это точка запуска сценария.

Кроме обязательной функции **start()**, скрипт может содержать функцию **check()**, что позволяет запускать скрипт условно.

Работает это так:

- Если функция **check()** присутствует в коде скрипта, то она запускается первой (это как бы предзапуск сценария).
- Функция **check()** должна вернуть **true**, тогда сценарий считается запущенным. При этом фиксируется момент запуска и управление передается функции **start()**.
- В противном случае сценарий не запускается.

Конечно, условие можно проверять и внутри функции **start()** перед запуском основной логики.

Отличие заключается в том, что при выходе из функции **check()** с результатом **false** момент запуска не фиксируется и число запусков не инкрементируется.

Например:

```
const boiler = Device("sc_boiler");
const temp = Device("sc_temperatura");

startOnChange([temp.value]);

const script = {
  // Запустить только если отопительный сезон и бойлер находится в режиме а
  check() {
    return boiler.auto && globals.heatingSeason;
  },

  start() {
    // Основная логика скрипта
    if (temp.value < 10 && boiler.state == 0) boiler.on();
    if (temp.value > 10 && boiler.state == 1) boiler.off();
  }
}
```

Также есть функция **boot()**, которая предусмотрена для запуска сценария на старте сервера.

Можно использовать эту возможность, например, для установки параметров.

Если функция **boot()** возвращает **true**, управление передается функции **start()**. В противном случае сценарий не запускается.

```
const script = {  
  boot() {  
    return true;  
  },  
  
  start() {  
    const dev = this.getDevice('AI_1');  
    dev.assign('setPoint', 22);  
    this.mainlog('Сценарий был запущен на старте.')  
  }  
};
```

## Пользовательские функции

Кроме функций входа **start()**, **check()**, **boot()** можно добавить пользовательские функции с произвольными именами.

Созданные пользователем функции добавляются к встроенным и становятся методами объекта **script**.

Например:

Имя пользовательской функции может содержать только латинские буквы, цифры, знак подчеркивания.

Не следует использовать имена встроенных методов (info(), doAll() и т.д. - см. [Встроенные методы](#)).

Вызов пользовательской функции внутри скрипта выполняется двумя способами:

1. Через **this**

Такая функция содержит логический блок и вызывается из любого места скрипта напрямую.

В этом случае функция может иметь любое количество аргументов на усмотрение разработчика.

2. По имени функции как строки для функций обратного вызова

```
/**
 * @name Пример использования функций пользователя
 * @version 5
 */

const script = {

  start() {
    this.doAll({tag:"продувка"}, "on");
    // sayAbout – функция пользователя, два аргумента
    this.sayAbout("admin", "Включена продувка");
    // doOff – функция обратного вызова, вызовется через 600 сек
    this.startTimer("T1", 600, "doOff");
  },

  doOff() {
    this.doAll({tag:"продувка"}, "off");
    this.sayAbout("admin", "Выключена продувка");
  },

  sayAbout(target, text) {
    this.info("telegram", target, text);
    this.info("email", target, text);
    this.log(text);
  }
}
```

Функции обратного вызова вызываются по таймеру/по событию слушателя после выполнения команды плагина или системной команды.

Их аргументы определены заранее

- Функции, вызываемые по таймеру (startTimer, restartTimer), не имеют аргументов.

- Функция обратного вызова для execOS имеет на входе строку stdout - результат выполнения системной команды

```
const script = {
  start() {
    this.execOS(`vcgencmd measure_temp`, 'processTemp');
  },

  processTemp(stdout) {
    //stdout = "temp=44.0'C"
    if (stdout) {
      const arr = stdout.split('=');
      if (arr.length == 2) {
        const temp = parseInt(arr[1]);
        this.log('Температура процессора '+temp);
      }
    }
  }
}
```

## Пользовательские переменные

Переменные можно объявлять внутри функций скрипта по правилам JavaScript (let, const, var):

```
const script = {
  start() {
    const x = Math.round((temp1.value+temp2.value)/2);
    this.log("Средняя температура "+x+"C");
  }
}
```

Область видимости такой переменной - блок, в котором она объявлена (в данном случае функция).

Если нужна переменная, которая видима в рамках всего сценария, можно объявить переменную как свойство объекта script.

Такая переменная будет сохранять данные между запусками сценария.

Обращаться к ней нужно через **this**

```
const script = {
  count:0, // Эта переменная – свойство объекта script

  start() {
    this.count += 1; // Доступно в любом месте скрипта как this.count
    this.log('Счетчик равен '+ this.count);
  }
}
```

## Работа с устройствами

Внутри функций скрипта можно использовать свойства и методы устройств, объявленных для сценария. API для работы с устройством одинаков для использования в обработчиках и сценариях.

Любое свойство устройства доступно для чтения и использования в условных выражениях через точку (lamp.state).

Команда устройства вызывается как функция устройства (lamp.on())

Также доступны все глобальные переменные (**globals**), их объявлять не нужно.

```
const lamp = Device("LAMP1");
const script = {
  start() {
    globals.guard = 1;
    lamp.off();
    // Сообщение появится в журнале устройства LAMP1
    lamp.log('Отключен свет после постановки на охрану')
  }
}
```

[Более подробно о работе с устройствами из сценария >](#)

## Добавление новых свойств в устройство из сценария

Иногда бывает удобно создать новые свойства устройства прямо из сценария.

Конечно, это будут виртуальные свойства, привязать их к каналам нельзя.

Обычно это параметры алгоритма автоматизации.

Например, добавим параметры для светильников, для которых есть мультисценарий освещения по датчикам движения.

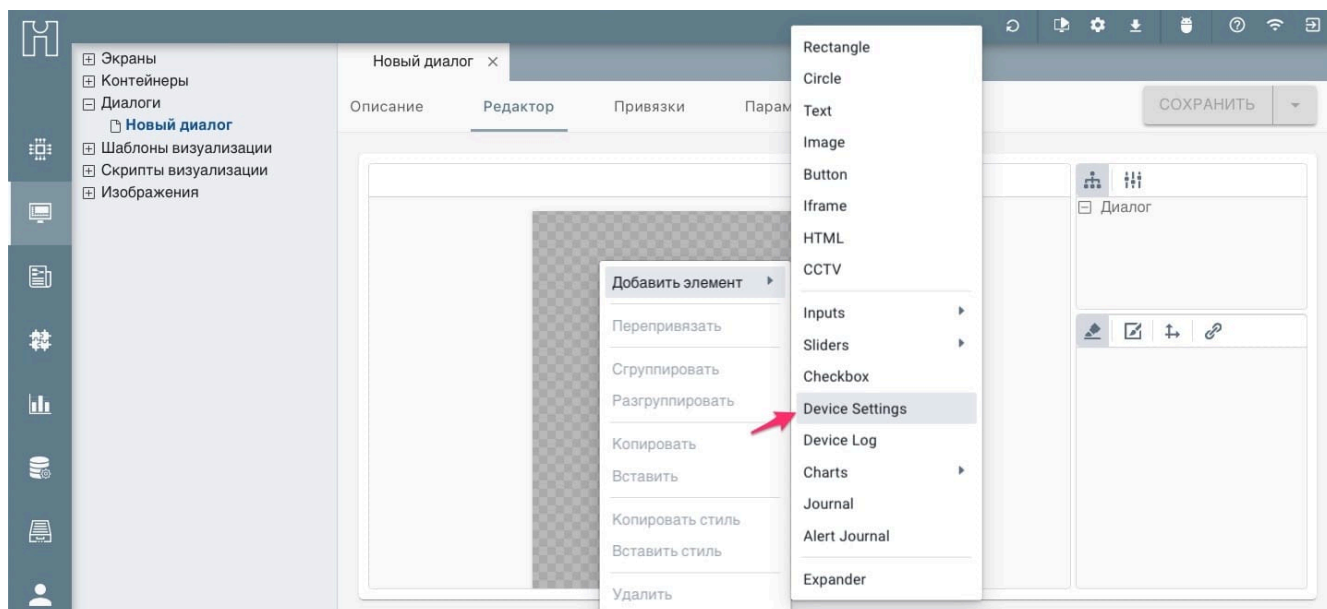
```

const lamp = Device(
  "Светильник",
  [
    {
      name: "timeOff",
      note: "Время работы без движения",
      type: "time",
      val: 30
    },
    {
      name: "auto",
      note: "Авто",
      type: "cb",
      val: 0
    },
    {
      name: "reauto",
      note: "Флаг reauto, вручную не взводить",
      type: "number",
      val: 0,
      hide: 1
    },
    {
      name: "reautoTimeAfterOn",
      note: "Возврат в Авто после включения",
      type: "time",
      val: 15
    },
    {
      name: "reautoTimeAfterOff",
      note: "Возврат в Авто после выключения",
      type: "time",
      val: 25
    }
  ]
);

```

Плюсы такого подхода: - свойства появляются только у тех устройств, которые работают по сценарию  
 Если сценарий удаляется, свойства у устройства пропадают - данные свойства выводятся на визуализацию с помощью специального визуального компонента **Device Settings**:





# Цикл работы сценария

Следует помнить, что сценарий не представляет собой модуль, постоянно работающий и самостоятельно контролирующий свое выполнение. Если действительно требуется сценарий такого типа, нужно создать плагин — модуль, запускаемый в отдельном потоке.

Проект автоматизации может содержать сотни и даже тысячи сценариев, поэтому вариант с постоянно запущенной массой сценариев в отдельных потоках будет очень затратным.

К счастью, обычные сценарии, позволяющие решить большинство задач автоматизации, этого и не требуют.

В отличие от плагина, все сценарии выполняются под управлением движка сценариев в специально выделенном, но общем потоке.

Такой подход накладывает некоторые ограничения:

- каждая функция скрипта должна выполняться синхронно и быстро;
- нельзя использовать `setTimeout`, `setInterval`, `process.nextTick`, `async/await` и другие асинхронные возможности.

Организацию асинхронной работы берет на себя движок сценариев.

Благодаря событийной модели, большая часть сценариев 99,9% времени находятся в неактивном состоянии и ресурсы не потребляют. Вместе с тем, реакция при необходимости происходит почти мгновенно.

Как же все это работает? Рассмотрим цикл работы сценария.

## Запуск

Сценарий может быть запущен разными способами:

- интерактивно
- по расписанию
- при изменении значений свойств устройств и/или глобальных переменных системы
- ...

Независимо от способа запуска, при старте выполняется функция **start()** сценария.

Если функция **start()** содержит простую последовательность действий, сценарий отработает и сразу завершится (станет не активным).

При следующем запуске он опять начнется со **start()**.

Для многих случаев это подходит: событие - реакция - возможно, информирование. И при повторении все по новой.

Но если нужно отслеживать одновременно несколько событий, их взаимосвязь во времени?

В таких случаях сценарий может не завершаться, а продолжить работу, отслеживая и реагируя на ситуацию изнутри.

## Активное состояние

Рассмотрим простую задачу:

Есть несколько датчиков протечки и кран перекрытия воды.

При сработке любого из датчиков кран перекрыть.

При нормализации ситуации (нет сработок в течение x минут) кран открыть.

Предлагаемая реализация:

- Сценарий запускается при возникновении протечки на любом из датчиков
- Сценарий остается активным, пока протечка не ликвидирована и кран не открыт

При протечке на любом датчике:

- отправляем сообщение, перекрываем кран, следим за датчиками
- перекрываем кран
- следим за датчиками
- если протечка ликвидирована - взводим таймер
- если протечка возобновилась - сбрасываем таймер
- по таймеру - открываем кран и завершаем сценарий
- учесть также, что кран могли открыть вручную

Почему удобно, что сценарий остается активным?

- Проблема (протечка) уже определена при запуске.
- Повторный запуск с другими датчиками не поможет, а только помешает.
- В активном состоянии сценарий имеет возможность слушать события устройств и запускать таймеры.  
Можно последовательно строить алгоритм действий, исходя из realtime состояния
- Не нужны глобальные флаги для синхронизации событий, вся логика сосредоточена в одном месте



Таким образом, сценарий может находиться в активном состоянии, если взводит таймеры и/или устанавливает слушателя событий.

Такой сценарий не будет срабатывать повторно, так как он уже запущен.

Если запустить отладчик, можно увидеть, что не дает завершиться сценарию:

```
03.01 18:33:50.431 scen003 Working:
  Listener: Leak1.state
  Listener: Leak2.state
  Listener: Leak3.state
  Listener: Water_Rele.state
  Active timer "T1" on 03.01 18:34:07.518 (1641224047518)
```

## Завершение

Сценарий завершается автоматически, если у него нет слушателей, активных таймеров и ожидающих функций обратного вызова.

- слушатель удаляется командой **this.removeListener(...)**
- активный таймер удаляется командой **this.stopTimer(..)**

Если таймер досчитал, то он перестает быть активным

Для завершения сценария в любой точке можно выполнить команду **this.exit()**

При этом удаляются все его таймеры и слушатели.

При следующем запуске опять выполнится **start()**.

# Мультисценарии

## Что такое мультисценарий

Мультисценарий — это вариант сценария, при котором в скрипте используются не реальные устройства системы, а устройства-шаблоны.

Такой подход имеет много плюсов:

- Можно использовать один и тот же сценарий много раз, задавая наборы устройств
- При этом код скрипта загружается 1 раз и не нуждается в изменении при добавлении новых наборов
- Можно загрузить (выгрузить) сценарий и использовать его в других проектах
- Можно использовать сценарии, написанные коллегами, не методом "copy-paste, и здесь подставь свой идентификатор хуз56758765-tyutyu", а просто бери и пользуйся

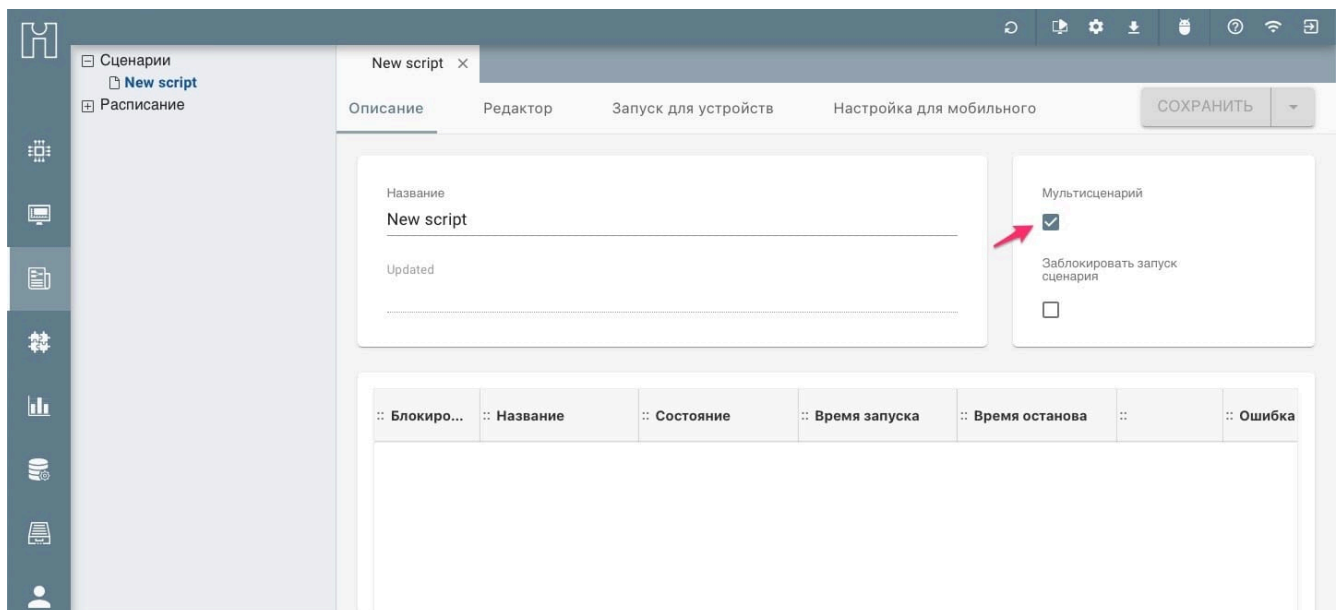
Подключение новых однотипных сценариев становится тривиальной задачей - нужно просто добавить набор устройств.

Еще один существенный плюс — превращение из мультисценария и обратно не требует практически никаких усилий.

Например, написан сценарий для конкретных устройств: датчик движения и светильник. Отлажен, прекрасно работает:

```
const motion = Device("MOTION1");
const lamp = Device("LAMP1");
```

Ставим галочку Мультисценарий:



Меняем названия:

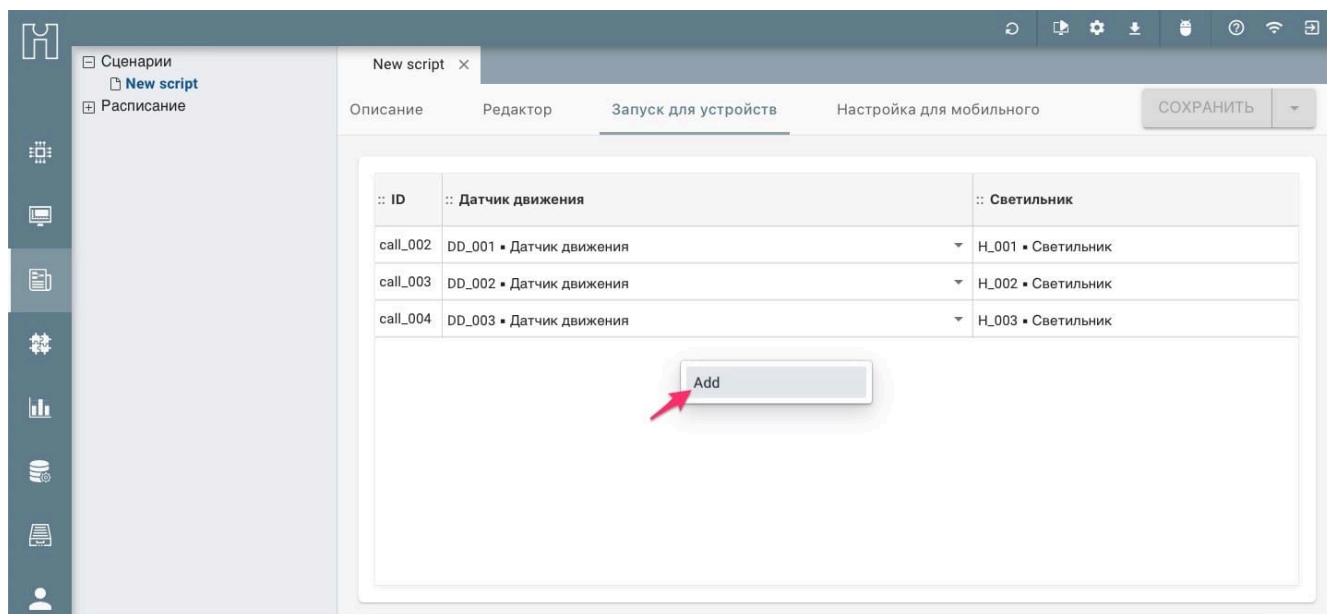
```
const motion = Device("Датчик движения");
const lamp = Device("Светильник");
```

И начинаем заполнять наборы устройств.

Возможна обратная задача: есть мультисценарий, для 20 случаев подходит, но 21 с фокусами, хочется допилить. Не проблема — копируем, убираем галочку Мультисценарий, в декларативной части ставим конкретные устройства и добавляем уникальный код.

## Наборы устройств

Если на первой вкладке сценария поставлена галочка Мультисценарий, становится доступна таблица, в которой столько столбцов, сколько вы определили устройств в сценарии.



Правой кнопкой мыши добавляем новый набор (новую строку).

Добавится новый экземпляр сценария, который будет работать с устройствами из этого набора.

Если сценарий вводит новые свойства для устройства, эти свойства появятся у добавленных устройств.

Как только вы сохранили набор, экземпляр сценария готов к работе.

## Команда плагина

### Отправка команды плагину

Отправить прямую команду плагину из сценария.

- **this.pluginCommand**( {unit:<ID плагина>, command:<команда>}, <Функция обратного вызова-опционально>)
  - **unit** - ID экземпляра плагина
  - **command** - команда плагина - строка или объект

Штатно команды плагину отправляются через механизм каналов.

Не все плагины принимают прямые команды. Формат команды зависит от плагина.

#### Пример 1:

Просто отправить команду

```
this.pluginCommand({
  unit: 'megad1',
  command: '/sec/?pt=' + channel + '&ws=' + cmd + '&chip=' + chip
});
```

#### Пример 2:

Отправить команду, получить ответ и разобрать его:



```

const script = {
  start() {
    // Передать запрос на чтение, ответ придет в функцию  getResponse
    this.pluginCommand(
    {
      unit: 'megad1',
      command: {
        url: '/sec/?pt=7&cmd=get',
        onResponse: 'raw'
      }
    },
    'getResponse'
  );
  this.startTimer('T1', 1, 'onTimer'); // Следующий запрос
},

getResponse(body) {
  // body = 0FF/2
  if (!body) return;

  const arr = body.split('/');
  if (arr.length<2) {
    this.log('Ожидается значение счетчика после символа "/"');
    return; // Ошибка, значение не получено
  }
  meter.assign('value', arr[1]);
}
}

```

## HTTP

### Отправка http(s) запросов

Сценарий может выполнить http запросы GET/POST из сценария.

Каждая из функций имеет разновидность для выполнения https запроса. Результат будет передан функции обратного вызова.

### Отправка GET запроса

Добавлено: v5.9.46

- **this.httpGet** (<строка запроса uri>, <Объект опций opt>, <Функция обратного вызова>)
- **this.httpsGet** (<строка запроса uri>, <Объект опций opt>, <Функция обратного вызова>)
  - **uri** - строка запроса (Uniform Resource Identifier)
  - **opt** - объект, содержащий опции запроса, может быть пустым объектом

### Опции запроса

Используются для добавления нужных заголовков и/или параметров запроса, например, время таймаута. Заголовки передаются во вложенном объекте `opt.headers`:

```
opt = {headers: {'Cookie': ['foo=bar', 'bar=baz']}}
```

Для GET запроса никакие заголовки автоматически не добавляются.

```
const script = {
  start() {
    const opt = {}; // Можно оставить пустой объект
    // Изменить таймаут – по умолчанию timeout составляет 10000 (10 сек)
    opt.timeout = 3000;
    this.httpGet('127.0.0.1:8088/restapi/products', opt, 'onGet');
  },

  onGet(error, result) {
    if (error) {
      this.log('ERROR: '+error);
    } else {
      this.log('OK. result: '+result);
    }
  }
}
```

### Отправка POST запроса

Добавлено: v5.9.47

- **this.httpPost**(<строка запроса uri>, <Объект опций opt>, <данные data>, <Функция обратного вызова>)
- **this.httpsPost**(<строка запроса uri>, <Объект опций opt>, <данные data>, <Функция обратного вызова>)
  - **uri** - строка запроса (Uniform Resource Identifier)
  - **opt** - объект, содержащий опции запроса
  - **data** - данные для передачи, может быть строка, буфер или объект.

### Опции запроса

Если data - объект, автоматически будет выполнена сериализация и установлены заголовки:

- 'Content-Type' = 'application/json';
- 'Content-Length' = длина буфера

Если используется другой тип данных, заголовки 'Content-Type' и 'Content-Length' автоматически не формируются.

Если 'Content-Length' не передавать, будет использован метод передачи **HTTP Chunked** и добавлен заголовок 'Transfer-Encoding='chunked'.

```
const script = {
  start() {
    const data = 'Это сообщение будет отправлено методом POST';
    const opt = {
      headers: {
        'Content-Type': 'text/plain',
        'Content-Length': Buffer.byteLength(data)
      }
    };
    this.httpPost('127.0.0.1:8088/restapi/send', opt, data, 'onPost');
  },

  onPost(error, result) {
    if (error) {
      this.log('ERROR: '+error);
    } else {
      this.log('OK. result: '+result);
    }
  }
}
```

Пример 2. Отправляем объект, заголовки можно не устанавливать.

## Информирование

Отправка сообщения через плагин информирования для пользователя или группы:

- **this.info**(<ID плагина>, <ID пользователя>, <Текст сообщения>);
- **this.info**(<ID плагина>, <ID группы>, <Текст сообщения>);

```
this.info('telegram', 'u0002', 'Текст сообщения');  
this.info('pushnotification', 'u0002', 'Текст сообщения');
```

## Снимок с видеокамеры

- **this.snap**(<ID камеры - строка>, <Функция обратного вызова-строка>)

Получить снимок с видеокамеры можно, если

- установлен плагин [CCTV](#)
- Для камеры введен параметр **Snapshot Url** для получения снапшота.

Команда выполнит снимок с камеры, запишет его в файл и вернет имя файла в результате. Вы можете отправить этот файл в телеграм или по e-mail.

```
const script = {
  start() {
    this.snap("cam_5", "onSnap_5");
    // Если 10 секунд нет результата – завершим сценарий
    this.startTimer('T1', 10, 'onTimer');
  },

  onSnap_5(result) {
    if (result.filename) {
      this.info(
        "telegram",
        "admin",
        {
          img: result.filename,
          txt: "Снимок с камеры номер 5"
        }
      );
    } else {
      this.info(
        "telegram",
        "admin",
        "Был план получить снимок с камеры номер 5, но увы"
      );
    }
  },

  onTimer() {
    this.info("telegram", "admin", "Ошибка связи с камерой 5");
    this.exit();
  }
}
```

## Взаимодействие с активными клиентами

Сценарий может выполнить вызов диалога, запуск скрипта, звуковое оповещение для конкретного пользователя или группы. Эти действия будут выполнены для клиентов, подключенных в текущий момент с указанными учетными записями.

### Вызов диалога

Сценарий может вызвать диалог на экран конкретного пользователя (группы).

- **this.showDialog**(<ID диалога>, <ID пользователя>, <значения локальных переменных>)

Пример 1: Диалоговое окно будет выведено на все терминалы, на которых в данный момент запущен пользовательский интерфейс под логином 'admin'.

```
this.showDialog('di0012','admin');
```

Пример 2: Можно дополнительно передать значения локальных переменных

```
this.showDialog('di0013','admin', {dev_id: 'd0095'});
```

### Запуск скрипта

Добавлено: v5.9.34

Сценарий может запустить скрипт визуализации для конкретного пользователя или группы.

- **this.startVisscript**(<ID скрипта>, <ID пользователя или группы>, <значения локальных переменных>)

Пример 1: Скрипт визуализации будет запущен терминалах, на которых в данный момент запущен пользовательский интерфейс под логином 'admin'.

```
this.startVisscript('vs0002','admin');
```

Пример 2: Можно дополнительно передать значения локальных переменных

```
this.startVisscript('vs0002','admin', {var1: 'тест скрипт'});
```

### Звуковое оповещение - вариант 1

Воспроизведение звука на активном веб клиенте конкретного пользователя или группы

- **this.info**('sound', <ID пользователя или группы>, <Список звуковых файлов через запятую>);

```
this.info('sound', 'admin', 'ding.wav, внимание.wav, авария.wav');
this.info('sound', 'grp003', 'ding.wav, внимание.wav, авария.wav');
```

## Звуковое оповещение - вариант 2

Добавлено: v5.17.18

### playSound

Воспроизведение звука на активном веб клиенте конкретного пользователя или группы

- **this.playSound**(<Список звуковых файлов через запятую>, <ID пользователя или группы> );

```
this.playSound('ding.wav, внимание.wav, авария.wav', 'admin');
```

### repeatSound

Циклическое воспроизведение звука на активном веб клиенте конкретного пользователя или группы

- **this.repeatSound**(<Список звуковых файлов через запятую>, <ID пользователя или группы> );

```
this.repeatSound('ding.wav, внимание.wav, авария.wav', 'admin');
```

### stopSound

Прекращение воспроизведения звука на активном веб клиенте конкретного пользователя или группы

- **this.stopSound**(<ID пользователя или группы> );

```
this.stopSound('admin');
```

## Встроенные методы скрипта

Это методы информирования, работы с таймерами, журналами, плагинами, выполнение других системных действий.

Вызываются они через **this**.

## Запись в консоль отладчика сценария

- **this.log**(<Сообщение>)

```
this.log('Значение температуры' + temp.value);
```

## Запись в главный журнал системы

- **this.mainlog**(<Сообщение>, <Уровень>)
- **this.mainlog**(<Сообщение>, {level:<Уровень>, tags:<Тэг>, location:<Расположение>})

```
this.mainlog('Датчик сработал', 2);
this.mainlog(
  'Датчик сработал',
  {
    level: 2,
    tags: 'Климат',
    location: '/place/dg1'
  }
);
```

## Получение временной точки

Возвращает число (timestamp) для заданной временной точки: закат, восход, граница следующего часа

- **this.getSysTime**('sunrise' || 'sunset' || 'hourly', <Когда>)
  - аргумент **Когда** может быть:
    - строкой (<'today' || 'tomorrow' || 'год-месяц-день' || 'next'>)
    - объектом типа Дата ( new Date(2021,11,31)).
    - Если аргумент **Когда** не задан, используется 'today'



## Выполнение команды ОС

- **this.execOS**(<команда ОС-строка>, <Функция обратного вызова-строка -опционально>)

**Пример 1:** Выдать звуковое сообщение на сервере командой aplay.

Здесь не используется функция обратного вызова.

Сценарий НЕ будет ожидать результат выполнения, запустит команду и завершится.

```
const script = {
  start() {
    this.execOS(`aplay /home/sound/ding.wav`); //
  }
}
```

**Пример 2:** Определить температуру процессора

Здесь используется функция обратного вызова.

После выполнения команды ОС будет запущена функция сценария, которая разберет результат.

Поскольку у сценария "дел" больше не осталось (нет активных таймеров, слушателей, функций обратного вызова), сценарий будет завершен.

```
const script = {
  start() {
    this.execOS(`vcgencmd measure_temp`, 'processTemp');
  },

  processTemp(stdout) {
    //temp=44.0'C
    if (stdout) {
      const arr = stdout.split('=');
      if (arr.length == 2) {
        const temp = parseInt(arr[1]);
        this.log('Температура процессора '+temp);
      } else {
        this.log('Ошибка скрипта, получено '+stdout);
      }
    }
  }
}
```

## Получение каналов устройства

Для отправки команды плагину иногда бывает полезно получить список привязанных каналов.

- **this.getDeviceChannels**( {did:<ID устройства>, unit:<ID плагина>}, <Функция обратного вызова>)
  - **did** - ID устройства
  - **unit** - ID экземпляра плагина (опционально)

Функция обратного вызова, как обычно, получает ошибку в первом аргументе и результат (массив всех каналов, привязанных к свойствам указанного устройства) во втором.

```
const script = {
  start() {
    this.getDeviceChannels({did: 'd0002', unit: 'modbus1'}, 'onGet');
  },

  onGet(error, data) {
    const util = require("util");
    if (!error) {
      this.log(util.inspect(data));
    } else {
      this.log('Ошибка: ' + error)
    }
  }
};
```

## Генерация отчета

Добавлено: v5.9.45

Сценарий может запросить генерацию отчета.

- **this.getReport**({<объект, содержащий параметры отчета>}, <Функция обратного вызова-строка>)
  - **id** - ID отчета
  - **start** - начало периода (timestamp), обязательный параметр
  - **end** - конец периода (timestamp), если не указан, будет равен Date.now()
  - **content** - 'pdf' || 'csv'

Плагин генерации отчета подготовит отчет, запишет его в файл и вернет имя файла в результате. Вы можете отправить этот файл в телеграм или по e-mail

```

const script = {
  start() {
    const start = Date.now() - 3600000;
    this.getReport({id: 'r011', start, content: 'pdf'}, 'onGet');
  },

  onGet(error, result) {
    if (result.filename) {
      this.info(
        'telegram',
        'u0003',
        {
          txt: 'Отчет за последний час',
          pdf: result.filename
        }
      );
    }

    this.info(
      'email',
      'u0003',
      {
        txt: 'Отчет за последний час',
        img: result.filename
      }
    );

    this.log('Подготовлен и отправлен отчет ' + result.filename);
  } else {
    this.log('При подготовке отчета произошла ошибка: ' + error)
  }
}
};

```

## Определение папки для отчета

Добавлено: v5.17.46

По умолчанию отчет сохраняется в папку проекта **temp**.

Можно сохранить отчет в другую папку, указав параметр **folder**.

```

// Папка anyfiles внутри проекта
const folder = this.getProjectPath('anyfiles');
this.getReport({id: 'r011', start, content: 'pdf', folder}, 'onGet');

```

## Запуск сценария из сценария

Добавлено: v5.17.46

Из сценария можно запустить другой сценарий, в том числе шаговый.

- **this.runScene(sceneld)**
  - **sceneld** - ID сценария

```
this.runScene('trscen020');
```

### Пример вызова цепочки шаговых сценариев

Поскольку каждый шаговый сценарий имеет связанный с ним системный индикатор, можно написать сценарий для запуска последовательности шаговых сценариев.

Чтобы визуализировать процесс выполнения, создадим глобальную переменную `globals.curStepScene`, которая будет содержать имя системного индикатора текущего шагового сценария.

Используя `globals.curStepScene` можно будет вывести состояние и сообщения текущего шагового сценария.

```

const script = {
  start() {
    this.arr = ['trscen020', 'trscen021', 'trscen022', 'trscen023'];
    this.index = 0;
    this.next();
  },
  next() {
    if (this.index > 0) {
      const prevscene = this.arr[this.index-1];
      this.removeListener('__SCRT_'+ prevscene+'.state', 'onState');
    }

    const curscene = this.arr[this.index];
    const dn = '__SCRT_'+ curscene;
    this.sceneIndicator = this.getDevice(dn);

    if (!this.sceneIndicator) {
      this.log('Not found device '+dn)
      this.exit();
    }

    globals.curStepScene = dn;
    this.addListener(dn+'.state', 'onState');
    this.runScene(curscene);
  },
  onState() {
    const state = this.sceneIndicator.state;
    if (state == 2) {
      this.log('Suspended '+this.sceneIndicator.dn);
    }
    if (state == 0) {
      this.log('Stopped '+this.sceneIndicator.dn);
      if (this.index + 1 < this.arr.length) {
        this.index = this.index + 1;
        this.next();
      } else {
        globals.curStepScene = '';
        this.exit();
      }
    }
  }
};

```

## Выход из сценария

- `this.exit()`

```
this.exit();
```

## Вывод даты, времени в виде строки

Для работы с временем в javascript удобно использовать таймстемп (Timestamp). Он однозначно определяет время с точностью до миллисекунд.

**Timestamp** - это целое число, представляющее собой количество миллисекунд, прошедших с начала 1970 года в UTC.

В исторических таблицах системы используется именно timestamp (ts). Работать с timestamp в javascript легко:

```
// Получить текущий таймстемп – 13-значное число
const ts = Date.now();
// 1662155144643
console.log(ts);

// Преобразовать ts во встроенный js объект типа Date
const date = new Date(ts);
// 2022-09-02T21:45:44.643Z – это стандартный формат вывода даты в UTC
console.log(date);

// Можно вывести дату в локальных форматах, зависит от настроек ОС
// Sat Sep 03 2022 00:45:44 GMT+0300 (Москва, стандартное время)
console.log(date.toString());

// 03.09.2022, 00:45:44
console.log(date.toLocaleString());
```

Хотя javascript имеет разные, в том числе локальные форматы вывода даты, для показа пользователю часто удобно использовать predetermined форматы.

Можно, например, не выводить год и/или вывести миллисекунды. Или вывести только дату, или только время и т д

В API для этого предусмотрена функция вывода даты с использованием различных форматов. В любом случае используется локальное время сервера (с учетом timezone, установленной на уровне ОС).

- `this.getDateTimeStr(ts <, format >)`
  - ts - timestamp
  - format - опциональная форматная строка: если format не задан, дата будет выведена в формате DD.MM.YYYY HH:MM:SS

Значение параметра format	Формат вывода	Пример
'dtms'	DD.MM.YY HH:MM:SS.mmm	03.09.2022 00:45:44.643
'shortdtms'	DD.MM HH:MM:SS.mmm	03.09 00:45:44.643
'shortdt'	DD.MM HH:MM:SS	03.09 00:45:44
'reportdt'	DD.MM.YYYY HH:MM	03.09.2022 00:45
'reportd'	DD.MM.YYYY	03.09.2022
'onlytime'	HH.MM.SS	00:45:44
'id'	YYMMDDHHMMSSMMMM	2209030045440643
не задан	DD.MM.YYYY HH:MM:SS	03.09.2022 00:45:44

```
const str1 = this.getDateTimeStr(ts);
const str2 = this.getDateTimeStr(ts, 'dtms');
```

Конечно, всегда можно вручную отформатировать дату, используя функции объекта Date. Например, вывести время:

```
const ts = Date.now() - 3600000;
const dt = new Date(ts);
const str = dt.getHours() + ':' + dt.getMinutes() + ':' + dt.getSeconds();
```

# Шаговые сценарии

## Назначение

Механизм шаговых сценариев предназначен для автоматизации различных процессов.

Шаговый сценарий - это цепочка последовательных действий.

Шаговые сценарии могут запускаться по кнопке или по событию устройства.

## Описание

The screenshot shows the IntraSCADA software interface. On the left is a tree view with categories like 'Сценарии', 'Расписание', and 'Шаговые сценарии'. Under 'Шаговые сценарии', there is a sub-category 'Аварийный останов' containing several items, including 'Мельница 1'. The main window is titled 'Мельница 1' and has three tabs: 'Описание', 'Шаги', and 'Скрипт'. The 'Описание' tab is active, showing a form for the scenario's description. The form includes fields for 'Название' (Name) and 'Изменено' (Modified), both containing 'Мельница 1' and '28.06.2023 18:39:21' respectively. There is a checkbox 'Запускать по событиям устройств' (Run by device events) which is checked. A 'Комментарий' (Comment) field contains text about 'ТА: Готовность (RELE) (от WAGO)' and 'DO1: Вращение (RUN) (от WAGO)'. On the right, there are checkboxes for 'Проверять стартовые условия' (Check start conditions) and 'Объединить по И' (Combine by AND), both checked. Below the form are two tables. The first table, 'Запустить при изменении любого триггера' (Run when any trigger changes), has columns for 'Триггер запуска' (Start trigger), 'Операция' (Operation), and 'Операнд 2' (Operand 2). It contains four rows of triggers: 'inv\_0\_2.TA', 'inv\_0\_3.TA', 'inv\_0\_2.DO1', and 'inv\_0\_3.DO1', all with the operation 'равно' (equal) and operand '0'. The second table, 'Перед запуском проверить условия' (Check conditions before start), has columns for 'Операнд 1' (Operand 1), 'Операция' (Operation), and 'Операнд 2' (Operand 2). It contains one row with 'S\_AUTO.state' as the operand 1, 'равно' (equal) as the operation, and '1' as the operand 2.

На вкладке "Описание" кроме наименования и комментария, настраиваются:

## Триггеры запуска

Сценарий запускается при изменении любого свойства устройства из списка.

К примеру, сценарий запустится, если у частотного преобразователя inv\_0\_2 свойство TA станет равным 0.

## Условия запуска

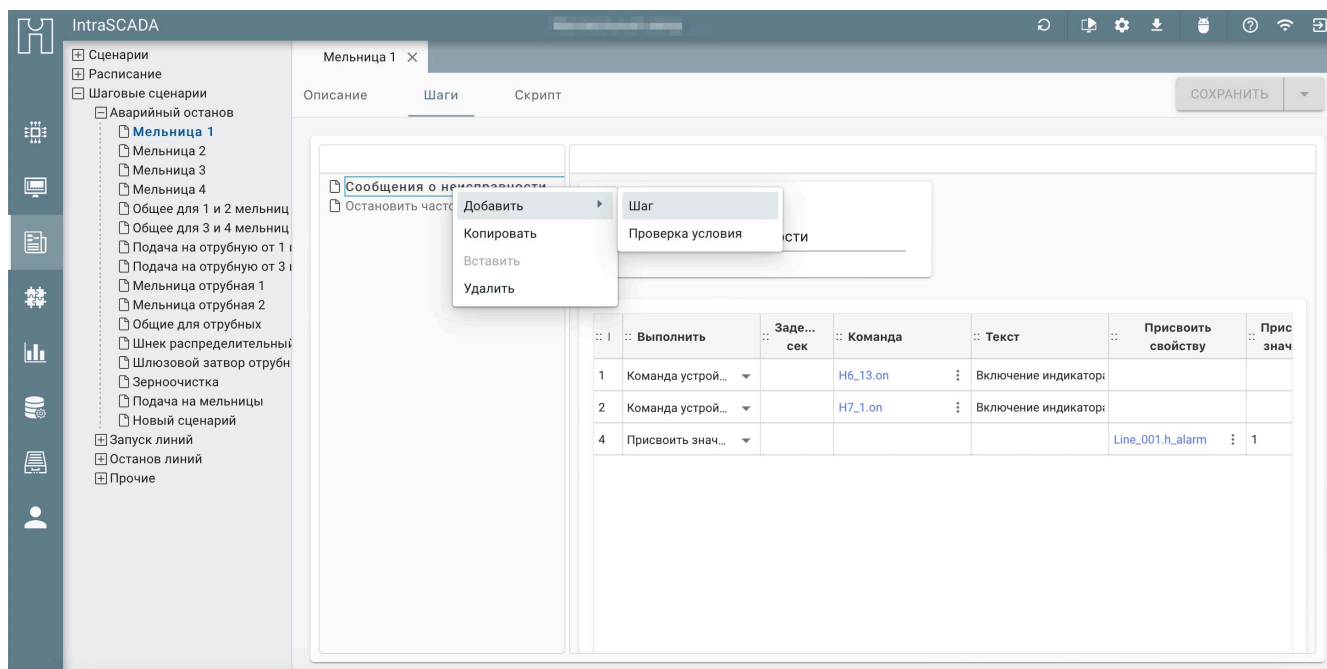
Сценарий запускается при определенном наборе условий.

В примере выше, проверяется условие: свойство S\_AUTO.state равно 1 (переключатель S\_AUTO находится в положении "Автоматический режим")

В списке проверки условий может быть несколько строк. Эти строки могут объединяться по "И" или по "ИЛИ"

## Шаги



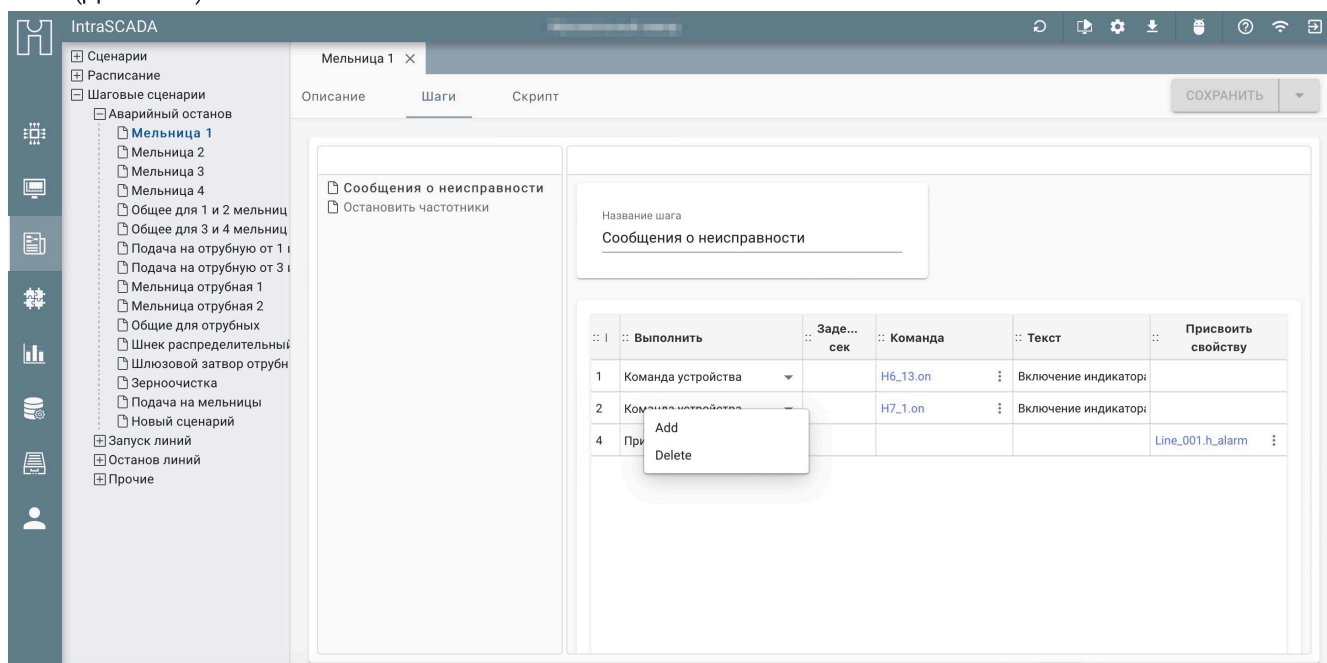


В списке шагов правой кнопкой мыши можно добавить:

- новый шаг
- проверку условия

## Добавить шаг

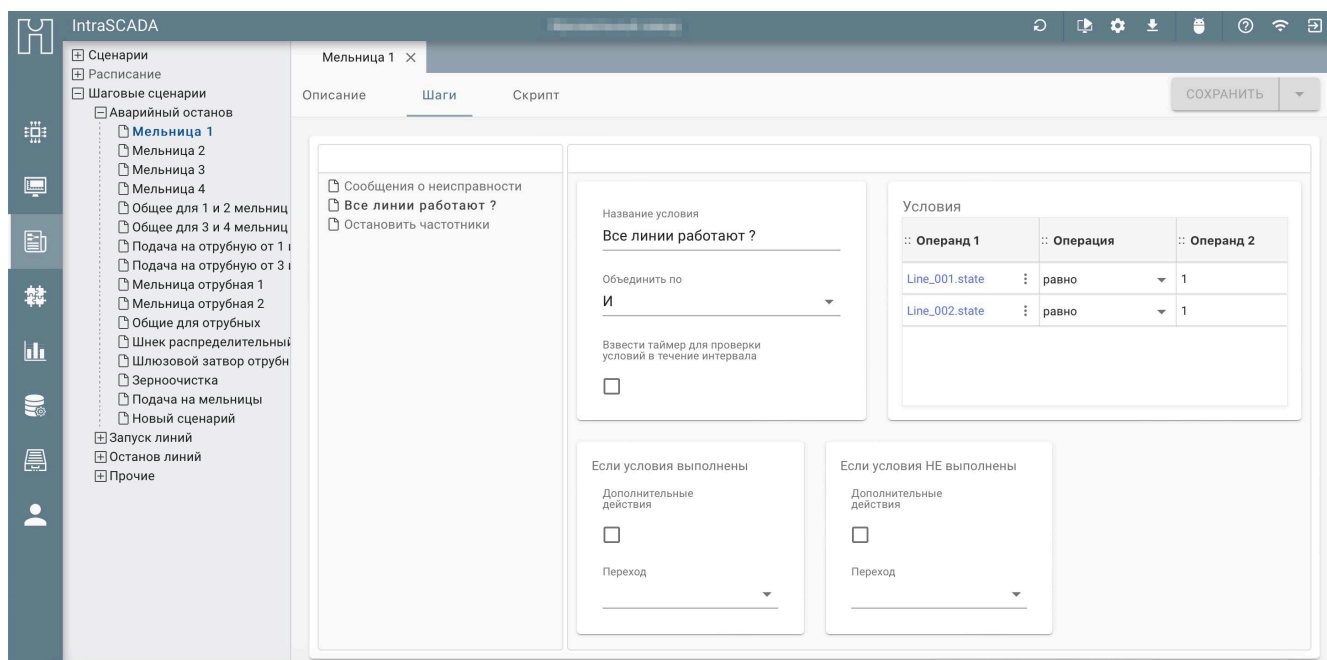
После добавления шага можно приступить к заполнению списка действий. Правой кнопкой мыши выбираем "Add" (Добавить):



Можно добавить следующие действия:

Выполнить	Комментарий	Примечание
Команда устройства	Выполнить команду устройства	
Присвоить значение	Присвоить значение свойству устройства	
Сообщение сценария	Выдать сообщение	См. <a href="#">Свойства шаговых сценариев</a>
Задержка	Выполнить задержку на время (в секундах)	
Завершить	Завершить выполнение всего шагового сценария	

## Добавить проверку условия



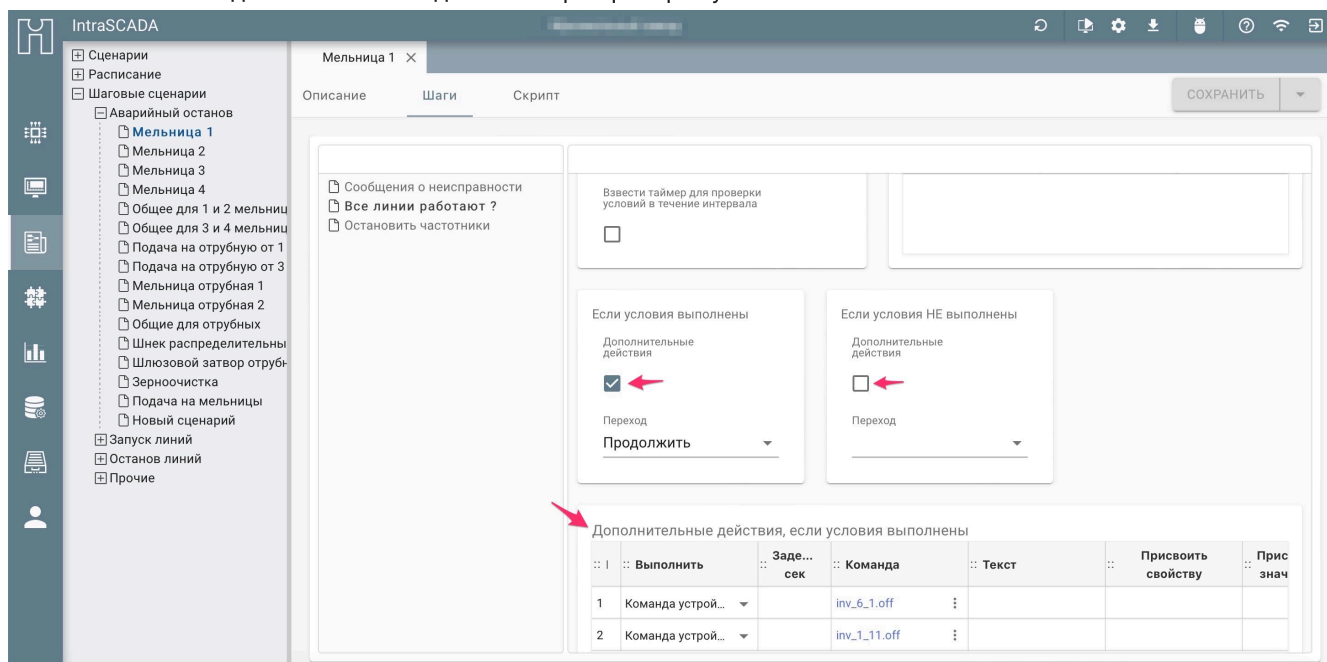
В списке условий добавляются свойства устройств для соответствия необходимым параметрам. Эти свойства можно объединить по "И" или по "ИЛИ"

Если условия выполнены можно:

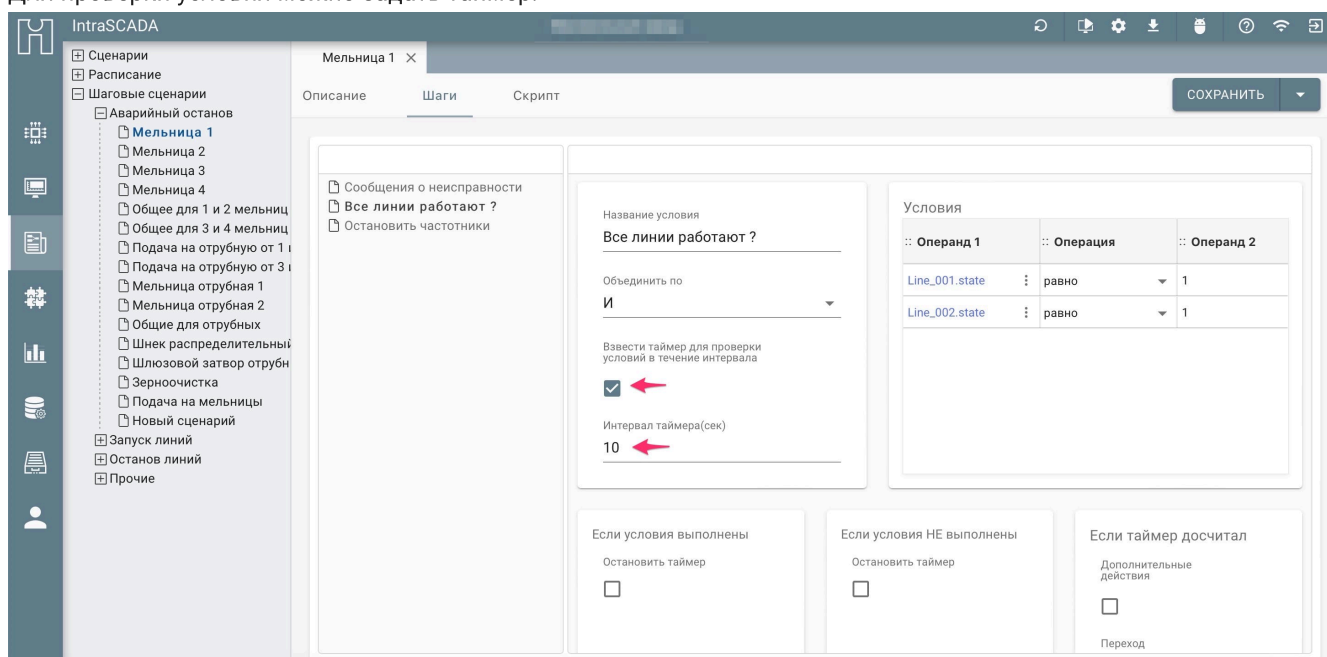
- Завершить выполнение шагового сценария
- Продолжить выполнение
- Перейти на другой шаг

Аналогичные действия можно выполнить и если условия не выполнены.

Можно выполнить дополнительные действия при проверке условия:

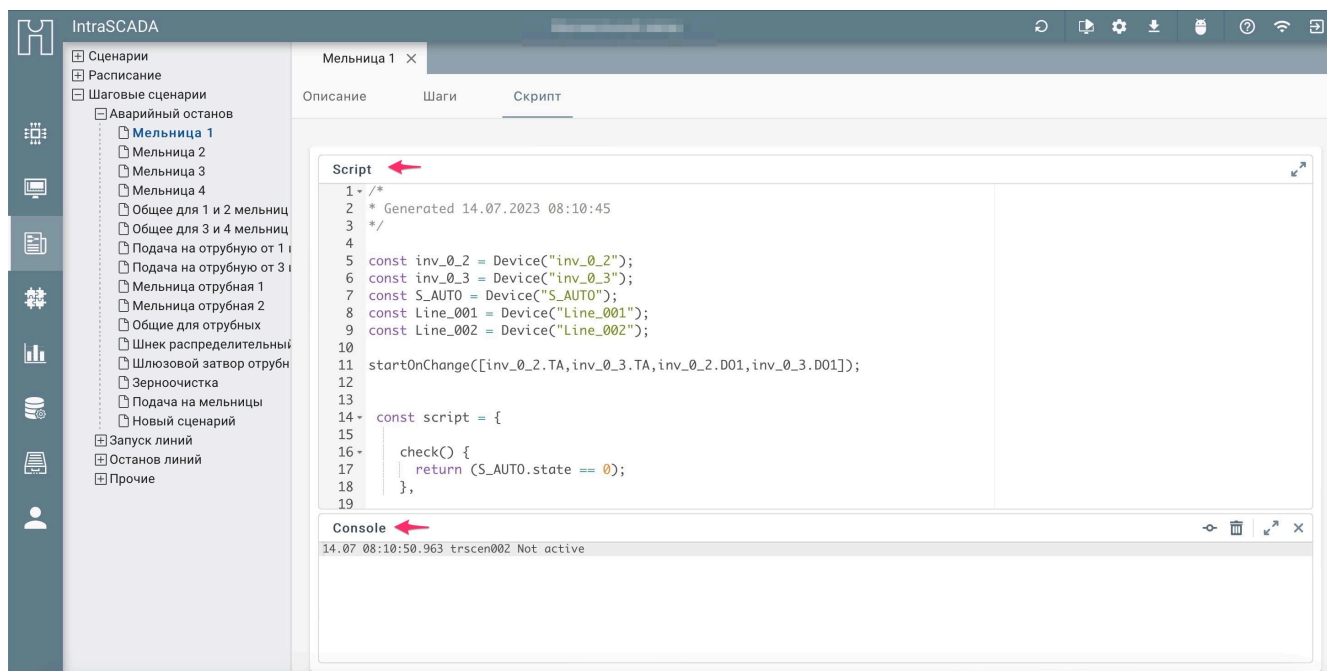


Для проверки условия можно задать таймер:



В этом случае проверка условия будет выполняться не на входе в этот шаг, а в течении заданного времени.

## Скрипт

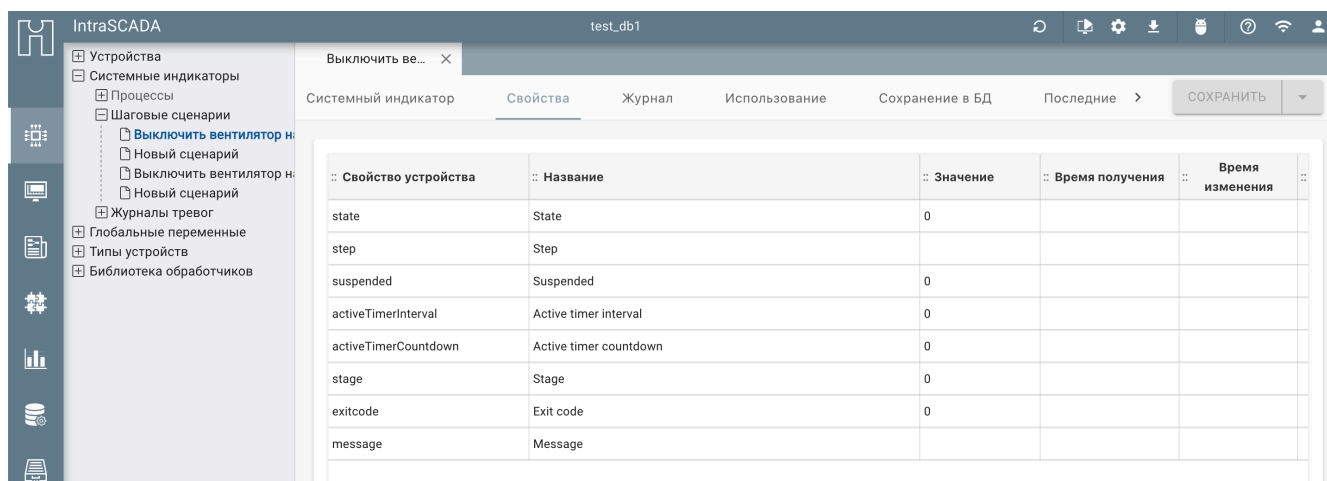


На вкладке "Скрипт" показан текст созданного шагового сценария. В окне "Console" отображается процесс работы сценария. Полезно для контроля и отладки работы сценария.

## Свойства шаговых сценариев

Шаговые сценарии имеют свой набор свойств:

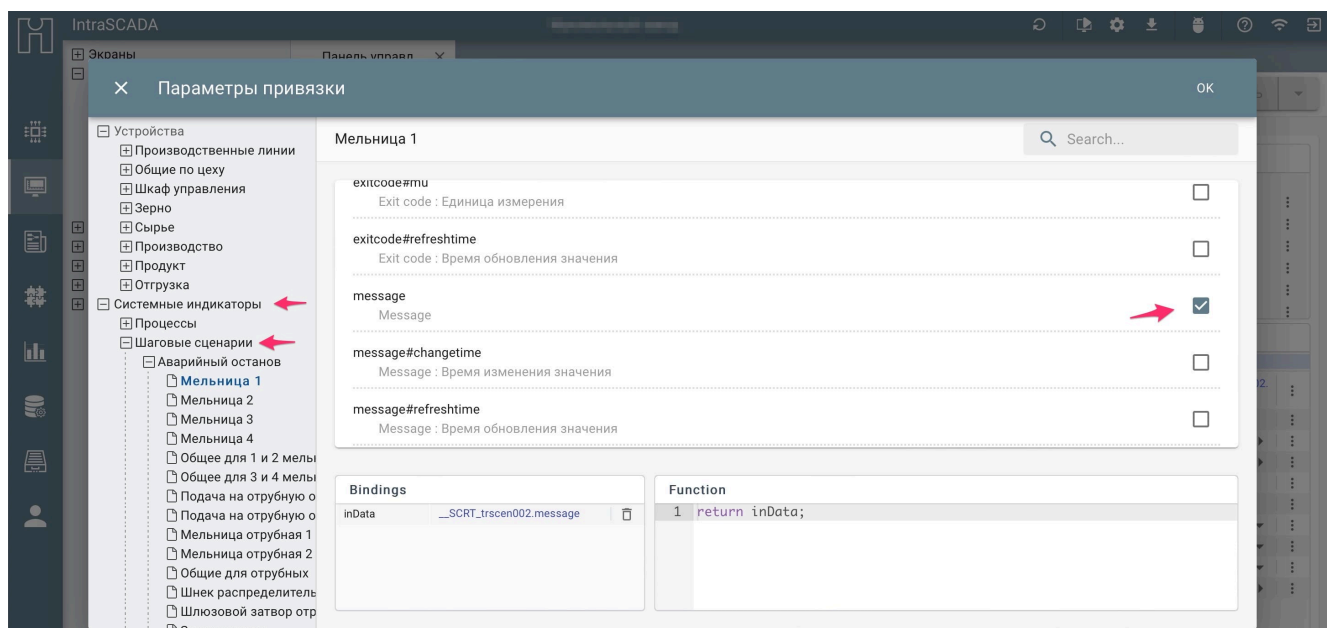
- state - Текущее состояние шагового сценария (0 - не работает, 1 - активный, 2 - ошибка, 3 - приостановлен)
- step - Текущий шаг
- suspended - Индикатор приостановки шагового сценария (1 - приостановлен)
- activeTimerInterval - Общее время текущего таймера
- activeTimerCountdown - Обратный отсчет от времени текущего таймера
- exitcode - код ошибки сценария (0 - нет ошибки, 1 - ошибка кода сценария)
- message - текущее сообщение шагового сценария



Свойство устройства	Название	Значение	Время получения	Время изменения
state	State	0		
step	Step			
suspended	Suspended	0		
activeTimerInterval	Active timer interval	0		
activeTimerCountdown	Active timer countdown	0		
stage	Stage	0		
exitcode	Exit code	0		
message	Message			

Любое свойство можно использовать в дальнейшей работе с помощью привязки. Аналогично привязке свойств устройств.

К примеру свойство message можно отобразить на визуализации в текстовом поле:



Так же доступны действия на нажатии кнопки мыши: Пауза сценария, продолжение сценария после паузы

# Плагины

## О плагинах

**Плагины** — это программные модули (драйверы) для подключения оборудования и обмена данными со сторонними сервисами.

Существует набор готовых плагинов для работы с оборудованием:

- Modbus RTU/TCP/ASCII/RTU over TCP
- Modbus server
- MQTT/MQTTS
- MQTT server
- OPC UA client
- OPC UA server
- HTTP/HTTPS
- SNMPv2
- MEGA-D
- EMULATOR
- SIEMENS S7 Ethernet protocol RFC1006
- Ethernet/IP AllenBradley
- Mitsubishi MC FX3U
- МЭК 60870-5-104 клиент
- Счетчики Меркурий
- Счетчики СЭТ
- KNX/IP
- Bacnet/IP
- Codesys 2.3
- PostgreSQL client
- LDAP client
- Kerberos
- Ping
- CCTV
- Zigbee

Плагины для информирования:

- E-mail
- Telegram

Состав плагинов постоянно пополняется, полный список плагинов доступен на вкладке дашборда **Плагины**

## Разработка плагинов

Любое оборудование или сервис, который требуется подключить к системе, должен иметь API (Application Programming Interface). API — описание способов (набор классов, процедур, функций, структур или констант), которыми система может взаимодействовать с оборудованием. Используя это API можно написать плагин для нового оборудования. Для написания плагина рекомендуем ознакомиться с концепцией плагинов и API взаимодействия плагина и сервера.

## Установка плагина

Список плагинов доступен во вкладке **Плагины** основного дашборда.

Плагин	Описание	Версия	Последняя версия	Статус	Действия
ethernetip	Ethernet/IP plugin		5.0.15	Доступно для установки	Установить
http	HTTP plugin		5.5.6	Доступно для установки	Установить
httpcache	HTTP cache plugin		5.0.0	Доступно для установки	Установить
iec60870p	Client for 60870-5-104		5.5.7	Доступно для установки	Установить
jethomed1v	JetHome D1 plugin		5.0.7	Доступно для установки	Установить
knx_ip	KNX IP plugin		5.0.2	Доступно для установки	Установить
laurent	Laurent Plugin		5.5.2	Доступно для установки	Установить
megad	Megad HTTP plugin		5.0.11	Доступно для установки	Установить
modbus	Modbus plugin	5.5.28	5.5.32	Есть обновление	Обновить, Удалить
modbusserver	Modbus server plugin	5.5.7	5.5.8	Есть обновление	Обновить, Удалить
mqttclient	MQTT client		5.5.29	Доступно для установки	Установить
mqttserver	MQTT Server plugin		5.5.2	Доступно для установки	Установить
mssql	MS SQL client plugin		5.0.2	Доступно для установки	Установить

Установлено плагинов: 5

Доступно обновлений: 2

Доступно для установки: 25

**ПРОВЕРИТЬ ОБНОВЛЕНИЯ**

## Рекомендуемый способ установки плагина

Установку плагина можно выполнить из таблицы плагинов, показанной выше.

- Плагин будет установлен автоматически без необходимости скачивания zip архива
- Будет установлена самая свежая версия плагина

## Резервный способ установки плагина

Плагин можно установить с помощью zip архива, который можно скачать с нашего [github](#). Например если у вас отсутствует подключение к интернету на сервере. Архив плагина (zip) - это пакет, содержащий программные модули и файлы описания плагина. Установка такого архива выполняется интерактивно с помощью кнопки

**Import:**

Плагин	Описание	Версия	Последняя версия
applehome	Apple Home Plugin		5.2.1
applehomekit	Apple HomeKit Plugin		5.0.8
cctv	CCTV plugin		5.7.3
email	E-mail plugin		5.5.3
emuls	Emulator	5.5.8	5.5.8
http	HTTP plugin		5.5.6
jethomed1v	JetHome D1 plugin		5.0.7
knx_ip	KNX IP plugin		5.0.2
laurent	Laurent Plugin		5.5.2
megad	Megad HTTP plugin		5.0.11
modbus	Modbus plugin	5.6.3	5.6.1
mqttclient	MQTT client		5.5.32
mqttserver	MQTT Server plugin		5.5.2

Процесс установки плагина отображается в всплывающем окне. По окончании установки появляется зеленая галочка "Complete". Нажимаем кнопку OK:

После установки плагин становится доступен в любом проекте на сервере.

После установки необходимо обновить страницу РМ.

## Добавление экземпляра плагина

После установки плагина в дереве вы увидите папку с названием плагина. Щелкнув правой кнопкой мыши по папке, вы сможете добавить экземпляр плагина.



Название	Состояние	Время запуска	Время останова	Uptime	Ошибка
modbus1	Приостановлен				
modbus2	Приостановлен				
mqttclient1	Работает	20.09.2021 19:27:21		0d 17h 49m	
opcua1	Приостановлен				
s7plc1	Приостановлен				
snmp1	Приостановлен				
emuls1	Приостановлен				

Обратите внимание!

Не все плагины имеют возможность запускать отдельные экземпляры.

Если в дереве плагинов вы видите плагин в корневой ветке без папки, это означает то, что плагин может запускаться только в одном экземпляре.

## Настройка плагина

Настройка экземпляра плагина осуществляется на вкладке **Параметры плагина**.

В зависимости от плагина указываются параметры соединения, уровень логирования и время перезапуска плагина после останова или по расписанию. В большинстве плагинов, один экземпляр обслуживает один IP адрес, поэтому если у вас например 10 устройств Modbus TCP, то вам нужно запустить 10 экземпляров плагина

на каждый IP адрес.

Параметры плагина

Каналы

Таблица каналов

Отладчик

Журнал

Лог

СОХРАНИТЬ

Транспорт

Modbus TCP

IP

192.168.1.72

Порт

502

Ожидание ответа на запрос (ms)

5000

Интервал между запросами (ms)

200

Мак кол-во слов при чтении диапазона

250

Настроить порядок байт

☐

Отправлять только изменения

☒

Время рестарта (сек)

5

Уровень логирования

-

Перезагружать плагин по расписанию

☐

Комментарий

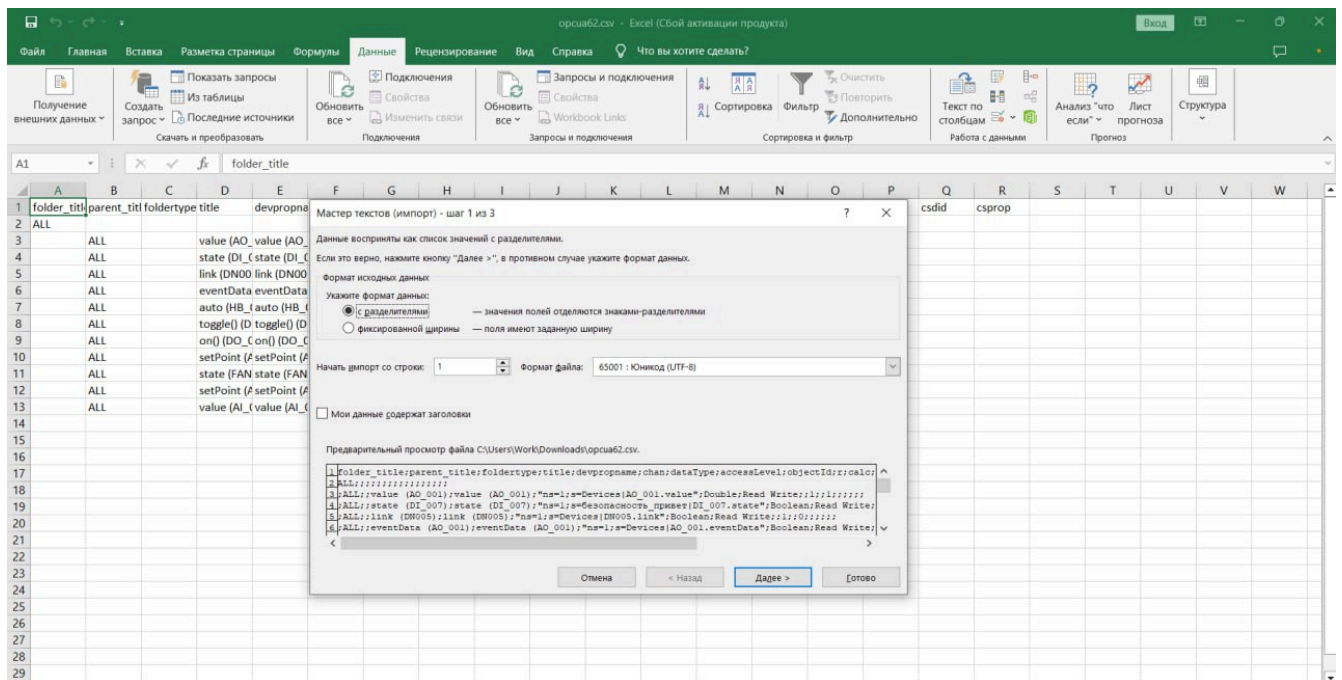
ID	Состояние	Время запуска	Время останова	Uptime	Ошибка
modbus2	Приостановлен				

## Добавление и редактирование каналов плагина

Создание каналов для связи с оборудованием считается одной из самых трудоемких задач при интеграции. Мы постарались максимально автоматизировать данную работу, добавив несколько способов создания каналов и привязок. Каналы в плагине можно добавить 3 способами:

1. Вручную, во вкладке **Каналы** правой кнопкой мыши создаем канал в дереве плагина и заполняем поля для получения или отправки данных.
2. Каналы добавляются из вкладки **Сканирование**, если такой функционал поддерживает плагин.
3. Загрузка из файла CSV. Нажав правой кнопкой мыши на экземпляре плагина в дереве вы можете выгрузить/загрузить каналы с помощью CSV. Для примера можете создать несколько каналов, выгрузить их в CSV, отредактировать и загрузить обратно. В этом файле так же можно указать конкретные свойства устройства к которым будет привязан канал

На ОС Windows могут возникнуть проблемы с кодировкой при использовании русских символов в параметрах каналов, так как ОС Windows меняет кодировку UTF-8 на WIN-1251 при открытии или сохранении CSV файла в Excel. Для того, чтобы сохранить кодировку необходимо открыть пустой файл Excel и добавить CSV файл через механизм **Получения внешних данных из текста** в разделе **Данные**, при этом указав кодировку **UTF-8** и символ разделителя **Точку с запятой**.



Для сохранения в Excel в формате UTF-8 необходимо выбрать пункт **Сохранить как**, Тип файла **CSV UTF-8 (разделитель - запятая)**. Если данного типа файла в вашей версии Excel нет, то можно преобразовать кодировку с помощью приложения **Notepad++**

Для того, чтобы данные попали в привязанные вами свойства устройства, необходимо установить галочку на свойстве **Чтение**, в тех каналах, с которыми вы хотите обмениваться данными. Если есть необходимость в записи в канал, то необходимо установить галочку на свойстве **Запись**

Для ускорения механизма связки каналов со свойствами устройств мы реализовали возможность привязывать каналы в папке к устройству в целом. Для этого вам необходимо заполнить поле **Свойство устройства для привязки** названием свойства в устройстве, к которому вы будете привязывать канал. Создать папку и поместить туда каналы. Когда у всех каналов в папке будет заполнено данное поле, щелкнув правой кнопкой

мышки на папке вы сможете привязать все эти каналы к конкретному устройству одним нажатием.

Channels

+

emul\_DG\_101

**emul\_DT\_108**

emul\_DT\_105

emul\_DT\_106

emul\_DD\_106

emul\_DT\_201\_1

emul\_DT\_204\_1

emul\_DT\_209\_1

emul\_DT\_208\_1

emul\_DD\_208\_1

emul\_DH206\_1

emul\_DT206\_1

Properties

Привязка к устройству

DT108\_1 ▀ Датчик температуры ▀ value

Channel ID

emul\_DT\_108

Свойство устройства для привязки

Тип

Analog Input ▾

Период, сек

70

Min

14

Канал для чтения

☒

Канал для записи (команда)

☐

Для быстрого редактирования свойств каналов существует вкладка **Таблица каналов**, где предоставляется групповая возможность редактирования свойств.

Параметры плагина   Каналы   **Таблица каналов**   Отладчик   Журнал   Лог

СОХРАНИТЬ ▾

:: Канал	::	Пе... сек	:: ...	:: ...	:: Де...	:: в ди... min-	:: Устройство	:: Св... ус...	:: Ка... зн...	:: Канал: в
emuls1.emul_DD_106	DI	7	17	24	1	<input type="checkbox"/>	DD106_1 ▀ Датчик движения	state		
emuls1.emul_DD_208_1	DI	8	17	24	1	<input type="checkbox"/>	DD208_1 ▀ Датчик движения	state		
emuls1.emul_DG_101	DI	140	17	24	1	<input type="checkbox"/>	DG101_1 ▀ Датчик открытия двери	state		
emuls1.emul_DH206_1	AI	8	40	65	4	<input type="checkbox"/>	DH206_1 ▀ Датчик влажности	value		
emuls1.emul_DT206_1	AI	10	19	25	0.4	<input checked="" type="checkbox"/>				
emuls1.emul_DT_105	AI	75	17	28	1	<input checked="" type="checkbox"/>				
emuls1.emul_DT_106	AI	10	19	30	0.333	<input type="checkbox"/>	DT106_1 ▀ Датчик температуры	value		
emuls1.emul_DT_108	AI	70	14	30	5	<input checked="" type="checkbox"/>	DT108_1 ▀ Датчик температуры	value		
emuls1.emul_DT_201_1	AI	10	17	22	0.7	<input checked="" type="checkbox"/>	DT201_1 ▀ Датчик температуры	value		
emuls1.emul_DT_204_1	AI	10	17	23	0.2	<input checked="" type="checkbox"/>	DT204_1 ▀ Датчик температуры	value		
emuls1.emul_DT_208_1	AI	73	19	23	0.4	<input checked="" type="checkbox"/>	DT208_1 ▀ Датчик температуры	value		
emuls1.emul_DT_209_1	AI	69	19	23	0.8	<input checked="" type="checkbox"/>	DT209_1 ▀ Датчик температуры	value		

### Отладчик

Во вкладке отладчика есть возможность выводить отладочные сообщения в консоль, в зависимости от уровня логирования плагина. Сообщения выводятся в виде строки. Если сообщений поступает много, вы можете воспользоваться фильтром. В поле фильтра необходимо указать слово, по которому мы ищем совпадения. Если

таких слов несколько, то указываем их через пробел

Параметры плагинаКаналыТаблица каналовРасширениеОтладчикЖурнал

СОХРАНИТЬ

Консоль

single

```
topic: 'zigbee2mqtt/button',
value: '{"action":"single","battery":100,"linkquality":2,"voltage":3002}'
}
]
set undefined
21.09 15:45:51.533 IH: get [
{
id: 'zigbee2mqtt_button',
topic: 'zigbee2mqtt/button',
value: '{"battery":100,"click":"single","linkquality":2,"voltage":3002}'
}
]
set undefined
21.09 15:46:02.638 IH: get [
{
id: 'zigbee2mqtt_button',
topic: 'zigbee2mqtt/button',
value: '{"action":"single","battery":100,"linkquality":28,"voltage":3002}'
}
]
set undefined
21.09 15:46:02.642 IH: get [
{
id: 'zigbee2mqtt_button',
topic: 'zigbee2mqtt/button',
value: '{"battery":100,"click":"single","linkquality":28,"voltage":3002}'
}
]
set undefined
```

## Журнал

В журнал плагинов записывается информация по запуску и останову плагина. Данная информация хранится в базе данных.

Параметры плагинаКаналыТаблица каналовРасширениеОтладчикЖурналЛог

СОХРАНИТЬ

Время	Плагин	Сообщение	Уровень логирования
16.06 09:31:00.907	mqttclient1	IH: Plugin suspended	null
16.06 09:31:00.871	mqttclient1	IH: Suspend plugin	null

## Лог



В данной вкладке выводится текстовый лог, согласно установленному уровню логирования плагина. Данный лог соответствует файлу лога, который хранится в папке /opt/ih-v5/logs для данного экземпляра плагина.

Параметры плагина

Каналы

Таблица каналов

Отладчик

Журнал

Лог

СОХРАНИТЬ

1

2 14.09 18:03:09.326 Modbus Master plugin has started.

3 14.09 18:03:09.340 Received params...

4 14.09 18:03:09.379 Received 2 channels...

5 14.09 18:03:09.384 Requested channels update. Get channels: no

6 14.09 18:03:09.432 ERROR: Error: connect ECONNREFUSED 192.168.1.72:502

7 | at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1146:16) {

8 | errno: -111,

9 | code: 'ECONNREFUSED',

10 | syscall: 'connect',

11 | address: '192.168.1.72',

12 | port: 502

13 | }

14

# Плагин CCTV

## Назначение

Плагин CCTV предназначен для:

- передачи изображения с IP камер в браузеры (Safari, Chrome) на компьютерах, планшетах и смартфонах.
- передачи изображения на мобильное приложение

Плагин выполняет роль стрим-сервера. По запросу подключается к камере и раздает поток активным клиентам.

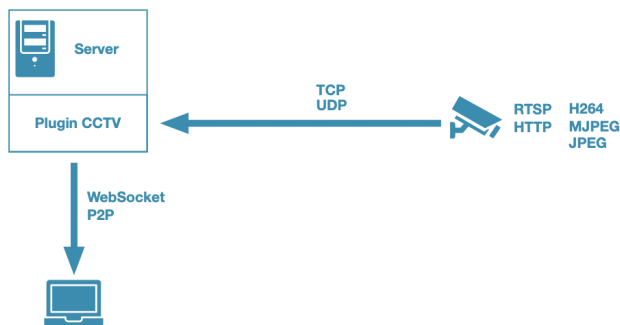
С одной камеры плагин забирает только один поток и раздает его всем активным клиентам.

При отсутствии активных клиентов плагин отключается от камеры и уходит в режим ожидания.

Плагин не производит транскодирование видео потока, что позволяет экономить ресурсы сервера.

Для работы плагина не требуются какие либо дополнительные программы (FFmpeg и пр.)

## Поддерживаемые протоколы



### Подключение IP камер:

Транспортные протоколы: TCP, UDP

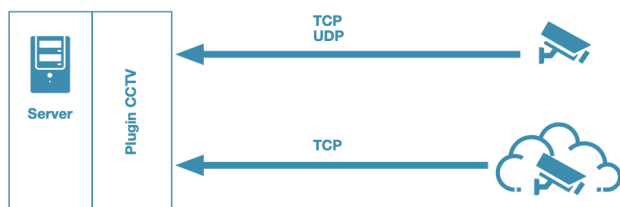
Прикладные протоколы: RTSP, HTTP

Форматы сжатия: H264, MJPEG, JPEG

### Подключение клиентов (компьютеры, планшеты, смартфоны):

Протоколы: WebSocket, P2P

## Подключение IP камер

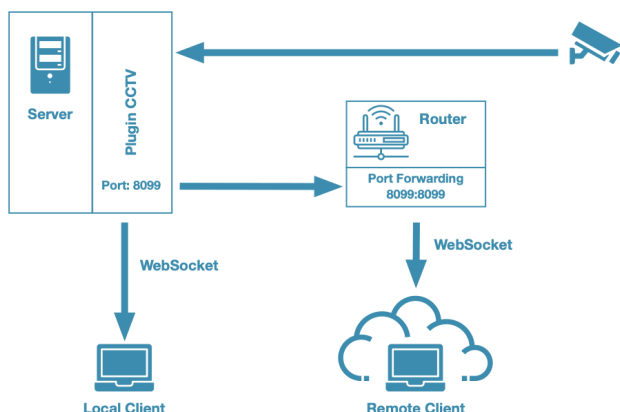


Для подключения камер, расположенных **в локальной сети**, можно использовать любой протокол: **TCP или UDP**

Для подключения камер, расположенных **вне локальной сети**, необходимо использовать протокол **TCP**

## Подключение клиентов

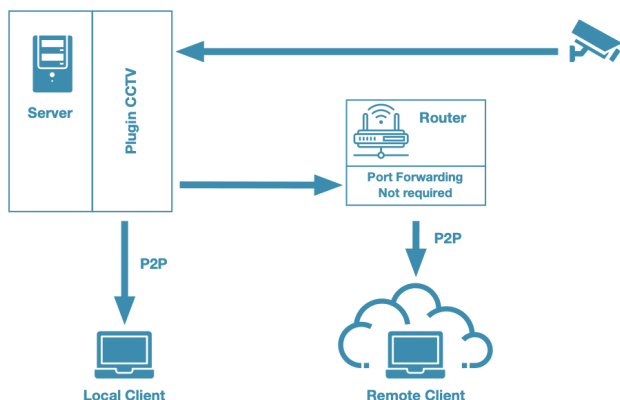
## Вариант 1 (подключение по протоколу WebSocket):



Если ваш сервер имеет выделенный IP адрес, рекомендуется подключение по протоколу WebSocket.

Для удаленного просмотра камер необходимо выполнить проброс порта на роутере, через который вы выходите в интернет.

## Вариант 2 (подключение по протоколу P2P):



Если выделенного IP адреса нет, можно установить подключение по протоколу P2P. В этом случае проброс порта на роутере не требуется.

В обоих вариантах подключение локальных клиентов возможно как по протоколу WebSocket, так и по протоколу P2P. По быстродействию и производительности оба варианта равноценны.

## Настройка плагина

### Параметры плагина

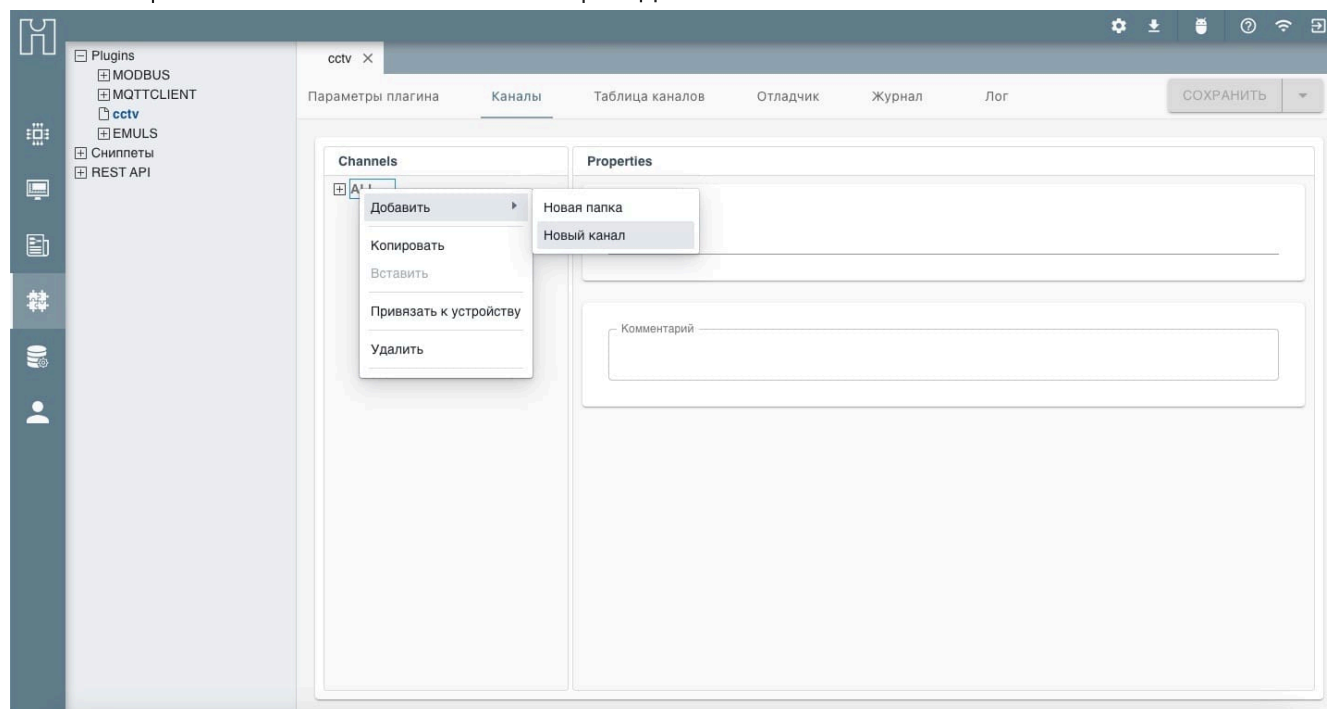
На вкладке "Параметры плагина" можно, при необходимости, изменить адрес порта для протокола WebSocket

### Каналы

На вкладке "Каналы" выполняется настройка IP камер.



Нажатием правой кнопки мыши на папке All выбрать Добавить - Новый канал:



Заполнить следующие поля:

Properties	
<p>Название</p> <p><b>MyCam1</b></p>	<p>Тип</p> <p><b>RTSP/H264</b></p>
<p>Url</p> <p><b>rtsp://user:pwd@192.168.0.xxx:port/videoMa</b></p>	<p>Протокол</p> <p><b>UDP</b></p>
<p>Транспорт (Клиент - Сервер)</p> <p><b>P2P</b></p>	
<p>Snapshot Url</p> <p><b>http://admin:123456@192.168.0.64:80/ISAPI/Streaming/channels/101/picture?snapShotImageType=</b></p>	

- Название - название камеры
- URL - строка запроса RTSP потока от камеры.
- Тип - вариант прикладного протокола и формата сжатия
- Протокол - TCP или UDP
- Транспорт - P2P или WebSocket
- Snapshot Url - Строка запроса для получения снапшота с IP камеры.

URL запросы для потока и снимка зависят от конкретной камеры. Эту информацию можно найти на сайте производителя.

# Плагин Emulator

## Назначение

Плагин предназначен для эмуляции работы различных устройств. Полезен для отладки работы системы при отсутствии физических устройств.

Плагин обеспечивает генерацию дискретных и аналоговых значений с заданной периодичностью.

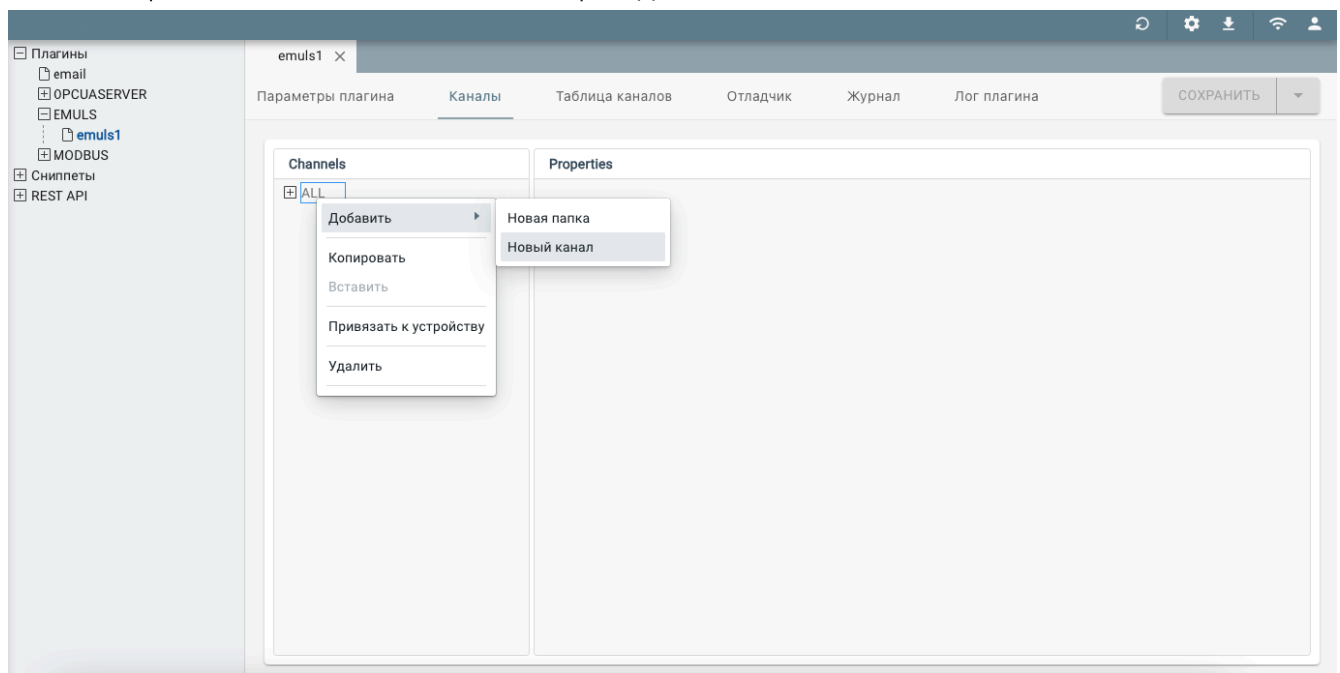
## Настройка

### Настройка подключения

Специальных настроек не требуется.


### Настройка каналов

Нажатием правой кнопки мыши на папке All выбрать Добавить - Новый канал:



Заполнить следующие поля:

**Properties**

Привязка к устройству
 

Channel ID  
 1

Свойство устройства для привязки

Тип  
 Digital Input

Период, сек  
 60

Канал для чтения  
☒

Канал для записи (команда)  
☐

- Привязка к устройству. Привязку канала к свойству устройства в системе.
- Channel ID - наименование канала
- Тип - Digital/Analog input

Для бинарных значений достаточно задать период отправки значения свойству привязанного устройства.

Для аналоговых необходимо задать:

Тип  
 Analog Input

Период, сек  
 60

Min  
 0

Max  
 100

Дельта  
 1

Случайное в диапазоне min-max  
☐

- Период - периодичность отправки значения свойству привязанного устройства.

- Min - минимальное значение
- Max - максимальное значение
- Дельта - дискрета изменения

Если установить галку "Случайное в диапазоне min-max", значения свойству привязанного устройства будут отправляться случайно в диапазоне от min до max

## Плагин Kerberos

Плагин работает только под Linux x64/arm/arm64

Плагин Kerberos предназначен для реализации механизма SSO (Single sign-on) - технология единого входа. Данная технология позволяет пользователям входить в систему IntraSCADA с использованием учетных данных LDAP, минуя форму авторизации.

Чтобы технология SSO заработала, кроме установки плагина Kerberos, необходимо:

1. Установить и настроить плагин [LDAP client](#). Импортировать группы и пользователей в IntraSCADA.
2. Создать SPN для сервиса IntraSCADA и привязать его к учетной записи LDAP.
3. На сервере LDAP сгенерировать Keytab-файл и скопировать его на компьютер с IntraSCADA.
4. На компьютерах клиентов выполнить настройку браузера для использования SSO
5. В системных настройках IntraSCADA установить галку в поле **Использовать технологию единого входа(SSO)**

*Здесь дано описание процедур создания SPN и Keytab-файла для Windows Active Directory. Для других серверов LDAP (FreeIPA,...) описание можно найти в документации*

### Создание SPN

SPN (Service Principal Name) — уникальный идентификатор экземпляра сервиса.

До того как Kerberos сможет использовать SPN для аутентификации сервиса, SPN должен быть привязан к учетной записи, которая будет использоваться для входа. SPN может быть привязан **только к одной учетной записи**. Если учетная запись, привязанная к SPN, изменяется, то необходимо заново выполнить привязку.

- Для начала необходимо создать на контроллере домена (DC) пользователя к которому впоследствии будет привязан SPN. Например, создадим пользователя intrascada:

Новый объект - Пользователь

Создать в: my-office.com/Users

Имя:  Инициалы:

Фамилия:

Полное имя:

Имя входа пользователя:  
 @my-office.com

Имя входа пользователя (пред-Windows 2000):

< Назад **Далее >** Отмена

- Далее необходимо запретить пользователю смену пароля и не ограничивать срок действия пароля. Последнее важно, так как иначе при истечении срока действия пароля придется не только менять пароль, но и заново генерировать keytab-файлы привязанные к этому пользователю:

Новый объект - Пользователь

Создать в: my-office.com/Users

Пароль:

Подтверждение:

☐ Требовать смены пароля при следующем входе в систему

☒ Запретить смену пароля пользователем

☒ Срок действия пароля не ограничен

☐ Отключить учетную запись

< Назад **Далее >** Отмена

В целях безопасности рекомендуется исключить сервисного пользователя из доменных групп.

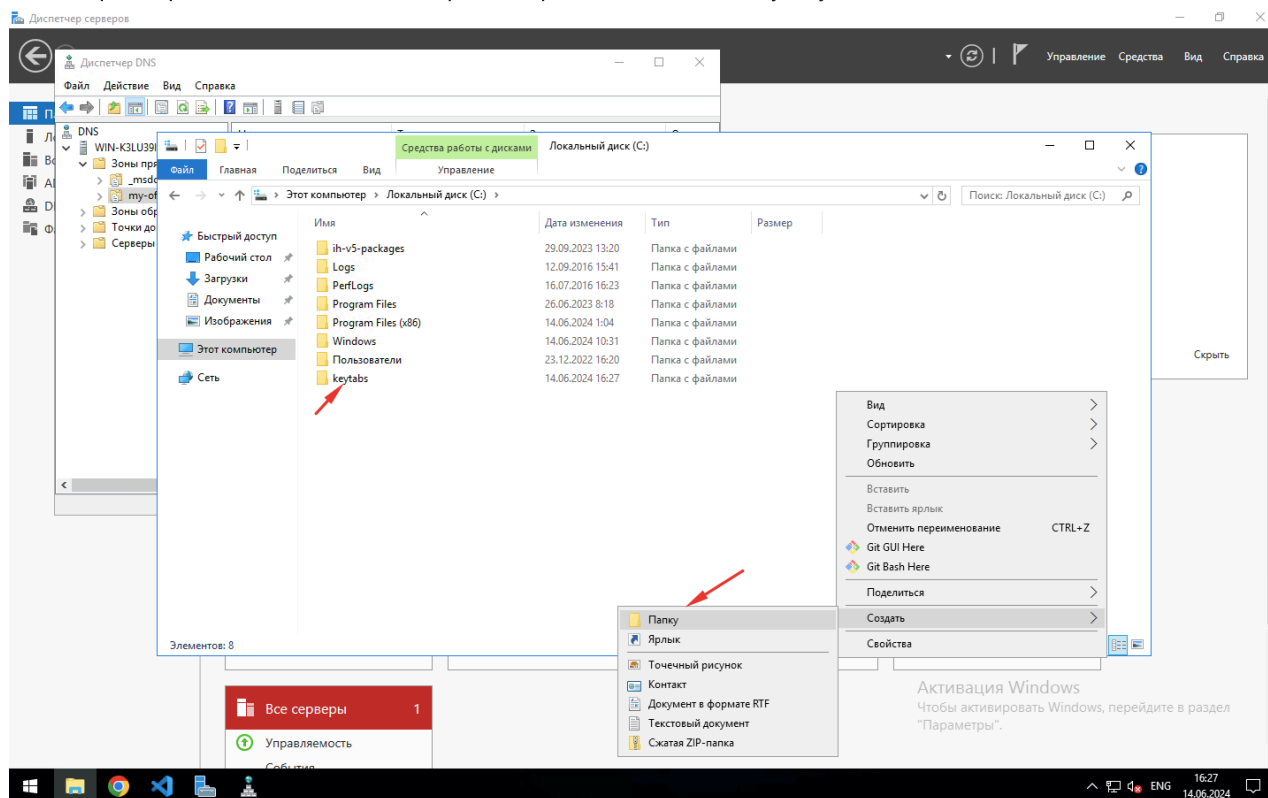
Привязку SPN к пользователю intrascada автоматически выполняет утилита ktpass при создании keytab-файла.

## Генерация Keytab-файла на сервере LDAP

Keytab-файл — это файл содержащий пары Kerberos принципалов и их ключей (полученных с использованием Kerberos пароля). Эти файлы используются для аутентификации в системах, использующих Kerberos, без ввода пароля. Если пароль принципала изменится, то keytab-файл необходимо будет сгенерировать заново.

Внимание! Каждый кто имеет разрешения на чтения keytab-файла может воспользоваться любыми ключами в нем. Чтобы предотвратить нежелательное использование, ограничивайте права доступа при создании keytab-файла

- На контроллере домена создаем в корневом разделе диска папку 'keytabs'



- Запускаем командную строку, вводим команду:

```
ktpass `
-princ HTTP/scada.my-office.com@my-office.com `
-mapuser intrascada `
-crypto ALL `
-ptype KRB5_NT_PRINCIPAL `
-pass Test1234 `
-target my-office.com `
-out C:\keytabs\intrascada.keytab
```

Где -princ HTTP/xxxxx - адрес нашего сервера

-mapuser xxxx - имя пользователя

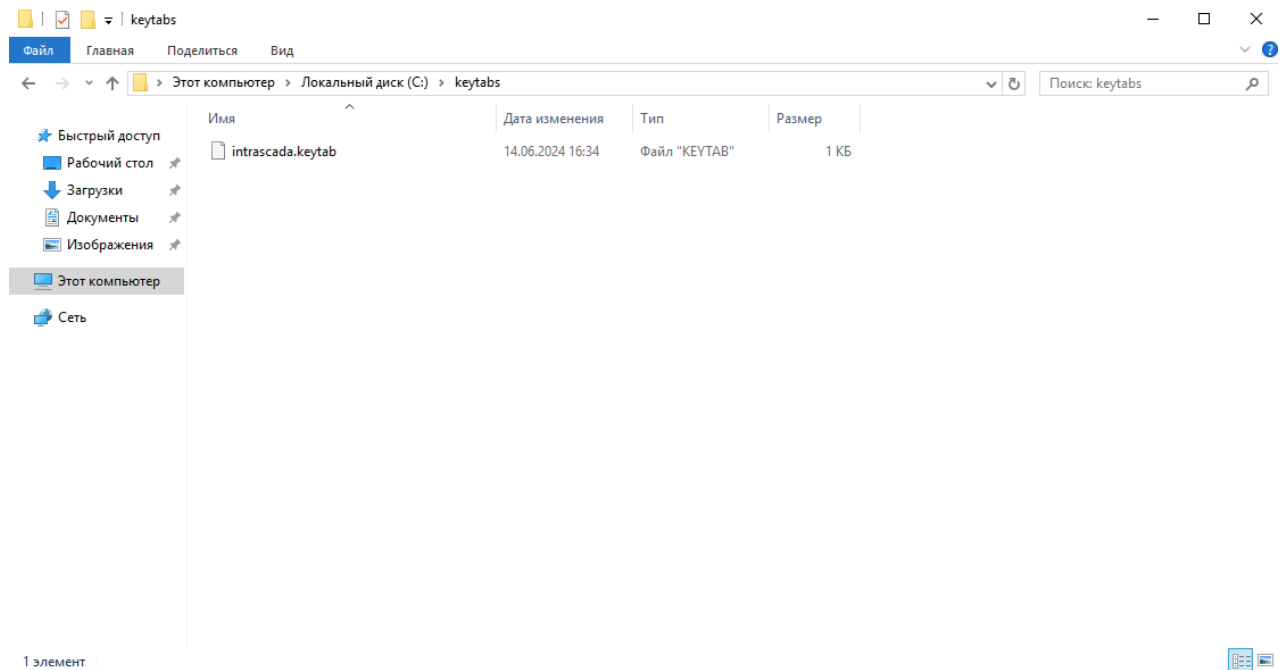
-pass xxxx - наш пароль

-target xxxx - наш лес

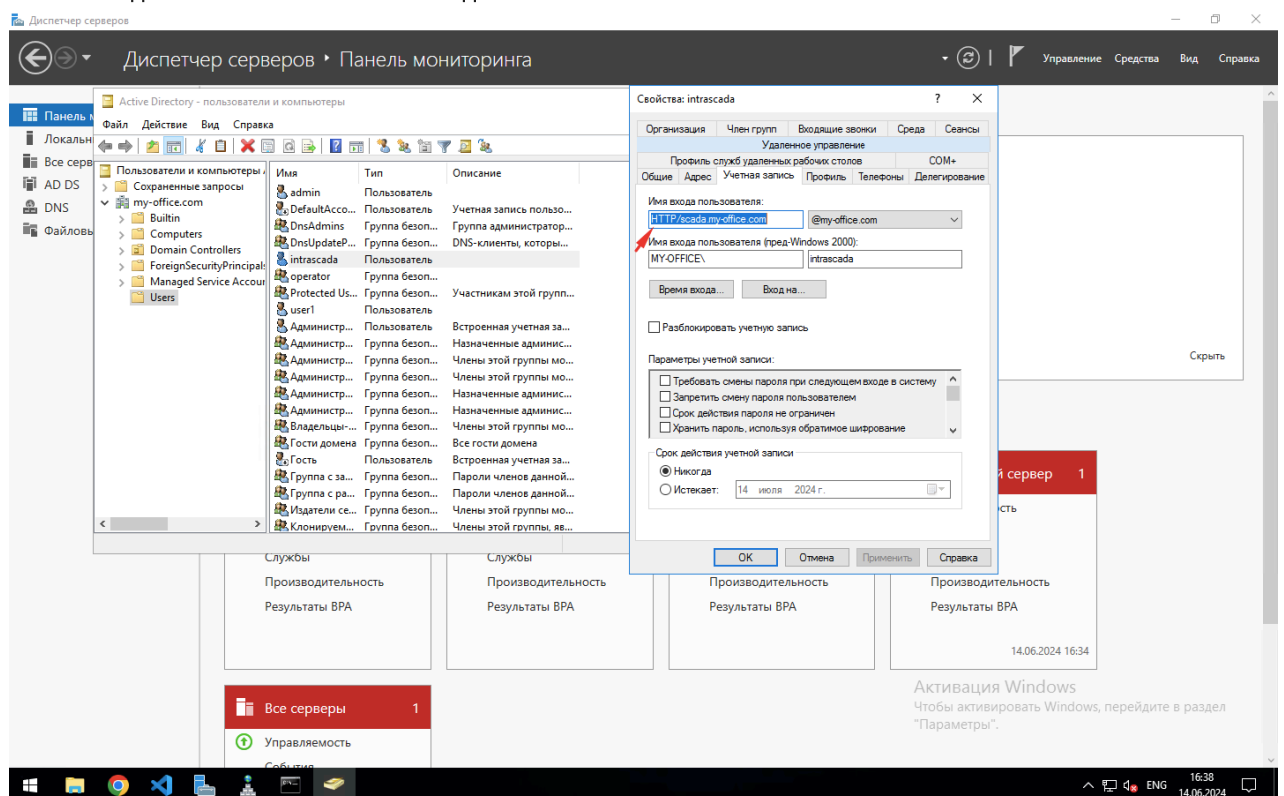
-out xxxx - путь к папке keytabs



- В созданной нами папке keytabs должен появиться файл, в нашем случае - intrascada.keytab



- В диспетчере серверов в разделе 'Пользователи и компьютеры Active Directory', в свойствах пользователя intrascada должно поменяться имя входа пользователя



## Копирование файла .keytab на сервер IntraSCADA

Задача заключается в простом копировании сгенерированного файла на сервер IntraSCADA под именем /etc/krb5.keytab.

Для удаленного копирования с Windows сервера можно, например, применить программу WinSCP.

- Находясь на Windows сервере подключаемся к серверу IntraSCADA по протоколу sftp и переносим keytab-файл в любую доступную папку

- Далее подключаемся к серверу IntraSCADA и копируем файл в /etc/krb5.keytab

```
sudo cp /home/intrascada/intrascada.keytab /etc/krb5.keytab
```

Где /home/intrascada - ваш путь к keytab-файлу

## Настройка браузера для работы с SSO

На компьютерах клиентов необходимо настроить браузер для работы с SSO.  
Мы рассмотрим настройку браузера на примере Chromium для РЕД ОС.

- Заходим в консоль, вводим команды:

```
cd /etc/chromium/policies/managed
```

```
su - root
```

Вводим пароль для root-доступа

```
touch policy.json
```

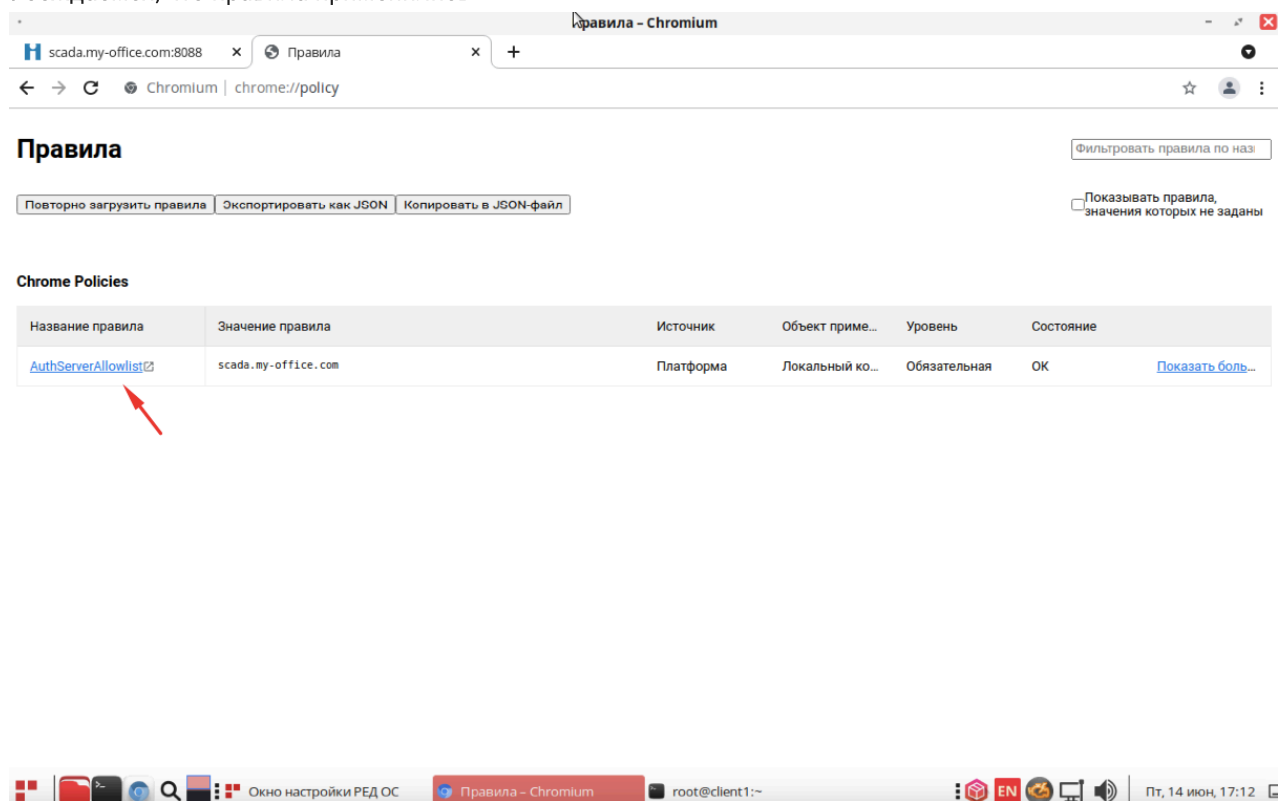
```
echo '{  
  "AuthServerAllowlist": "scada.my-office.com"  
}' > /etc/chromium/policies/managed/policy.json
```

Где scada.my-office.com - адрес вашего сервера

- Для проверки в браузере в поисковой строке вводим

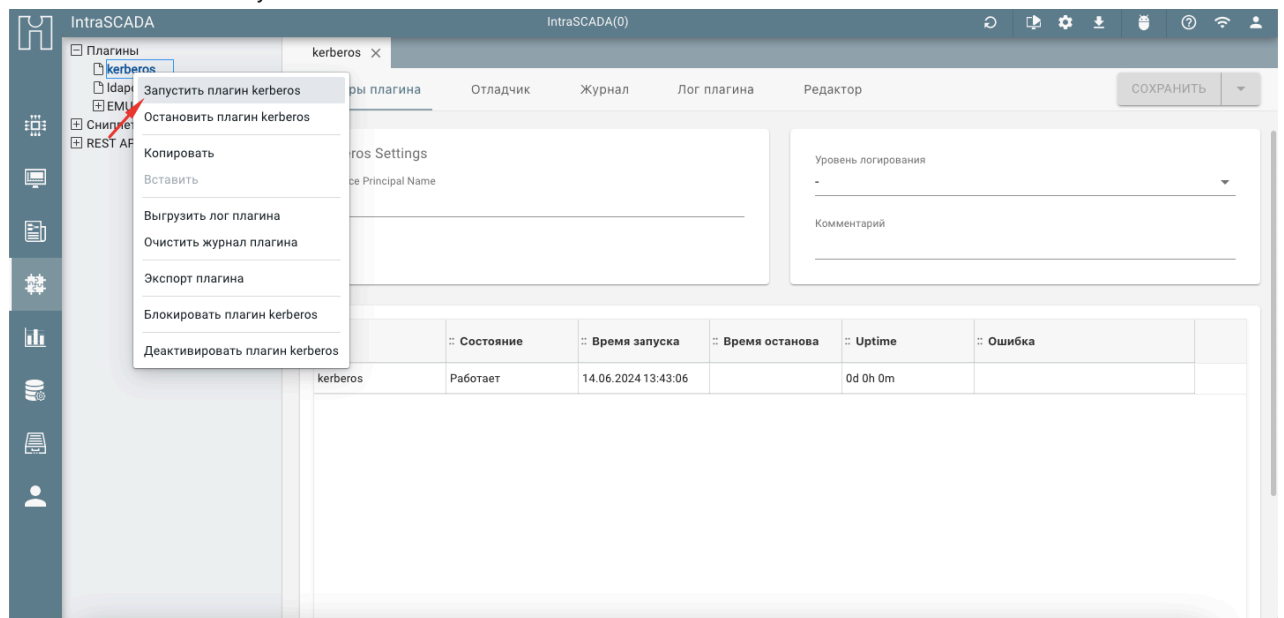
```
chrome://policy
```

- Убеждаемся, что правила применились

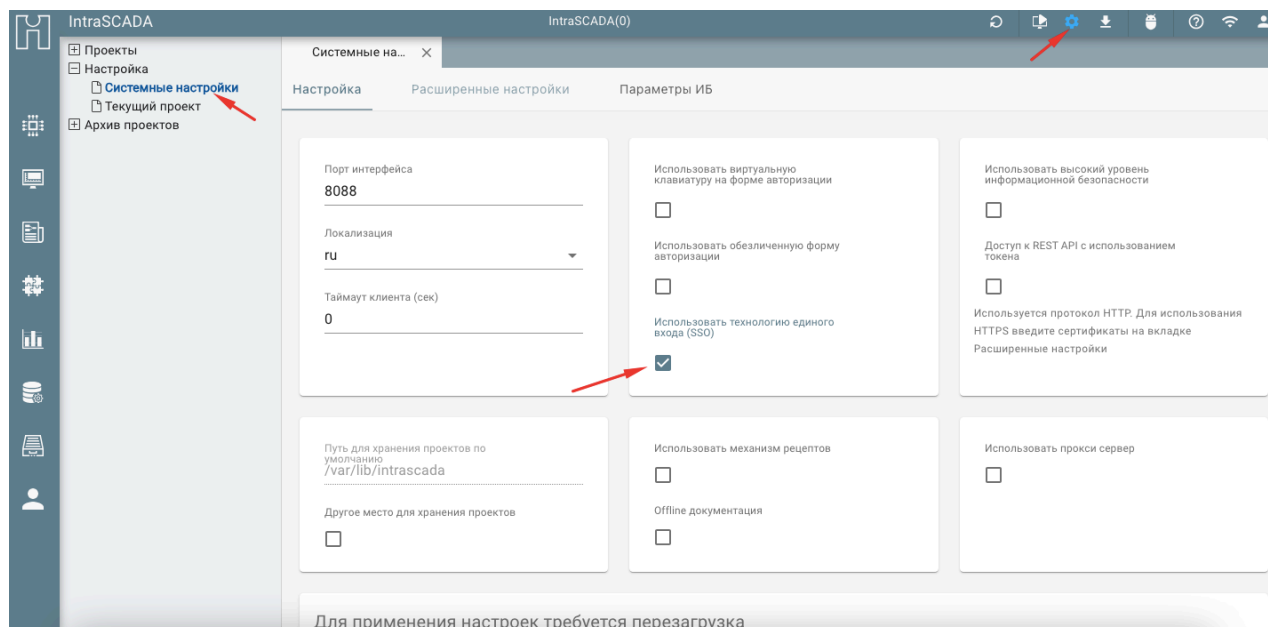


## Настройка сервера IntraSCADA

- Устанавливаем и запускаем плагин Kerberos в системе IntraSCADA



- Заходим в настройки сервера - системные настройки и ставим галку в поле **Использовать технологию единого входа(SSO)**



Затем перезагружаем сервер

На этом настройка SSO завершена

## LDAP client

Плагин обеспечивает вход в IntraSCADA с использованием учетных данных, хранящихся в вашем Active Directory / другом каталоге на основе LDAP.

Параметры плагина содержат настройки:

- Подключения к серверу LDAP
- Пути и фильтры для загрузки нужных учетных записей
- Маппинг атрибутов записей LDAP и системы IntraSCADA

Настройки существенно зависят от используемого LDAP сервера и структуры вашего LDAP каталога. Далее дано подробное описание **Настройки для Active Directory** и краткое **Настройки для FreeIPA**

## Подключение к серверу LDAP

- URL Формат URL-адреса - ldap://IP-адрес: номер порта. Поддерживается ldaps
- Логин и пароль учетной записи LDAP для выполнения операции импорта

IntraSCADA выполняет только чтение данных с сервера LDAP.

Рекомендуется создать специальную учетную запись на сервере LDAP для выполнения операций импорта (read-only-admin)

## Правила импорта

Для импорта данных нужно ввести **Путь** и **Фильтр** в формате, определяемом LDAP сервером.

Для **Атрибутов** нужно выполнить маппинг, то есть выбрать название атрибута в объекте LDAP, который будет соответствовать полю записи в IntraSCADA.

#### Параметры **Импорт групп**:

- Путь к группам - строка для получения групп с сервера LDAP (DIT)
- Фильтр для импорта - класс объекта по которому будет отфильтрован запрос в LDAP сервере

#### Параметры **Атрибуты группы**:

- Наименование - название атрибута в объекте LDAP, который будет соответствовать названию группы в системе
- Id группы - название атрибута в объекте LDAP, который будет соответствовать id группы в системе (*для AD не требуется*)

#### Параметры **Импорт учетных записей**:

- Путь к учетным записям - строка для получения учетных записей с сервера LDAP (DIT)
- Фильтр для импорта - класс объекта по которому будет отфильтрован запрос в LDAP сервере

#### Параметры **Атрибуты учетных записей**:

- Логин пользователя - название атрибута в объекте LDAP, который будет соответствовать логину пользователя в системе
- Имя пользователя - название атрибута в объекте LDAP, который будет соответствовать имени пользователя в системе
- Id учетной записи - название атрибута в объекте LDAP, который будет соответствовать id учетной записи в системе (*для AD не требуется*)

При использовании бинарного атрибута к названию атрибута добавьте без пробела `'binary'`.

Например, Id группы: **objectSid;binary**

Подробнее о синхронизации учетных записей в разделе [Доступ->Интеграция с LDAP](#)

## Настройка плагина

### Настройка для Active Directory

Настройка плагина LDAP на примере созданного нами ранее [Active Directory](#)

Заполняем поля по созданному нами ранее примеру(Название - intrascada, лес - my-office.com)

В качестве **Атрибута группы** Наименование можно выбрать **cn**.

Пример возвращаемого объекта группы с атрибутами с сервера Active Directory.

```
{
  dn: 'CN=Пользователи домена,CN=Users,DC=ih-systems,DC=lan',
  objectClass: [ 'top', 'group' ],
  cn: 'Пользователи домена',
  description: 'Все пользователи домена',
  distinguishedName: 'CN=Пользователи домена,CN=Users,DC=ih-systems,DC=lan',
  instanceType: '4',
  whenCreated: '20240527144211.0Z',
  whenChanged: '20240527144211.0Z',
  uSNCreated: '12348',
  memberOf: 'CN=Пользователи,CN=Builtin,DC=ih-systems,DC=lan',
  uSNChanged: '12350',
  name: 'Пользователи домена',
  objectGUID: '....',
  objectSid: '....',
  sAMAccountName: 'Пользователи домена',
  sAMAccountType: '268435456',
  groupType: '-2147483646',
  objectCategory: 'CN=Group,CN=Schema,CN=Configuration,DC=ih-systems,DC=lan',
  isCriticalSystemObject: 'TRUE',
  dSCorePropagationData: [ '20240527144211.0Z', '16010101000001.0Z' ]
}
```

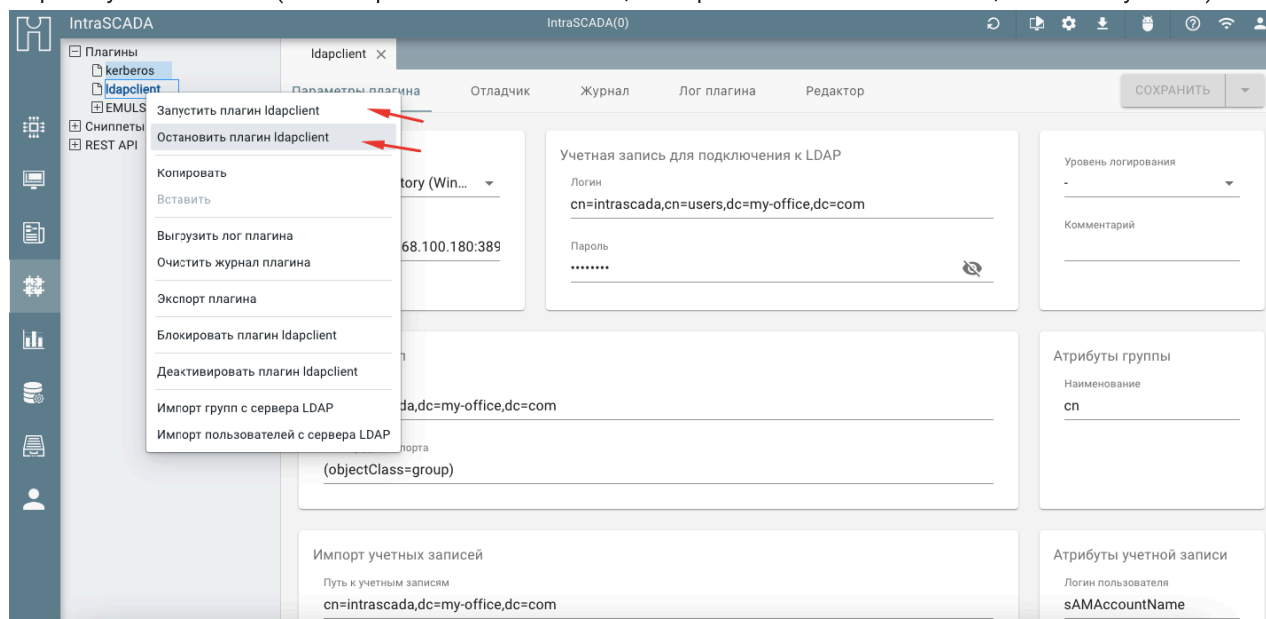
Пример возвращаемого объекта учетной записи с атрибутами с сервера Active Directory.

```
{
  dn: 'CN=admin1,CN=Users,DC=ih-systems,DC=lan',
  objectClass: [ 'top', 'person', 'organizationalPerson', 'user' ],
  cn: 'admin1',
  givenName: 'admin1',
  distinguishedName: 'CN=admin1,CN=Users,DC=ih-systems,DC=lan',
  instanceType: '4',
  whenCreated: '20240527150542.0Z',
  whenChanged: '20240529123214.0Z',
  displayName: 'admin1',
  uSNCreated: '12806',
  memberOf: [
    'CN=Администраторы основного уровня предприятия,CN=Users,DC=ih-systems,DC=lan',
    'CN=Администраторы основного уровня,CN=Users,DC=ih-systems,DC=lan',
    'CN=Администраторы домена,CN=Users,DC=ih-systems,DC=lan',
    'CN=Администраторы Hyper-V,CN=Builtin,DC=ih-systems,DC=lan',
    'CN=Пользователи удаленного рабочего стола,CN=Builtin,DC=ih-systems,DC=lan',
    'CN=Администраторы,CN=Builtin,DC=ih-systems,DC=lan'
  ],
  uSNChanged: '218625',
  name: 'admin1',
  objectGUID: '...',
  badPwdCount: '0',
  codePage: '0',
  countryCode: '0',
  objectSid: '...',
  adminCount: '1',
  accountExpires: '9223372036854775807',
  logonCount: '52',
  sAMAccountName: 'admin1',
  sAMAccountType: '805306368',
  objectCategory: 'CN=Person,CN=Schema,CN=Configuration,DC=ih-systems,DC=lan',
  lastLogonTimestamp: '133613004774614062',
  'msDS-SupportedEncryptionTypes': '16'
}
```

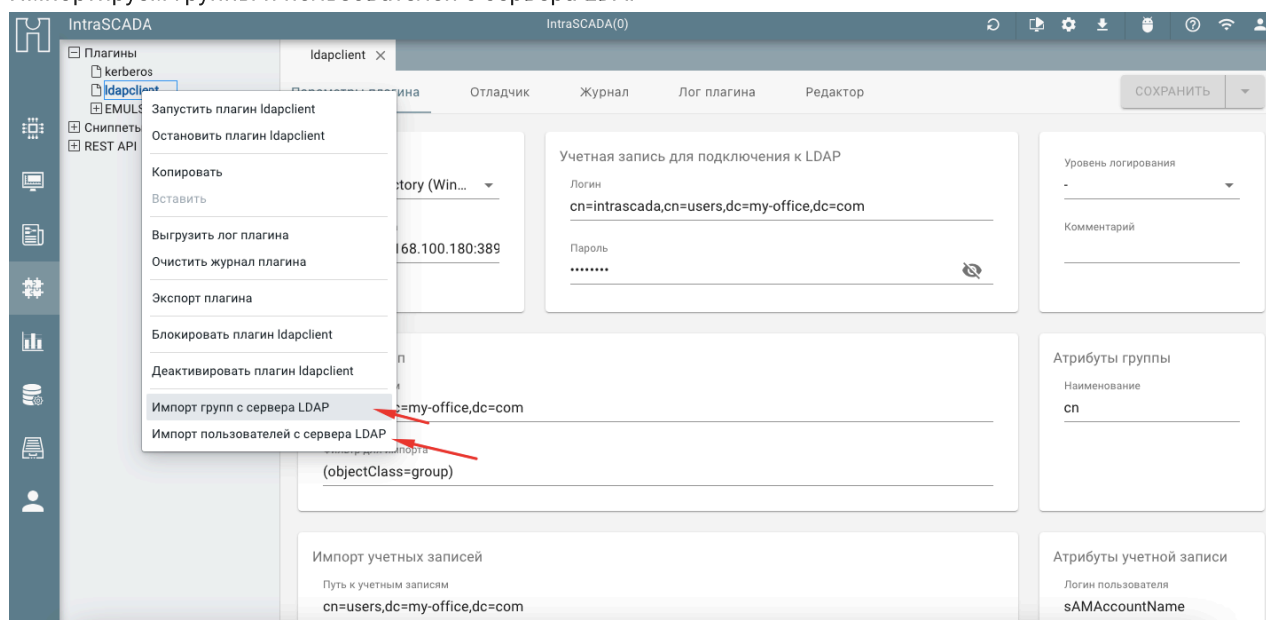
Атрибут **memberOf** используется при синхронизации, чтобы включить пользователя в соответствующие группы, загруженные в IntraSCADA.



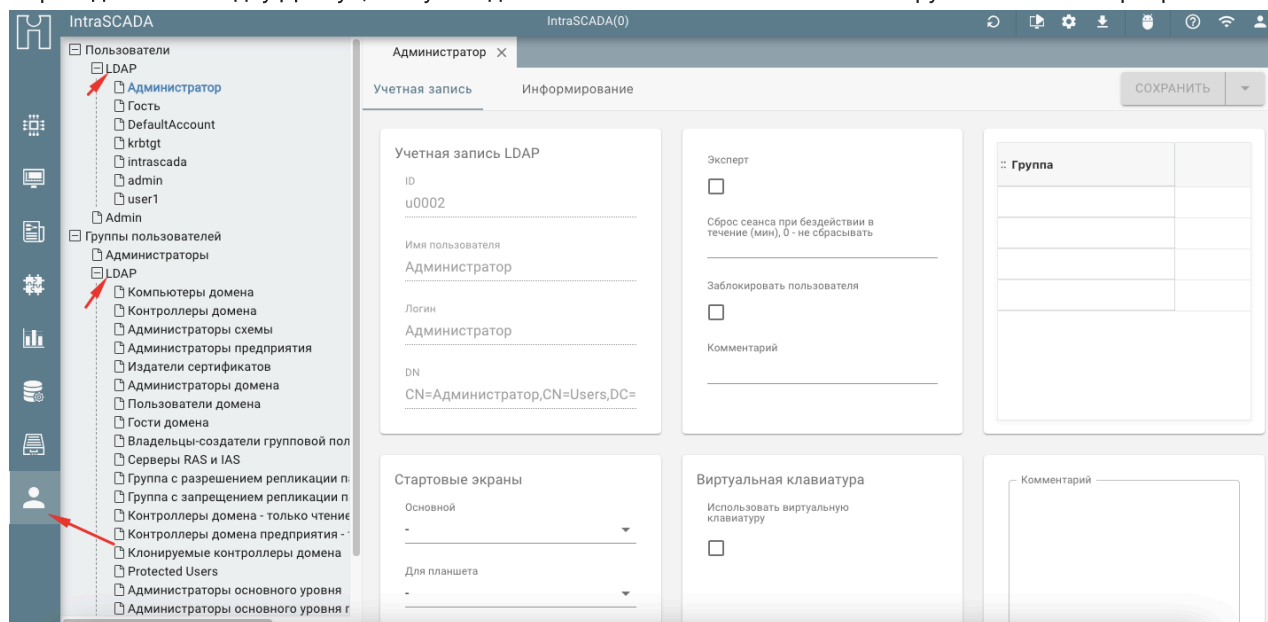
- Перезапускаем плагин(Жмем правой кнопкой мыши, выбираем - остановить плагин, потом запускаем)



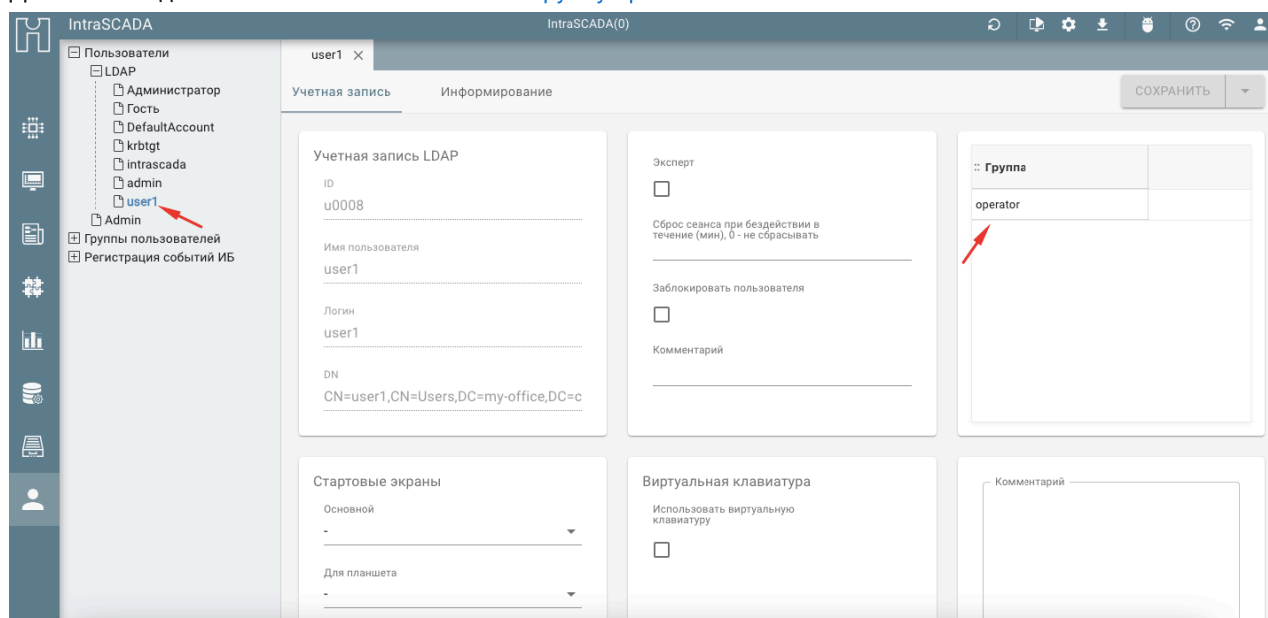
- Импортируем группы и пользователей с сервера LDAP



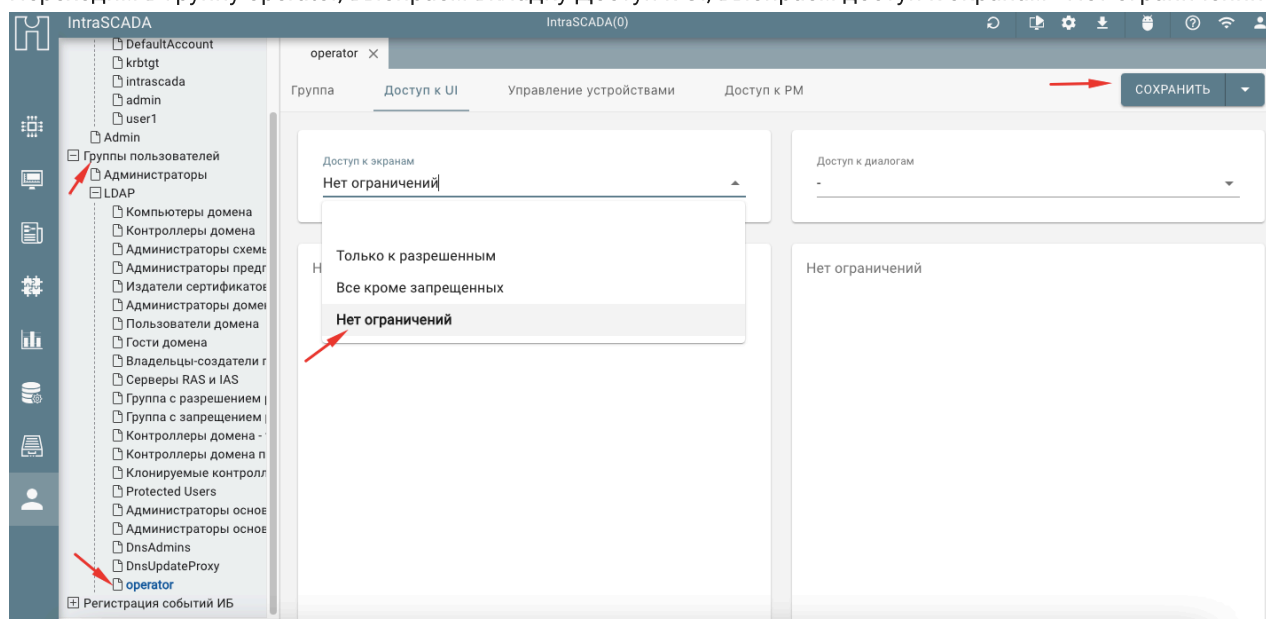
- Переходим во вкладку Доступ, там у нас должны появиться пользователи и группы из LDAP-сервера



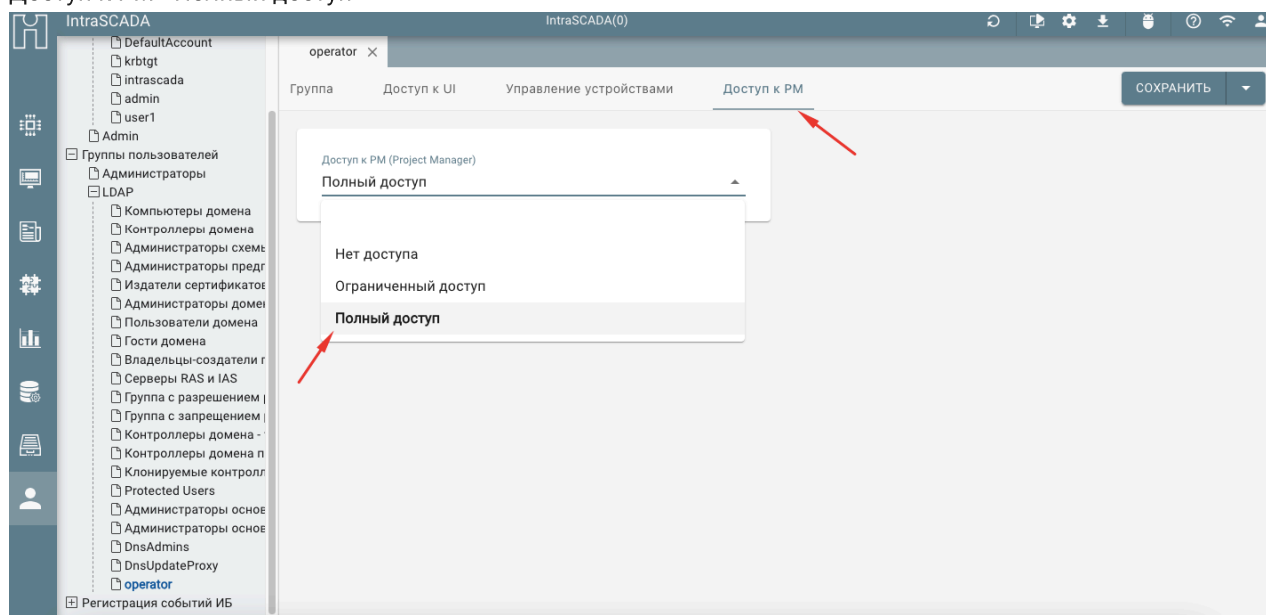
- До этого мы добавили пользователя **user1** в группу **operator**



- Переходим в группу operator, выбираем вкладку Доступ к UI, выбираем доступ к экранам - Нет ограничений



- Доступ к PM - Полный доступ



- После этого можно зайти в систему под логином user1 и с назначенными нами правами доступа.

## Настройка для FreeIPA

На данном скриншоте показано, как должны выглядеть настройки для FreeIPA

ldapclient X

Параметры плагина

Отладчик

Журнал

Лог плагина

Редактор

СОХРАНИТЬ

Сервер LDAP

Другой LDAP сервер

Адрес сервера

ldap://192.168.100.183:389

Учетная запись для подключения к LDAP

Логин

uid=admin,cn=users,cn=accounts,dc=my-company,dc=lan

Пароль

.....

Уровень логирования

-

Комментарий

Импорт групп

Путь к группам

cn=groups,cn=accounts,dc=my-company,dc=lan

Фильтр для импорта

(objectClass=groupofnames)

Атрибуты группы

Наименование

cn

ID группы

gidNumber

Импорт учетных записей

Путь к учетным записям

cn=users,cn=accounts,dc=my-company,dc=lan

Фильтр для импорта

(objectClass=person)

Атрибуты учетной записи

Логин пользователя

uid

Имя пользователя

cn

ID учетной записи

uidNumber

## Команды плагина

Плагин поддерживает следующие параметрические команды для синхронизации с LDAP сервером:

- **syncGroups** - синхронизация групп
- **syncUsers** - синхронизация пользователей

Эти команды отправляются плагину при вызове операций импорта из меню плагина.

Команду также можно отправить через метод сценария **pluginCommand**.

## Примеры сценариев синхронизации с сервером LDAP

*Сценарии можно запускать по расписанию, если необходимо регулярно обновлять данные.*

Для корректной работы сценариев требуется версия плагина не ниже v5.17.0

### Синхронизация пользователей

Если состав групп, с которыми работает IntraSCADA, не меняется (или меняется редко), имеет смысл вызывать синхронизацию пользователей отдельно.

Сценарий отправляет команду с параметром **syncUsers** плагину **ldapclient**. Результат пишется в главный журнал. Предусмотрен выход по таймауту, чтобы сценарий завершился в случае проблем со стороны плагина.

```

const script = {
  start() {
    this.timeout = 10; // Время таймаута в секундах.
    this.pluginCommand(
    {
      unit: 'ldapclient',
      type: 'command',
      param: 'syncUsers'
    },
    'onSyncUsers'
  );
    this.startTimer('T1', this.timeout, 'onTimeout');
  },

  onSyncUsers(response) {
    if (!response.response) {
      this.mainlog(
        'LDAP: Ошибка при синхронизации пользователей: ' +
        response.message
      );
    } else {
      this.mainlog('LDAP: Синхронизация пользователей завершена.');
```

Таймаут следует подбирать индивидуально, так как операция загрузки с сервера LDAP при больших объемах может занять длительное время!

## Синхронизация групп и пользователей

Если группы тоже нужно постоянно синхронизировать, необходимо загрузить сначала группы, затем пользователей.

```

const script = {
  start() {
    this.timeout = 10;
    this.pluginCommand(
    {
      unit: 'ldapclient',
      type: 'command',
      param: 'syncGroups'
    },
    'onSyncGroups'
  );
    this.startTimer('T1', this.timeout, 'onTimeout');
  },

  onSyncGroups(response) {
    if (response.response) {
      this.pluginCommand({unit:'ldapclient', param:'syncUsers'}, 'onSyncUsers');
    } else {
      this.mainlog('LDAP: Ошибка при синхронизации групп: ' + response.message);
      this.exit();
    }
  },

  onSyncUsers(response) {
    if (!response.response) {
      this.mainlog(
        'LDAP: Ошибка при синхронизации пользователей: ' +
        response.message
      );
    } else {
      this.mainlog('LDAP: Синхронизация пользователей завершена. ');
    }
    this.exit();
  },

  onTimeout() {
    this.mainlog(
      'LDAP: Синхронизация не выполнена – истек таймаут ' +
      this.timeout +
      ' сек'
    );
    this.exit();
  }
};

```

# Плагин MODBUS master (client)

## Описание

Плагин реализует функции Modbus master (client) и позволяет подключиться к Modbus slave (server).

На текущий момент поддерживаются следующие возможности:

- Поддерживаемые протоколы:
  - Modbus TCP
  - Modbus RTU
  - Modbus RTU over TCP
  - Modbus ASCII
- Возможность запуска множества экземпляров плагина с индивидуальными настройками
- Автоматическая/Ручная группировка каналов при опросе
- Настройка порядка байт (byte order) как для экземпляра плагина в целом, так и для отдельного канала
- Возможность отправлять на сервер только изменения полученных данных с Modbus устройств. Может использоваться для снижения нагрузки на основной процесс при использовании большого количества экземпляров плагина и высокой частоте опроса
- Механизм простого создания каналов для идентичных устройств с разными Unitld
- Изменение статуса каналов в зависимости от состояния плагина и доступности конечного устройства  
Подробнее [Каналы с индикацией связи](#)

## Настройка

### Настройка подключения

Для настройки подключения необходимо во вкладке **Параметры плагина** выбрать один из протоколов взаимодействия (транспорт). В зависимости от транспорта настраиваются соответствующие параметры связи. Для TCP достаточно настроить IP и порт. Для RTU настроек больше.

Транспорт <b>Modbus RTU</b>	Ожидание ответа на запрос (ms) <b>5000</b>
Порт <b>/dev/ttyUSB0</b>	Интервал между запросами (ms) <b>200</b>
Скорость, бод/с <b>115200</b>	Мах кол-во слов при чтении диапазона <b>125</b>
Контроль четности <b>even</b>	Настроить порядок байт <input type="checkbox"/>
Биты данных <b>8</b>	Отправлять только изменения <input type="checkbox"/>
Стоп-биты <b>1</b>	

## Настройка каналов плагина

Канал имеет следующие параметры:

- Адрес устройства на шине (Для TCP не используется, можно оставить равным 1)
- Адрес регистра в шестнадцатеричном (0x0001) или десятичном (1) формате
- Тип переменной (BOOL, INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, FLOAT, DOUBLE, STRING UTF-8, STRING ASCII)
- Функция чтения (FC1 - Read Coils, FC2 - Read Discrete Inputs, FC3 - Read Holding Registers, FC4 - Read Input Registers)
- Порядок байт для канала.  
Такая необходимость может возникнуть при опросе устройств разных производителей на одной шине.
- Извлечь битовое значение. Можно выполнять чтение байтами, а затем извлекать бит, указав смещение. Например, битовые флаги доступны для чтения как WORD (UINT16). Для каждого битового флага создается канал с одним и тем же адресом, но разными смещениями.
- Формула преобразование входного значения (например, value/10).
- Группировка при чтении. Необходима для оптимизации количества запросов к Modbus устройствам.
  - Автоматическая группировка выполняется плагином
  - Ручная группировка может быть выполнена при настройке каналов
- Коэффициент времени опроса. Данный параметр становится доступным только при ручной группировке.



Привязка к устройству



Unit ID (адрес устройства на шине)

1

Канал

ch1

Свойство для привязки

Чтение



Запись



Адрес регистра (dec or hex)

0x3000

Тип переменной

UINT16, 2 байта

Порядок байт для канала



Порядок байтов для 2-байтовых значений

Big-Endian, B1B2 => B1B2

Функция Modbus

FC3 - Read Holding Register

Извлечь битовое значение



Смещение

2

Группировать при чтении



Ручная группировка



Название группы

gr1

Коэффициент времени опроса

1

Если канал необходимо читать периодически, то устанавливаем галочку **Чтение**. Если будет установлена галочка **По Запросу**, то канал будет исключен из общего цикла периодического опроса.

## Механизмы группировки

Группировка позволяет оптимизировать количество запросов к Modbus устройствам.

За один запрос могут быть опрошены до 125 регистров, если они входят в один диапазон.

## Автоматическая группировка

Плагин сам группирует последовательно идущие адреса внутри каждого UnitId (допускаются разрывы) с учетом функции чтения и параметра плагина "Мах кол-во слов при чтении диапазона".

Кроме установки галочки **Группировать при чтении** никаких дополнительных действий по настройке не требуется.

## Ручная группировка

Позволяет вручную объединять регистры для группового чтения.

Для этого вводится **Название группы** - произвольный идентификатор (латинские буквы+цифры+знак подчеркивания). Названия группы уникальны в рамках одного узла или если не используются узлы, то в рамках плагина

Обычно используется в двух случаях:

- Если в диапазоне есть адреса, при опросе которых slave возвращает ошибку, и мы не можем опросить весь диапазон. Ручная группировка позволяет разбить диапазон на более мелкие части, не изменяя общий параметр "Мах кол-во слов при чтении диапазона"
- Есть необходимость опроса одних регистров реже чем других. При использовании ручной группировки становится доступным параметр **Коэффициент цикла опроса**. Он позволяет настроить периодичность опроса данной группы.  
По умолчанию значение равно 1 - это значит, что группа будет включена в каждый цикл опроса.

## Цикл опроса

Цикл опроса обычно состоит из нескольких запросов к устройству, например:

- чтение регистров с адреса 0x0010 - 0x0020 командой FC1 - автоматическая группировка
- чтение регистра с адреса 0x0023 командой FC1 - Флаг группового чтения снят
- чтение регистров с адреса 0x3000 - 0x3042 командой FC3 - автоматическая группировка
- чтение регистров с адреса 0x3A02 - 0x3A1D командой FC3 - ручная группировка 'group\_1' (например, адреса 0x3A1E - 0x3A1F выдают ошибку при чтении FC3)
- чтение регистров с адреса 0x3A20 - 0x3A55 командой FC3 - ручная группировка 'group\_2'
- ...

То есть все зависит от количества регистров и автоматической или ручной группировки.

Если нужно опрашивать какой-то диапазон адресов реже, то можно использовать **Ручную группировку** и **Коэффициент цикла опроса** больше 1.

Например, установить для 'group\_1' коэффициент равный 5, тогда эта группа будет опрашиваться только в каждом пятом цикле.

## Запись

Если канал предполагает запись, нужно выставить галочку **Запись**

По умолчанию запись выполняется по тому же адресу, что и чтение. Команду плагин выбирает исходя из типа канала (BOOL - FC5, иначе FC6 или FC16 в зависимости от количества байт, которые нужно записать). То есть обычно настройка записи не требуется

Бывает, что есть необходимость в рамках одного канала читать с одного регистра, а писать в другой (например, контроллер принимает значение в регистр для записи и затем переписывает его в регистр для чтения или отвергает присланное значение).

В этом случае можно установить галочку **Для записи использовать другой адрес** и ввести адрес и тип переменной для записи.

Формула расчета входного значения <input type="text" value="value/100"/> Инвертировать входное значение <input type="checkbox"/>	Канал <input type="text"/> Устройство <input type="text"/>
Для записи использовать другой адрес <input checked="" type="checkbox"/> Адрес регистра (dec or hex) <input type="text" value="0x0000"/> Тип переменной <input type="text" value="UINT8, 1 byte"/> Формула расчета выходного значения <input type="text"/>	Тест записи Записать значение <input type="text"/> <input type="button" value="ЗАПИСАТЬ"/> FC6 - Write Single Register

## Тест записи

Можно проверить работу канала без привязки к устройству. Кнопка **Записать** дает возможность выполнить операцию записи значения из поля **Записать значение**.

## Коды ошибок

Плагин может завершаться с ошибками. Возможные коды ошибок:

- 1 - отсутствует связь с контроллером
- 2, 8 - отсутствуют каналы
- 42 - все каналы не отвечают и имеют статус 1

## Команды плагина

Плагин поддерживает следующие команды из сценария и скриптов визуализации:

- **readOnReq** - команда предназначена для получения данных по запросу. С помощью данной команды можно передать в плагин запрос на разовое получение данных с каналов типа **По Запросу**

Пример для скрипта визуализации:

```

module.exports = async function({ local, context, source }, core, debug) {
  //Получение каналов из устройства
  const arr = await core.getDeviceChannels({did: 'd1113', unit: 'modbus7'});
  //Команда плагину
  const result = await core.pluginCommand(
    {
      unit: 'modbus7',
      command: 'readOnReq',
      data: arr
    },
    10
  );
};

```

Пример для сценария:

```

const script = {
  start() {
    this.getDeviceChannels({did: 'd1113', unit: 'modbus7'}, 'onGet');
  },

  onGet(error, data) {
    if (!error) {
      this.pluginCommand(
        {
          unit: 'modbus7',
          command: 'readOnReq',
          data
        },
        'getResponse'
      );

      } else {
        this.log('Ошибка: ' + error)
      }
    },

    getResponse(message) {
      this.log(message);
    }
  };

```

- **writeWordArray** - команда предназначена для записи массива регистров одной командой.

Для корректной работы данной команды необходимо, чтобы в объекте **data** были следующие свойства:

- unitid - id модбас устройства на шине
- address - начальный адрес массива
- value - массив регистров

Пример для скрипта визуализации:

```
module.exports = async function({ local, context, source }, core, debug) {
  const data = {unitid:1, address: 10, value: [122, 223, 334]}

  //Команда плагину
  const result = await core.pluginCommand(
    {
      unit: 'modbus7',
      command: 'writeWordArray',
      data: data
    },
    10
  );
};
```

Пример для сценария:

```
const script = {
  start() {
    const data = {unitid:1, address: 10, value: [122, 223, 334]}
    this.pluginCommand(
      {
        unit: 'modbus7',
        command: 'writeWordArray',
        data
      },
      'getResponse'
    );
  },

  getResponse(message) {
    this.log(message);
  }
};
```

# Плагин Modbus TCP/RTU Server

## Описание

Плагин позволяет публиковать данные из системы в протокол Modbus TCP/RTU

Может работать в качестве шлюза из любого поддерживаемого системой протокола в Modbus TCP/RTU

На текущий момент поддерживаются следующие возможности:

- Все основные функции протокола Modbus FC-1,2,3,4,5,6,15,16
- 4 области памяти для каждого типа переменной Modbus **Coil, Discrete Input, Holding Registers, Input Registers** с максимально возможной памятью для адресации.
- Порядок байт (byte order) с которым данные из системы передаются в плагин, настраивается в общих настройках плагина

Для корректной работы с панелями Weintek необходимо установить порядок байт для 4 байтных значений **Big-Endian swap** и использовать драйвер Modbus TCP/IP (Zero-based Addressing) в панели.

## Настройка

### Настройка сервера TCP

Для настройки необходимо указать свободный порт TCP и unitId

Порт

1502

unitID

1

## Настройка сервера RTU

Для настройки необходимо указать свободный последовательный порт RTU и unitId

Выбрать порядок байт для записи и чтения данных системой из регистров Modbus плагина

Позиция байта в слове (для 1-байтовых значений)

Big-Endian, B0 => B0

Порядок байтов для 2-байтовых значений

Big-Endian, B1B2 => B1B2

Порядок байтов для 4-байтовых значений

Big-Endian, swap B1B2B3B4 => B3B4B1B2

Порядок байтов для 8-байтовых значений

Big-Endian, B1B2B3B4B5B6B7B8 => B1B2B3B4B5B6B7B8

## Расширение плагина

Расширение плагина используется для публикации данных из системы в область памяти Modbus/TCP сервера и обратно. Для привязки свойств устройств к регистрам Modbus необходимо:

1. Выбрать устройство
2. Прописать свойство
3. Указать адрес в Modbus
4. Выбрать тип переменной
5. Выбрать тип данных

:: Устройство	:: Свойство	:: Адрес регистра (dec or hex)	:: Тип переменной	:: Функция Modbus
DD_001 • Датчик движения ▼	state	0	BOOL ▼	Coil ▼
Dimm_001 • Диммер ▼	value	3	UINT8, 1 байт ▼	Holding Register ▼
DG_001 • Датчик открытия ▼	state	1	BOOL ▼	Coil ▼
AO_001 • Актуатор аналоговый ▼	value	0	FLOAT, 4 байта ▼	Input Register ▼

Типы **Holding Register** и **Coil** доступны для чтения и записи

Типы **Input Register** и **Discrete Input** доступны только для чтения

# Плагин MQTT

## Описание

Плагин позволяет подключиться к Mqtt брокеру по протоколам MQTT и MQTTS.

На текущий момент поддерживаются следующие возможности:

- Подключение к серверу используя шифрованное MQTTS и нешифрованное MQTT подключение
- Аутентификация на брокере используя логин и пароль
- Сканирование данных на брокере
- Подписка на изменение данных на брокере
- Запись данных на брокер

## Настройка

### Настройка подключения

Для настройки подключения к брокеру достаточно указать IP адрес и порт брокера, выбрать тип протокола (MQTT или MQTTS), указать логин и пароль если необходимо.

<p>MQTT broker URL</p> <p>127.0.0.1</p> <hr/> <p>MQTT broker port</p> <p>1883</p> <hr/> <p>Protocol</p> <p>mqtt</p> <hr/>	<p>Использовать аутентификацию</p> <p><input checked="" type="checkbox"/></p> <p>Логин</p> <p>login</p> <hr/> <p>Пароль</p> <p>.....</p> <hr/> <p>Брать время из сообщения (JSON)</p> <p><input type="checkbox"/></p>
---	---

### Сканирование узлов

Для сканирования узлов брокера необходимо перейти на вкладку каналы и правой кнопкой мыши на папке ALL вызвать всплывающее окно, где нажать на "Сканировать каналы". Появится всплывающее окно, где необходимо нажать кнопку "Сканировать" для начала сканирования дерева брокера. Как только данные



поступят на брокер, вы сразу увидите их в дереве:

×

Сканировать

zigbee2mqtt

bridge

motion = {"battery":86,"linkquality":6}

tele

tasmota\_316BBC

tasmota

discovery

devices

dn = 0

ОСТАНОВИТЬ

ДОБАВИТЬ КАНАЛЫ

Topic	Channel
-------	---------

## Добавление переменных для мониторинга и записи

Для мониторинга и записи необходимых переменных, которые присутствуют в дереве, вам достаточно щелкнуть на них два раза левой кнопкой мыши и переменные добавятся в список справа. После этого нажимаем на кнопку "Добавить каналы" и данные переменные появятся в списке каналов плагина.

×

Сканировать

zigbee2mqtt

bridge

state = online

config = {"commit":"ed8b4e5"

info = {"commit":"ed8b4e5","c

devices = [{"definition":"null","er

groups = []

logging = {"level":"info","mess

motion = {"battery":86,"linkquality

temp\_parent = {"battery":37,"hum

boiler = {"consumption":4.1,"linkq

set = ON

tele

tasmota\_316BBC

LWT = Offline

tasmota

discovery

807D3A316BBC

config = {"ip":"192.168.1.:

sensors = {"sn":"Time":2

ОСТАНОВИТЬ

ДОБАВИТЬ КАНАЛЫ

Topic	Channel
zigbee2mqtt/motion	zigbee2mqtt_motion
zigbee2mqtt/temp_parent	zigbee2mqtt_temp_parent
zigbee2mqtt/boiler/set	zigbee2mqtt_boiler_set

## Добавление узла с функцией обработчика

В том случае, когда в MQTT топик приходят данные например в виде JSON, то для парсинга можно воспользоваться функцией обработчика в узле, которая может вернуть данные в каналы этого узла. Пример функции обработчика:

```
module.exports = function(data, debug) {
  const parsed = JSON.parse(data);
  debug(data);
  return {ch_4:parsed.value, ch_2:parsed.state}
};
```

Так же возможно вернуть данные с меткой времени и статусом канала. Пример функции обработчика:

```
module.exports = function(data, debug) {
  const parsed = JSON.parse(data);
  debug(data);
  return {ch_4:{value: parsed.value, ts:parsed.ts, chstatus: parsed.q}}
};
```

## Редактирование каналов плагина

В каналах можно настроить первичную обработку поступивших и отправляемых значение с помощью поля **Формула извлечения значения** и **Сообщение для публикации**. Здесь есть возможность написать тернарную

операцию вот такого типа:

<p>Канал</p> <p>zigbee2mqtt_state_boiler</p> <hr/> <p>Название</p> <hr/> <p>Свойство устройства при автоматической привязке</p> <p>state</p> <hr/>	<p>Чтение (подписка)</p> <p><input checked="" type="checkbox"/></p> <p>Запись (публикация)</p> <p><input checked="" type="checkbox"/></p>
<p>Топик для подписки</p> <p>zigbee2mqtt/boiler</p> <hr/> <p>Формула извлечения значения</p> <p>JSON.parse(value).state == 'ON' ? 0 : 1</p> <hr/>	
<p>Топик для публикации</p> <p>zigbee2mqtt/boiler/set</p> <hr/> <p>Сообщение для публикации</p> <p>value == 1 ? 'OFF' : 'ON'</p> <hr/>	
<p>Тест записи</p> <p>Сообщение для публикации</p> <hr/> <div>ЗАПИСАТЬ</div>	

Кнопка **Записать** дает возможность отправить команды непосредственно из плагина для этого необходимо заполнить поле **Тест записи -> Сообщение для публикации** и нажать на кнопку. Данное сообщение уйдет непосредственно в топик для публикации

## Примеры для расчета входного/выходного значения в канале

### Пример 1

Необходимо обработать входное значение, которое поступило в виде JSON строки -

`{"battery":37,"humidity":66.87,"linkquality":31,"temperature":23.74,"voltage":2865}` и передать в канал только значение temperature

Для этого в поле **Формула извлечения значения** необходимо написать следующую формулу:

```
JSON.parse(value).temperature
```

## Пример 2

Необходимо обработать входное значение, которое поступило в виде обычной строки - 'ON' и передать в канал 1 или 0.

Для этого в поле **Формула извлечения значения** необходимо написать следующую формулу:

```
value == 'ON' ? 1 : 0
```

## Пример 3

Необходимо обработать выходное сообщение, которое поступило со свойства устройства и опубликовать данные в определенном JSON формате на брокер.

Для этого в поле **Сообщение для публикации** необходимо написать следующую формулу:

```
'{"brightness": '+value+', "turn": "on"}'
```

## Пример 4

Необходимо обработать выходное сообщение, которое вызывается с помощью команды и не имеет передаваемых параметров, а имеет жестко фиксированную строку 'ON'.

Для этого в поле **Сообщение для публикации** необходимо написать следующую формулу:

```
'ON'
```

Обратите внимание: строка должна быть в кавычках.

## Расширение плагина

Расширение плагина используется для интеграции устройств системы со сторонними сервисами по MQTT протоколу. В данном разделе вы указываете какие устройства и их свойства будут публиковаться на брокер, принимать значения с брокера для присвоения значений свойствам устройств или выполнения команд.

Для добавления устройства правой кнопкой мыши нажмите на поле таблицы и добавьте новую строку. В данной строке необходимо выбрать устройство, указать свойство, операцию и топик. Для базовой настройки этого

ДОСТАТОЧНО

mqttclient1 X

Параметры плагина

Каналы

Таблица каналов

Расширение

Отладчик

Журнал

>

СОХРАНИТЬ

▼

:: Устройство	:: Свойство или команда	:: Операция	:: Топик	:: Сообщение
DD_001 • Датчик движения ▼	state	Публиковать значение ▼	DD_001/state	
Dimm_001 • Диммер ▼	value	Принять значение ▼	Dimm_001/value	
H_001 • Светильник ▼	off	Принять команду ▼	H_001/cmd	0
H_001 • Светильник ▼	on	Принять команду ▼	H_001/cmd	1
H_001 • Светильник ▼	state	Принять значение ▼	light/lamp1/cmd	
H_001 • Светильник ▼	state	Публиковать значение ▼	light/lamp1/state	
DG_001 • Датчик открытия ▼	state	Публиковать значение ▼	dg_001/state	
HB_001 • Радиатор отопления ▼	off	Принять команду ▼	hb_001/cmd	0
HB_001 • Радиатор отопления ▼	on	Принять команду ▼	hb_001/cmd	1
HB_001 • Радиатор отопления ▼	state	Публиковать значение ▼	h_001/ddd1/state1	

Так же для каждого расширения существует дополнительные свойства QoS, RETAIN, Глубина Буфера. Первые два работают согласно спецификации MQTT 3.1. А вот глубина буфера указывается для тех свойств, которые необходимо публиковать на брокер при этом при обрыве связи копить изменения, которые не были отправлены. При восстановлении связи все накопленные данные по каждому топику уйдут одной публикацией с метками времени в виде массива. Если использовать наш MQTT клиент на принимающей стороне, то эти данные будут отправлены в систему, как исторические и записаны в базу данных с теми же метками времени, которые были в сообщении.

# Плагин MQTT Server

## Описание

Плагин позволяет создать брокер MQTT для интерфейсов TCP, Websocket, TCP over TLS

На текущий момент поддерживаются следующие возможности:


- Авторизация по логину и паролю
- Использование самоподписных сертификатов для создания защищенного подключения

## Настройка

### Настройка сервера

Для настройки необходимо выбрать интересующий вас MQTT тип интерфейса и указать порт.

Для MQTT over TLS необходимо указать полный путь к private-key.pem и public-cert.pem

<div>Protocol MQTT</div> <div><input checked="" type="checkbox"/></div>	<div>Использовать аутентификацию</div> <div><input checked="" type="checkbox"/></div>
<div>Port MQTT</div> <div>1883</div>	<div>Логин</div> <div>login</div>
<div>Protocol MQTT over TLS</div> <div><input checked="" type="checkbox"/></div>	<div>Пароль</div> <div>.....</div>
<div>Port MQTTS</div> <div>8883</div>	
<div>Private KEY file path</div> <div>/var/lib/ih-v5/plugins/mqttserver/private</div>	
<div>Public CERT file path</div> <div>/var/lib/ih-v5/plugins/mqttserver/public</div>	
<div>Protocol MQTT over websocket</div> <div><input checked="" type="checkbox"/></div>	
<div>Port MQTTWS</div> <div>8888</div>	

[Инструкция по генерации самоподписных сертификатов](#)

# Плагин Omron Fins клиент

## Описание

Плагин позволяет подключиться к контроллерам Omron серии CP, CV, CS, CJ, NJ, NX используя протокол FINS

На текущий момент поддерживаются следующие возможности:

- Подключение к контроллерам Omron через TCP с возможностью выбрать порт (по умолчанию 9600)
- Получение данных из регистров контроллера периодическим опросом
- Запись данных в регистры контроллера

## Настройка

### Настройка подключения

Для настройки подключения необходимо выбрать тип соединения **TCP/UDP** и ввести IP адрес и порт (стандартный порт для Omron - 9600). Так же необходимо указать интервал между опросами и таймаут в миллисекундах, количество регистров в одном запросе.

<p>ID</p> <p>omronfins1</p> <hr/> <p>Название</p> <hr/> <p>IP</p> <p>127.0.0.1</p> <hr/> <p>Порт</p> <p>9600</p> <hr/>	<p>Ожидание ответа на запрос (ms)</p> <p>5000</p> <hr/> <p>Интервал между запросами (ms)</p> <p>2000</p> <hr/> <p>Мак кол-во слов при чтении диапазона</p> <p>50</p> <hr/> <p>Отправлять только изменения в систему</p> <p><input checked="" type="checkbox"/></p>
--	--

Так же доступны расширенные настройки подключения

- DNA (Destination Network Address) — Сетевой адрес устройства-получателя.
- DA1 (Destination Node Address) — Адрес узла устройства-получателя.
- DA2 (Destination Unit Address) — Адрес модуля/блока устройства-получателя.
- SNA (Source Network Address) — Сетевой адрес устройства-отправителя.
- SA1 (Source Node Address) — Адрес узла устройства-отправителя.
- SA2 (Source Unit Address) — Адрес модуля/блока устройства-отправителя.

Название	Назначение	Диапазон значений	Типичное значений
<b>DNA</b> (Сетевой адрес получателя)	Определяет, в какой сети находится целевое устройство. Используется при маршрутизации между разными сетями (например, из Ethernet в Controller Link).	<b>0x00</b> - <b>0x7F</b> (0 до 127 в десятичной системе).	<b>0x00</b> — локальная сеть (сеть, к которой подключен отправитель). Если оба устройства в одной сети, DNA почти всегда равно 0.
<b>DA1 (Адрес узла получателя)</b>	Это самый важный параметр. Он определяет конкретное устройство в сети. Каждому устройству в сети должен быть назначен уникальный номер узла.	<b>0x01</b> - <b>0x7E</b> (1 до 126). Значения <b>0x00</b> и <b>0xFF</b> обычно зарезервированы.	Если вы хотите обратиться к ПЛК с номером узла 5, вы установите DA1 = 5.
<b>DA2 (Адрес блока получателя)</b>	Определяет конкретный модуль или блок внутри целевого устройства. Это актуально для сложных устройств, таких как ПЛК с установленными слотами расширения (например, блоки ввода/вывода, коммуникационные модули).	<b>0x00</b> - <b>0xFE</b> (0 до 254).	<b>0x00</b> — Основной блок ПЛК (CPU Unit). <b>0x10</b> — Модуль расширения в слоте 0. <b>0x11</b> — Модуль расширения в слоте 1. <b>0xFE</b> — Бродкаст-рассылка (сообщение для всех блоков в узле).
<b>SNA</b> (Сетевой адрес			



отправителя)	<p>Определяет исходную сеть, из которой было отправлено сообщение. Этот параметр становится критически важным, когда устройство-получатель (например, шлюз или маршрутизатор) подключено к нескольким сетям и должно понять, в какую сеть отправить ответ.</p>	<p><b>0x00</b> - <b>0x7F</b> (0 до 127 в десятичной системе)</p>	<p><b>0x00</b> — Локальная сеть. Наиболее распространенное значение. Означает, что отправитель находится в той же сети, что и получатель. Ответ будет отправлен в эту же самую сеть. <b>0x01</b> - <b>0x7F</b> — Удаленная сеть. Используется, когда отправитель находится в другой, отличной от получателя, сети. Например, если ПЛК в сети Controller Link (DNA=1) отправляет команду ПЛК в сети Ethernet (DNA=0), то в ответном сообщении он укажет свой адрес как SNA=1.</p>
SA1 (Адрес узла отправителя)	<p>Это уникальный идентификатор самого устройства-отправителя в рамках его сети. Это самый важный параметр из тройки для отправителя, так как именно по нему получатель понимает, кто именно послал запрос, и на какой узел нужно отправить ответ.</p>	<p><b>0x01</b> - <b>0x7E</b> (1 до 126). Значения <b>0x00</b> и <b>0xFF</b> (127) обычно зарезервированы для специальных целей (например, широковещательная рассылка).</p>	<p>Должен быть уникальным числом в пределах одной сети. Например, если ваш ПЛК-отправитель имеет IP-адрес 192.168.250.10, то его узел (SA1) часто настраивается как 10.</p>
SA2 (Адрес блока отправителя)	<p>Определяет конкретный модуль или блок внутри устройства-отправителя, который инициировал FINS-команду. Это</p>		

(коммуникационные, CPU) могут independently инициировать связь.

0x00 -  
0xFE  
(0 до  
254).

0x00 — Основной блок CPU. Наиболее часто используется. Означает, что команда исходит от центрального процессора ПЛК. 0x10 — Модуль расширения в слоте 0. 0x11 — Модуль расширения в слоте 1. 0xFE — Бродкаст (широковещательная рассылка). Используется, когда сообщение должно быть отправлено от имени всех блоков в узле (на практике для отправителя используется редко).

Практические примеры для разных конфигураций с ПК, ПЛК и модулями.

### Сценарий 1: ПК ↔ ПЛК (Базовая связь)

**Описание:** Персональный компьютер (ПК) с OPC-сервером или SCADA-системой опрашивает основной блок ПЛК. Оба устройства находятся в одной сети Ethernet.

- **ПК (Отправитель):** Сетевой адрес = 0, Узел = 10
- **ПЛК (Получатель):** Сетевой адрес = 0, Узел = 5

Параметр	Значение (HEX)	Значение (Dec)	Обоснование
<b>DNA</b>	0x00	0	Сеть ПЛК (локальная сеть)
<b>DA1</b>	0x05	5	<b>Адрес узла ПЛК</b>
<b>DA2</b>	0x00	0	Основной блок CPU ПЛК
<b>SNA</b>	0x00	0	Сеть ПК (локальная сеть)
<b>SA1</b>	0x0A	10	<b>Адрес узла ПК</b>
<b>SA2</b>	0x00	0	Программа на ПК (условно "основной блок")

**Что происходит:** ПК (узел 10) говорит: "Эй, ПЛК на узле 5, ответь мне, узлу 10, в эту же сеть".

### Сценарий 2: ПК ↔ Модуль расширения ПЛК

**Описание:** ПК обращается напрямую к аналоговому входному модулю, установленному в слоте 3 основного ПЛК. Это используется для прямого доступа к данным модуля в обход логики CPU.

- **ПК (Отправитель):** Сетевой адрес = 0, Узел = 15

- **ПЛК (Шлюз):** Сетевой адрес = 0, Узел = 2
- **Модуль в слоте 3 (Получатель):** Адрес блока = 0x13 (0x10 + 3)

Параметр	Значение (HEX)	Значение (Dec)	Обоснование
<b>DNA</b>	0x00	0	Сеть, где находится ПЛК
<b>DA1</b>	0x02	2	<b>Адрес узла ПЛК</b> , в котором установлен модуль
<b>DA2</b>	0x13	19	<b>Адрес модуля</b> в слоте 3
<b>SNA</b>	0x00	0	Сеть ПК
<b>SA1</b>	0x0F	15	Адрес узла ПК
<b>SA2</b>	0x00	0	-

**Что происходит:** ПК (узел 15) говорит: \*Эй, устройство на узле 2, передай это сообщение конкретно твоему модулю в слоте 3 (блок 0x13). Ответ пришли мне.\*

### Сценарий 3: ПЛК ↔ Удаленный модуль в другой сети (через шлюз)

**Описание:** ПЛК в основной сети Ethernet управляет удаленным модулем ввода/вывода (например, NJ-series), который находится в подсети через Ethernet-модуль с IP-адресом.

- **ПЛК CPU (Отправитель):** Сеть = 0, Узел = 1
- **Удаленный модуль I/O (Получатель):** Сеть = 1, Узел = 25, Блок = 0 (основной CPU удаленного модуля)

Параметр	Значение (HEX)	Значение (Dec)	Обоснование
<b>DNA</b>	0x01	1	<b>Сеть удаленного модуля</b>
<b>DA1</b>	0x19	25	<b>Адрес узла удаленного модуля</b> в его сети
<b>DA2</b>	0x00	0	Основной блок удаленного модуля
<b>SNA</b>	0x00	0	Сеть главного ПЛК
<b>SA1</b>	0x01	1	Адрес узла главного ПЛК
<b>SA2</b>	0x00	0	-

**Что происходит:** Главный ПЛК говорит: "Маршрутизатор, перешли это сообщение в сеть №1, узлу №25. Ответ жду по адресу: сеть 0, узел 1."\*

## Сценарий 4: Модуль связи ПЛК ↔ Внешнее устройство

**Описание:** Специализированный коммуникационный модуль (например, модуль FINS-to-Modbus TCP шлюз) в слоте 2 ПЛК опрашивает внешнее устройство. Ответ должен прийти обратно на этот модуль, а не на основной CPU.

- **Модуль связи (Отправитель):** Сеть = 0, Узел ПЛК = 3, Слот = 2 (Блок = 0x12)
- **Внешнее устройство (Получатель):** Сеть = 0, Узел = 40

Параметр	Значение (HEX)	Значение (Dec)	Обоснование
<b>DNA</b>	0x00	0	Локальная сеть
<b>DA1</b>	0x28	40	Адрес узла внешнего устройства
<b>DA2</b>	0x00	0	Основной блок внешнего устройства
<b>SNA</b>	0x00	0	Локальная сеть
<b>SA1</b>	0x03	3	<b>Адрес узла всего ПЛК</b> в сети
<b>SA2</b>	0x12	18	<b>Важно! Адрес модуля связи (слот 2)</b>

**Что происходит:** Модуль связи говорит: \*"Внешнее устройство на узле 40, ответь, пожалуйста, не просто ПЛК на узле 3, а мне лично, модулю в слоте 2 (блок 0x12)."\*

## Сводная таблица типичных значений

Объект	SNA/DNA	SA1/DA1	SA2/DA2
Основной CPU ПЛК	0x00	[Номер узла]	0x00
Модуль в слоте N	0x00	[Номер узла ПЛК]	0x10 + N
ПК / HMI	0x00	[Назначенный узел]	0x00
Широковещание	0x00	0xFF	0xFE

### Ключевой вывод:

- **DA1/SA1** — это почти всегда адрес **всего устройства** (ПЛК, ПК) в сети.
- **DA2/SA2** — это адрес **конкретного "мозга" или модуля** внутри этого устройства, который должен обработать команду или отправить её.

## Настройка каналов плагина

Канал имеет следующие параметры:

- Адрес регистра в десятичном формате
- Тип переменной (INT8, UINT8, INT16, UINT16, INT32, UINT32, FLOAT)
- Область памяти контроллера (D - Data Memories, W - Work Area, T - Timers, C - Counters, CIO - Input/Output, H - Holding Registers)
- Извлечь битовое значение. Можно выполнять чтение словами, а затем извлекать бит, указав смещение.  
Например, битовые флаги доступны для чтения как UINT16. Для каждого битового флага создается канал с одним и тем же адресом, но разными смещениями.
- Группировка при чтении. Необходима для оптимизации количества запросов к устройству.
- Автоматическая группировка выполняется плагином
- Ручная группировка может быть выполнена при настройке каналов
- Коэффициент времени опроса. Данный параметр становится доступным только при ручной группировке.

Канал <b>ch1</b>	Чтение <input checked="" type="checkbox"/>
Свойство для привязки _____	Запись <input checked="" type="checkbox"/>

Адрес регистра <b>10</b>	Группировать при чтении <input checked="" type="checkbox"/>
Тип переменной <b>FLOAT, 4 байта</b>	Ручная группировка <input checked="" type="checkbox"/>
Тип памяти <b>D</b>	Название группы _____
Извлечь битовое значение <input checked="" type="checkbox"/>	Коэффициент цикла опроса <b>1</b>
Смещение <b>0</b>	

Формула расчета входного значения _____	Канал <b>29.08.24 16:43:56.143 (status 1)</b>
Инvertировать входное значение <input type="checkbox"/>	Устройство _____

Для записи использовать другой адрес или команду <input type="checkbox"/>	Тест записи
Формула расчета выходного значения _____	Записать значение _____
	<div>ЗАПИСАТЬ</div>

## Механизмы группировки

Группировка позволяет оптимизировать количество запросов к устройству.

### Автоматическая группировка

Плагин сам группирует последовательно идущие адреса внутри каждого запроса с учетом функции чтения и параметра плагина "Мак кол-во слов при чтении диапазона".

Кроме установки галочки **Группировать при чтении** никаких дополнительных действий по настройке не требуется.

## Ручная группировка

Позволяет вручную объединять регистры для группового чтения.

Для этого вводится **Название группы** - произвольный идентификатор (латинские буквы+цифры+знак подчеркивания).

## Запись

Если канал предполагает запись, нужно выставить галочку **Запись**

По умолчанию запись выполняется по тому же адресу, что и чтение.

Бывает, что есть необходимость в рамках одного канала читать с одного регистра, а писать в другой (например, контроллер принимает значение в регистр для записи и затем переписывает его в регистр для чтения или отвергает присланное значение).

В этом случае можно установить галочку **Для записи использовать другой адрес** и ввести адрес и тип переменной для записи.

<p>Для записи использовать другой адрес или команду</p> <p><input checked="" type="checkbox"/></p> <p>Адрес регистра</p> <p>0</p> <p>Тип переменной</p> <p>INT16, 2 bytes</p> <p>Формула расчета выходного значения</p>	<p>Тест записи</p> <p>Записать значение</p> <p>ЗАПИСАТЬ</p>
---	---

## Тест записи

Можно проверить работу канала без привязки к устройству. Кнопка **Записать** дает возможность выполнить операцию записи значения из поля **Записать значение**.

# Плагин OPC UA клиент

## Описание

Плагин позволяет подключиться к OPC UA серверу по TCP/IP.

На текущий момент поддерживаются следующие возможности:

- Подключение к серверу используя различные Security Mode и Security Policy
- Создание сессии используя логин и пароль
- Сканирование узлов сервера
- Подписка на изменение данных на сервере
- Запись данных на сервер

## Настройка

### Настройка подключения

Для настройки подключения к серверу необходимо ввести EndPoint URL в формате:

opc.tcp://opcua.umatl.app:4843

Для создания сессии необходимо ввести логин и пароль, если это необходимо и настроить параметры Безопасности и выбрать тип подключения Security Mode и Security policy. Параметр Buffer time отвечает за время буферизации данных в плагине. Это необходимо, чтобы отправлять данные с плагина в систему пакетами, а не по одному.

Endpoint URL	Использовать аутентификацию
<input type="text" value="opc.tcp://uademo.prosysopc.com:53530/OPCUA/SimulationServe"/>	<input checked="" type="checkbox"/>
Buffer time, ms	Логин
<input type="text" value="1000"/>	<input type="text" value="admin"/>
Время рестарта (сек)	Пароль
<input type="text" value="5"/>	<input type="password" value="....."/>
Уровень логирования	Security Policy
<input type="text" value="-"/>	<input type="text" value="Basic256Sha256"/>
	MessageSecurityMode
	<input type="text" value="SignAndEncrypt"/>

### Настройка параметров подписки и мониторинга

Так же вы можете настроить параметры подписки и мониторинга. Подробнее про данные параметры вы можете изучить в спецификации OPC UA



Requested Publishing Interval	Sampling Interval
1000	100
Requested Lifetime Count	Queue Size
100	10
Requested Max KeepAlive Count	Discard Oldest
10	<input checked="" type="checkbox"/>
Max Notifications Per Publish	
100	
Priority	
10	

## Параметры Подписки (Subscription)

Подписка определяет, *как* и *когда* данные будут доставляться от сервера клиенту.

Параметр / Свойство	Описание	Рекомендации / Влияние
<b>RequestedPublishingInterval</b>	<p><b>Интервал публикации (в мс).</b> Основной параметр! Как часто сервер должен пытаться отправить пакет (NotificationMessage) с данными.</p>	<p><b>Самый важный параметр для производительности.</b> Меньше интервал = больше нагрузка на сеть и ЦП. Ставьте в соответствии с требуемой скоростью обновления данных (например, 100 мс для быстрых процессов, 1000 мс для медленных).</p>
<b>RequestedMaxKeepAliveCount</b>	<p><b>Максимальное количество интервалов без данных.</b> Определяет, как часто сервер будет отправлять "пустое" сообщение (KeepAlive), если данных для отправки нет. Таймаут = <math>\text{PublishingInterval} * \text{MaxKeepAliveCount}</math>.</p>	<p>Гарантирует, что "молчащая" подписка будет считаться активной. Если таймаут истечет, подписка будет закрыта. Обычно значение 3-10.</p>
<b>RequestedLifetimeCount</b>	<p><b>Счетчик времени жизни.</b> Определяет, как быстро сервер должен обнаружить "мертвого" клиента и очистить подписку. Время жизни = <math>\text{PublishingInterval} * \text{LifetimeCount}</math>.</p>	<p>Должен быть значительно больше <math>\text{MaxKeepAliveCount}</math> (обычно в 3-5 раз). Это страховка на случай временных проблем с сетью у клиента.</p>
<b>MaxNotificationsPerPublish</b>	<p><b>Максимальное количество уведомлений в одном сообщении публикации.</b> Ограничивает размер одного пакета данных.</p>	<p>Если установлено в 0, лимита нет. Полезно для ограничения трафика, но если лимит будет постоянно превышаться, сервер будет посылать ошибку <b>Overflow</b> и сбрасывать неотправленные данные.</p>
<b>Priority</b>	<b>Приоритет подписки (от 0)</b>	

**до 255).** Сервер может использовать это значение для управления очередностью обработки подписок. 0 - наивысший приоритет.

**Ключевая связь:**  $\text{KeepAlive Timeout} = \text{PublishingInterval} * \text{MaxKeepAliveCount}$

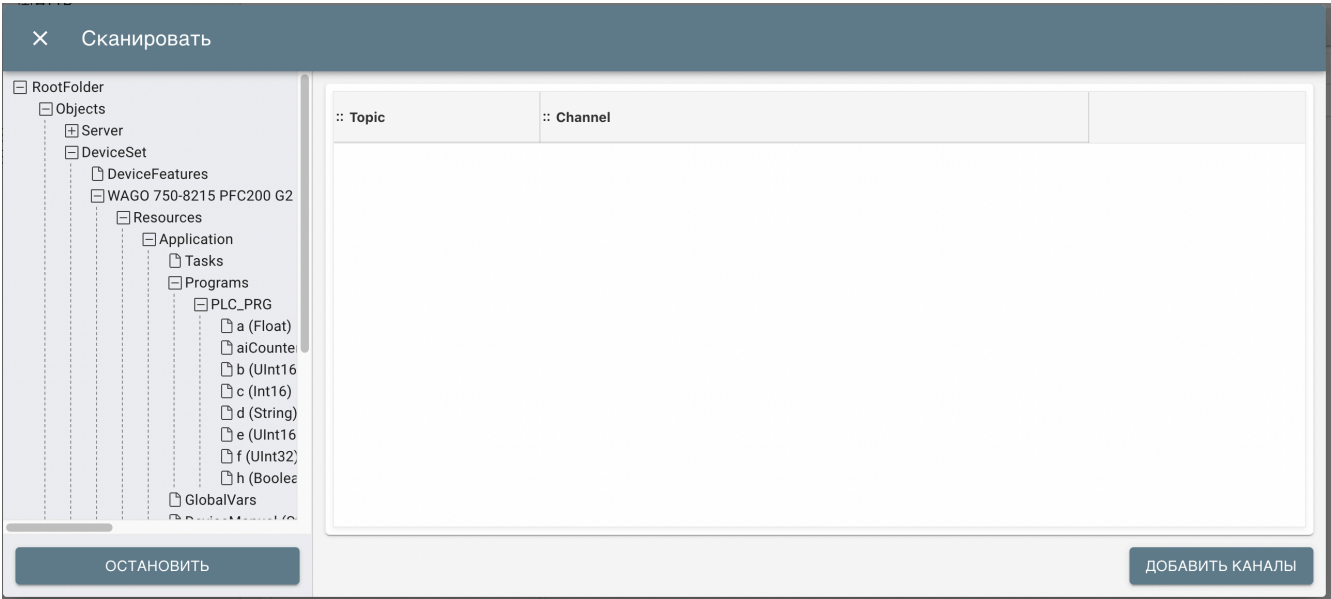
## Параметры мониторинга

Параметр	Описание	Рекомендации
<b>SamplingInterval</b>	<b>Интервал выборки (в мс).</b> Как часто сервер должен проверять (читать) значение переменной.	<b>Не путать с PublishingInterval !</b> SamplingInterval определяет актуальность данных на сервере, а PublishingInterval - как часто они уходят клиенту. Можно установить -1, чтобы использовать интервал подписки. Для быстрых данных ставьте меньше.
<b>QueueSize</b>	<b>Размер очереди выборки.</b> Размер буфера на сервере, где хранятся отснятые, но еще не отправленные значения.	Минимум 1. Если 1, то хранится только последнее значение. Большие значения полезны для медленных подписок, чтобы не потерять данные между циклами публикации.
<b>DiscardOldest</b>	<b>Удалять самые старые значения из очереди.</b>	В 80% случаев используйте discardOldest: true – это обеспечивает получение самых актуальных данных при переполнении очереди. discardOldest: false оставляйте для специальных сценариев, где критически важна полная хронологическая последовательность событий. Правильная настройка этого параметра в сочетании с queueSize предотвращает потерю важных данных и оптимизирует работу клиента OPC UA.

## Сканирование узлов

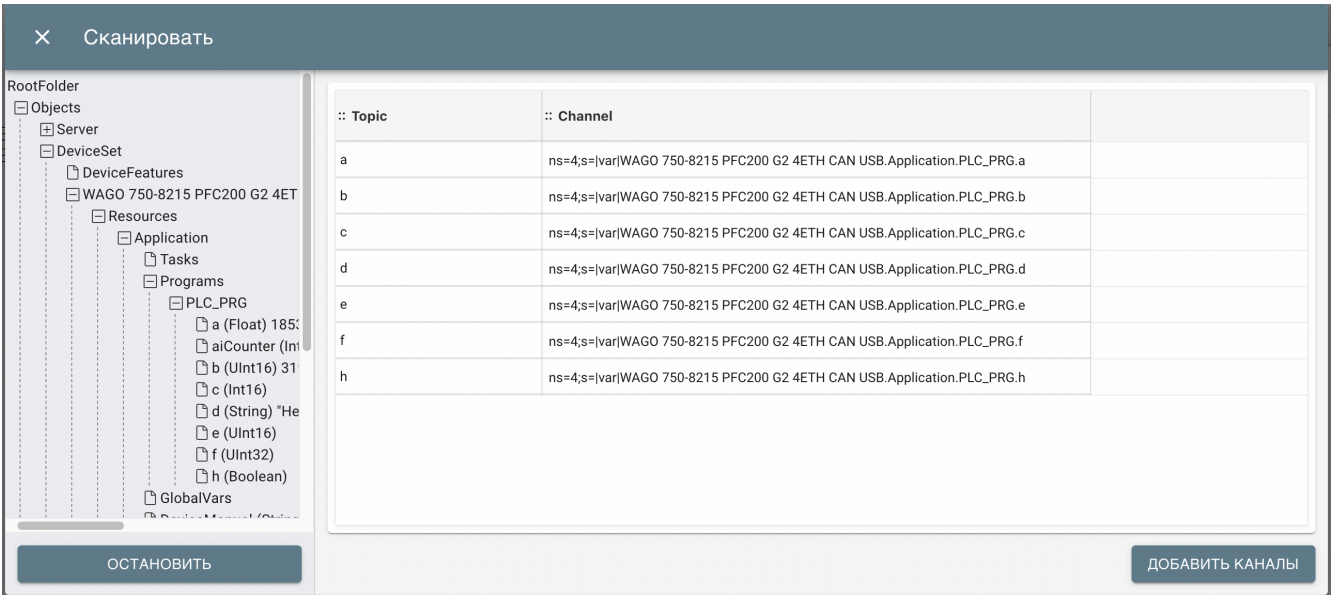
Для сканирования узлов сервера необходимо перейти на вкладку каналы и правой кнопкой мыши на папке ALL вызвать всплывающее окно, где нажать на "Сканировать каналы". Появится всплывающее окно, где необходимо нажать кнопку "Сканировать" для начала сканирования дерева сервера. После получения узлов

дерева вы сможете передвигаться по нему:



Добавление переменных для мониторинга и записи

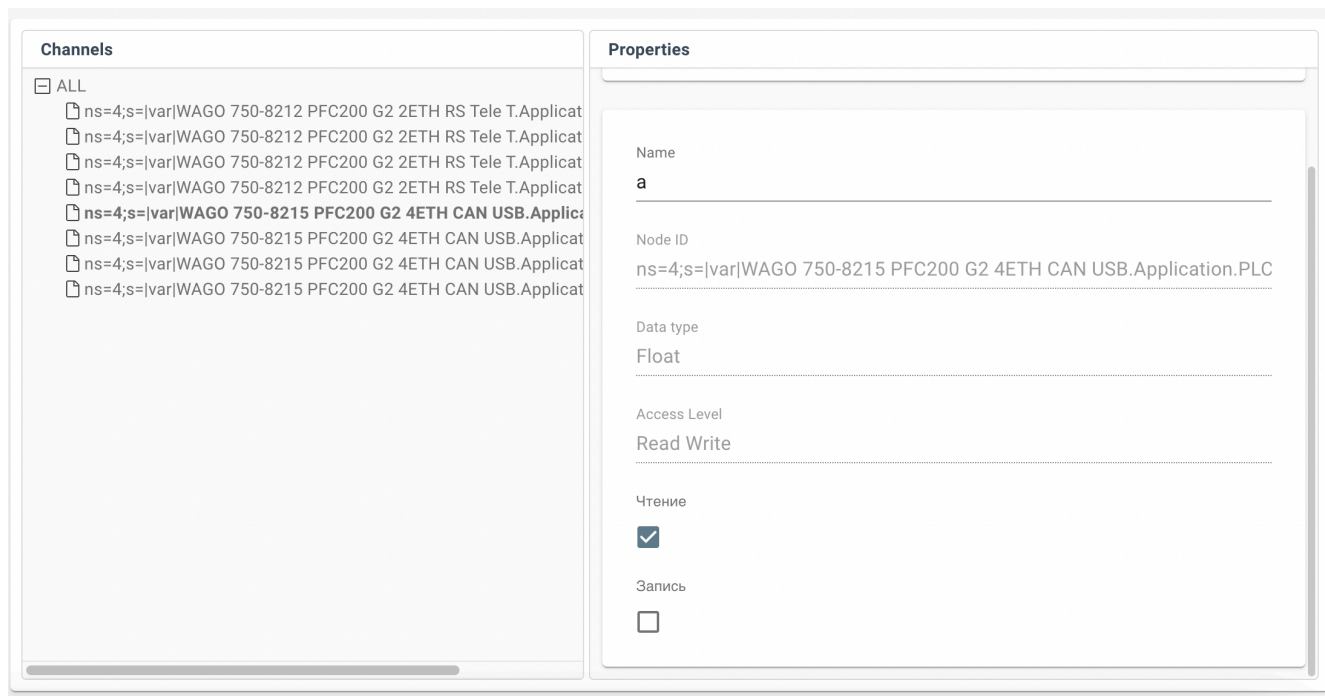
Для мониторинга необходимых переменных, которые присутствуют в дереве, вам достаточно щелкнуть на них два раза левой кнопкой мыши и переменные добавятся в список справа. После этого нажимаем на кнопку "Добавить каналы" и данные переменные появятся в списке каналов плагина.



Так же вы можете добавить переменные вручную или с помощью csv файла. Для более гибкой настройки опроса данных с орс сервера есть возможность добавить узлы с индивидуальными параметрами **Sampling interval, Queue size, Discard oldest**

Редактирование каналов плагина

В дереве каналов есть возможность с помощью свойства **Чтение** включить данный канал в список для подписки, а с помощью свойства **Запись** указать серверу возможность записи в данный канал.



## Команды плагина

Плагин поддерживает следующие команды из сценария и скриптов визуализации:

- **syncHistory** - команда предназначена для синхронизации исторических данных. С помощью данной команды можно запросить исторические данные за определенный интервал времени с сервера орс ua по конкретном nodeId и они автоматически добавятся с локальную БД того свойства устройства к которому привязан данный nodeId.

Пример для скрипта визуализации:

```
module.exports = async function({ local, context, source }, core, debug) {  
  //Получение каналов из устройства  
  const arr = await core.getDeviceChannels({did:'d1121', unit:'opcua1'});  
  //Команда плагину  
  let syncRes = {  
    chanarr: data,  
    startTime: Date.now() - 3600000,  
    endTime: Date.now()  
  };  
  
  const result = await core.pluginCommand(  
    {  
      unit: 'opcua1',  
      command: 'syncHistory',  
      data: syncRes  
    },  
    10  
  );  
  
};
```

Пример для сценария:

```
const script = {
  start() {
    this.getDeviceChannels(
      { did: 'd1121', unit: 'opcua1' },
      'onGet'
    );
  },

  onGet(error, data) {
    if (!error) {
      let syncRes = {
        chanarr: data,
        startTime: Date.now() - 3600000,
        endTime: Date.now()
      };

      this.pluginCommand(
        {
          unit: 'opcua1',
          command: 'syncHistory',
          data: syncRes
        },
        'getResponse'
      );
    } else {
      this.log('Ошибка: ' + error);
    }
  },

  getResponse(message) {
    this.log('Response ' + message);
  }
};
```

# Плагин OPC UA Server

## Описание

Плагин позволяет публиковать данные из системы в протокол OPC UA.

Может работать в качестве шлюза из любого поддерживаемого системой протокола в OPC UA

На текущий момент поддерживаются следующие возможности:

- Публикация на сервер состояния любого устройства системы.
- Групповой выбор устройств по меткам или расположению в дереве проекта
- Команды устройства доступны на сервере в качестве Методов
- Безопасный режим (Sign, Sign&Encrypt).
- Политика безопасности (Basic128Rsa15, Basic256, Basic256Sha256)
- Авторизация по логину и паролю
- Публикации тревог Alarm & Events
- Доступ к историческим данным History Data Access

## Настройка

### Настройка сервера

Для настройки необходимо указать свободный порт TCP и Source Path (Исходный путь)

Если требуется установить ограничение на доступ к серверу, то можно выбрать две учетные записи:

- Учетная запись администратора - Полный доступ (Чтение/Запись)
- Учетная запись пользователя - Ограниченный доступ (Чтение)

Для публикации тревог необходимо поставить галочку напротив параметра **A&E**

Для доступа к историческим данным поставить галочку напротив параметра **HDA**



Исходный путь <b>/UA/IntraServer</b>	OPC AE (Alarms & Events) <input type="checkbox"/>	Время рестарта (сек) <b>6</b>
Порт <b>4334</b>	OPC HDA (Historical Data Access) <input type="checkbox"/>	Уровень логирования <b>Низкий</b>
Использовать учетную запись Администратора (Чтение/Запись) <input checked="" type="checkbox"/>		Комментарий _____
Логин Администратора <b>admin</b>		
Пароль Администратора .....		
Использовать учетную запись Пользователя (Чтение) <input checked="" type="checkbox"/>		
Логин Пользователя <b>user</b>		
Пароль Пользователя ....		

## Расширение плагина

Расширение плагина используется для публикации данных из системы в область памяти OPC UA сервера и обратно. Существует три способа публикации устройств в OPC UA сервер:

1. Выбор устройства из списка устройств
2. Выбор папки расположения устройств в дереве
3. Выбор устройств по Меткам

:: Фильтр	:: Устройство	:: Название Пути	:: Название Метки
Tag ▼			безопасность
Location ▼		Папка 1 ▼	
Device ▼	AO_001 • Актуатор аналоговый ▼		

# Плагин МЭК 60870-5-104 клиент

## Описание

Плагин позволяет подключиться к серверу по протоколу МЭК 60870-5-104.

## Настройка

### Настройка подключения

Для подключения к серверу необходимо настроить следующие параметры плагина:

- IP адрес
- TCP Порт
- Originator адрес - адрес устройства
- Параметр K - максимальное количество неподтвержденных APDU между полученными и отправленными телеграммами, которая может возникнуть без подтверждения партнера по связи. Если значение достигнуто, другие телеграммы не отправляются и ожидается подтверждение.
- Параметр W - количество неподтвержденных APDU после получения нескольких "W" телеграмм.
- Параметр T0 - Таймаут для установления соединения, сек
- Параметр T1 - Таймаут для передаваемых APDU, должно быть больше чем T2, сек
- Параметр T2 - Таймаут для подтверждения сообщений, сек
- Параметр T3 - Время до отправки тестовых телеграмм в случае простоя соединения, сек

IP	Parameter T0
192.168.0.25	30
Port	Parameter T1
2404	15
Originator address	Parameter T2
1	10
Parameter K	Parameter T3
12	20
Parameter W	
8	

### Настройка каналов

Для того, чтобы добавить каналы, необходимо на папке ALL нажать правой кнопкой мыши и добавить:

- **Канал для Чтения.** В поле **Object Address** ввести адрес, а в поле **Information Object** выбрать тип:  
Поддерживаются следующие типы: M\_SP\_NA(1), M\_DP\_NA(3), M\_ST\_NA(5), M\_BO\_NA(7), M\_ME\_NA(9), M\_ME\_NB(11), M\_ME\_NC(13), M\_IT\_NA(15), M\_SP\_TB(30), M\_DP\_TB(31), M\_ST\_TB(32), M\_BO\_TB(33), M\_ME\_TD(34), M\_ME\_TE(35), M\_ME\_TF(36), M\_IT\_TB(37)

Channels	Properties
<div> <div>ALL</div> <div> M_SP_NA(1)  M_DP_NA(3)  M_ST_NA(5)  M_BO_NA(7)  M_ME_NA(9)  M_ME_NB(11)  M_ME_NC(13)  M_IT_NA(15)  M_SP_TB(30)  M_DP_TB(31)  M_ST_TB(32)  M_BO_TB(33)  M_ME_TD(34)  M_ME_TE(35)  M_ME_TF(36)  M_IT_TB(37)  Command </div> </div>	<div> Привязка канала к свойству устройства  <input type="text"/> </div> <div> Object Address  1 </div> <div> Название  M_SP_NA(1) </div> <div> Information Object  M_SP_NA (1) </div> <div> Формула расчета входного значения  <input type="text"/> </div> <div> Свойство устройства для привязки  <input type="text"/> </div>

- **Канал для Записи.** В поле **Object Address** ввести адрес, а в поле **Control Object** выбрать тип.  
Поддерживаются следующие типы: C\_SC\_NA (45), C\_DC\_NA (46), C\_RC\_NA (47), C\_SE\_NA (48), C\_SE\_NB (49), C\_SE\_NC (50), C\_BO\_NA (51), C\_SC\_TA (58), C\_DC\_TA (59), C\_RC\_TA (60), C\_SE\_TA (61), C\_SE\_TB (62), C\_SE\_TC (63), C\_BO\_TA (64) Так же можно выбрать **Qualifier of command**:

- No additional definition
- Short pulse
- Long pulse
- Persistent output.

Channels	Properties
<div> <div>ALL</div> <div> M_SP_NA(1)  M_DP_NA(3)  M_ST_NA(5)  M_BO_NA(7)  M_ME_NA(9)  M_ME_NB(11)  M_ME_NC(13)  M_IT_NA(15)  M_SP_TB(30)  M_DP_TB(31)  M_ST_TB(32)  M_BO_TB(33)  M_ME_TD(34)  M_ME_TE(35)  M_ME_TF(36)  M_IT_TB(37)  Command </div> </div>	<div> Привязка канала к свойству устройства  H_003 • Светильник • оп </div> <div> Object Address  145 </div> <div> Название  C_SC_NA(145) </div> <div> Свойство устройства для привязки  <input type="text"/> </div> <div> Control Object  C_SC_NA (45) </div> <div> Qualifier of command  No additional definition </div> <div> Формула расчета выходного значения  1 </div>

Для типов: C\_SC\_NA (45), C\_SC\_TA (58) в формуле необходимо прописать значение 1 или 0, которое будет передаваться на сервер и привязать этот канал к команде устройства Для типов: C\_DC\_NA (46), C\_DC\_TA (59) в формуле необходимо прописать значение 1 или 2, которое будет передаваться на сервер и привязать этот канал к команде устройства

# Плагин МЭК 60870-5-104 мульти клиент

## Описание

Плагин позволяет подключиться к серверам по протоколу МЭК 60870-5-104.

## Настройка

### Настройка плагина

- Originator адрес - адрес устройства клиента (мастера)

### Настройка узла

Для подключения к серверу необходимо добавить узел в каналы и настроить следующие параметры:

- IP адрес - адрес устройства
- TCP порт - порт устройства
- Резервный IP адрес, если нужен механизм резервирования каналов связи
- ASDU адрес - это адрес, используемый для идентификации конкретного устройства или логического объекта в системе связи
- Часовой пояс устройства - так как метки времени от устройства приходят со смещением от UTC, а в системе все хранится по UTC, то необходимо указать это смещение для каждого узла
- Параметр K - максимальное количество неподтвержденных APDU между полученными и отправленными телеграммами, которая может возникнуть без подтверждения партнера по связи. Если значение достигнуто, другие телеграммы не отправляются и ожидается подтверждение.
- Параметр W - количество неподтвержденных APDU после получения нескольких "W" телеграмм.
- Параметр T0 - Таймаут для установления соединения, сек
- Параметр T1 - Таймаут для передаваемых APDU, должно быть больше чем T2, сек
- Параметр T2 - Таймаут для подтверждения сообщений, сек
- Параметр T3 - Время до отправки тестовых телеграмм в случае простоя соединения, сек

<p>Название</p> <p>Новая папка</p> <hr/> <p>IP адрес</p> <p>176.193.75.143</p> <hr/> <p>TCP порт</p> <p>2404</p> <hr/> <p>Включить резервирование</p> <p><input type="checkbox"/></p> <p>ASDU адрес</p> <p>2</p> <hr/> <p>Часовой пояс устройства</p> <p>+3</p> <hr/> <p>Комментарий</p> <hr/>	<p>Параметр K</p> <p>12</p> <hr/> <p>Параметр W</p> <p>8</p> <hr/> <p>Параметр T0</p> <p>30</p> <hr/> <p>Параметр T1</p> <p>15</p> <hr/> <p>Параметр T2</p> <p>10</p> <hr/> <p>Параметр T3</p> <p>20</p> <hr/>
--	--

## Настройка каналов

Для того, чтобы добавить каналы, необходимо на папке узла нажать правой кнопкой мыши и добавить: **Канал для Чтения**. Доступны следующие параметры канала:

- **Object Address** - адрес параметра
- **Information Object** - тип параметра
- **Группировать по ASDU** - группировка адресов по ASDU
- **ASDU для группировки** - адрес ASDU для группировки
- **Извлечь битовое значение** - номер бита, который необходимо извлечь из значения.

Поддерживаются следующие типы Информационные объекты (мониторинг):

ID типа	Название	Описание	Применение
1	M_SP_NA_1	Одноточечная информация	Состояние одной точки (например, вкл/выкл) без метки времени
3	M_DP_NA_1	Двухточечная информация	Состояние двух точек (например, вкл/выкл/промежуточное) без метки времени
5	M_ST_NA_1	Информация о положении ступени	Положение ступени (например, положение трансформатора) без метки времени
7	M_BO_NA_1	Битовая строка из 32 бит	32-битная строка без метки времени
9	M_ME_NA_1	Измеренное значение, нормализованное	Нормализованное измеренное значение без метки времени
11	M_ME_NB_1	Измеренное значение, масштабированное	Масштабированное измеренное значение без метки времени
13	M_ME_NC_1	Измеренное значение, короткое с плавающей точкой	Измеренное значение в формате с плавающей точкой без метки времени
15	M_IT_NA_1	Интегральные значения	Накопленные значения (например, счётчики) без метки времени
30	M_SP_TB_1	Одноточечная информация с меткой времени CP56Time2a	Состояние одной точки с меткой времени в формате CP56Time2a
31	M_DP_TB_1	Двухточечная информация с меткой времени CP56Time2a	Состояние двух точек с меткой времени CP56Time2a
32	M_ST_TB_1	Информация о положении ступени с меткой времени CP56Time2a	Положение ступени с меткой времени CP56Time2a
33	M_BO_TB_1	Битовая строка из 32 бит с меткой времени CP56Time2a	32-битная строка с меткой времени CP56Time2a

35	M_ME_TE_1	Измеренное значение, масштабированное с меткой времени CP56Time2a	Масштабированное измеренное значение с меткой времени CP56Time2a
36	M_ME_TF_1	Измеренное значение, короткое с плавающей точкой с меткой времени CP56Time2a	Измеренное значение в формате с плавающей точкой с меткой времени CP56Time2a
37	M_IT_TB_1	Интегральные значения с меткой времени CP56Time2a	Накопленные значения с меткой времени CP56Time2a

Channel

ch\_1

Object Address

1

Группировать по ASDU

☐

Название

Control Object 1

Information Object

M\_SP\_NA (1)



Извлечь битовое значение

☐

Формула расчета входного значения

Свойство устройства для привязки

Существует возможность добавлять каналы интерактивно, через окно сканирования. Для этого нужно правой кнопкой мыши нажать на узел и выбрать пункт меню **Сканировать каналы**. Далее во всплывающем окне нажать на кнопку **Сканировать**, в появившемся дереве выбрать необходимые каналы и нажать кнопку **Добавить каналы**.

**Канал для Записи.** Доступны следующие параметры канала:

- **Object Address** - адрес команды
- **Control Object** - тип команды
- **Группировать по ASDU** - группировка адресов по ASDU
- **ASDU для группировки** - адрес ASDU для группировки
- **Selected Command** - Выбранная команда



- **Qualifier of command** - Квалификатор команды (No additional definition/Short pulse/Long pulse/Persistent output )

Поддерживаются следующие типы Управляющие объекты (команды):

ID типа	Название	Описание	Применение
45	C_SC_NA_1	Одиночная команда	Одиночная команда (например, включить/выключить)
46	C_DC_NA_1	Двойная команда	Двойная команда (например, для управления двухпозиционным устройством)
47	C_RC_NA_1	Команда регулирования ступени	Команда для регулирования положения ступени
48	C_SE_NA_1	Команда установки значения, нормализованное	Команда установки нормализованного значения
49	C_SE_NB_1	Команда установки значения, масштабированное	Команда установки масштабированного значения
50	C_SE_NC_1	Команда установки значения, короткое с плавающей точкой	Команда установки значения в формате с плавающей точкой
51	C_BO_NA_1	Команда битовой строки из 32 бит	Команда для передачи 32-битной строки
58	C_SC_TA_1	Одиночная команда с меткой времени CP56Time2a	Одиночная команда с меткой времени CP56Time2a
59	C_DC_TA_1	Двойная команда с меткой времени CP56Time2a	Двойная команда с меткой времени CP56Time2a
60	C_RC_TA_1	Команда регулирования ступени с меткой времени CP56Time2a	Команда регулирования ступени с меткой времени CP56Time2a
61	C_SE_TA_1	Команда установки значения, нормализованное с меткой времени CP56Time2a	Команда установки нормализованного значения с меткой времени CP56Time2a
62	C_SE_TB_1	Команда установки значения, масштабированное с меткой времени CP56Time2a	Команда установки масштабированного значения с меткой времени CP56Time2a
63	C_SE_TC_1	Команда установки значения, короткое с плавающей точкой с меткой времени CP56Time2a	Команда установки значения в формате с плавающей точкой с меткой времени CP56Time2a

ch_1
Object Address
1
Группировать по ASDU
<input type="checkbox"/>
Название
Command Object 1
Свойство устройства для привязки
Control Object
C_SC_NA (45) ▼
Selected Command
<input type="checkbox"/>
Qualifier of command
No additional definition ▼

Для типов: C\_SC\_NA (45), C\_SC\_TA (58) в формуле необходимо прописать значение 1 или 0, которое будет передаваться на сервер и привязать этот канал к команде устройства  
Для типов: C\_DC\_NA (46), C\_DC\_TA (59) в формуле необходимо прописать значение 1 или 2, которое будет передаваться на сервер и привязать этот канал к команде устройства

# Плагин МЭК 60870-5-104 сервер

## Описание

Плагин позволяет создать сервер, который будет обмениваться данными по протоколу МЭК 60870-5-104.

## Настройка

### Настройка плагина

- **Порт** - TCP/IP порт для подключения
- **ASDU адрес** - это адрес, используемый для идентификации конкретного устройства или логического объекта в системе связи
- **Адрес отправителя** - Originator address
- **Максимальное количество клиентов** - максимальное количество клиентов, которое может подключиться к серверу одновременно
- **Максимальный размер буфера сообщений** - максимальный размер буфера, в случае обрыва связи клиента с сервером
- **Параметр K** - максимальное количество неподтвержденных APDU между полученными и отправленными телеграммами, которая может возникнуть без подтверждения партнера по связи. Если значение достигнуто, другие телеграммы не отправляются и ожидается подтверждение.
- **Параметр W** - количество неподтвержденных APDU после получения нескольких "W" телеграмм.
- **Параметр T0** - Таймаут для установления соединения, сек
- **Параметр T1** - Таймаут для передаваемых APDU, должно быть больше чем T2, сек
- **Параметр T2** - Таймаут для подтверждения сообщений, сек
- **Параметр T3** - Время до отправки тестовых телеграмм в случае простоя соединения, сек
- **Часовой пояс устройства** - добавляет смещение к метки времени при отправке клиентам, так как на сервере все метки хранятся с нулевым смещением
- **Тип исполнения команды** - Прямой (Немедленное исполнение) и Выбор/Исполнение (Сначала выбор, потом исполнение)
- **Время исполнения по-умолчанию** - Время импульса команд исполнения по-умолчанию в миллисекундах
- **Время исполнения короткое** - Время импульса команд исполнения короткое в миллисекундах
- **Время исполнения длинное** - Время импульса команд исполнения длинное в миллисекундах

ID iec60870-104server1	Параметр K 12	Часовой пояс устройства +3
Название _____	Параметр W 8	Тип исполнения команды Прямое
Порт 2404	Параметр T0 30	Время исполнения по-умолчанию, мс 2000
ASDU адрес 1	Параметр T1 15	Время исполнения короткое, мс 1000
Адрес отправителя 1	Параметр T2 10	Время исполнения длинное, мс 5000
Максимальное количество клиентов 10	Параметр T3 20	Время рестарта (сек) 5
Максимальный размер буфера сообщений 5000		Уровень логирования -
		Комментарий _____

## Расширение плагина

Расширение плагина используется для публикации данных из системы в область памяти сервера МЭК 60870-5-104 и обратно. Для привязки свойств устройств к регистрам МЭК 60870-5-104 необходимо:

1. Выбрать устройство
2. Прописать свойство
3. Указать Адрес объекта
4. Выбрать Тип расширения - Принять значение/Публиковать значение
5. Выбрать ASDU адрес для группировки по ASDU
6. Циклический интервал в минутах для отправки данных циклически всем клиентам
7. Тип информационного объекта Поддерживаются следующие типы Информационные объекты (мониторинг):

ID типа	Название	Описание	Применение
1	M_SP_NA_1	Одноточечная информация	Состояние одной точки (например, вкл/выкл) без метки времени
3	M_DP_NA_1	Двухточечная информация	Состояние двух точек (например, вкл/выкл/промежуточное) без метки времени
5	M_ST_NA_1	Информация о положении ступени	Положение ступени (например, положение трансформатора) без метки времени
7	M_BO_NA_1	Битовая строка из 32 бит	32-битная строка без метки времени
9	M_ME_NA_1	Измеренное значение, нормализованное	Нормализованное измеренное значение без метки времени
11	M_ME_NB_1	Измеренное значение, масштабированное	Масштабированное измеренное значение без метки времени
13	M_ME_NC_1	Измеренное значение, короткое с плавающей точкой	Измеренное значение в формате с плавающей точкой без метки времени
15	M_IT_NA_1	Интегральные значения	Накопленные значения (например, счётчики) без метки времени
30	M_SP_TB_1	Одноточечная информация с меткой времени CP56Time2a	Состояние одной точки с меткой времени в формате CP56Time2a
31	M_DP_TB_1	Двухточечная информация с меткой времени CP56Time2a	Состояние двух точек с меткой времени CP56Time2a
32	M_ST_TB_1	Информация о положении ступени с меткой времени CP56Time2a	Положение ступени с меткой времени CP56Time2a
33	M_BO_TB_1	Битовая строка из 32 бит с меткой времени CP56Time2a	32-битная строка с меткой времени CP56Time2a

35	M_ME_TE_1	Измеренное значение, масштабированное с меткой времени CP56Time2a	Масштабированное измеренное значение с меткой времени CP56Time2a
36	M_ME_TF_1	Измеренное значение, короткое с плавающей точкой с меткой времени CP56Time2a	Измеренное значение в формате с плавающей точкой с меткой времени CP56Time2a
37	M_IT_TB_1	Интегральные значения с меткой времени CP56Time2a	Накопленные значения с меткой времени CP56Time2a

8. Тип управляющего объекта Поддерживаются следующие типы Управляющие объекты (команды):

ID типа	Название	Описание	Применение
45	C_SC_NA_1	Одиночная команда	Одиночная команда (например, включить/выключить)
46	C_DC_NA_1	Двойная команда	Двойная команда (например, для управления двухпозиционным устройством)
47	C_RC_NA_1	Команда регулирования ступени	Команда для регулирования положения ступени
48	C_SE_NA_1	Команда установки значения, нормализованное	Команда установки нормализованного значения
49	C_SE_NB_1	Команда установки значения, масштабированное	Команда установки масштабированного значения
50	C_SE_NC_1	Команда установки значения, короткое с плавающей точкой	Команда установки значения в формате с плавающей точкой
51	C_BO_NA_1	Команда битовой строки из 32 бит	Команда для передачи 32-битной строки
58	C_SC_TA_1	Одиночная команда с меткой времени CP56Time2a	Одиночная команда с меткой времени CP56Time2a
59	C_DC_TA_1	Двойная команда с меткой времени CP56Time2a	Двойная команда с меткой времени CP56Time2a
60	C_RC_TA_1	Команда регулирования ступени с меткой времени CP56Time2a	Команда регулирования ступени с меткой времени CP56Time2a
61	C_SE_TA_1	Команда установки значения, нормализованное с меткой времени CP56Time2a	Команда установки нормализованного значения с меткой времени CP56Time2a
62	C_SE_TB_1	Команда установки значения, масштабированное с меткой времени CP56Time2a	Команда установки масштабированного значения с меткой времени CP56Time2a
63	C_SE_TC_1	Команда установки значения, короткое с плавающей точкой с меткой времени CP56Time2a	Команда установки значения в формате с плавающей точкой с меткой времени CP56Time2a



:: Устройство	:: Свойс...	:: Адрес объек...	:: Тип расширения	:: ASDU адрес	Цикличе... :: интервал, мин	Тип :: информационн... объекта	Тип :: управляющего объекта
AI_002 • Датчик аналоговый ▼	value	5	Публиковать значение ▼	1	2	M_ME_TE (35) ▼	
AO_001 • Актуатор аналоговый ▼	value	1	Публиковать значение ▼	1	1	M_ME_TF (36) ▼	
DO_001 • Актуатор бинарный ▼	state	2	Принять значение ▼	1			C_SC_NA (45) ▼

# Плагин Stimulsoft Report

## Описание

Плагин позволяет использовать отчеты Stimulsoft Report.js прямо из интерфейса системы.

Настройка и создание отчетов доступно в разделе **Аналитика->Отчеты Stimulsoft**

# Плагин SNMP

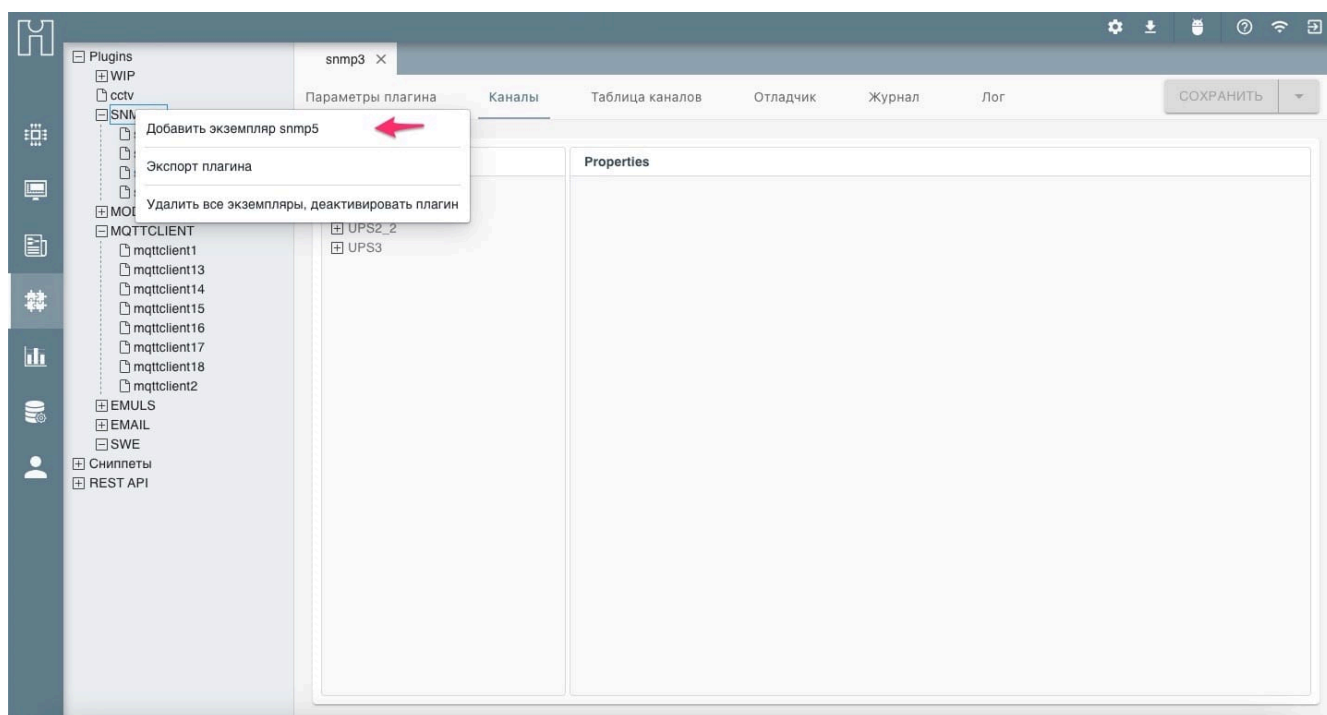
## Назначение

Плагин SNMP предназначен для работы с устройствами по протоколу SNMP (Simple Network Management Protocol). К поддерживаемым SNMP устройствам относятся маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, источники бесперебойного питания (UPS) и другие.

## Настройка

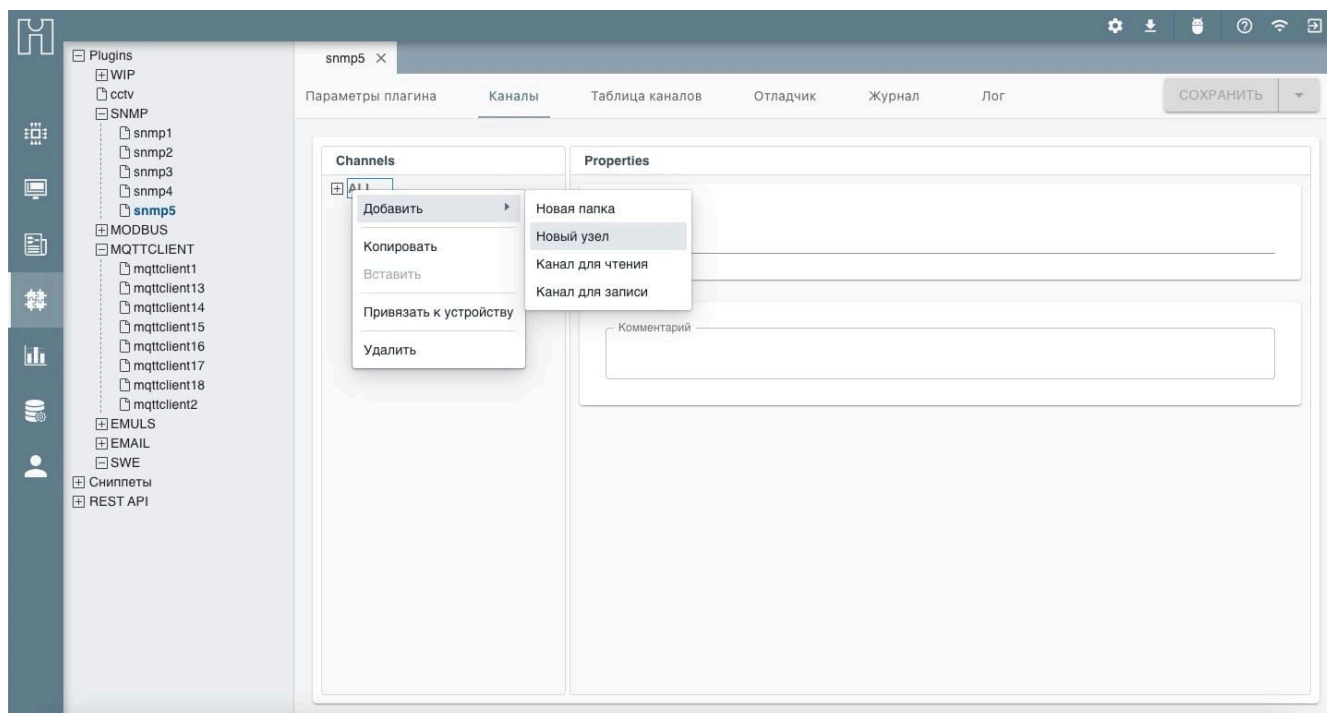
Любой плагин запускается системой в отдельном процессе. Для большинства плагинов, при необходимости, можно запустить несколько экземпляров для обслуживания устройств с разными IP адресами.

1. Для добавления экземпляра плагина нужно правой кнопкой мыши нажать на плагине и выбрать пункт "Добавить экземпляр":

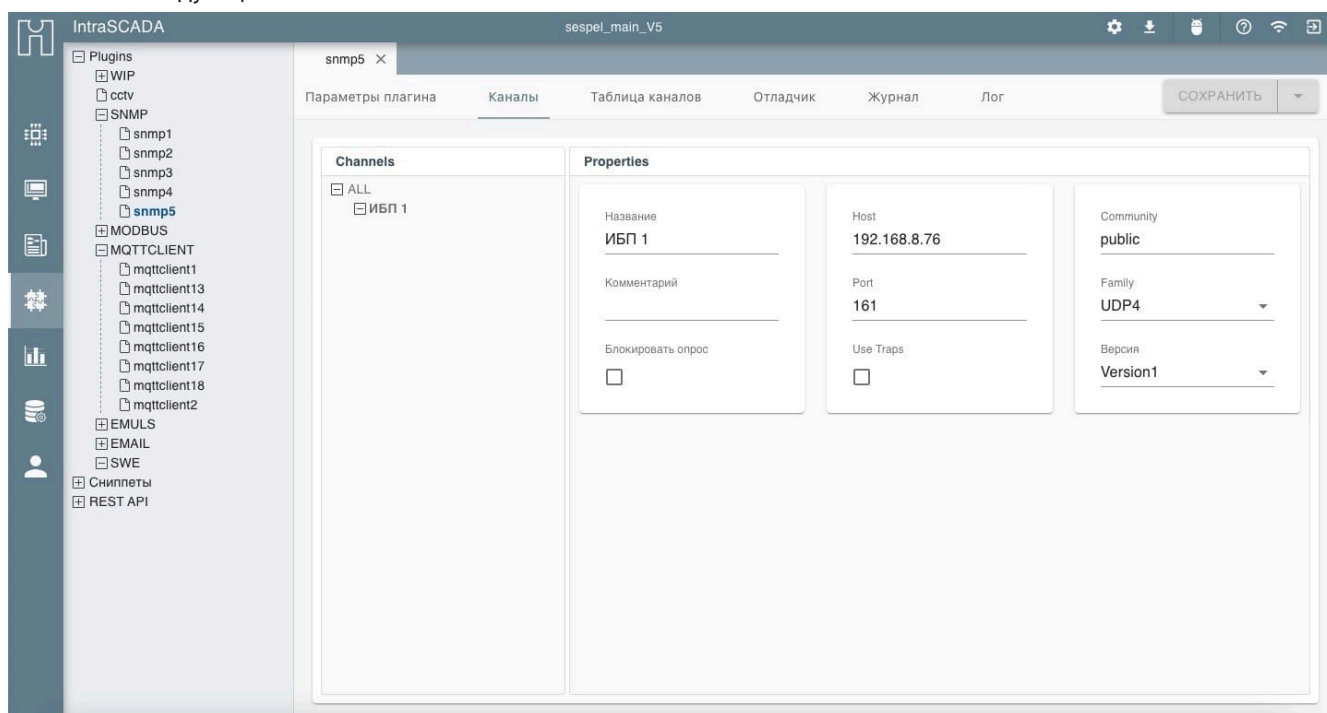


2. В дереве каналов правой кнопкой мыши нажать на папке ALL и выбрать "Добавить - Новый узел".

Узел - это отдельное устройство с ip адресом. Например, источник бесперебойного питания (ИБП).

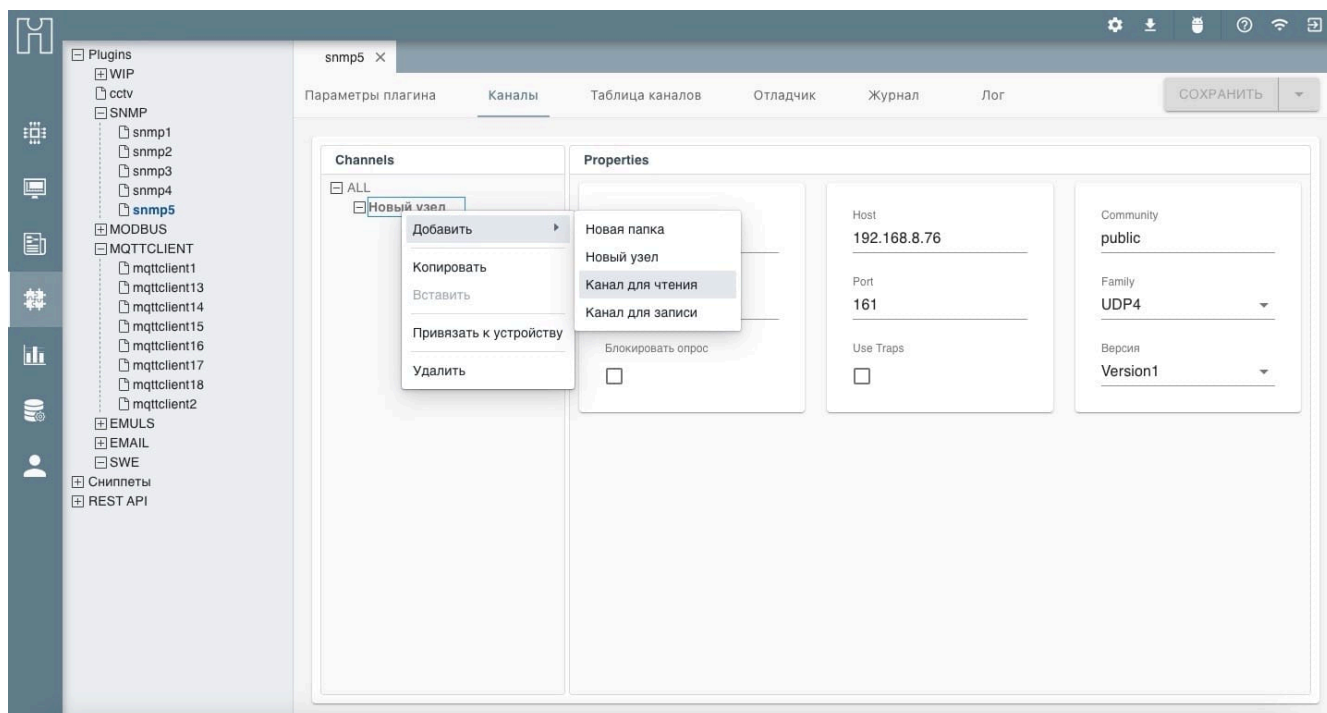


Заполнить следующие поля:

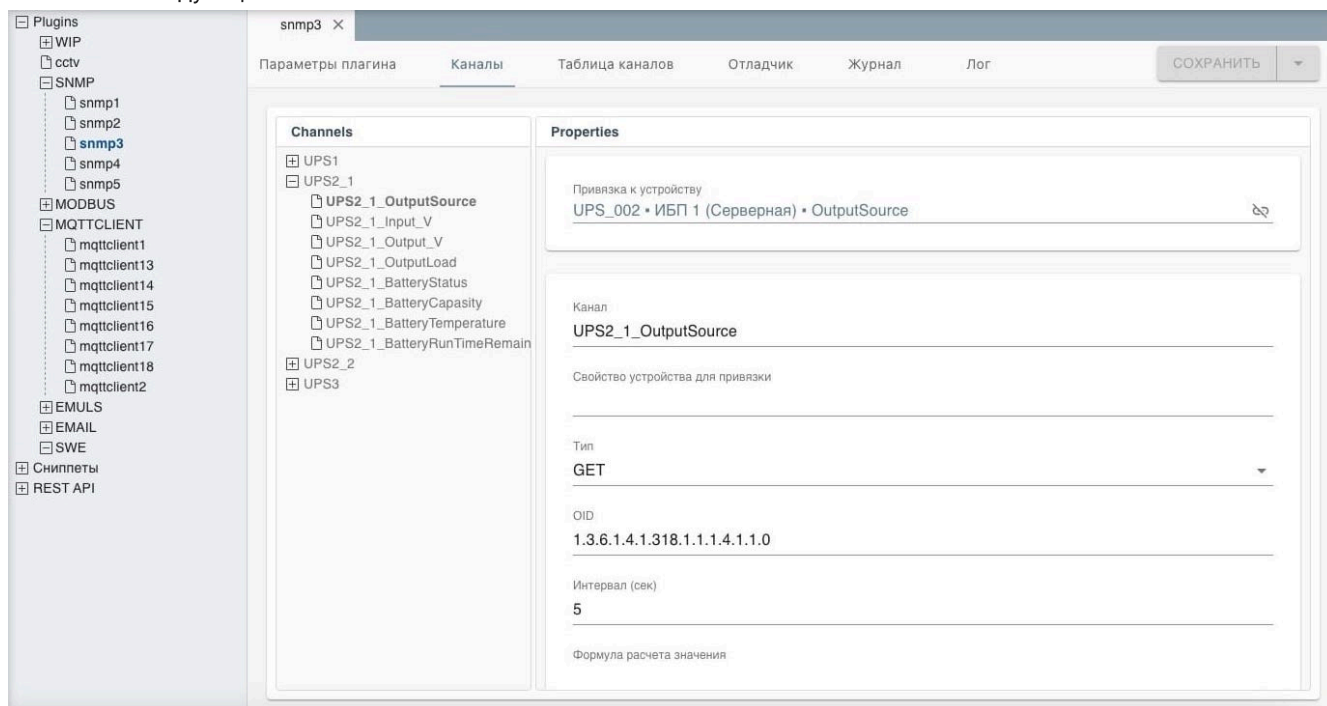


- Название узла
- Host - IP адрес подключаемого устройства
- Port - порт устройства. По умолчанию — 161
- Community - пароль. По умолчанию для большинства устройств — public
- Версия - версия протокола

3. В дереве каналов правой кнопкой мыши нажать на папке ALL и выбрать "Добавить - Канал для чтения":



Заполнить следующие поля:



- Привязка к устройству. Привязку канала к свойству устройства в системе.
- Канал - название канала
- Тип — тип запроса. (GET, TABLE, TRAP)
- OID - Идентификатор объекта. Этот идентификатор можно найти в документации на устройство. Идентификаторы в некоторых устройствах могут начинаться с точки. Здесь точку в начале OID ставить не нужно.
- Интервал - период опроса
- Формула - преобразование входного значения (например, value/10)

# API плагина на JS

Данный раздел описывает API встроенной библиотеки **pluginapi**. Библиотека предоставляет объект **plugin** для взаимодействия с сервером.

Код для подключения библиотеки дан в разделе [Плагин на JS](#).

## Получение данных с сервера - **get()**

Операция получения данных - это асинхронная операция.

В библиотеке используется **async/await** синтаксис, что дает возможность организовать асинхронную работу в синхронном стиле.

Вызов через **await** накладывает ограничение:  
функция, в которой используется вызов метода с **await**, должна быть объявлена как **async**

```
async function processChannels() {  
  // Отправили запрос на сервер и получили результат – все в одной строке  
  const result = await plugin.channels.get();  
  // ....Обработка ...  
}
```

Плагин может запросить:

- параметры плагина
  - **await plugin.params.get()**
- каналы плагина для опроса (с установленным флагом на чтение)
  - **await plugin.channels.get()**
- все данные по каналам (узлы, папки, каналы)
  - **await plugin.devhard.get()**
- данные с вкладки расширений плагина
  - **await plugin.extra.get()**
- постоянные данные, которые каждый плагин может сохранить в нужном ему формате
  - **await plugin.persistent.get()**
- данные папок в дереве устройств

- **await plugin.places.get()**
- данные устройств, можно задать фильтры для получения подмножества
  - **await plugin.devices.get()**
    - `plugin.devices.get({ did: ["d0121", "d0234", "d0123"] })` // по списку устройств
    - `plugin.devices.get({ tag: "Климат" })` // с тегом Климат
    - `plugin.devices.get({ location: "/place/dg004" })` // из заданной папки
- исторические данные, которые находятся в исторической БД
  - **await plugin.get('hist', {dn\_prop:'AI\_002.value', start, end})**
- данные устройств расширенные
  - **await plugin.get('devices', {location:'/place/dg043/'}, {alerts: true, dbsave:true});**
- узлы каналов плагина *Добавлено v5.17.82*
  - **await plugin.nodes.get()**

## Отправка данных на сервер - **send\*()**

Отправка данных выполняется без ожидания результата, **await** не требуется

### Отправить данные каналов - **sendData**

- **sendData([id, value, ts, chstatus],...)**

```
const ts = Date.now();
plugin.sendData([id: 'DI1', value: 1, ts}, {id: 'DI2', value: 0, ts}]);
```

### Отправить запись в журнал плагинов - **sendLog**

*Добавлено: v5.17.68*

Применяется для записи в серверный Журнал плагинов.

- **sendLog(txt || {txt, level})**

```
// Вариант 1
plugin.sendLog(
  'Этот текст будет записан в Журнал плагинов как обычное сообщение'
);

// Вариант 2
plugin.sendLog(
  {
    txt: 'Этот текст будет записан в Журнал плагинов как аварийное сообщение',
    level: 2
  }
);
```

### Отправить архивные данные - `sendArchive`

- `sendArchive({id, value, ts},...)`

### Отправить команду - `sendCommand`

- `sendCommand(command)`

### Отправить ответ на поступивший запрос - `sendResponse`

- `sendResponse(message, response)`

## Отправка служебных сообщений - `send`

Для отправки служебных сообщений нужно явно указать тип сообщения `type`:

- `send({type<, op, data, ...>})`

### Отправить данные для системного индикатора

```
process.send({
  type: 'procinfo',
  data: {
    current_endpoint: plugin.params.data.endpointUrl
  }
});
```

### Отправить данные сканирования



```
this.plugin.send({ type: 'scan', op: 'list', data, uuid });
```

## Отправить данные для изменения каналов

Добавлено: v5.18.32

При использовании **type:'channels'** плагин может менять свои каналы

- **send({type:'channels', op:'add', data})** - добавление канала(-ов)
- **send({type:'channels', op:'update', data })** - изменение канала(-ов)
- **send({type:'channels', op:'remove', data })** - удаление канала(-ов)
  - data - может быть массивом или одиночным объектом
    - Для update необходимо указать id канала и изменяемые свойства
    - Для remove нужен только id канала

На стороне сервера функционал реализуется через [Операции CRUD -> Работа с каналами](#)

## Обработка сообщений с сервера

Сервер может присылать на плагин различные сообщения.

Обработка сообщений сервера выполняется путем регистрации обработчиков.

При регистрации задается функция, которая будет выполнена при получении данного типа сообщения.

### Обработчик команд для каналов - onAct(cb)

Сервер присылает **message** в формате:

- {type: 'act', data: [{id:<id канала>, <...данные канала>, value:xx},...]}
  - data - массив команд, даже если отправлена одна команда
  - Данные канала - это набор атрибутов, зависит от структуры канала данного плагина. Сервер всегда присылает всю информацию о канале

Для отработки команд регистрируется обработчик **plugin.onAct**

```
plugin.onAct(message => {
  plugin.log('Получен массив команд канала: ' + util.inspect(message.data));
  message.data.forEach(item => {
    // Выполнение команд ...
  })
});
```

Например, вот такое сообщение поступает в плагин mqtt:

```
{
  type: 'act',
  data: [
    {
      id: 'ch_1',
      unit: 'mqttclient1',
      r: 1,
      w: 1,
      topic: '/devices/dev42',
      // плагин будет использовать этот топик для отправки сообщения
      pubtopic: '/devices/dev42/set',
      value: 1
    }
  ]
}
```

### Обработчик команды для плагина - onCommand(cb)

Команды плагину могут отправлять сценарий или скрипт визуализации.

Здесь нет привязки к каналам, можно отправить любую команду.

Плагин должен обработать и может отправить результат, который получит отправитель команды.

Сервер присылает **message** в формате:

```
{type: 'command', unit: 'myplug1', uuid:'TKQmn23WZ',sender: 'scene_scen003', ..... }
```

- type - 'command'
- unit - имя плагина, которому была направлена команда
- sender - отправитель команды (сценарий)
- uuid - уникальный идентификатор сообщения

Можно добавить любые дополнительные свойства, которые будет обрабатывать плагин, например:

```
{
  type: 'command',
  command: 'mytest',
  reqStr: 'Hello',
  unit: 'myplug1',
  uuid: 'TKQmn23WZ',
  sender: 'scene_scen003',
  // ...
}
```

Для отработки команд регистрируется обработчик **plugin.onCommand**. Отправка результата выполняется с помощью **plugin.sendResponse**.

Аргументы **sendResponse**:

- первый аргумент - **result** - объект, содержащий ответ.  
В него нужно обязательно включить данные из запроса: **type**, **unit**, **uuid**
- второй аргумент - **response** - флаг результата:
  - 1 - команда выполнена успешно,
  - 0 - не удалось выполнить команду

```
plugin.onCommand(msg => {
  // Обрабатываем запрос и получаем результат
  const mydataArray = [1,2,3,4,5];
  // Формируем сообщение для ответа
  const result = {
    mydataArray,
    myStr: 'OK',
    type: 'command',
    unit: msg.unit,
    uuid: msg.uuid
  };

  // Отправляем ответ
  plugin.sendResponse(result, 1);
});
```

Если входящее сообщение не содержит много данных, можно отправить ответ, объединив его с входящим сообщением.

И, конечно, функция обработчика может быть асинхронной, если это необходимо:

```
plugin.onCommand(async (msg) => {
  plugin.log('Получено сообщение: ' + util.inspect(msg));
  const mydataArray = await getMydataArray(msg);
  const result = Object.assign(msg, { mydataArray });
  plugin.sendResponse(result, 1);
});
```

Пример скрипта визуализации с командой плагина, подробнее в [API скрипта визуализации](#):

```

module.exports = async function({ local, context, source }, core, debug) {

  const result = await core.pluginCommand(
    {
      unit: 'myplug1',
      command: 'test',
      arg: 42
    },
    1
  );

  debug(result);

  // ..... обработать ответ и отправить клиенту, который вызвал скрипт
};

```

### Обработчик команд процесса сканирования - onScan(cb)

### Обработчик изменений данных - onSub(name,< filter,> cb)

Подписывается на изменения данных на сервере. При получении информации об изменениях - запускает функцию.

Это подписка на изменения всех свойств всех устройств.

```

plugin.onSub('devices', data => {
  plugin.log('devices data=' + util.inspect(data));
});

```

Подписка только на выбранные свойства выбранных устройств - используется фильтр:

```

plugin.onSub('devices', {did_prop:['d003.state', 'd042.value']}, data => {
  plugin.log('devices data=' + util.inspect(data));
});

```

### Обработчик изменений настроек (параметров, каналов) - onChange(name, cb)

Подписывается на изменения данных на сервере (параметров, каналов).

При получении информации об изменениях - запускает функцию.

```
plugin.onChange('channels', data => {
  plugin.log('Каналы изменились: ' + util.inspect(data));
  // Можно анализировать изменения и перестроить только измененные каналы
  // Можно запросить все каналы заново и перестроить все
  // Можно просто завершить работу плагина и начать все заново
});
```

## Служебные функции плагина

### Получить расшифрованный пароль

Для подключения к оборудованию/ресурсам плагину может быть нужен пароль/пароли.

Система обеспечивает механизм ввода, хранения и передачи паролей на плагин в зашифрованном виде.

- **getPassword(params, propname)**
  - params - объект в параметрах, полученный с сервера
  - propname - опционально - имя свойства с паролем. По умолчанию **password**

```
// В pw получим расшифрованный пароль
const pw = plugin.getPassword(plugin.params.data, 'pw_user');
```

Подробнее в разделе [Пароли](#)

### Записать в лог плагина

Каждый экземпляр плагина создает лог - это файл с именем *ih\_<имя\_экземпляра\_плагина>.log*, который находится в папке логов системы.

Запись в лог происходит, если уровень сообщения соответствует уровню логирования, заданному в параметрах при запуске плагина:

- 0 - Низкий уровень (пишется всегда)
- 1 - Средний уровень
- 2 - Высокий уровень
- **log(txt, level)**
  - txt - сообщение
  - level - уровень сообщения (по умолчанию 0)

Подробнее о просмотре и настройке логов в разделе [Логирование](#)

Также все сообщения этой команды выводятся в отладчик, независимо от level.

## Завершить работу плагина

- **exit(txt, level)**
  - code - код завершения плагина. Ненулевое значение сигнализирует об ошибке. Можно использовать свои коды для определения ошибки.
  - txt - сообщение об ошибке. Будет выведено в отладчик

```
process.on('SIGTERM', () => {
  terminate();
});

function terminate() {
  plugin.log('TERMINATE PLUGIN', 2);
  plugin.exit(17);
}
```

Для windows команда **SIGTERM** не срабатывает, поэтому можно воспользоваться встроенным обработчиком **plugin.onStop()**

```
if (plugin.onStop) {
  plugin.onStop(async () => {
    terminate();
  });
}
```

Команда завершение плагина особенно важна если нужно послать сигнал окончания взаимодействия серверу к которому вы подключаетесь, чтобы штатно завершить сеанс.

При завершении плагина (по любой причине, с ошибкой или без) сервер будет пытаться его перезапустить с интервалом, заданным в параметрах плагина.

Чтобы остановить плагин, нужно его переключить в **suspend**, что выполняется интерактивной командой **Stop**. Остановленный плагин находится в состоянии **Остановлен без перезапуска** и не будет запущен даже после перезагрузки сервера.

# Логирование

## Лог экземпляра

Каждый экземпляр плагина создает лог - это файл с именем *ih\_<имя\_экземпляра\_плагина>.log*, который находится в папке логов системы.

Последние записи лога можно увидеть на странице экземпляра плагина на вкладке **Лог плагина**. Логи ротируются, стандартный размер лога 128 KB, количество архивных файлов - 2.

Например:

- *ih\_modbus1.log* - это текущий лог экземпляра *modbus1*.
- *ih\_modbus1.log.1731682739286* - архивный лог экземпляра *modbus1*, где последнее расширение - это метка времени архивирования

## Настройка логов

Можно изменить размер лога и количество архивных файлов для конкретного плагина.

- Вариант 1. Параметризовать. В этом случае значения будут доступны для настройки.
  - Добавить в файл настройки параметров **v5/formPluginCommon.json** параметры:

```
{ "prop": "logsize", "title": "$LogSizeKB", "type": "number" },
{ "prop": "logrotate", "title": "$LogRotate", "type": "number" },
```

- В коде плагина после получения параметров передать их в логгер:

```
plugin.logger.setParams(params)
```

- Вариант 2. Установить фиксированные значения прямо в коде плагина.

```
plugin.logger.setParams({logsize:500, logrotate:7})
```

## Журнал плагинов на сервере

Сервер ведет журнал плагинов, который хранится в БД в таблице **pluginlog**. В журнале фиксируются время запуска, параметры, а также время и код выхода каждого экземпляра плагина.

Эти записи можно видеть на вкладках Журнал (корневой папки Плагины, папки плагина или конкретного экземпляра).

В версии 5.17.68 добавлена возможность делать записи в этот журнал из кода плагина.

```
plugin.sendLog('Пишем в Журнал плагинов')
```



## Свойства индикатора плагина

Для каждого экземпляра плагина автоматически создается устройство [Индикатор плагина](#)

Свойства индикатора можно использовать как свойства обычных устройств системы: выводить на визуализацию, записывать в БД, строить по ним графики, запускать сценарии....

### Встроенные свойства

Встроенные (основные) свойства позволяют выполнять текущий контроль параметров процесса:

- state - Состояние работы плагина, заполняется сервером
- version - Текущая версия плагина, заполняется сервером
- memrss, memheap, memhuse - объемы памяти, используемые процессом, заполняется объектом plugin (встроенный функционал)

### Дополнительные свойства

Разработчик плагина может при необходимости добавить дополнительные свойства. Для этого в манифесте создается раздел ext:

```
"ext": [  
  { "prop": "count", "note": "Счетчик попыток", "vtype": "N" },  
  { "prop": "myflag", "note": "Флаг чего-то там", "vtype": "B" },  
  { "prop": "mymessage", "note": "Сообщение от плагина", "vtype": "S" }  
]
```

Для обновления свойств плагин должен отправлять сообщения:

```
plugin.send({ type: 'procinfo', data: { count: 42, myflag: 1} });
```

### Состояние работы плагина

Состояние плагина (свойство state), как было указано выше, устанавливается сервером. Это удобно, так как большинство событий являются внешними по отношению к процессу:

- 0 - плагин не установлен
- 1 - плагин работает
- 2 - остановлен, без перезапуска (Suspended)
- 3 - остановлен с ошибкой и будет перезапускаться системой в соответствии с настройками

Таким образом, сервер устанавливает:

state <= 1 При запуске плагина

state <= 2 При остановке плагина (suspend)

state <= 3 При выходе плагина с ошибкой

Есть возможность расширить набор состояний плагина, добавив состояние:

- 10 - В процессе запуска

Для этого нужно в манифест добавить флаг:

```
"hasTryStartState":1,
```

Если этот флаг установлен, сервер при запуске плагина установит:

state <= 10 (В процессе запуска).

Плагин сам должен переключить состояние в 1, отправив сообщение:

```
plugin.send( { type: 'state', data: 1 } );
```

При необходимости (например, если выполняется реконнект без перезапуска плагина ) можно переключать состояния 10 => 1 => 10 => 1.

# REST API

Любое стороннее приложение может взаимодействовать с IntraSCADA через REST API.



## Настройка

По умолчанию REST API в системе не настроен, разработчик проекта сам создает маршруты и обработчики, исходя из необходимости.

Сервер IntraSCADA слушает /restapi запросы на том же порту, что и основной интерфейс.

Запрос будет выглядеть так:

```
<url сервера IntraSCADA>/restapi/<ваш маршрут>?<ваши параметры>
```

Например, можно создать маршрут /device для получения значений свойств или конкретного свойства:

```
http://192.168.0.245:8088/restapi/device?dev=Vent_007&prop=temp2
```

При получении запроса /restapi/<ваш маршрут> запускается обработчик маршрута и возвращается результат. В обработчике можно использовать любые функции ядра системы как для мониторинга, так и для управления объектами проекта.

## Встроенный маршрут

Если включен флаг **Доступ к REST API с использованием токена**, будет обрабатываться встроенный endpoint /restapi/auth. Подробнее [Доступ к API](#)

## Доступ к REST API

По умолчанию доступ к API не требует авторизации.

Для подтверждения прав доступа и защиты информации можно использовать механизм токенов.

Для этого в **Системных настройках** установите флаг **Доступ к REST API с использованием токена**. В этом случае в заголовок каждого запроса нужно включать валидный токен.

Для получения токена необходимо выполнить аутентификацию.

## Учетные записи для работы с API

Получить токен можно для любой учетной записи, имеющей флаг **Эксперт**.

С точки зрения безопасности рекомендуется создать специальную учетную запись, установить флаг **Эксперт**, но не включать ее ни в одну из групп, таким образом ограничив доступ только работой с REST API, а вход в UI и административный интерфейс будут запрещены.

## Аутентификация и получение токена

Аутентификацию обеспечивает встроенный запрос **/restapi/auth**. В теле запроса передается JSON-объект, содержащий login и хэш пароля в кодировке **hex**. В ответ будут передан токен.

**/restapi/auth** - единственный встроенный endpoint среди маршрутов **/restapi**. Он работает только при установленном флаге **Доступ к REST API с использованием токена**

- Request: **/restapi/auth**
- Method: POST
- body: {"login": "<логин учетной записи>", "password": "<хэш пароля в кодировке hex>"}
- Response: {"login": "<логин учетной записи>", "token": "<токен>"}
- Error status:
  - 400 Bad request - сервер не смог понять запрос (ожидается POST запрос и JSON в body)
  - 401 Unauthorized - неверный логин и/или пароль

## Передача пароля

В целях безопасности пароль передается не в открытом виде, а в виде хэша в кодировке **hex**, полученный методом SHA512.

Пример вычисления хэша на Node.js.

```
const hash = require('crypto')
  .createHash('sha512')
  .update(password)
  .digest('hex');
```

Пример запроса с помощью curl и полученный результат

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -d '{
    "login": "apiadmin",
    "password": "980f5736c9d474c001bfc82b09e4dd7eb8f8663621d1581b2fb06b5934
6 2c60fdc2bbb12ffdc744203602b03a8336c38175dea4271ec41fab0f251e529"
  }' \
  http://127.0.0.1:3000/restapi/auth

HTTP/1.1 200 OK

{
  "login": "apiadmin",
  "token": "6807a173a559426fd032a3c3ac8e0f933a940ddbef07cbc59349a77b0b9a35\
8c"
}
```

Для Windows необходимо будет использовать вместо одинарных кавычек двойные, а двойные экранировать

```
curl -i -X POST \
  -H "Content-Type: application/json" \
  -d '{
    "login": "apiadmin",
    "password": "980f5736c9d474c001bfc82b09e4dd7eb8f8663621d1581b2fb06b5934
  }' \
  http://127.0.0.1:3000/restapi/auth

HTTP/1.1 200 OK

{
  "login": "apiadmin",
  "token": "6807a173a559426fd032a3c3ac8e0f933a940ddbef07cbc59349a77b0b9a358
}
```

## Использование токена в запросах

- Request: **/restapi/devices?type=t002**
- Method: GET
- Header: 'token: <токен, полученный при авторизации>'
- Response: <ответ в формате, заданном в обработчике>

Пример запроса с помощью curl

```
curl \
  http://127.0.0.1:3000/restapi/devices?type=t002 \
  -H 'token: 6807a173a559426fd032a3c3ac8e0f933a940ddbef07cbc59349a77b0b9a35'
```

Обратите внимание, что при каждом запросе /restapi/auth токен будет регенерирован, а предыдущий токен учетной записи удален.

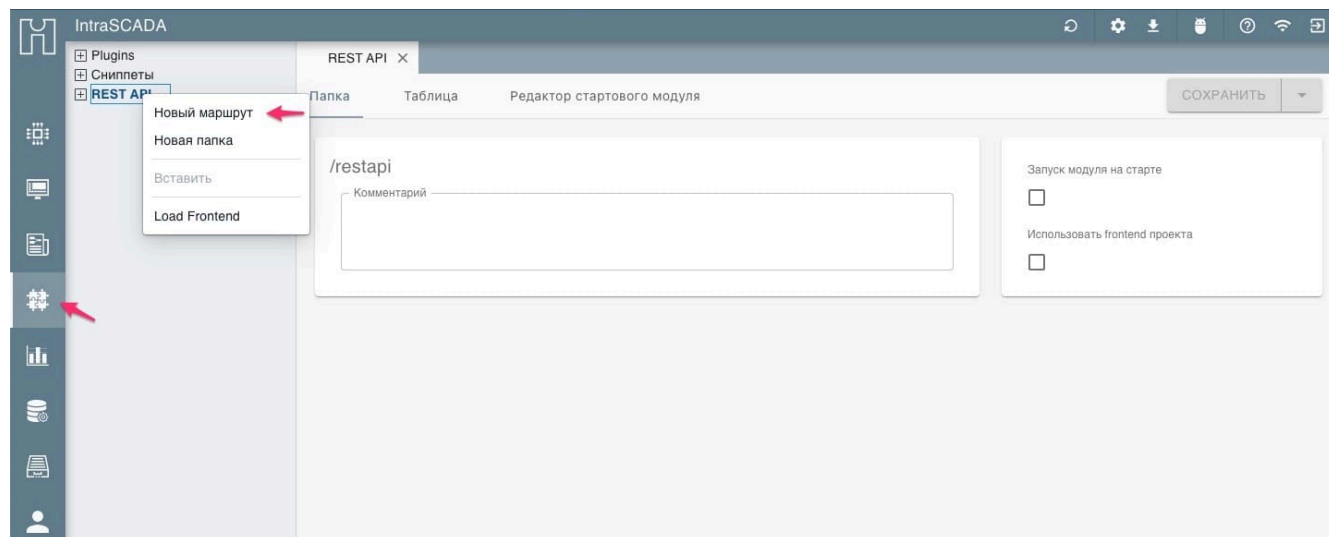
Если есть необходимость выполнения независимых запросов от разных источников, лучше создать несколько технических учетных записей, у каждой будет свой токен

# Маршрут

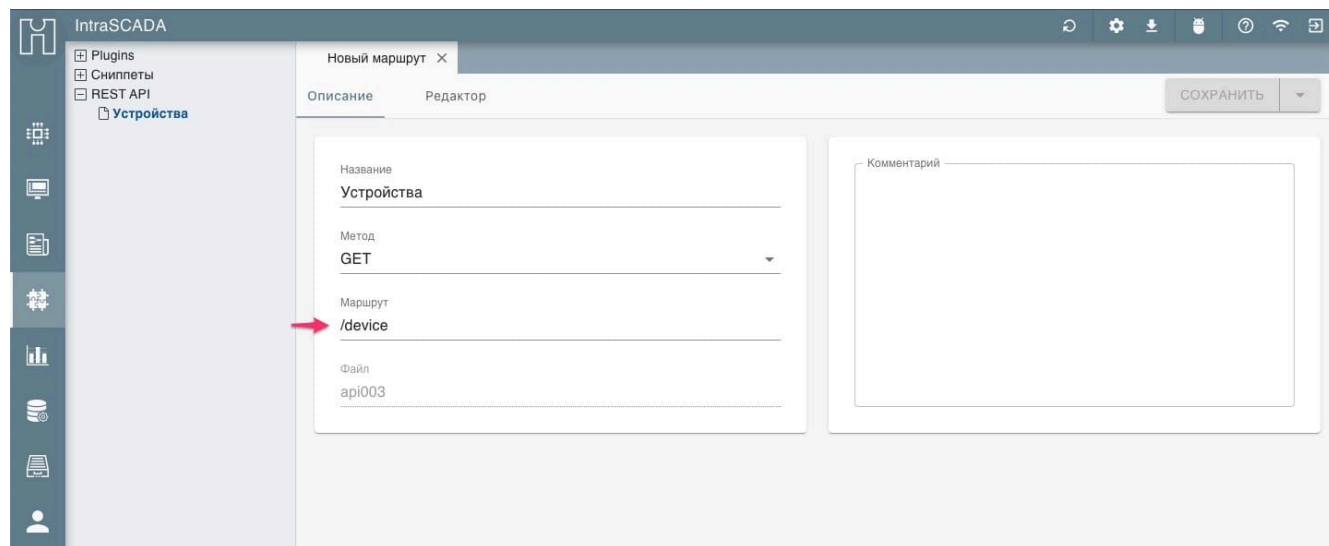
Для создания запроса нужно определить маршрут и обработчик запроса.

## Создание маршрута

В разделе **Источники данных** правой кнопкой мыши на папке **REST API** добавляем новый маршрут:



Заполняем соответствующие поля:



- Название - любое имя.
- Метод - GET или POST
- Маршрут - будет использоваться в запросе.

Обратите внимание на обязательный слеш в начале (**/device**).

Можно построить многоуровневую маршрутизацию, например: **/myrequest/list/parts**.

Затем после знака **?** можно (но не обязательно) передать параметры, например **/device?dev=PUMP1&prop=state**.

Разработчик проекта сам определяет, какие параметры будут передаваться.

Их нужно будет обработать в скрипте обработчика.



## Обработчик запроса

Скрипт обработчика создается на вкладке **Редактор**. Когда придет запрос, содержащий заданный маршрут, будет запущен связанный с ним обработчик.

Если в этот момент вкладка **Редактор** этого маршрута открыта, в консоли отладчика, расположенной в нижней части экрана, можно увидеть, что запрос поступил.

Например, при поступлении запроса `/restapi/device?dev=VENT1&prop=value` в отладчике будут сообщения:

```
04.02 17:30:56.169 => HTTP GET /device
04.02 17:30:56.169   query:{ dev: 'VENT1', prop:'value' }
```

Также в отладчик можно выводить данные из скрипта, используя объект **debug**..

## Структура скрипта.

При добавлении нового маршрута выводится скрипт по умолчанию, который для примера возвращает данные из пользовательской таблицы.

Рассмотрим структуру скрипта более подробно. Скрипт должен выполнить 3 задачи:

- разобрать входящий запрос;
  - получить данные от системы в соответствии с запросом;
  - отправить ответ, даже если на каком-то этапе возникла ошибка.
- (То есть в любом случае в рамках обработчика ответ нужно вернуть)

Обычно скрипт состоит из блока **try**, который должен завершиться отправкой ответа.

Если возникает ошибка, она перехватывается блоком **catch**, в котором нужно отправить ответ с информацией об ошибке.

```

module.exports = async (req, res, holder, debug) => {
  try {
    const filter = req.query;
    // Получение данных от системы
    const data = await holder.dm.get('mytable', filter, {order: 'name'});
    // отправка ответа в формате JSON
    res.json({ res: 1, data });
    // вывод результата в отладчик
    debug(JSON.stringify({ res: 1, data }))
  } catch (e) {
    // отправка ответа с сообщением об ошибке
    res.json({ res: 0, message: e.message });
    // вывод результата в отладчик
    debug(e.message);
  }
};

```

## Параметры скрипта.

При запуске обработчик получает четыре параметра:

- **req** - объект запроса (request)
  - Внутри req много объектов, интерес представляют:
    - req.url - строка запроса
    - req.query - объект, содержащий данные после знака вопроса уже в разобранном виде (этот объект сразу виден в отладчике)
    - req.headers - объект содержит заголовки запроса в виде ключ-значение (если нужен анализ заголовка)
    - req.body - содержит тело POST запроса
- **res** - объект, через который будет отправлен ответ (response)
  - res.send("строка") - отправит ответ в виде строки, будет установлен Content-Type='text/plain' или
  - res.json({...}) - отправит ответ в виде json, аргумент - объект, Content-Type='application/json'

При необходимости можно установить соответствующие заголовки, используя res.setHeader:

```

// эквивалентно res.json():
res.setHeader('Content-Type', 'application/json');
res.send(JSON.stringify(result));

```

По умолчанию для ответа устанавливается status = 200.

Можно отправить свой статус:

```
res.status(500).send('Что-то пошло не так...');
```

- **holder** - объект для доступа к данным системы. В частности:
  - holder.dnSet[<ID устройства>] - доступ к устройствам
  - holder.dm.get('mytable') - доступ к пользовательским таблицам

Раздел "[Методы API](#)" содержит список доступных методов объекта holder, включая операции CRUD для создания/редактирования объектов проекта

- **debug** - функция для отладки запросов - опционально. Параметр - строка. Например, добавим в обработчик этот код:

```
debug('Url = ' + req.url)
debug('Headers =' + JSON.stringify(req.headers))
```

При получении запроса <http://189.89.89.89:8088/restapi/device?dev=VENT1&prop=value> в отладчике увидим:

```
04.02 17:30:56.167 => HTTP GET /device.  <= это выводится по умолчанию
04.02 17:30:56.169   query: { dev: 'VENT1', prop: 'value' }  <= это выводи
04.02 17:30:56.169 Url = /device?dev=VENT1&prop=value
04.02 17:30:56.170 Headers = {
  "host": "189.89.89.89:8088",
  "connection": "keep-alive",
  "upgrade-insecure-requests": "1",
  "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit
  "accept-encoding": "gzip, deflate",
  "accept-language": "ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7",
  "cookie": "_ga_W2EKGR4X42=GS1.1.1619095530.1.0.1619095530.0; _ga=GA1.1.16
}
```

## Обработка POST запроса

Объект POST запроса имеет req.body - тело запроса после парсинга.

Формат req.body определяет Content-Type запроса: для application/json это объект, для text/plain - строка

```
03.09 11:00:44.421 => HTTP POST /remotedevice
03.09 11:00:44.421   body: { name: 'test1', value: 77 }
03.09 11:00:44.421 application/json
03.09 11:00:44.421 typeof req.body = object
```

```
03.09 10:51:56.498 => HTTP POST /remotedevice
03.09 10:51:56.498   body: 'test text\n'
03.09 10:51:56.498 text/plain
03.09 10:51:56.498 typeof req.body = string
```

## Примеры

### Запрос значения свойства устройства методом GET

Пусть запрос будет такой: `/restapi/device?dev=<ID устройства>&prop=<название свойства>`.

В ответ просто отдаем значение. Если возникла ошибка - вернем статус ошибки 500.

```

/**
 * Запрос текущего значения свойства устройства
 * /device?dev=VENT1&prop=value**
 */
module.exports = async (req, res, holder, debug) => {
  try {
    const dev = req.query.dev;
    const prop = req.query.prop || 'value';
    if (!dev) throw {message: 'Устройство не задано'}

    const devObj = holder.dnSet[dev]; // Найти объект устройства
    if (!devObj) throw {message: 'Устройство '+dev+' не найдено!'}

    const value = devObj[prop]; // Текущее значение свойства prop
    res.send(String(value));
    debug(String(value)); // Это только для отладки – продублировать резу

    // Другой вариант – можно вернуть подробный json
    // const result = { res: 1, dev, prop, value };
    // res.json(result);
    // debug(JSON.stringify(result));

  } catch (e) {
    debug('ERROR: ' + e.message)
    res.status(500).send('Error: ' + e.message);

    // Другой вариант – можно вернуть json с описанием ошибки
    // res.json({ res: 0, message: e.message });
  }
};

```

Обратите внимание: Можно сформировать ответ в любом нужном принимающей стороне формате.

К примеру, запрашивается температура приточного воздуха (temp2) вентиляционной установки (Vent\_007):

```
http://189.52.31.134/restapi/device?dev=Vent_007&prop=temp2
```

В ответ отправляем значение:

29

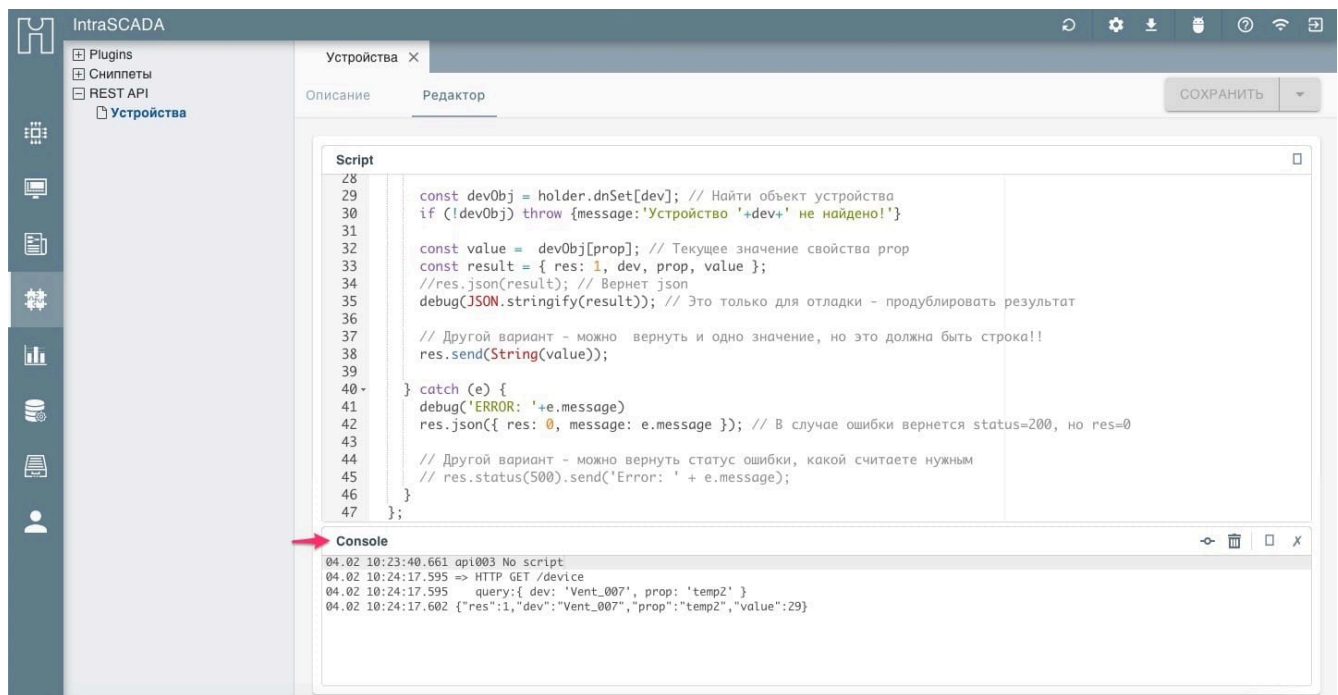
Если необходимо отправлять данные в JSON формате, достаточно:

1. Закомментировать строку `// res.send(String(value));`
2. Раскомментировать строки  
`const result = { res: 1, dev, prop, value }; res.json(result);`

И вот уже отправляем данные в JSON формате:

```
{"res":1,"dev":"Vent_007","prop":"temp2","value":29}
```

В окне отладчика **Console** можно наблюдать выполнение запроса:



## Запрос записи значения свойства устройства методом GET

Пусть запрос будет такой: `/restapi/device/set?dev=<ID устройства>&prop=<Название свойства>&value=<Значение свойства>`.

В ответ просто отдаем значение. Если возникла ошибка - вернем статус ошибки 500.

```

/**
 * Запись значения в свойство устройства
 * /device/set?dev=VENT1&prop=setpoint&val=25**
 */
module.exports = async (req, res, holder, debug) => {
  const sleep = ms => new Promise(resolve => setTimeout(resolve, ms));
  try {
    const dev = req.query.dev;
    const prop = req.query.prop || 'value';
    const val = req.query.val;
    if (!dev) throw {message: 'Устройство не задано'}

    const curdev = holder.getDevice(dev); // Получить устройство из системы
    if (!devObj) throw {message: 'Устройство '+dev+' не найдено!'}

    curdev.setValue(prop, val); //Запись значения в свойство устройства
    await sleep(100);
    const curvalue = curdev[prop]; // Текущее значение свойства prop
    res.send(String(curvalue));
    // Это только для отладки – продублировать результат
    debug(String(curvalue));

    // Другой вариант – можно вернуть подробный json
    // const result = { res: 1, dev, prop, value };
    // res.json(result);
    // debug(JSON.stringify(result));

  } catch (e) {
    debug('ERROR: ' + e.message)
    res.status(500).send('Error: ' + e.message);

    // Другой вариант – можно вернуть json с описанием ошибки
    // res.json({ res: 0, message: e.message });
  }
};

```

К примеру, необходимо записать уставку температуры приточного воздуха (setpoint) вентиляционной установки (Vent\_007):

```
http://189.52.31.134/restapi/device/set?dev=Vent_007&prop=setpoint&val=25
```

В ответ отправляем значение после небольшой паузы, чтобы успел отработать обработчик или значение успело отправиться на устройство через плагин:

25

### Запрос записи значения свойства устройства методом POST

- Запрос: `**/restapi/device/set`
- Content-Type = `application/json`
- Тело запроса: `{ dev:<ID устройства>, prop:<Название свойства>, value:<Значение свойства> }`

Обработчик будет отличаться только тремя строками - нужно использовать `req.body` вместо `req.query`

```
/**
 * Запись значения в свойство устройства методом POST
 * /device/set
 * body:{dev,prop,value}
 */
module.exports = async (req, res, holder, debug) => {
  const sleep = ms => new Promise(resolve => setTimeout(resolve, ms));
  try {
    const dev = req.body.dev;
    const prop = req.body.prop || 'value';
    const val = req.body.val;
    // ... Аналогично обработчику GET запроса

  } catch (e) {
    res.json({ res: 0, message: e.message });
  }
};
```

### Запрос исторических данных за интервал времени методом GET

Пусть запрос будет такой: `/restapi/records?dev=< deviceId >&start=< ts >&end=< ts >`. В ответ просто отдаем объект с историческими данными по конкретному устройству в JSON формате. Если возникла ошибка - вернем статус ошибки 500.



```

/**
 * Обработчик запроса REST API для исторических данных
 * /restapi/records?dev=< deviceId >&start=< ts >&end=< ts >
 */
module.exports = async (req, res, holder, debug) => {
  try {
    const dev = req.query.dev;
    const start = req.query.start || Date.now()-3600000;
    const end = req.query.end || Date.now();
    if (!dev) throw {message: 'Устройство не задано'};
    const curdev = holder.getDevice(dev);
    const records = await holder.dm.get(
      'records',
      {
        start,
        end,
        dn_prop: curdev.dn + ".value"
      }
    );

    res.send(JSON.stringify(records));

  } catch (e) {
    debug('ERROR: ' + e.message)
    res.status(500).send('Error: ' + e.message);
  }
};

```

### Запрос исторического журнала за интервал времени методом GET

Пусть запрос будет такой: `/restapi/journal?start=< ts >&end=< ts >`.

В ответ просто отдаем массив из строк исторического журнала в JSON формате. Если возникла ошибка - вернем статус ошибки 500.

```

/**
 * Обработчик запроса REST API для исторического журнала
 * /restapi/journal?start=<ts>&end=<ts>
 */
module.exports = async (req, res, holder, debug) => {
  try {
    const start = req.query.start || Date.now()-3600000;
    const end = req.query.end || Date.now();
    const records = await holder.dm.logconnector.read(
`SELECT * FROM mainlog
WHERE ts >= ${start}
  AND ts <= ${end}
ORDER BY ts DESC`
    );

    res.send(JSON.stringify(records));

  } catch (e) {
    debug('ERROR: ' + e.message)
    res.status(500).send('Error: ' + e.message);
  }
};

```

К примеру, необходимо получить все сообщения журнала за интервал времени:

```
http://127.0.0.1:8088/restapi/journal?start=1720002423943&end=1720002431015
```

В ответ придет следующее:

```
[
  {
    "tags": "##",
    "did": "d1071",
    "location": "/place/dg005/dg006",
    "txt": "Состояние : Авария Датчик бинарный 2 (DI_003) ",
    "level": 2,
    "ts": 1720002431015,
    "tsid": "1720002431015_00001",
    "sender": "alert"
  },
  {
    "tags": "##",
    "did": "d1063",
    "location": "/place/dg004",
    "txt": "Нормализация",
    "level": 0,
    "ts": 1720002431015,
    "tsid": "1720002431015_00002",
    "sender": "alert"
  },
  {
    "tags": "##",
    "did": "d1067",
    "location": "/place/dg005/dg006",
    "txt": "Нормализация",
    "level": 0,
    "ts": 1720002431014,
    "tsid": "1720002431014_KmAtf2FGHQ",
    "sender": "alert"
  },
  {
    "tags": "##",
    "did": "d1071",
    "location": "/place/dg005/dg006",
    "txt": "Нормализация",
    "level": 0,
    "ts": 1720002430508,
    "tsid": "1720002430508_00001",
    "sender": "alert"
  }
]
```

ВАЖНО. Timestamp это время по UTC-0. В системе все хранится с временем UTC-0

## Аналитика

Данный раздел включает в себя:

- Графики - линейные, столбчатые, круговые
- Таймлайны
- Журналы
- Журналы тревог
- Отчеты

### Графики

Настройка графиков для отображения на пользовательском интерфейсе. Существует два механизма настройки: интерактивный и скриптом через обработчик.

### Таймлайны

Это графики в виде диаграммы Ганта. С помощью которых можно показать очередность появления событий.

### Журналы

В данные журналы попадают все сообщения системы. Вы можете настроить статические фильтры для отображения.

### Журналы тревог

Настройка журналов для оперативных сообщений. Так же можно настроить статические фильтры для вывода информации.

### Отчеты

Конфигуратор отчетов. Существует два механизма получения данных в отчете для таблиц: интерактивный и скриптом через обработчик

# Графики

Для того, чтобы увидеть значение на графике, нужно настроить [сохранение значения в БД](#) в экземпляре устройства!

Для вывода графика обычно нужны две составляющие:

- Визуальный элемент **Chart**, который размещается в контейнере или диалоге;
- Сущность **График** в разделе Аналитика -> Графики, который содержит правила формирования данных

Сущность **График** привязывается к элементу **Chart**. Подробно настройка визуальных элементов **Chart** описана в разделе [Визуализация -> Элементы -> Виды графиков](#)

Один и тот же **График** можно привязать к элементам **Chart** на разных экранах (контейнерах, диалогах).

В данном разделе речь идет о создании сущности **График**.

Настройки **Графика** используются, чтобы сформировать данные по запросу от виджета (визуального элемента графика).

Данные должны содержать не только числовые массивы (точки), но и некоторые параметры оформления (легенды, оси, цвета,...).

## Вид графика

Атрибут **Вид** определяет, с каким виджетом будет работать график.

В зависимости от **Вида** графика доступны различные опции настройки. Все графики позволяют выводить как несколько свойств одного устройства, так и данные разных устройств.

## Линейный

Многоперьевой линейный график работает с виджетами [Chart Multiline](#), [Chart Multiline GL](#)

В нижней таблице настраиваются перья:

- Значение выбирается из списка свойств устройств, которые пишутся в БД.
- Также нужно настроить легенду и цвет линии.
- Опционально можно настроить метод и цвет заполнения .
- При желании, чтобы изменить форму кривой, можно выполнить интерполяцию, выбрав метод интерполяции из списка.

Обязательно укажите толщину линии в настройках пера, иначе у вас график будет пустой.

На плашке справа можно настроить Min и Max значения по оси Y.

Если настроек нет, границы будут выбраны исходя из данных.

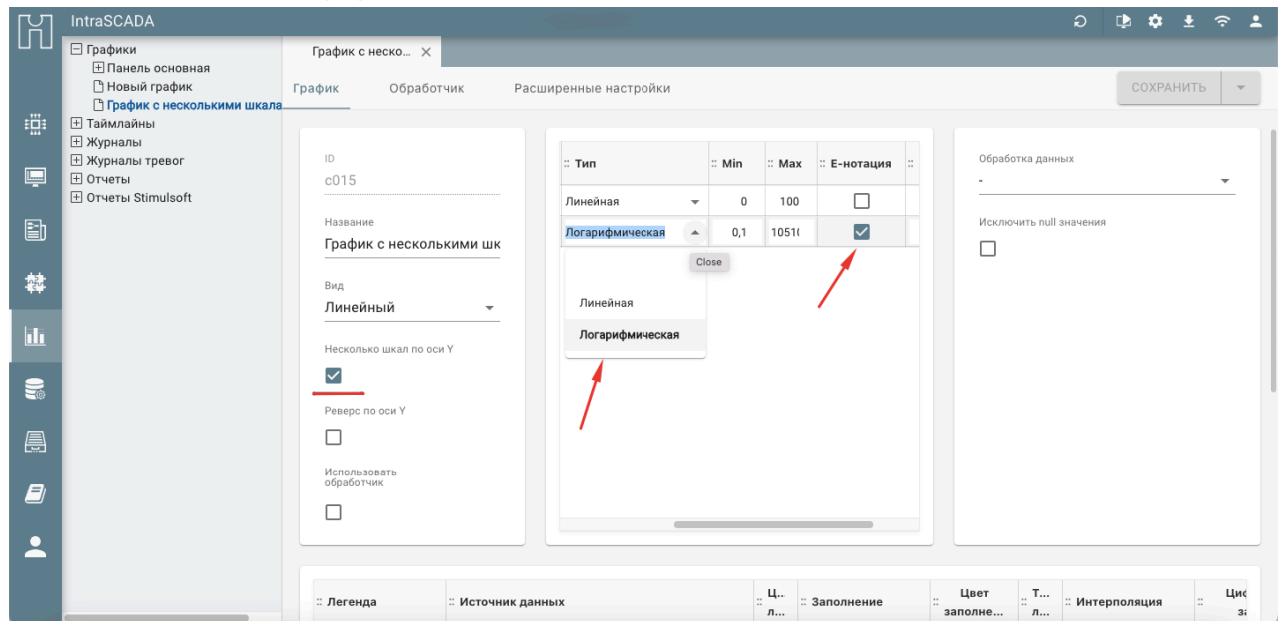
Также границы будут расширены автоматически, если данные не попадают в диапазон Min-Max.

## Пример создания линейного многоперьевого графика

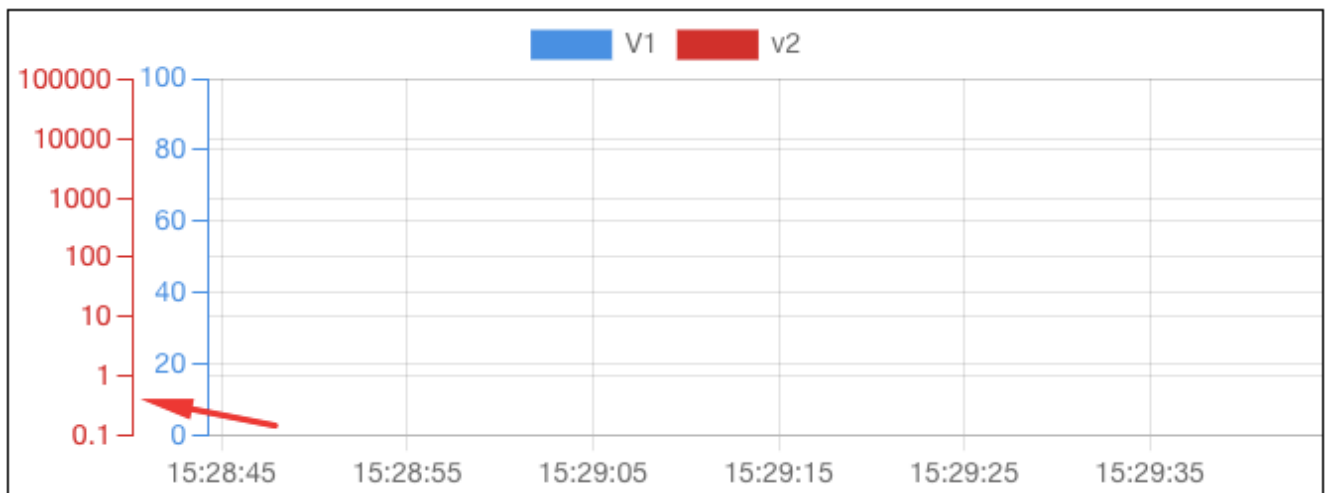
## Линейный график с несколькими шкалами

Добавлено: v5.17.66

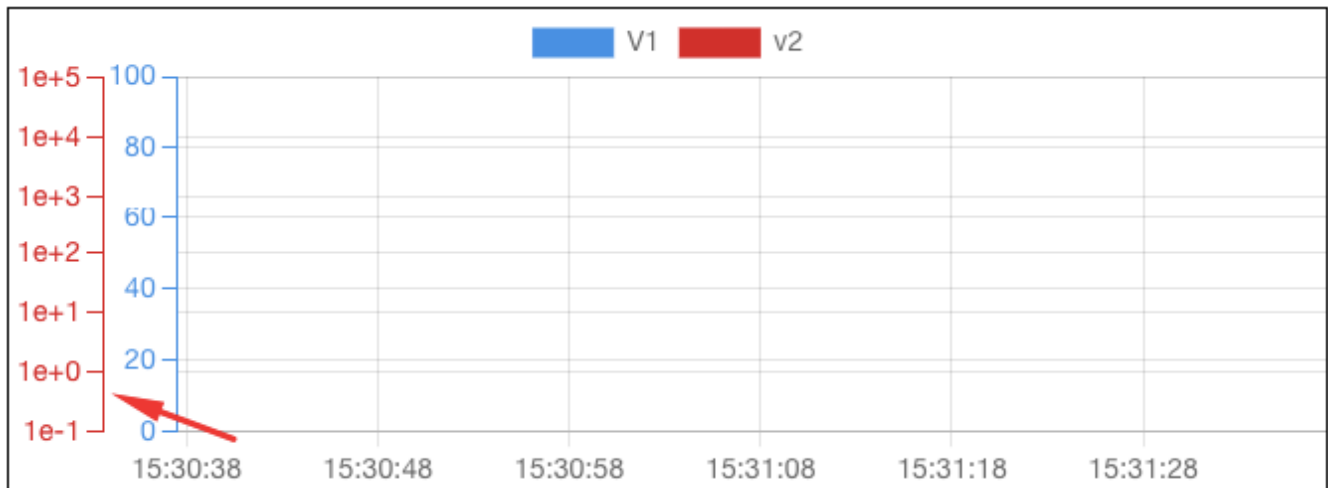
- Добавлена возможность отображения нескольких шкал по оси Y
- Добавлена логарифмическая шкала для графика
- Добавлена Е-нотация для графика



Логарифмическая шкала:



Логарифмическая шкала с Е-нотацией:



## Столбчатый

Столбчатый график работает с виджетом [Chart Columns](#). Выводятся группы столбцов по временной оси. Обычно имеет смысл при использовании агрегирования.

В нижней таблице определяются Значения из списка свойств устройств, которые пишутся в БД. Для каждого значения нужно задать цвет и легенду.

Обработка данных, ось Y и период настраиваются аналогично линейному. Можно также настроить **Вид столбцов**

- стековый (значения накладываются вертикально)
- обычный (значения показываются рядом)

## Круговой

Круговой график работает с виджетом [Chart Pie](#).

В нижней таблице также, как и для столбчатого графика, определяются Значения из списка свойств устройств, которые пишутся в БД. Для каждого значения, который будет представлять сегмент кругового графика, нужно задать цвет.

Здесь уже нет временной оси. Данные агрегируются в соответствии с заданной функцией обработки. Результаты агрегирования можно **Вывести в процентах**

## Обработка данных

Если данные идут без обработки, будут выведены все точки, которые хранятся в БД за выбранный период.

Но можно агрегировать значения, и выводить, например, среднее значение за период.

При выборе способа **Агрегировать по времени** нужно выбрать:

- Временной интервал (минута, час, день, месяц, год)
- Функцию обработки
  - Минимум
  - Максимум
  - Среднее
  - Первое значение
  - Последнее значение



- Сумма
- Количество значений
- Расход

## Использование обработчика

Для любого вида графика можно использовать обработчик, то есть написать скрипт, который вернет нужные данные.

В этом случае большинство настроек скрывается. Скрипт должен вернуть данные в виде объекта заданного формата.

Подробное описание: [Обработчик для графика](#)

## Использование Переменных клиента

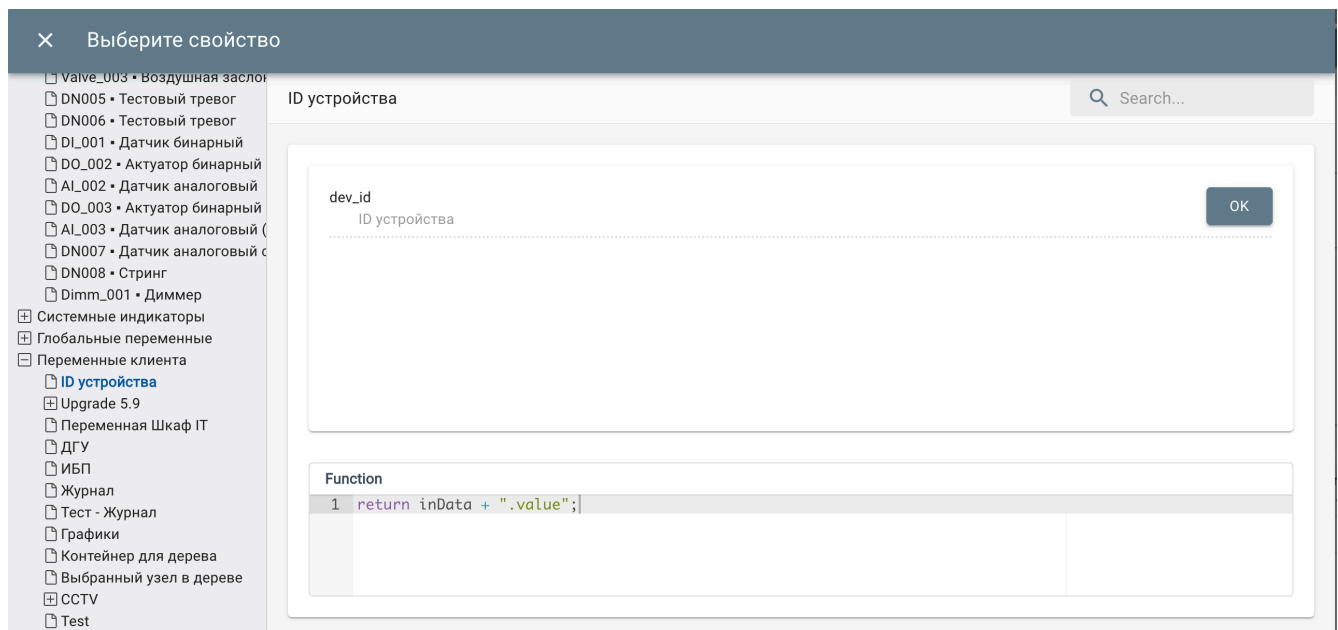
При использовании Переменных клиента можно не задавать жестко устройства, а применить одну сущность **График** для формирования однотипных графиков.

Например, вы хотите создать график, который будет отображать заданное свойство любого устройства, у которого это свойство есть.

Для этого необходимо при привязке перьев выбрать Переменную клиента, которая будет хранить идентификатор устройства (в нашем случае dev\_id) и прописать в формуле свойство, которое необходимо вывести:

```
return inData + ".value"
```

И нажать OK



# Обработчик для графика

При включении чекбокса "Использовать обработчик" большинство настроек скрывается.

Скрипт сам получает данные, определяет оформление и должен вернуть данные в виде объекта заданного формата, который ожидает виджет графика (может отличаться для разных видов графиков).

Дефолтный обработчик содержит пример кода и возвращаемого объекта.

Перед использованием обработчика выберите тип графика - Линейный, Столбчатый, Круговой.  
Дефолтный обработчик будет содержать код для выбранного типа.

## Входные аргументы

```
module.exports = async ({local, filter}, dbclient, api, debug) => {
  ...
}
```

На входе обработчик получает 4 параметра:

- {local, filter} - объект данных с клиента
  - filter = {start, end} - содержит интервал запрашиваемых данных (timestamp) Скрипт может использовать эти данные или переопределить их
  - local = {dev\_id:'d0402', listid:'vs17',...} - локальные переменные, передаются клиентом
- dbclient - объект для получения данных из СУБД
- api - объект для доступа к устройствам и сервисным операциям
- debug - функция для вывода отладочных сообщений в консоль

## Выходные аргументы

Скрипт должен вернуть объект, структура зависит от типа графика.

### Для линейного графика

Обработчик должен вернуть точки графика (points) и параметры оформления для каждого пера, а также интервалы по осям.

```

return {
  // Массив items содержит по одному элементу для каждого пера
  "items": [
    {
      "id": "1",
      "points": points, // Точки графика в формате {x,y}
      "legend": legend, // Строка – легенда
      "lineColor": "rgba(216,38,38,1)",
      "fillMode": "none",
      "fillColor": "rgba(248,231,28,1)",
      "lineWidth": "1",
      "interpolation": "default"
    }
  ],

  // Интервал по оси X (временная шкала) – timestamp
  "start": start, // начало
  "end": end, // конец

  // Интервал по оси Y
  "min": 0,
  "max": 40
}

```

## Параметры оформления

- lineWidth - толщина линии в пикселах
- lineColor - цвет линии в шестнадцатеричном формате или rgba
- fillMode - режим заполнения, варианты:
  - none - нет заполнения
  - first\_up - заполнить вверх до следующей линии
  - first\_down - заполнить вниз до следующей линии
  - all\_up - заполнить вверх полностью
  - all\_down - заполнить вниз полностью
- fillColor - цвет заполнения в шестнадцатеричном формате или rgba
- interpolation - метод интерполяции при отрисовке, варианты:
  - default
  - liner
  - cubic
  - cubic\_monotone
  - step

## Несколько шкал по оси Y

*Добавлено v5.16.12*

Линейный график может иметь несколько шкал по оси Y.

В этом случае нужно добавить **объект scales** с описанием каждой шкалы. Шкалы нумеруются последовательно, описание шкалы содержит:

- `scaleId` - идентификатор шкалы (ключ в строковом виде)
- `tu` - единица измерения
- `min` - минимальное значение, число
- `max` - максимальное значение число
- `scaleColor` - опционально - цвет шкалы в шестнадцатеричном формате или `rgba`
- `scaleType` - тип шкалы (`linear`, `logarithmic`)
- `toExponential` - подпись делений шкалы в виде экспоненты
- `hideGrid` - скрыть сетку шкалы

*Добавлено v5.17.66.* Опциональные параметры для шкалы

- `scaleType` - тип шкалы, по умолчанию `linear`
  - `linear` - линейная
  - `logarithmic` - логарифмическая
- `toExponential` - флаг отображения чисел на шкале в E-нотации
- `stepsize` - шаг шкалы, число

Для каждого пера в массиве **items** нужно указать **scaleId**.



Добавлено v5.17.90.

Добавлена возможность размещать линейные графики с несколькими шкалами друг над другом. На форме настройки графика добавлен чекбокс **Вид шкал - стековый**.

В обработчике этот параметр можно установить, используя флаг **scalestacked**:

```
// Результат
return {
  scales: {
    1: {
      scaleId: "1",
      mu: "°C",
      scaleColor: "",
      min: 0,
      max: 70,
      scaleType: 'linear'
    },
    2: {
      scaleId: "2",
      mu: "mA",
      scaleColor: "",
      min: 0.0001,
      max: 10000,
      scaleType: 'logarithmic'
    }
  },

  items: [
    // ....
  ],

  start,
  end,
  scalestacked: true
};
```

## Для столбчатого графика

Обработчик должен вернуть массив значений {x, y} для каждого столбца и параметры оформления (цвет) каждого столбца, а также интервалы по осям.

```

return {
  // Массив items содержит по одному элементу для каждого столбца
  "items": [
    {
      "points": [{x: start, y:10}, ...{x: end, y:20}],
      "legend": 'Первый',
      "fillColor": '#E38627'
    },
    {
      "points": [{x: start, y:20}, ...{x: end, y:40}],
      "legend": 'Второй',
      "fillColor": '#C13C37'
    },
    {
      "points": [{x: start, y:30}, ...{x: end, y:60}],
      "legend": 'Третий',
      "fillColor": '#6A2135'
    }
  ],
  "start": start,
  "end": end,
  "min": 0,
  "max": 70,
  // "unit":"hour",
  "stacked": 0
}

```

### Опциональные параметры (stacked, unit)

- **stacked** позволяет настроить вид столбцов:
  - 1 - стековый (значения группы накладываются вертикально)
  - 0 - обычный (значения группы показываются рядом)
- **unit** позволяет указать временной интервал между столбцами.  
Варианты - "second", "minute", "hour", "day", "week", "month"

Если интервал указан, при операции zoom столбцы будут только растягиваться по горизонтали без дробления временной шкалы.

### Параметры оформления

- **fillColor** - цвет столбца в шестнадцатеричном формате или rgba

### Для кругового графика

Обработчик должен вернуть массив сегментов и параметры оформления (цвет) каждого сегмента

```
return {
  // Массив items содержит числовое значение value и цвет для каждого сегмента
  "items": [
    { value: 10, color: '#E38627' },
    { value: 15, color: '#C13C37' },
    { value: 20, color: '#6A2135' },
  ]
}
```

## Параметры оформления

- color - цвет сегмента в шестнадцатеричном формате или rgba

## Получение исторических данных

Исторические данные устройств хранятся в СУБД в таблице "records".

Схема для SQLite:

- dn - String - имя устройства
- prop - String - свойство
- ts - Number - timestamp
- val - Number - значение

Схема для других СУБД:

- id - Number - идентификатор метрики (имя устройства + свойства)  
Формируется при создании правила для записи в БД
- ts - Number - timestamp
- val - Number - значение

## Запросы к БД

Доступ к данным обеспечивает dbclient.

- **dbclient.prepareQuery({...})** возвращает SQL запрос (строку) по заданным параметрам
- **await dbclient.query(sql)** возвращает массив данных по запросу в строке sql

Таким образом, при необходимости сложных выборок можно написать SQL запрос в коде, но удобнее воспользоваться готовой функцией для подготовки запроса.

## Подготовка запроса для SQLite

Список "имя устройства.свойство" передаются в строке dn\_prop через запятую



```
const dn_prop = 'Meter1.P1,Meter1.P2';
const sqlStr = dbclient.prepareQuery({dn_prop, start, end });
debug('sqlStr = '+sqlStr);
const arr = await dbclient.query(sqlStr); // => [{dn, prop, ts, val}]
```

## Подготовка запроса для других СУБД

Оптимизированный формат хранения требует получения id по паре {did, prop}, где did - идентификатор устройства, prop - свойство

Это обеспечивает **асинхронная** функция объекта api: **api.getIds([ {did, prop},....])**

Возвращает массив идентификаторов, который передается функции подготовки запроса.

```
const ids = await api.getIds([
  { did: 'd0042', prop: 'P1' },
  { did: 'd0042', prop: 'P2' }
]);

const sqlStr = dbclient.prepareQuery({
  ids,
  start,
  end
});

const arr = await dbclient.query(sqlStr);
// => [{id, ts, val}]
```

## Применение переменных клиента

При использовании обработчика можно не задавать жестко устройства, а работать с Переменными клиента. Таким образом, можно применить одну сущность **График** для формирования графиков однопоточных устройств. Переменные клиента приходят в объекте **local**.

Если нужно получить информацию об устройстве - можно использовать функцию **await api.getDevice(id)**

```
// Динамический выбор – ID устройства приходит в локальной переменной
const dev_id = local.dev_id;
// Из исторической БД получить значения например для этого свойства
const prop = 'dt_out1';

// SQLite нужен dn (device name)
// По ID получаем описание устройств
if (dev_id) {
  const dobj = await api.getDevice(dev_id);
  const dn = dobj.dn;
}
```

Подробнее см [Переменные клиента](#)

## Использование пользовательских таблиц

В IntraSCADA можно создавать [Пользовательские таблицы](#). Схема каждой таблицы создается разработчиком проекта.

Таблицы может заполнять вручную в РМ, а также из скрипта визуализации, сценария или обработчика REST API.

По данным этих таблиц также можно строить графики (причем в качестве x может быть не временная шкала).

Еще одно применение пользовательских таблиц - в них можно записать настройки, которые затем использовать при расчетах или даже хранить параметры оформления .

Для получения данных пользовательской таблицы, используются функции из API:

- **getRecords({table:<>, filter:{..}, opt:{}})** - получение массива данных
- **findOneRecord({table:<>, filter:{..}, opt:{}})** - получение одной записи

В Примере для нескольких перьев мы задавали цвета прямо в коде. Можно создать пользовательскую таблицу, в которой задавать, каким цветом следует выводить каждое свойство. Назовем таблицу 'propcolors', и пусть таблица имеет столбцы:

- prop - имя свойства
- color - цвет

Заполним таблицу для свойств temp1 - temp5.

В коде вместо создания статического массива colorArr загрузим цвета из таблицы

```
// получаем всю таблицу как массив [{prop, color}
const colorRecs = await api.getRecords({table: 'propcolors'});

// Преобразуем в Map, чтобы было удобно искать
const colorMap = new Map( colorRecs.map(item => ([item.prop,item.color]))
const defColor = '#BD10E0'; // дефолтный цвет

// В отличие от примера сформируем массив items в цикле
const items = [];
propArr.forEach((prop, idx) => {
  // Для каждого свойства найдем свой цвет, если нет – применяем дефолтный
  const lineColor = colorMap.get(prop) || defColor;
  items.push({
    id: idx,
    points: pointsArr[idx],
    legend: legendArr[idx],
    lineColor,
    fillMode: "none",
    lineWidth: "1",
    interpolation: "default"
  });
});
return {items, start, end, min: -40, max: 50};
```

## Отладка

При запуске обработчика в консоли Редактора обработчика можно увидеть информацию о запуске, входные данные и результат работы.

В случае ошибки будет выведена диагностика.

При необходимости, можно выводить промежуточные результаты, используя функцию `debug(<Строка | Объект>)`

## Сообщение об ошибке

*Добавлено v5.18.2*

Если не удалось сформировать данные, так же, как в скрипте визуализации, можно передать на клиент сообщение об ошибке:

```
return {err: 'Что-то пошло не так'}
```

В нижней части экрана клиента отобразится всплывающее окно с сообщением об ошибке.

Данные, даже если они тоже отправлены, будут проигнорированы.

## Таймлайны

Для того, чтобы использовать таймлайны, нужно настроить во вкладке [Сохранение значения в БД](#) сохранять для Таймлайн!

Для вывода Таймлайн обычно нужны две составляющие:

- Виджет Chart Timeline, который размещается в контейнере или диалоге;
- Сущность **Таймлайн** в разделе Аналитика -> Таймлайн, который содержит правила вывода диаграммы

Сущность **Таймлайн** привязывается к виджету графика на вкладке **Ссылка**.

Один и тот же **Таймлайн** можно привязать к виджетам на разных экранах (контейнерах, диалогах).

В данном разделе речь идет о создании сущности **Таймлайн** и механизмах привязки к виджету. Подробнее о настройке виджета графика см Визуализация -> Виджеты ...

### Chart Timeline

Это графики в виде диаграммы Ганта. С помощью которых можно показать очередность появления событий.

В нижней таблице настраиваются цвета горизонтальных полос при определенных значениях свойств, выбранных устройств. В первой таблице необходимо заполнить:

- Название цветовой группы.
- Выбрать свойство устройства
- Легенда

Во второй таблице необходимо заполнить:

- Название цветовой группы.
- Значение
- Цвет из палитры при этом значении

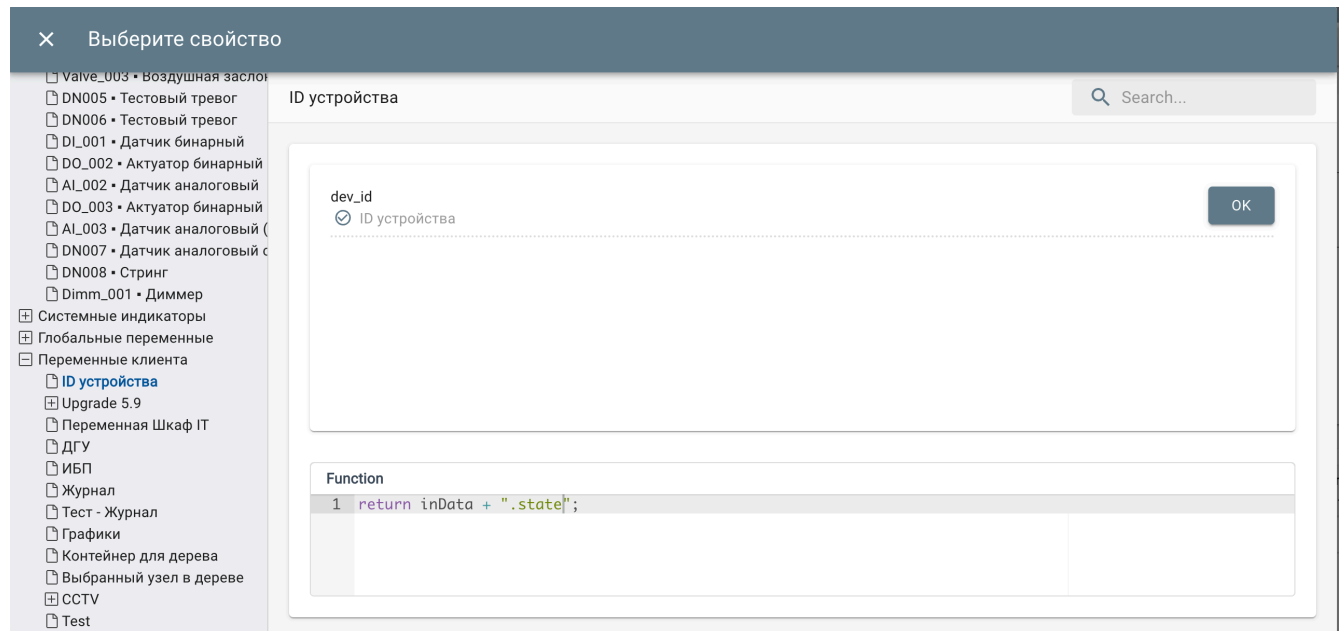
Можно для каждой горизонтальной полосы настроить сразу несколько цветов в зависимости от Значения с устройства. Например когда состояние насоса в норме, то полоса будет зеленым цветом, а когда авария то красным.

## Применение Переменных клиента

При использовании Переменных клиента можно не задавать жестко устройства, а работать через Переменными клиента. Таким образом, можно применить одну сущность **Таймлайн** для формирования графиков однотипных устройств. Например вы хотите создать график, который будет отображать состояние для разных устройств. Для этого необходимо при привязке перьев выбрать Переменную клиента и прописать в формуле следующий текст в зависимости от свойства, которое необходимо вывести:

```
return inData + ".state"
```

И нажать ОК



# Журналы

The screenshot shows the 'Новый журнал' (New Log) form in the IntraSCADA application. The form is divided into several sections:

- Left Sidebar:** Contains navigation icons for various system components.
- Main Form:**
  - ID:** jr003
  - Название:** Новый журнал
  - Тип журнала:** Главный журнал (dropdown)
  - Формат вывода даты (времени):** ДД.ММ.ГГ ЧЧ:ММ (dropdown)
  - Форма для вывода в pdf:** Отчет по главному журналу (dropdown)
  - Фильтры для включения сообщения в журнал:**
    - Метки: (text input)
    - Локация: Все (dropdown)
    - Устройство: (dropdown)
  - Уровень и источник сообщения:**
    - Источник сообщения: Устройства (dropdown)
    - Выводить сообщения: Все (dropdown)
- Table at the bottom:**

№ пл	Данные столбца	Шапка столбца	Ширина столбца
1	Дата, время	Date	250
2	Сообщение	MessageText	400

В системе есть два вида журналов - **исторические журналы** и [Журналы тревог](#)

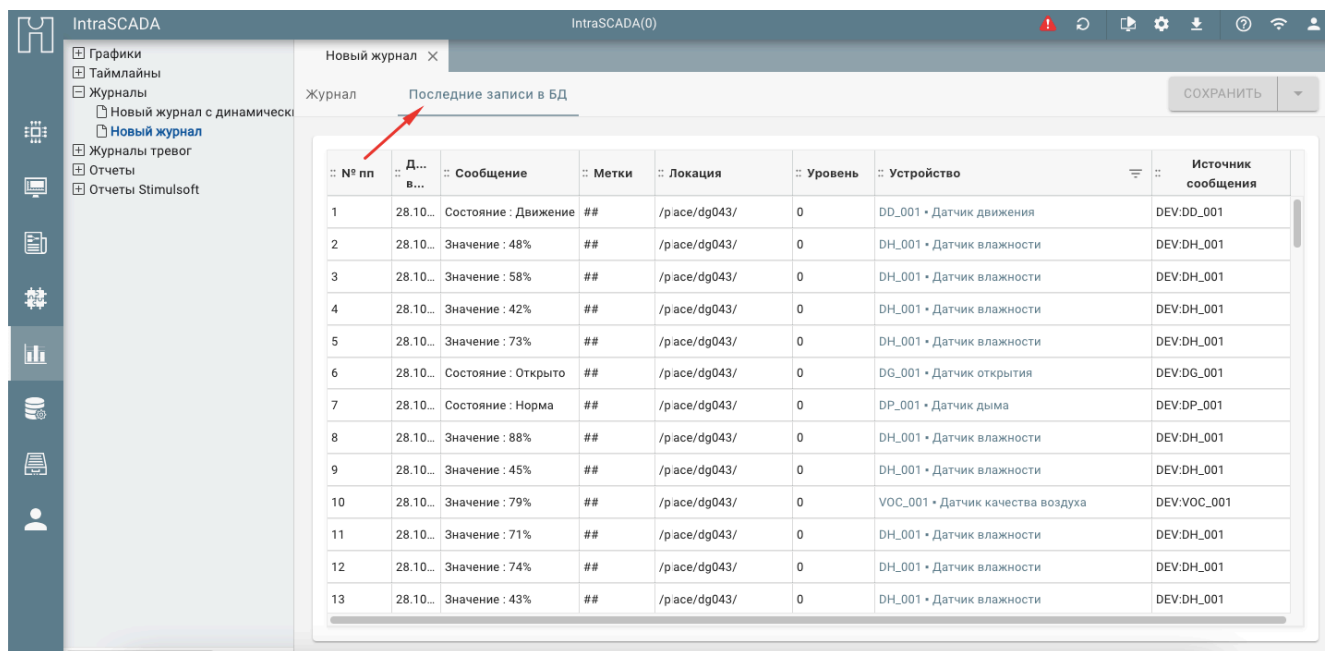
Данные **исторических журналов** хранятся в СУБД и предоставляют возможность для ретроспективного просмотра и анализа данных:

- Таблица БД **mainlog** содержит записи **Главного журнала**
- Таблица БД **pluginlog** предназначена для записей **Журнала плагинов**

Удаление записей из этих таблиц производится исходя из сроков хранения сообщений, которые настраиваются для каждого уровня отдельно.

В данном разделе описывается, как создать сущности **Журнал** с различными фильтрами и наборами столбцов на базе **Главного журнала** и **Журнала плагинов**. На вкладке **Последние записи в БД** выводятся последние 100 сообщений в журнале с учетом установленного фильтра.

Набор столбцов применяется только при выводе журнала через виджет, на вкладке **Последние записи в БД** выводятся все столбцы из БД без обработки.



Для вывода на визуализацию **Журнал** нужно привязать к виджету **Journal**.

## Отличие Главного журнала от Журнала тревог

**Журнал тревог** - это оперативный журнал, который хранит только активные записи аварийно-предупредительной сигнализации. После отработки события (квитирования и/или нормализации) запись из журнала тревог удаляется.

Если нет активных событий, Журнал тревог будет пуст.

Подробно настройка тревог описана в разделе Тревоги.

Параллельно все события механизма тревог (срабатывание, нормализация, эскалация, квитирование) фиксируются в **Главном журнале** в хронологическом порядке.

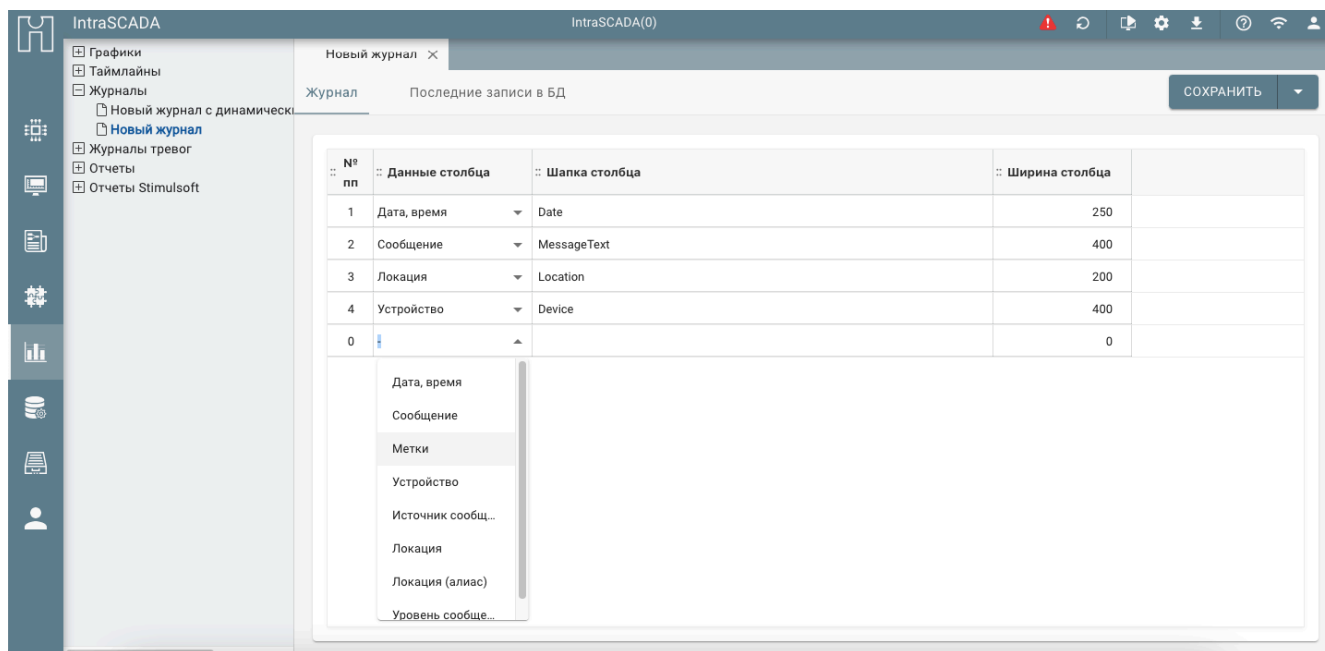
## Главный журнал

Какие данные попадают в **Главный журнал**:

- Все события механизма тревог
- Изменение свойств устройств или запуск команд, если в экземпляре устройства установлена галочка **Сохранять в главный журнал**
- Сообщения из сценариев (функция **this.mainlog**)
- Сообщения из скриптов визуализации (функция **core.mainlog**)
- Сообщения из обработчиков устройств (функция **device.mainlog**)

Какие столбцы доступны:

- Дата и время
- Сообщение
- Метки
- Устройство
- Источник сообщения (Отправитель)
- Локация
- Локация (алиас)
- Уровень сообщения



Какие есть фильтры:

- Фильтр по Меткам (Tags)
- Фильтр по Локации - включает устройства в дереве внутри выбранной папки
- Фильтр по Устройству - включает информацию только по данному устройству Этот фильтр имеет приоритет, фильтры по меткам и локации в данном случае не учитываются
- Фильтр по Уровню сообщений. Все / Предупреждение, Авария / Авария
- Фильтр по Источнику сообщений

Можно фильтр настроить статически, то есть выбрать конкретные значения для фильтра из списков.

## Динамические фильтры с применением Переменных клиента

В **v5.18.34** добавлена возможность использовать динамические фильтры с применением Переменных клиента. Это более сложный механизм, но он позволяет:

- Использовать один и тот же Журнал для вывода на разные мнемосхемы с переключением фильтров
- Использовать в фильтре несколько значений, перечисленных через запятую

Это позволяет задавать фильтрацию динамически — в зависимости от состояния интерфейса, выбранного объекта или действий пользователя на мнемосхеме.



Фильтр	Поддерживаемое значение	Пример переменной клиента	Пример значения
<b>Метки (Tags)</b>	Список меток	tags	Климат, Вентиляция
<b>Локация (ID или алиас папки)</b>	Идентификатор или алиас	loc_id	Area_1
<b>ID устройства</b>	Один или несколько идентификаторов устройств	dev_id	d001 или d002, d003, ...

#### Принцип работы

В настройках *Журнала с динамическими фильтрами* на вкладке *Фильтры с использованием переменных клиента* в нужном поле указывается имя переменной клиента.

Примеры:

- dev\_id
- loc\_id
- tags

При загрузке мнемосхемы фильтр применяет текущее значение переменной. Если в переменной указано несколько значений через запятую, фильтрация выполняется по каждому из них.

При изменении значения переменной журнал автоматически обновляется, отображая только записи, соответствующие новому фильтру.

#### Примеры сценариев использования

Сценарий	Переменная клиента	Значение	Результат
Просмотр событий конкретного устройства	dev_id	d001	Отображаются записи по устройству <i>d001</i> .
Просмотр группы устройств	dev_id	d001, d212, d321	Отображаются события для всех указанных устройств.
Просмотр по локации	loc_id	Area_1	В журнал включаются устройства из папки <i>Area_1</i> .
Отбор по меткам	tags	Климат, Вентиляция	Будут показаны записи по тегам.

## Журнал плагинов

Какие данные попадают в **Журнал плагинов**:

- Изменение состояния плагинов - запуск, останов, выход с ошибкой.
- Сообщения, отправленные плагином ( **plugin.sendLog** )

Какие есть фильтры:

- Фильтр по экземплярам плагинов.
- Фильтр по уровню сообщений. Все / Предупреждение, Авария / Авария

## Журналы тревог

Журналы тревог - это оперативные журналы, содержащие информацию о текущих тревогах.

Журнал тревог имеет механизм квитирования, позволяющий квитировать отдельные тревоги.

В данном разделе вы можете настроить формирование журналов тревог для вашей системы. Используя фильтры, можно создать журналы по отдельному устройству, объекту, группе устройств.

Если все фильтры сброшены, журнал будет отображать все тревоги проекта.

Также для конкретного журнала можно настроить отображение столбцов и их размер.

Доступны столбцы:

- Время начала
- Время завершения
- Состояние
- Сообщение
- Кнопка подтверждения
- Время квитирования
- Оператор
- ID Оператора
- Устройство
- Метки
- Локация
- Уровень сообщения

Во вкладке **Текущие записи** выводятся все сообщения этого журнала тревог в зависимости от установленного фильтра. Какие есть фильтры для журналов:

- Фильтр по Меткам. Например климат, безопасность, освещение.
- Фильтр по Локации - размещению устройств в дереве. Можно выбрать узел
- Фильтр по Устройству. Выводить информацию только по данному устройству. Удобно если устройство сложное, например как система вентиляции

# Формирование и показ отчета

Как правило, чтобы использовать данные в отчете, нужно настроить сохранение значений в БД!

Для отчета, как и для графика, нужны две составляющие:

- Виджет **Report**, который размещается в контейнере или на экране.  
В оперативном режиме пользователь сможет выбрать отчет из списка, задать период, просмотреть результат на экране, при необходимости напечатать и/или выгрузить в виде файла (pdf, csv).  
В список отчетов виджета можно включить только те отчеты, которые должны быть доступны на конкретном экране.
- Сущности **Отчет** в разделе Аналитика -> Отчеты содержат правила формирования для каждого отчета.

## Виджет Report

Виджет имеет встроенные элементы для выбора периода и droplist со списком отчетов.

Настройка списка отчетов выполняется на вкладке **Привязка**. Можно добавлять отчеты в список, добавляя строки и выполняя операцию Привязать.

Есть возможность упорядочить этот список, используя операции **Вверх** и **Вниз**.

Отчеты формируются в формате pdf и отображаются для просмотра в окне виджета. Для Печати и Выгрузки отчета на виджете имеются соответствующие кнопки.

## Сущность Отчет

Содержит правила формирования отчета:

- на вкладке **Описание** настраиваются правила получения данных для подготовки отчета.
- на вкладке **Редактор** графически готовится шаблон отчета для формирования pdf. Здесь, кроме основной таблицы отчета, можно разместить изображения, текстовые и графические элементы.
- на вкладке **Обработчик** можно написать скрипт, если выбран вариант **Обработка данных** -> **Пользовательский обработчик**  
В противном случае редактор скрипта недоступен.

## Вкладка Описание

- **Название отчета** - это название, которое будет видеть пользователь в списке доступных отчетов.  
В самом отчете это название не используется
- **Источник данных**  
Источником данных могут быть **Исторические данные устройств** или **Пользовательские таблицы**. При использовании Обработчика можно взять для отчета текущие данные устройств системы напрямую без обращения к БД.
- **Обработка данных**

- **Свертка по временным интервалам.** Обычно при формировании отчета используется этот вариант. При выборе опции становятся доступны поля для выбора **Временного интервала** (минута, час, день, месяц, год) Можно применить **Множитель**, например, агрегировать данные за каждые 15 минут, или 6 часов, или 3 месяца. Система построит запрос к БД на основании используемых в отчете значений и периода, который выбрал пользователь.
- **Пользовательский обработчик.** Этот вариант позволяет строить произвольные отчеты. Запросы к БД делаются разработчиком из скрипта. Шаблонный скрипт содержит варианты, которые можно использовать.  
 Подробное описание: [Обработчик для отчета](#)
- **Таблица переменных отчета.** Собственно, из этих переменных будут формироваться столбцы нашего будущего отчета. Подробное описание: [Переменные отчета](#)

## Вкладка Редактор

Графический редактор позволяет создать шаблон отчета, содержащий:

- Прimitives - Изображение, Прямоугольник, Круг;
- Текстовые элементы, которые могут содержать статический текст и динамические подстановки  $\${...}$ .
- Таблицу отчета - задает правила формирования элементов таблицы

## Динамические подстановки для текстовых элементов

Для вывода динамического значения используется синтаксис  $\${имя\_переменной}$ .

Кроме переменных отчета, существуют predefined переменные:

- $\${now}$ 
  - будет выведена строка с текущим временем
- $\${period}$ 
  - будет выведена строка периода отчета.  
 Например: **01.05.2023 - 12.06.2023**.  
  
 Если период определяет полный месяц, вместо дат будет выведено название месяца.  
 Например, вместо **01.05.2023 - 31.05.2023** будет выведено **май 2023**.  
  
 Аналогично, для полного года выводится только год.  
 Например, вместо **01.01.2023 - 31.12.2023** будет выведено **2023**.
- $\${period_datetime}$  или  $\${period_dt}$ 
  - будет выведена строка периода отчета в формате дата-время.  
 Например: **01.05.2023 00:00 - 12.06.2023 23:59**. В этом случае замена формата для полного месяца/года не выполняется.

## Переменные отчета

Чтобы использовать данные на вкладке **Редактор** (вторая вкладка), нужно создать переменные в **Таблице переменных отчета** на первой вкладке. Из этих переменных будет формироваться наш будущий отчет.

### Имя переменной

Каждая переменная должна иметь имя (латинские буквы, цифры, символ '\_').  
Имя обязательно должно быть, выбирать его можно совершенно произвольно.

### Источник значения

Источник тоже нужно выбрать. Это может быть:

- номер строки по порядку;
- дата/время в заданном формате (при свертке по времени можно вывести Нач дату, Кон дату или Период в целом)
- значение из БД (в списке будут все свойства устройств, сохраняемых в БД);
- расчетное значение по строке;
- текущее значение свойства конкретного устройства.

### Дополнительные параметры

После ввода имени и выбора источника значения из списка будут доступны те или иные дополнительные столбцы:

### Обработка данных из БД

Если выбрана опция свертки, нужно задать функцию **Обработки данных из БД** внутри заданного для свертки временного интервала.

Доступные функции:

- sum: Сумма
- min:
- max: Max
- avg: Среднее (average)
- first: Первое значение
- last: Последнее значение
- count: Количество значений
- diff: Расход (разность значений на конец и начало интервала)

### Формула расчета

Для расчетного значения по строке можно заполнить **Формулу расчета**. Это может быть любое выражение, использующее имена переменных отчета, например:

```
(meter1_val+ meter2_val+meter1_val)/1000
```

## Свойство устройства

Для текущего значения свойство выбирается из стандартного диалога для выбора свойств устройств. Здесь доступны все свойства всех устройств, в том числе статические, например, названия устройств.

Пример создания переменных для отчета посуточного расхода э/энергии по трем счетчикам :

Имя переменной	Источник значения	Обработка данных из БД
ppp	Номер строки	
day	Нач дата	
meter1_cons	METER1.value	Расход
meter2_cons	METER2.value	Расход
meter3_cons	METER3.value	Расход
day_cons	Расчет по строке	meter1_cons+meter2_cons+meter3_cons

## Использование переменных

### Столбцы таблицы

Все введенные переменные доступны в списках при настройке столбцов таблицы. То есть каждая переменная - это потенциальный столбец.

### Формулы расчета

Не обязательно создавать переменные только для столбцов. Переменные можно использовать в формулах расчета других переменных.

### Текстовые поля

Можно вывести значения переменных в текстовые поля и даже в шапки таблицы, используя "подстановку": `${myname}`

## Таблица отчета

После добавления в **Редакторе** элемента **Таблица** нужно настроить столбцы.

На первой вкладке добавляются столбцы, они будут сразу отображаться в редактируемом элементе.

После добавления столбца нужно задать текстовую шапку и выбрать **Переменную отчета**.

Следующие три вкладки позволяют в разрезе каждого столбца сформатировать разделы таблицы:

- Шапка (header)
- Тело (body)
- Итоги (footer)

Ширина столбца определяется атрибутом **Ширина** на вкладке **Шапка**.

Остальные атрибуты ячейки таблицы: выравнивание, размер и цвет текста, цвет фона, цвет и толщина линии настраиваются отдельно для каждого раздела.

**Тело** таблицы будет заполнено данными переменных отчета соответствующего столбца.

В разделе **Итоги** можно вывести тексты или итоговые значения по столбцу.

Для этого в атрибут Содержимое соответствующего столбца нужно прописать текст или функцию расчета:

```
// Содержимое (текст или формула)
Итого: – текст
 $\{sum\}$  – будет выведена сумма по столбцу
 $\{avg\}$  – будет выведено среднее значение по столбцу
```

Доступные функции расчета итогов по столбцу:

- sum: Сумма
- min: Минимальное значение
- max: Максимальное значение
- avg: Среднее (average)
- first: Первое значение
- last: Последнее значение
- count: Количество значений



# Обработчик для отчета

Если данные таблицы отчета не получается сформировать стандартным способом, можно выбрать опцию "Пользовательский обработчик".

Скрипт должен вернуть массив переменных отчета для каждой строки таблицы (кроме шапки и итоговой строки).

Дефолтный обработчик содержит пример кода и возвращаемого объекта.

## Входные аргументы

```
module.exports = async (reportVars, devices, dbclient, filter, api)
  ...
}
```

На входе обработчик получает 5 параметров:

- reportVars - массив переменных отчета
- devices - массив устройств
- filter - параметры отчета из текущего запроса: { start, end, local, context }
- dbclient - объект для получения данных из БД
- api - объект, предоставляющий сервисные функции

## reportVars: массив переменных отчета

Это переменные из [Таблицы переменных отчета](#) на первой вкладке.

Передаются как массив объектов: {varname:<имя переменной>, col\_type:<источник значения>,...}

Например:

```
[{ varname: 'meter_number', col_type: "constant", "value": "1234567" },..]
```

В обработчике можно создать новые переменные, которые можно использовать для подстановки в текстовые поля или шапку таблицы

```
// Добавим новую переменную для вывода в отчете в виде ${mytext}
reportVars.push({
  varname: 'mytext',
  col_type: 'constant',
  value: 'Этот текст из обработчика'
});
```

devices: массив устройств.

Используется опционально, если нужно передать скрипту список устройств с текущими значениями свойств. Для каждого устройства массив содержит текущие значения всех статических и динамических свойств. Например:

```
{
  // Статические свойства
  _id: 'd0356',
  dn: 'METER01',
  name: 'Счетчик 1',
  type: 't032',
  tags: '#Ресурсы#',
  placeStr: 'Цех 1',

  // Динамические свойства
  snumber: '123456',
  value: 11781,
  uptoHour: 11777,
  U1: 220,
  U2: 221,
  U3: 217
}
```

## filter: параметры текущего запроса

Содержат интервал, локальные переменные и контекст запрашивающего клиента.

- start, end - интервал запрашиваемых данных (timestamp)
- local = {dev\_id: 'd0402', listid: 'vs17', ...} - локальные переменные  
Может применяться для параметризации запроса (например, можно разместить на экране с отчетом список объектов, при выборе объекта формировать отчет по этому объекту)
- context - объект, содержащий данные о пользователе
  - username: Имя пользователя
  - layoutid: ID текущего экрана
  - start\_layoutid: ID стартового экрана пользователя

## dbclient: объект для получения данных из БД

- **dbclient.prepareQuery({...})** возвращает SQL запрос (строку) по заданным параметрам
- **await dbclient.query(sql)** возвращает массив данных по запросу в строке sql

Схема данных и выполнение запросов аналогичны тем, что выполняются в [Обработчике для графика](#)

## Выходные аргументы

### Данные для таблицы в виде массива

Скрипт должен вернуть массив, содержащий объект с переменными для каждой строки таблицы отчета (в объекте должны быть все переменные, которые используются в столбцах таблицы).

Пример 1. Вывести текущие показания счетчиков, перечисленные в списке devices.

Счетчики имеют свойства:

- snumber - серийный номер счетчика
- value - текущее показание

Таблица имеет столбцы:

- Номер по порядку (переменная npp)
- Номер счетчика (переменная meter\_number)
- Текущее показание (переменная meter\_val)
- Место размещения (переменная meter\_place)

```
module.exports = async (reportVars, devices, dbclient, filter, api) => {
  // Вернуть текущие данные не обращаясь к БД
  const data = devices.map((dev, idx) => ({
    npp: idx + 1,
    meter_number: dev.snumber,
    meter_val: dev.value,
    meter_place: dev.placeStr
  }));

  return data;
  // => [{npp:1, meter_number:'12345', meter_val: 7788, meter_place:'Цех 1'}];
};
```

Если в графическом шаблоне отчета используется несколько таблиц, содержимое массива будет применяться ко всем таблицам.

## Данные для нескольких таблиц

Если требуется отобразить в отчете несколько таблиц с разными данными, обработчик должен вернуть объект, содержащий массивы данных с привязкой к каждой таблице.

```

module.exports = async (reportVars, devices, dbclient, filter, api) => {
  const { start, end, local, context } = filter;
  const startDate = new Date(start);
  const endDate = new Date(end);

  let dt = startDate;
  const data = [];
  while (dt <= endDate) {
    // ....вычисляем x1, x2, x3
    data1.push({ date: api.getDateTimeStr(dt, 'reportd'), x1, x2, x3 });
    dt.setDate(dt.getDate() + 1);
  }

  const data2 = [{npp:'1', txt:'Строка 1'}, {npp:'2', txt:'Строка 2'}];

  return {table_1:data1, table_2:data2};
};

```

В массив переменных отчета нужно включить все используемые переменные (dt, x1,x2,x3, npp, txt), столбцы таблиц table\_1, table\_2 привязать к нужным переменным.

## Форматирование ячеек таблицы

Форматирование ячеек таблицы (цвет, размер, стиль и т д) выполняется в Редакторе отчета.

Можно выполнить настройку Шапки, Тела и Итогов таблицы.

Настройка форматирования Тела таблицы выполняется для каждого столбца и применяется ко всем строкам таблицы.

Если есть необходимость изменения отдельных ячеек таблицы в зависимости от содержимого (например, вывести отрицательные значения красным цветом), это можно сделать из обработчика.

Для этого в объекте строки для переменной нужно вернуть не значение, а объект.

Объект должен содержать свойство value:<значение> и форматирующие свойства, отличающиеся для конкретной ячейки.

При форматировании ячейки возможно изменение следующих свойств:

- halign - выравнивание текста по горизонтали ('left', 'center', 'right')
- fontStyle - стиль текста: 'bold', 'italic', 'normal'
- fontSize - размер шрифта (число)
- textColor - цвет текста
- fillColor - цвет фона

Для цветов ожидается синтаксис rgba.

Можно также использовать наименование цвета из списка predefined colors: 'red', 'green', 'blue', 'yellow', 'white',...

В Приложении дан полный [Список цветов](#)

```
const data = [];  
  
data.push({  
  F1: 2,  
  F2: {  
    value: -2,  
    // значение F2 будет выведено красным  
    textColor: 'rgba(255,0,0,1)'  
  }  
});  
  
data.push({  
  F1: 2,  
  F2: {  
    value: -20,  
    textColor: 'red',  
    // значение F2 будет выведено красным и жирным  
    fontStyle: 'bold'  
  }  
});
```

# SQLite

База данных SQLite поставляется в комплекте с системой.

Можно настроить следующие параметры:

- Лимит в Мб - максимальный размер базы данных  
Этот размер зависит от количества свободного места на HDD
- Хранить БД отдельно от проекта  
Если галка не установлена, база данных хранится в папке проекта.
- Путь к папке для БД - установить путь к папке с базой данных
- Уровень логирования - Низкий/Средний/Высокий  
В целях отладки можно установить высокий уровень логирования. В дальнейшем, в целях экономии места, рекомендуется устанавливать низкий уровень логирования

## Контроль за состоянием базы данных

Все параметры базы данных доступны для просмотра в разделе **Системные индикаторы**:

The screenshot shows the IntraSCADA interface. On the left, a sidebar contains a tree view with the following items: **Устройства**, **Системные индикаторы** (highlighted with a red arrow), **Основной процесс**, **DB агент (логи)**, **DB агент (историческая БД)** (highlighted with a red arrow), **Плагин r2p**, **Плагин pushnotification**, **Плагин webconsole**, **Плагин reportmaker**, **MEGAD**, **SNMP**, **EMULS**, **Глобальные переменные**, **Локальные переменные**, and **Типы устройств**. The main panel displays the 'DB агент (историческая БД)' window. It has tabs: **Системный индикатор**, **Свойства** (highlighted with a red arrow), **Журнал**, **Использование**, **Сохранение в БД**, **Последние**, and a **СОХРАНИТЬ** button. The 'Свойства' tab shows a table of system indicators.

:: Свойство устройства	:: Название	:: Значение	:: Дата, время	:: Изменено	:: Ошибка	:: Сохранять в журн устройства
state	State	1	10.11.2021 15:32:44	10.11.2021 15:32:44		<input type="checkbox"/>
version	Version	5.0.18				<input type="checkbox"/>
memrss	Memory RSS	31600	10.11.2021 15:32:44	10.11.2021 15:32:44		<input type="checkbox"/>
memheap	Memory Heap Total	4812	10.11.2021 15:32:44	10.11.2021 15:32:44		<input type="checkbox"/>
memhuse	Memory Heap Used	3185	10.11.2021 15:32:44	10.11.2021 15:32:44		<input type="checkbox"/>
size	DB size Mb	0.66	10.11.2021 15:28:52			<input type="checkbox"/>
overflow	Overflow	0	03.09.2021 08:59:48			<input type="checkbox"/>
lastMaxTimeWrite	Last Max Time Write	31	10.11.2021 15:32:46	10.11.2021 15:32:46		<input type="checkbox"/>
lastMaxCountWrite	Last Max Count Write	1	10.11.2021 15:32:46			<input type="checkbox"/>
lastMaxTimeRead	Last Max Time Read	780	10.11.2021 15:21:25			<input type="checkbox"/>
lastMaxCountRead	Last Max Count Read	16556	10.11.2021 15:21:25			<input type="checkbox"/>
error	error	0				

# Пользовательские таблицы

## Для чего используются

При разработке проекта часто возникает необходимость создания таблиц прикладного уровня. Это могут быть различные справочники, рецепты, результаты работы процессов, агрегированные данные и т.д. Состав и структура этих таблиц может меняться на лету в процессе разработки и дальнейшей эксплуатации системы.

Использование для таких целей классических таблиц в рамках реляционных СУБД обычно трудоемко, поскольку требуется операция миграции при изменении структур таблиц.

IntraSCADA предоставляет возможность использовать для таких целей NoSQL подход. Данные хранятся в формате JSON, что позволяет гибко подойти к хранению таких данных.

## Варианты хранения

Существует два варианта хранения:

1. Хранение таблиц в виде файлов проекта, обработка встроенным NoSQL движком.

Такие таблицы хранятся на диске, но полностью кэшируются в ОЗУ, это имеет свои плюсы и минусы:

- о плюс - скорость как чтения, так и записи очень большая
- о минус - при больших размерах таблиц использование ОЗУ становится непродуктивным

Рекомендуется использовать для часто читаемых данных, имеющих ограниченный размер (до 500 Мб). Поскольку эти таблицы являются частью проекта, их удобно применять для хранения справочников, рецептов, настроечных параметров.

2. Хранение в специальной таблице в рамках используемой СУБД.

Используемые нами движки (SQLite, PostgreSQL) позволяют хранить записи в JSON формате.

Соответствующие db-агенты, подключаемые к системе, реализуют прозрачный механизм работы с такими данными.

Можно хранить любую информацию, ограничением является только объем дискового пространства.

## Создание таблицы

Новая таблица создается в разделе База данных -> Пользовательские таблицы. Если это удобно, можно группировать таблицы по папкам.

При создании таблицы выбирается один из вариантов, описанных выше:

- Хранилище: СУБД
- Хранилище: Файл проекта

Дальнейшая работа с таблицей унифицирована и не зависит от способа хранения.

## Настройка таблицы

Выполняется на вкладке **Настройка**:

- Название таблицы - это краткий текст, который появится в дереве.  
Например: *Список снимков с камер*
- Имя таблицы - это идентификатор таблицы, который используется процедурами хранилища.  
Может содержать только латинские буквы, цифры и символ '\_'.  
Например: *snapshots* или *camera\_snaps* или *snapCameras*.

По умолчанию система дает техническое уникальное имя таблице.

Измените его, но имейте в виду, что имя должно быть уникальным в рамках проекта.

Существует набор имен, которые зарезервированы для встроенных таблиц, например, *devices*, *users*, *layouts*,... При выборе такого имени система сообщит, что имя зарезервировано.

Это же имя таблицы будет использоваться в функциях скриптов и сценариев для работы с таблицами.

При необходимости вы можете переименовать уже существующую таблицу с данными.

Если есть скрипты с функциями доступа к таблице, имя нужно будет поменять и там!

- Поля таблицы создаются в нижней табличке. Имя поля также должно быть простым идентификатором (латинские буквы, цифры и символ '\_').  
Существуют только 2 недопустимых (зарезервированных) имени поля: *id* и *\_id*.

## Заполнение данными

На вкладке **Данные** можно просмотреть, добавить, отредактировать или удалить записи таблицы. Таблицу можно сортировать и фильтровать по столбцам.

## Использование таблиц

### Работа с таблицами из скриптов системы

Пользовательские таблицы можно использовать в скриптах визуализации, сценариях и функциях REST API. Из скрипта можно не только получать данные, но и добавлять, редактировать и удалять записи.

[API пользовательских таблиц](#)

### Заполнение виджета **Таблица** данными пользовательской таблицы

Виджет **Таблица** связывается с сущностью **Таблица** из дерева ресурсов: Ресурсы -> Таблицы. Здесь создаются правила формирования столбцов и строк таблицы. Если использовать вариант *Заполнить из пользовательской таблицы*, таблица будет заполняться данными пользовательской таблицы.

Если нужно выбирать только часть данных, можно прописать **фильтр** в JSON формате.

Простейший формат {"<имя свойства1>": <значение1>}

Возможны более сложные фильтры с использованием условий неравенства, включения и регулярных выражений.

Подробнее в разделе [Фильтрация данных](#)



Также можно прописать условия сортировки в JSON формате.

# Работа с таблицами из скриптов системы

Пользовательские таблицы можно использовать в скриптах визуализации и сценариях.

Можно не только получать данные, но и добавлять, редактировать и удалять записи.

[API доступа к пользовательским таблицам](#) включает следующие функции:

- **getRecords** - чтение записей
- **findOneRecord** - поиск и чтение одной записи
- **insertRecords** - добавление новых записей
- **updateRecords** - изменение записей
- **removeRecords** - удаление записей

Работа с пользовательской таблицей унифицирована и не зависит от способа хранения  
(Хранилище:СУБД или Хранилище:Файл проекта)

## Вызов функции и обработка ошибок

Основной аргумент функции не зависит от того, вызывается функция из скрипта визуализации или сценария.

Например, чтобы считать все записи таблицы 'mytable', можно вызвать `getRecords` с единственным входным аргументом: `getRecords('mytable')`. В результате получим массив записей.

Отличие заключается в способе вызова и механизме обработки ошибок.

### В скрипте визуализации

В скрипте визуализации функция вызывается как асинхронная функция объекта `core`. Результат - массив записей таблицы. Потенциально может произойти ошибка, если, например, такой таблицы нет.

В случае ошибки генерируется исключение, которое можно обработать и известить пользователя, что что-то пошло не так (добавить подробностей на вкус разработчика):

```
try {
  const data = await core.getRecords('mytable');
  debug(data);
} catch (error) {
  debug('Ошибка при чтении таблицы: ' + error.message);
  return {err: 'Возникла ошибка 42! Но мы уже работаем над этим.'}
}
```

Ошибка для диагностики содержится в `error.message`. Ее можно вывести в консоль отладчика.

Возможна другая стратегия - исключения не обрабатывать. В этом случае их обработает система.

```
const data = await core.getRecords('mytable');
// дальше что-то делается...
```

При ошибке скрипт будет завершен, а пользователю отправлено сообщение **Ошибка скрипта!**  
В консоли будет выведено диагностическое сообщение об ошибке.

## В сценарии

В сценарии функция вызывается как метод сценария, а результат передается функции обратного вызова.

Функции обратного вызова для работы с таблицами имеют один или два аргумента:

- первый аргумент - это строка ошибки. Если она пуста, значит, операция выполнена успешно
- второй аргумент - это результат выполнения операции

```
start() {
  this.getRecords('mytable', 'onGetRecords');
},

onGetRecords(error, payload) {
  if (!error) {
    this.log('Получен массив данных: '+JSON.stringify(payload))
  } else {
    this.log('Ошибка при чтении таблицы: '+error)
  }
}
```

В разделе [API доступа к пользовательским таблицам](#) для каждой функции представлены примеры вызова для скрипта визуализации и для сценария.

# API пользовательских таблиц

Пользовательские таблицы можно использовать в скриптах визуализации и сценариях.

В зависимости от места использования, отличается [синтаксис вызова функций](#) и [способ обработки ошибок](#)

Далее для каждой функции представлены примеры вызова для скрипта визуализации и для сценария.

## Функции чтения данных

Возвращают строки таблицы.

Каждая строка - это объект, имеющий свойства, определенные для полей таблицы.

Также каждая строка имеет уникальный идентификатор строки.

Для таблиц СУБД уникальный идентификатор строки это "id", для таблиц проекта это "\_id"

### getRecords

**getRecords** возвращает массив строк таблицы.

Входным аргументом может быть строка или объект:

- **getRecords(<Имя таблицы - строка>)**  
Входной аргумент - строка.  
Возвращает массив всех записей из заданной таблицы.  
Каждый элемент массива содержит все поля, которые сохранялись.
- **getRecords({table:<Имя таблицы - строка>, filter:{..}, opt:{..}})**  
Входной аргумент - объект.  
Возвращает массив записей с учетом фильтра.  
Можно определить список полей, а также задать сортировку результирующего массива.

Свойства объекта:

- **table** - имя таблицы - обязательный аргумент
- **filter** - опциональный объект, задающий фильтрацию данных  
Простейший формат **filter:{<имя свойства1>: <значение1>,...}**  
Возможны более сложные фильтры с использованием условий неравенства, включения и регулярных выражений.  
Подробнее в разделе [Фильтрация данных](#)
- **opt** - опциональный объект, задающий дополнительные условия формирования результата
  - **fields**: список полей (столбцов), которые нужно включить в выборку  
**fields:{<имя свойства1>:1, ...}**  
ключи - имена свойств, значения = 1
  - **sort**: объект, описывающий сортировку  
**sort: {<имя свойства1>:1, ...}**  
ключи - имена свойств, значения =1(по возрастанию)/ -1(по убыванию) свойства

- order: сортировка по возрастанию чисел  
order: '<название столбца>'

## findOneRecord

**findOneRecord** возвращает одну запись в виде объекта или пустую строку, если запись не найдена.

Входной параметр - объект, аналогичный getRecords, за исключением того, что **filter** - обязательный атрибут.

- **findOneRecord**(**{table:<Имя таблицы - строка>, filter:{..}, opt:{..}}**) Свойства объекта:
  - **table** - имя таблицы - обязательный аргумент
  - **filter** - объект, задающий фильтрацию данных - обязательный аргумент
  - **opt** - опциональный объект, задающий дополнительные условия формирования результата

Если в результате фильтрации получается несколько записей, то будет возвращена первая запись.

Например, пусть в таблице 'weighing' хранятся данные о взвешивании машин:

- car - номер машины
- weight - вес в тоннах
- ts - время взвешивания
- point - пункт взвешивания
- comment - дополнительный комментарий

Будем получать данные из таблицы с фильтром по номеру машины.

Пример использования в скрипте визуализации:

```

try {
  // Найдем последнюю запись по конкретной машине
  const mycar = 'в252ee';

  const one = await core.findOneRecord({
    table: 'weighing',
    filter: { car: mycar },
    opt: { sort: { ts: -1 } }
  });
  // Вернет запись последнего по времени взвешивания (sort:{ts:-1})

  if (!one) {
    return { err: 'Не удалось найти запись для а/м ' + mycar };
  }

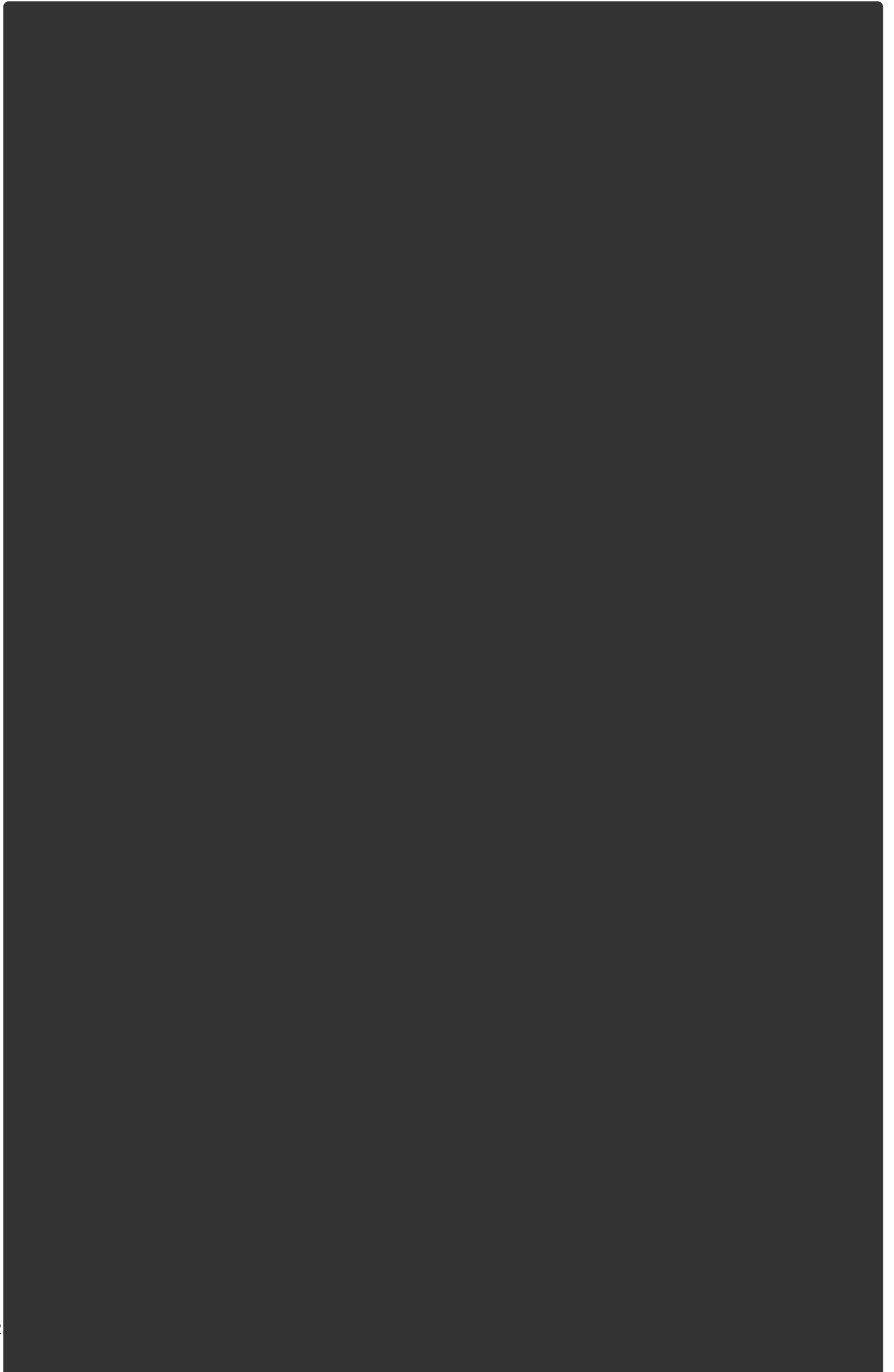
  // Найдем все записи по этой машине,
  // массив будет упорядочен в порядке убывания времени
  const arr_car = await core.getRecords({
    table: 'weighing',
    filter: { car: mycar },
    opt: { sort: { ts: -1 } }
  });

  if (!arr_car.length) {
    return { err: 'Не удалось найти ни одной записи для а/м ' + mycar };
  }

} catch (error) {
  debug('Ошибка: ' + error.message);
  return { err: 'Произошла ошибка при поиске в таблице взвешиваний!' };
}

```

Пример использования в сценарии:



## findRecordById

**findRecordById** возвращает одну запись в виде объекта или пустую строку, если запись не найдена. Данный метод значительно ускоряет поиск в таблице по id, так как не надо искать внутри json объектов.

Входной параметр - название таблицы, id записи в БД.

- **findRecordById(<Имя таблицы - строка>, <ID записи в БД>)**

Пример использования в скрипте визуализации:

```
try {
  // Найдем запись по id
  const id = 12;
  const one = await core.findRecordById('weighing', id)
  if (!one) {
    return {err: 'Не удалось найти запись по id '+ id}
  }
} catch (error) {
  debug('Ошибка: '+error.message)
  return {err: 'Произошла ошибка при поиске в таблице!'}
}
```

Пример использования в сценарии:

```
start() {
  const id = 12;
  // Найдем запись по id
  this.findRecordById('weighing', id, 'onFind');
},

onFind(error, one) {
  if (!error) {
    if (one) {
      this.log('Текущая запись: '+JSON.stringify(one))
    } else {
      this.log('Не удалось найти запись по id ')
    }
  } else {
    this.log('Ошибка при поиске в таблице: '+error)
  }
},
```



## Функции записи данных

### insertRecords

Добавление записей

- `insertRecords({table:<Имя таблицы>, data:[<объект - запись>,... ]})`

Входной аргумент - объект с атрибутами:

- `table` - имя таблицы - строка
- `data` - массив данных для добавления.

Каждый элемент массива - объект, представляющий одну строку (запись) таблицы.

Объект не должен содержать свойство `'id'`, идентификатор генерируется автоматически

Пример использования в скрипте визуализации:

```
try {
  const datarow = [{car:'a876ee', weight: 7.2, point:'cex1', ts: Date.now()
  await core.insertRecords({table:'weighing',data:datarow});
  debug('OK');
} catch (error) {
  debug('Ошибка при добавлении записи: '+error.message)
  return {err:'Ошибка при добавлении записи в таблицу взвешиваний!'}
```

Пример использования в сценарии:

```
start() {
  const data = [{car:'a876ee', weight: 7.2, point:'cex1', ts: Date.now()}
  this.insertRecords({table:'weighing',data}, 'onInsert');
},

onInsert(error) {
  if (!error) {
    this.log('Запись добавлена в таблицу взвешиваний.')
  } else {
    this.log('Ошибка при добавлении записи в таблицу взвешиваний: '+error)
  }
}
```

## Изменение записей

- **updateRecords**({table:<Имя таблицы>, filter:{}, data:{}})

Входной аргумент - объект с атрибутами:

- table - имя таблицы - строка
- filter - объект - условие фильтра
- data - объект данных, содержащий изменения.

Будут изменены записи, удовлетворяющие условиям фильтра

Пример: **updateRecords**({table:'weighing', filter:{id:42}, data:{comment:'Это комментарий'}})

Будет изменена одна запись, так как id - уникальный идентификатор.

Можно изменить несколько записей, задав условия фильтра.

Например, добавим комментарий ко всем записям для конкретного пункта взвешивания.

Пример использования в скрипте визуализации:

```
try {
  //
  const data = { comment:'Весы второго цеха'};
  const filter = { point:'cex1'};
  await core.updateRecords({table:'weighing', filter, data});
  debug('OK');
} catch (error) {
  debug('Ошибка при изменении записей: '+error.message)
  return {err:'Ошибка при изменении записей в таблице взвешиваний!'}
}
```

Пример использования в сценарии:

```
start() {
  const data = { comment:'Весы второго цеха'};
  const filter = { point:'cex1'};
  this.updateRecords({table:'weighing', filter, data}, 'onUpdate');
},

onUpdate(error) {
  if (error) {
    this.log('Ошибка при изменении записей в таблице взвешиваний: '+error)
  }
}
```

## removeRecords

## Удаление записей

- **removeRecords({table:<Имя таблицы>, filter:{}})**

Входной аргумент - объект с атрибутами:

- table - имя таблицы - строка
- filter - объект - условие фильтра Будут удалены записи, удовлетворяющие условиям фильтра

Пример: **removeRecords({table:'weighing', filter:{id:42}})**

Будет удалена одна запись, так как id - уникальный идентификатор.

Можно удалить несколько записей, задав условия фильтра.

Например, удалим все записи по конкретному пункту взвешивания.

Пример использования в скрипте визуализации:

```
const filter = { point:'cex1'};
await core.removeRecords({table:'weighing', filter});
```

Пример использования в сценарии:

```
start() {
  const filter = { point:'cex1'};
  this.removeRecords({table:'weighing', filter}, 'onUpdate');
},

onUpdate(error) {
  if (error) {
    this.log('Ошибка при удалении! '+error)
  }
}
```

# Изображения

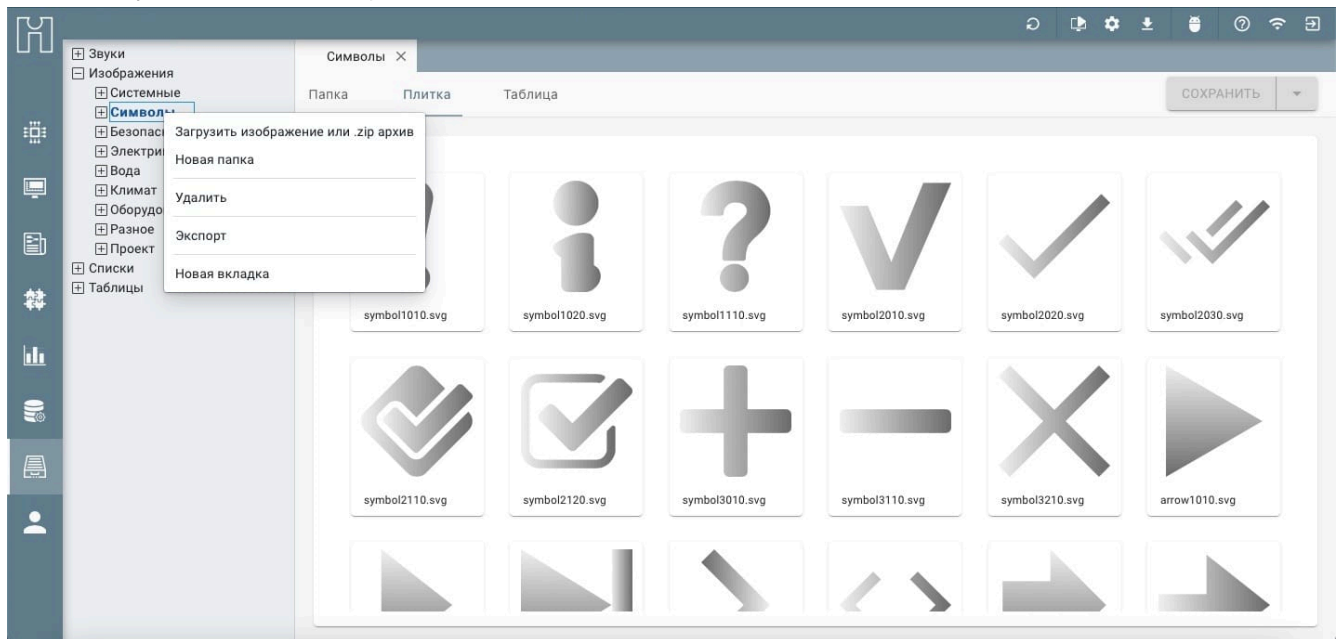
Поддерживаются файлы форматов SVG, PNG, JPG и GIF.

Для иконок устройств рекомендуется использовать SVG. Изображения в этом формате масштабируются без потери качества. Кроме этого будет возможность динамического управления цветом изображения.

Изображения в форматах PNG и JPG рекомендуется сжимать для минимизации размера и увеличения скорости загрузки. Для этого можно воспользоваться любой программой сжатия изображений.

Изображения в формате GIF поддерживаются, но не рекомендуются. Эти изображения могут существенно увеличить нагрузку на браузер.

Необходимые файлы с изображениями можно загрузить в систему, нажав правой кнопкой мыши на соответствующей папке с изображениями:



Можно загрузить как отдельные изображения, так и архивы с наборами изображений.

Папки с изображениями можно перегруппировать простым переносом папок в дереве методом drag&drop.

# Звуки

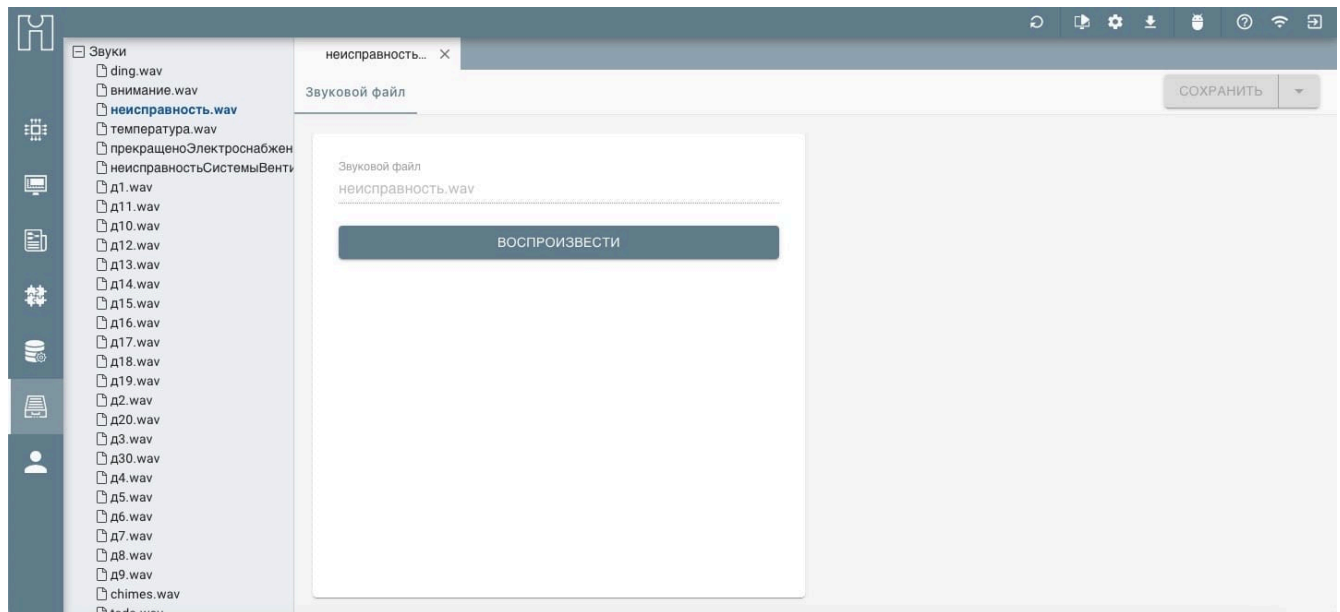
## Назначение

В проектах автоматизации можно использовать различные звуковые сообщения.

Поддерживаются звуковые файлы форматов wav и mp3.

Звуки воспроизводятся на стороне клиента (в пользовательском интерфейсе), в браузере.

Необходимые звуковые файлы нужно загрузить в систему, нажав правой кнопкой мыши на папке "Звуки":



## Применение

### Использование звуковых сообщений в сценариях

Формат сообщения:

```
this.info('sound', 'userID', 'Звуковой файл');
```

Можно указать группу:

```
this.info('sound', 'grp003', 'Звуковой файл');
```

## Примеры

Пример1. Звуковой сигнал:

```
this.info('sound', 'admin', 'ding.wav');
```

Пример2. Звуковой сигнал и сообщение:

```
this.info('sound', 'admin', 'ding.wav, неисправностьСистемыВентиляции.wav')
```

В примерах выше звуковые сообщения будут выданы на работающие пользовательские интерфейсы, вход на которые выполнен под пользователем с id='admin'.

## Списки

Данный ресурс позволяет выводить в пользовательском интерфейсе данные в виде списков. Для отображения данных списков используются визуальные компоненты типа **List** и **Autocomplete**.

Список можно заполнить двумя способами:

### Вручную.

static\_list ×

Описание Редактор СОХРАНИТЬ ▼

ID  
vlst002

Название  
static\_list

Использовать скрипт для заполнения  
☐

id	title
d0001	Устройство 1
d0002	Устройство 2
d0003	Устройство 3
d0004	Устройство 4

Комментарий

### Динамически, используя скрипт для заполнения.

Пример скрипта для заполнения списка из пользовательской таблицы:

Должен вернуть `{data : [массив]}`

Элементы массива - это строки списка: `{id : <идентификатор строки>, title : <видимый текст>}`

```

/**
 * Скрипт для заполнения списка
 * Должен вернуть объект, содержащий
 *   data: массив объектов {id, title}
 */
module.exports = async function ({local, context, source}, core, debug) {
  const data = [];
  const docs = await core.getRecords('mytable', {myfield:'test'}, {sort:{ts
  if (docs) {
    docs.forEach(doc => {
      if (doc.name) {
        data.push({id:doc._id, title: doc.name})
      }
    })
  }
  return {data:data};
}

```

Пример скрипта для заполнения списка устройствами из дерева устройств, начиная с конкретного узла дерева: 'dg061' - расположения узла в дереве (location)

```

module.exports = async function ({local, context, source}, core, debug) {
  const data = await core.getListFromTree('devdevices', 'dg061');
  return {data:data};
}

```



## Сетки

Данный ресурс позволяет выводить в пользовательском интерфейсе данные в виде сетки.

Для отображения данных используется визуальный компонент типа **Grid**. Сетку можно заполнить двумя способами:

### Вручную.

Задать число строк и столбцов, а также дефолтное значение.

Таким способом можно создавать только очень простые конструкции.

### Динамически, используя скрипт заполнения.

С помощью скрипта можно создать интерактивную сетку с индивидуальным поведением и раскраской ячеек.

Скрипт должен вернуть объект, содержащий:

- **columns**: массив объектов с описанием столбцов {title, prop, width <,readonly>}
  - **title**: Текстовое название, которое появится в шапке
  - **prop**: идентификатор столбца, является ключом в строках
  - **width**: ширина столбца
  - **readonly**: true/false - опциональный параметр, запрещает редактирование данных столбца.  
По умолчанию редактирование разрешено
- **data**: массив объектов. Каждый элемент массива - это строка таблицы.  
Строка должна содержать объект для каждого столбца, содержащий тип и значение {type, value <,data, background>}.
  - **type**: тип значения в ячейке. Допустимые типы:
    - type:'number' - ячейка содержит число
    - type:'string' - ячейка содержит строку
    - type:'droplist' - ячейка содержит строку, значение выбирается из выпадающего списка
  - **value**: значение
  - **sub**: подписка на реалтайм значение свойства устройства
  - **background**: цвет фона ячейки
  - **bold**: жирный шрифт
  - **italic**: курсив
  - **fontSize**: размер шрифта
  - **data**: для type:'droplist' определяет имя массива в объекте **droplists**
- **droplists**: опциональный объект содержит списки для ячеек с type:'droplist'

Пример скрипта заполнения сетки

```

/**
 * Скрипт заполнения сетки
 */
module.exports = async function({ local, context }, core, debug) {
  const columns = [
    { title: 'Техпроцесс', prop: 'main', width: 128, readonly: true },
    { title: 'Фаза 1', prop: 'step1', width: 100 },
    { title: 'Фаза 2', prop: 'step2', width: 100 },
    { title: 'Фаза 3', prop: 'step3', width: 100 }
  ];

  const data = [
    {
      main: { value: 'Температура' },
      step1: { value: 1, type: 'number', italic: true, sub: "d0002_value#st"},
      step2: { value: 2, type: 'number', bold: true },
      step3: { value: 3, type: 'number', fontSize: 15 }
    },
    {
      main: { value: 'Режим' },
      step1: { value: 'Dry', type: 'droplist', data: 'mode' },
      step2: { value: 'Heating', type: 'droplist', data: 'mode' },
      step3: { value: 'Off', type: 'droplist', data: 'mode' }
    },
    {
      main: { value: 'Цвет' },
      step1: { value: 'red', type: 'string', background: '#FF0000' },
      step2: { value: 'green', type: 'string', background: '#00FF00' },
      step3: { value: 'blue', type: 'string', background: '#0000FF' }
    }
  ];

  return {
    data,
    columns,
    droplists: {
      mode: ['Off', 'Heating', 'Dry']
    }
  };
};

```

В примере определены 4 столбца.

Оptionальный параметр `readonly:true` запрещает редактирование столбца `main`.

Обратите внимание, столбец не определяет тип, тип определяется для каждой ячейки.

Так, в первой строке вводится число, а во второй доступен только выбор из выпадающего списка mode (type:'droplist', data:'mode').

Массив вариантов для droplist-а **mode** необходимо включить в объект **droplists**.

В последней строке демонстрируется возможность раскраски ячеек с помощью параметра **background**.

## Скрипт обработки.

Данный скрипт используется для обработки Команды элемента **Save** для Grid, а так же данный скрипт вызывается при изменении элементов сетки. При вызове в скрипт передаются в виде объекта **context** с указанием команды **setgridcell** или **setgridall**:

```
{
  userid: 'admin',
  username: 'Admin',
  usergroups: 'admgrp',
  start_layoutid: 'l034',
  layoutid: 'l034',
  command: 'setgridcell'
}
```

и **source**:

```
{
  rowidx: 0,
  colidx: 2,
  prop: 'step2',
  value: 2,
  prev: 1,
  row: {
    nameWithMu: { value: 'Молоко', type: 'string' },
    step1: { value: 1, type: 'number' },
    step2: { value: 2, type: 'number' },
    prop: 'p1',
    min: 0,
    max: 0,
    vtype: 'N'
  }
}
```

Пример скрипта обработки с контролем ввода значения в ячейку сетки:

```
module.exports = async function({ local, context, source }, core, debug) {  
  try {  
    if (!source || !source.row) throw { message: 'Expect source object!' };  
    checkValue(source.value);  
    return { response: 1 };  
  } catch (e) {  
    return { response: 0, alert: e.message, value: source.prev };  
  }  
  
  // Проверка текущего значения – min/max, бизнес-правила ...  
  function checkValue(val) {  
    if (val > 1000) throw {message: 'Значение должно быть не больше 1000!'}  
  }  
};
```

## Таблицы

Данный ресурс позволяет выводить в пользовательском интерфейсе данные в виде таблицы. Для отображения данных таблиц используется визуальный компонент **Table**.

Таблицу можно заполнить двумя способами, используя соответствующие чекбоксы.

### Заполнить из пользовательской таблицы

IntraSCADA позволяет создавать [Пользовательские таблиц](#) для хранения данных проекта.

Это могут быть прикладные данные проекта - справочники, настройки, уставки, рецепты и т.д.

С другой стороны, их можно использовать для хранения любых данных о процессах (срезы, периодические замеры, специально обработанные исторические данные). Данные [Пользовательской таблицы](#) можно вывести в интерфейсе, используя компонент **Table**. Более подробную информацию можно получить по [ссылке](#)

### Опциональная обработка данных таблицы

Используя имена полей таблицы, можно довольно гибко задать условия фильтрации и/или сортировки данных. **Фильтрация** и **Сортировка** ожидают объекты в JSON формате: имена полей должны быть **в кавычках**, сам объект заключен **в фигурные скобки**.

Например, пусть таблица хранит данные о каких-то операциях взвешивания и имеет, среди прочих, столбцы:

- weight - вес
- cex - номер цеха
- ts - время выполнения операции
- inumber - номенклатурный номер

Чтобы вывести операции по цеху 1, в которых вес превышает 50, строка **Фильтрация** должна содержать такой объект:

```
{"cex":"1", "weight":{"$gt": 50} }
```

Чтобы отсортировать данные по номенклатурному номеру и времени, строка **Сортировка** должна содержать:

```
{"sort":{"inumber":1, ts":1 } }
```

Если номенклатурный номер - это числовой код, может быть полезен вариант альтернативной сортировки:

```
{"order":"inumber,ts"}
```

Подробное описание синтаксиса дано в разделе [Фильтрация и сортировка данных](#)

## Использовать скрипт для заполнения таблицы

Пример добавления таблицы со статическими полями.

```
/**
 * Скрипт для заполнения таблицы
 * Должен вернуть объект, содержащий
 *   columns: массив объектов с описанием столбцов {title, prop, width, filter},
 *   data: массив объектов, каждый элемент – это строка таблицы
 */
module.exports = async function ({local, context, source}, core, debug) {
  const columns = [
    { title: 'Column1', prop: 'c1', width: 280, filter: true },
    { title: 'Column2', prop: 'c2', width: 280, filter: true },
    { title: 'Column3', prop: 'c3', width: 280, filter: true }
  ];

  const data = [
    { id: 1, c1: 'Data 1', c2: 'Data 2', c3: 'Data 3', level: 0 },
    { id: 2, c1: 'Data 4', c2: 'Data 5', c3: 'Data 6', level: 1 }
  ];
  return {columns: columns, data: data};
}
```

Свойства **id** в данных таблицы обязательны, если вы хотите, чтобы на интерфейсе можно было выделять строки в таблице.

Пример добавления таблицы с запросом данных к пользовательской таблице, при условии, что свойства пользовательских таблиц будут совпадать с названиями свойств в prop массива columns

```
const columns = [
  { title: 'Column1', prop: 'c1', width: 280, filter: true },
  { title: 'Column2', prop: 'c2', width: 280, filter: true },
  { title: 'Column3', prop: 'c3', width: 280, filter: true }
];
const data = await core.getRecords('mytable', {myfield: 'test'}, {sort: {ts: 'asc'}});
return {columns: columns, data: data}
```

При получении данных с **Пользовательской таблицы проекта** будьте внимательны, так как в качестве уникальной **id** записи, приходит свойство **\_id**. Так таблица может работать только с **id**, нужно переписать **\_id** в **id**.

```
data.forEach(item => {item.id=item._id});
```

## Деревья

Данный ресурс позволяет выводить в пользовательском интерфейсе данные в виде дерева.

Для отображения данных используются визуальный компонент типа **Tree**.

Список можно заполнить двумя способами: вручную и с помощью скрипта заполнения.

### Заполнение с помощью скрипта

Пример скрипта для заполнения дерева

```
/**
 * Скрипт для заполнения дерева
 * Должен вернуть объект, содержащий
 * data: массив узлов дерева
 */
module.exports = async function ({local, context, source}, core, debug) {
  const data = [
    {
      id: 'root',
      title: 'Главный узел',
      children: [
        {
          id: 'lev1',
          title: 'Узел вложенный',
          children: [
            { id: '1', title: 'Раз' },
            { id: '2', title: 'Два' },
            { id: '3', title: 'Три' }
          ]
        }
      ]
    }
  ]
};
return { data };
```

Пример скрипта для заполнения дерева устройствами из дерева устройств, начиная с конкретного узла дерева:  
'dg061' - расположения узла в дереве (location)



```
module.exports = async function ({local, context, source}, core, debug) {  
  const data1 = await core.getTree('devices', 'dg061');  
  const data2 = await core.getTree('devices', 'dg083');  
  const data3 = await core.getTree('devices', 'dg085');  
  
  return {  
    data: [  
      ...expandFirst(data1),  
      ...expandFirst(data2),  
      ...expandFirst(data3)  
    ]  
  };  
}  
  
function expandFirst(data) {  
  data[0].expanded = true;  
  return data;  
}
```

# Файлы

В проектах автоматизации иногда возникает необходимость работы с пользовательскими файлами.

В IntraSCADA можно загрузить файл любого формата.

Рекомендуем загружать файлы, поддерживаемые для отображения в браузере: pdf, txt, csv и т.д.

Можно создавать папки для группировки файлов. Эта группировка работает для фильтрации и упорядочивания. Доступ к файлу выполняется просто по имени.

Загрузить файлы можно:

- в РМ через правое меню операций **Import**
- в пользовательском интерфейсе, используя компонент ... (доступно начиная с версии 5.11.5)

Для вывода файла в пользовательском интерфейсе можно использовать элемент визуализации [Iframe](#).

Например, для показа файла с именем **test.pdf** в [Iframe](#) в качестве Url достаточно прописать:

```
/files/test.pdf
```

[Скрипт визуализации](#) имеет набор [Функций работы с пользовательскими файлами](#), которые позволяют копировать, удалять, переименовывать пользовательские файлы.

Обратите внимание:

- Имя файла должно быть уникально для проекта
- Доступ к файлу выполняется просто по имени (без учета папок)

## Скриншоты

В данном разделе сохраняются скриншоты сделанные с помощью команды **Скриншот**

# Рецепты

Добавлено v5.14.38

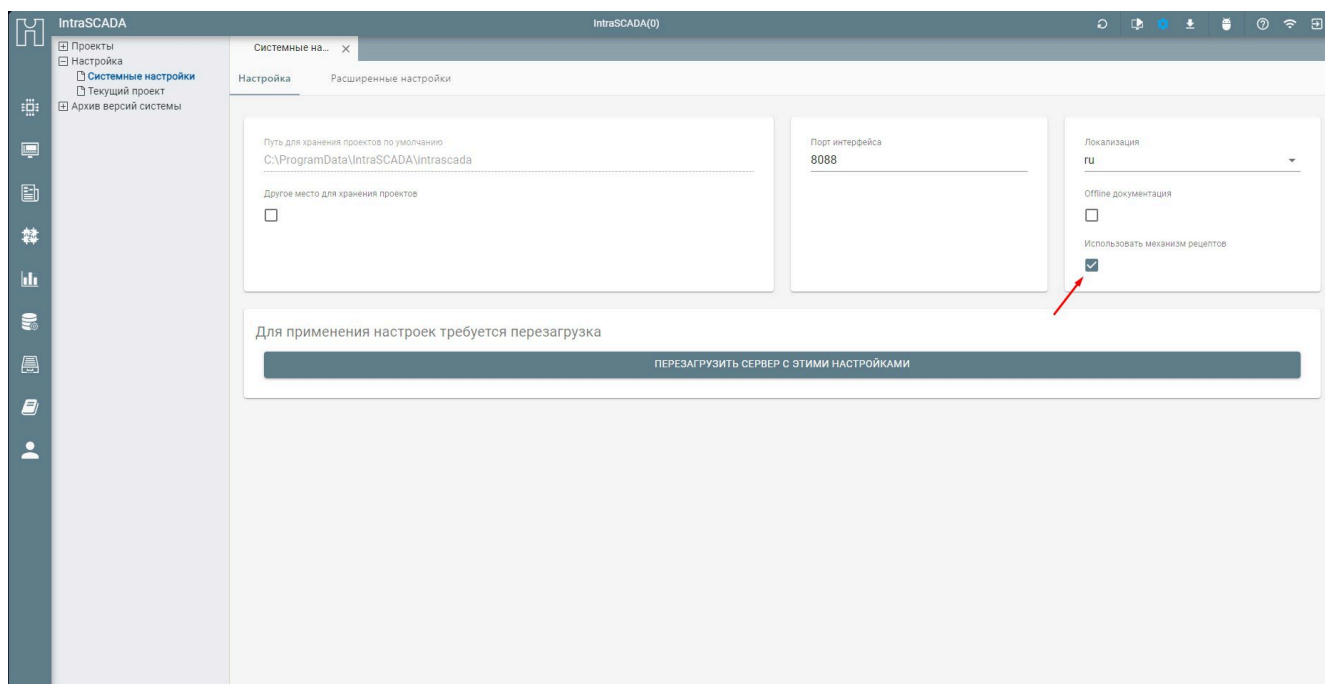
Механизм рецептов IntraSCADA позволяет:

- создавать схемы рецептов (в том числе имеющие множество фаз)
- на основе схем заполнять экземпляры рецептов (формулы) и сохранять их в БД
- работать с формулой рецепта на физическом устройстве, используя механизм каналов
  - считывать загруженную формулу
  - записывать выбранную формулу (возможно, с предварительным пересчетом)
  - редактировать уже загруженные элементы формулы (если это допускает техпроцесс)

Для создания схем рецептов и доступа к формулам в БД служит раздел **Рецепты** в левом меню РМ. Для работы с физическим устройством следует создать **Тип устройства** со ссылкой на одну из схем. В устройстве будут автоматически созданы свойства для работы с заданным рецептом.

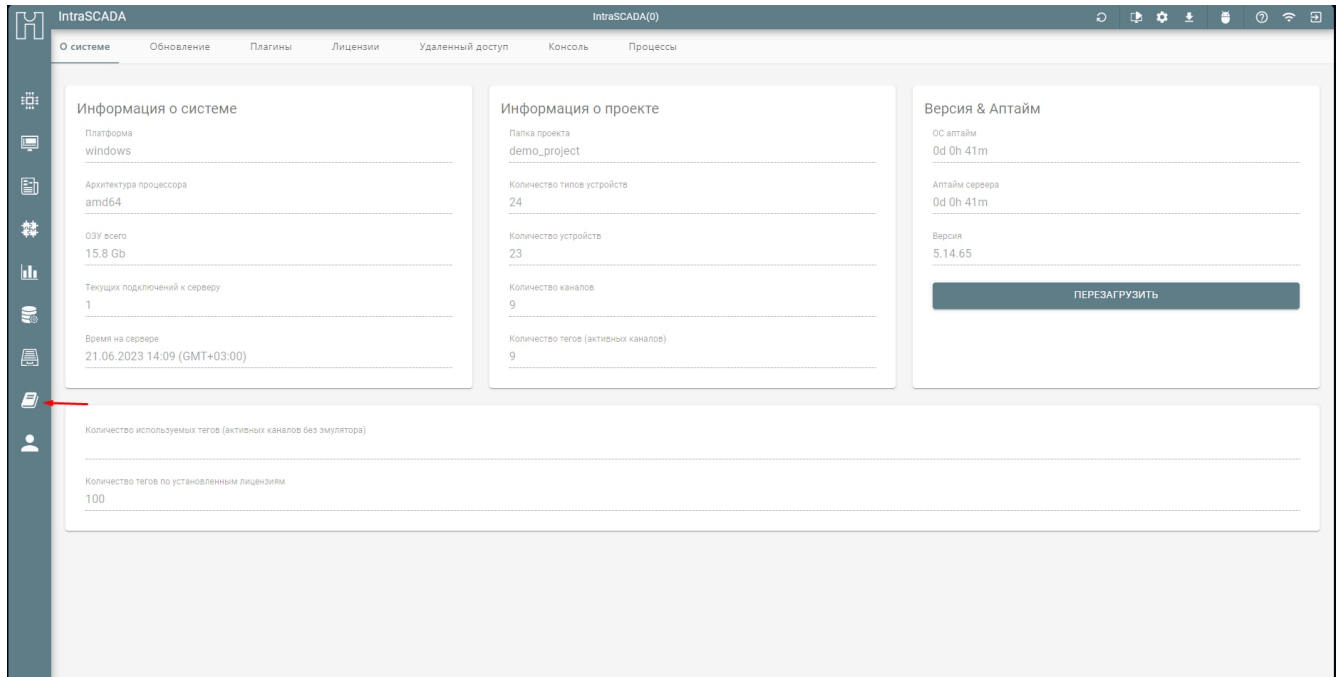
## Как включить механизм рецептов

Функционал рецептов является опциональным, включается флагом в **Системных настройках**:

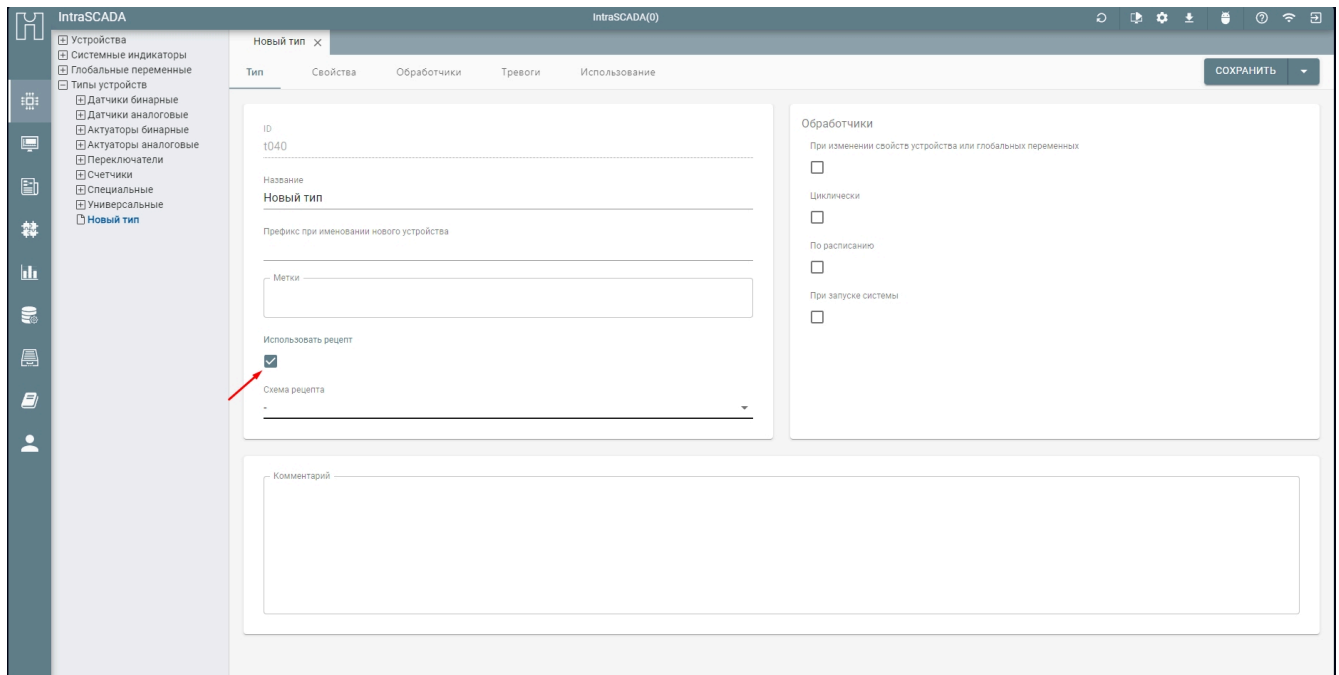


После установки галочки сервер нужно перезагрузить.

При включенном флаге в РМ появляется раздел **Рецепты**:



и возможность в **Типе устройства** задать ссылку на Рецепт:



## Раздел Рецепты

Каждый узел в дереве **Рецепты** содержит описание и схему рецепта.

При добавлении нового рецепта в **Дереве рецептов** на вкладке **Описание** следует определить:

- Наименование рецепта
- Максимальное число фаз техпроцесса
- Список **Общих параметров** рецепта, не зависящий от фаз (например, нормативный вес). Для каждого параметра обязательно задается:
  - Идентификатор (допустимы только латинские буквы, цифры и знак подчеркивания)

- Название (любая строка)
- Тип значения (Number, Bool, String).

Остальные атрибуты являются опциональными

## Схема рецепта

На вкладке **Схема рецепта** следует ввести составляющие рецепта - ингредиенты или технологические параметры.

Далее по этой схеме может быть создано любое количество формул (экземпляров рецепта с конкретными значениями).

Для каждой строки вводится:

- Номер строки в матрице рецепта
- Идентификатор (допустимы латинские буквы, цифры, знак подчеркивания )
- Название (любая строка)
- Вид (Ингредиент или Технологический параметр) Ингридиент пересчитывается в зависимости от общих параметров рецета.
- Тип значения (Number, Bool, List)

Для списков (тип List) нужно определить строку с вариантами через разделитель |

Примеры:

- NO|YES
- 2400|4800|9600|19200|38400|57600|15200
- ON|OFF|AUTO
- Ручной|Автоматический

Списки предназначены только для удобства ввода, хранятся и передаются на устройство индексы соотв вариантов (начиная с 0).

Пример: NO|YES, NO=>0, YES=>1

## Загрузка и выгрузка схемы рецепта из csv

Схема рецепта (список ингредиентов) может быть выгружена и загружена через csv.

Общие параметры при этом не выгружаются!

## Связь рецепта с устройством

Чтобы связать рецепт с устройством, достаточно в **Типе устройства** поставить галочку **Использовать рецепт** и выбрать конкретный рецепт (схему) из списка.

В устройствах этого типа автоматически будут добавлены:

- статические свойства
  - `rcpid` - ID рецепта (схемы)
  - `maxPhases` - max число фаз
- специальные свойства, имеющее тип `Recipe param`.  
Они формируются из списка общих параметров рецепта.  
Значения этих свойств будут заполняться из общих параметров выбранной формулы.  
Эти свойства с точки зрения использования имеют такой же функционал, как обычные свойства - их можно привязывать к каналам, использовать на визуализации, получать значение в скрипте.
- свойство `matrix` - матрица для значений формулы (строки - ингредиенты, столбцы - фазы).  
Для столбцов берется максимальное количество фаз, определенное в рецепте.  
Каждая ячейка может быть привязана к каналу.

Для получения значения ячейки матрицы можно использовать специальный метод устройства:

**`dev.getMatrixValue`**(<ID строки>, <ID столбца>)

```
// значение параметра MixerRate на фазе 2
dev.getMatrixValue('MixerRate', 2);
```

или стандартный метод **`dev.getPropValue`**, в этом случае имя свойства собирается из 3 частей:

```
// значение параметра MixerRate на фазе 2
dev.getPropValue('matrix.MixerRate.2');
```

# Хранение рецептов в БД

Формулы хранятся в основной БД в таблице **formulas**.

## Структура таблицы formulas

- id - уникальный идентификатор формулы - число (PRIMARY KEY NOT NUL)
- rid - идентификатор рецепта (схемы) - строка (NOT NULL)
- title - название формулы - строка (NOT NULL)
- description - описание формулы - строка
- comments - комментарий - строка
- active - флаг активности - число (1/0)
- ts - таймштамп - момент последней модификации
- jhead - JSON поле, содержит значения для **Общих параметры формулы**
- jrows - JSON поле, содержит массив значений ингредиентов из **Схемы** по каждой фазе.

Пример рецепта:

- ID рецепта = 'rcp003'
- Max число фаз = 5

Общие параметры рецепта:

- Phases - число фаз, Number
- Weight - вес партии, Number

Схема рецепта:

- Milk, Number, Ингредиент
- Filler, Number, Ингредиент
- MixerRate, Number, Техн параметр
- PauseTime, Number, Техн параметр

Одна из записей для этой схемы - формула №42:

- id = 42
- rid = 'rcp003'
- title = 'Apple Cocktail'
- active = 1
- jhead: {Phases:2, Weight: 200}
- jrows: {Milk:[100, 100], Filler:[0, 2.5], MixerRate:[1,2], PauseTime:[0.5,1]}

## Загрузка формул из csv

Загрузка существующих формул в таблицу formulas возможна из csv файла. Поскольку формат исходных данных может быть различен, для обработки загружаемых данных предусмотрены редактируемые инсталлятором скрипты загрузки. Редактор скриптов загрузки находится в корневом узле Рецепты -> вкладка Скрипты загрузки. Сама загрузка выполняется для выбранной схемы в 2 этапа:

- Загрузка списка формул (добавляется запись для каждой формулы + общие параметры)



- Загрузка данных формул (данные по схеме рецепта)

# Редактирование формул рецептов

## Редактирование формул через UI с использованием элемента Grid.

Формулы сохраняются в БД (таблица formulas).

Действия по загрузке и сохранению выполняются в скриптах с использованием функций.

### Пример скрипта заполнения сетки для нового или существующего рецепта

```
// в local.formula_id записывается id формулы при выборе
// или '_new' (условно) для создания нового рецепта
module.exports = async function({ local, context, source }, core, debug) {
  const rid = 'rcp002'; // идентификатор схемы
  const schema = await core.getRecipeSchema(rid);
  const maxPhases = schema.phases;

  const columns = [
    { title: 'Ингредиент', prop: 'nameWithMu', width: 128, readonly: true }
  ];

  for (let i = 1; i <= maxPhases; i++) {
    columns.push({ title: 'Фаза ' + i, prop: 'step' + i, width: 80 });
  }

  const fid = local.formula_id;
  const data = await core.getFormulaGridData(fid, schema, columns);

  return {
    data,
    columns,
    droplists: schema.droplists
  };
};
```

Для сохранения данных есть Команда элемента grid - **save**

Сохранение данных выполняется в скрипте обработки.

- при редактировании каждой ячейки
- при получении команды элемента grid - save

## Функции, используемые для записи в БД

- Получение формулы из БД

```
await core.getFormula(fid);
```

- Добавление новой формулы в рецепт для схемы rid со значениями по умолчанию для jhead и jrows

```
await core.insertFormula(rid, {title, description, active:1 })
```

- Добавление формулы в БД для схемы rid со своими jhead и jrows

```
await core.insertFormula(rid, {title, description, active, jhead, jrows
```

- Запись измененной формулы в БД

```
await core.updateFormula(fid, {title, description, active, jhead, jrows
```

- Подготовка данных сетки для записи в БД

```
jrows = core.getJrowsFromGridData(source, schema, phases);
```

- Удаление формулы из БД

```
await core.deleteFormula(fid);
```

## Функции, используемые для записи в PLC

- Запись рецепта в контроллер

```
const fid = local.formula_id; // id формулы
const did = 'd0235';
const unit = 'modbus1';
const batchWeight = 0;
core.writeFormulaToPlugin(fid, did, unit, batchWeight);
```

## Пример скрипта обработки сетки

Данный скрипт используется для редактирования уже существующего рецепта. Вызывается по команде Save элемента Grid

```

module.exports = async function({ local, context, source }, core, debug) {
  const rid = 'rcp002'; // идентификатор рецепта (схемы)
  const fid = local.formula_id;

  // context.command содержит команду setgridcell при редактировании ячейки,
  // setgridall при команде save
  return context.command == 'setgridall' ? save() : check();

  async function save() {
    const schema = await core.getRecipeSchema(rid);

    try {
      const title = local.rcp_title;
      const active = 1;
      const phases = schema.phases;
      // Обязательно нужен Phases, остальные общие параметры опционально
      const jhead = {Phases: phases};
      // формирует из данных, полученных от сетки (source) для записи в БД
      const jrows = core.getJrowsFromGridData(source, schema, phases);

      core.updateFormula(fid, {title, jhead, jrows })
      core.updateLocals({rcp_list:local.rcp_list})
      return { response: 1 };
    } catch (e) {
      return { response: 0, alert: e.message || 'Ошибка при сохранении формулы' };
    }
  }

  // При редактировании ячейки –
  // здесь можно добавить условие на проверку вводимых данных
  async function check() {

  }
}

```

Для ввода общих параметров рецепта и заголовочной информации придется создать локальные переменные и загрузить их при переходе к операции редактирования отдельным запросом к БД или константами для новой записи

## Примеры скриптов визуализации по кнопкам Новая формула, Удалить формулу, Редактировать формулу

- Пример скрипта визуализации по кнопке Редактировать формулу

```
module.exports = async function({ local, context, source }, core, debug) {
  if (!local.formula_id) return { err: 'Выберите формулу!' };

  // Получаем данные по выбранной формуле
  const recs = await core.getRecords(
    'formulas',
    { sql: 'SELECT jhead from formulas WHERE id=' + local.formula_id }
  );

  // Записываем полученные данные в локальные переменные
  // для последующего вывода на форму (диалог)
  let title = recs[0].title;
  let active = recs[0].active;
  let weight = recs[0].jhead.BatchWeight;

  core.updateLocals({
    formula_newbatch: weight,
    formula_title: title,
    formula_active: active
  });

  core.showDialog('di0074');
};
```

- Пример скрипта визуализации по кнопке Новая формула

```

module.exports = async function({ local, context, source }, core, debug) {
  const rid = 'rcp002'; // идентификатор схемы
  const title = local.rcp_title;
  const description = 'Самый новый коктейль';

  const id = await core.insertFormula(rid, {
    title,
    description,
    active: 1
  });

  core.updateLocals({
    rcp_list: local.rcp_list,
    formula_id: id,
    grid_id: local.grid_id
  });
};

```

– Пример скрипта визуализации по кнопке Удалить формулу

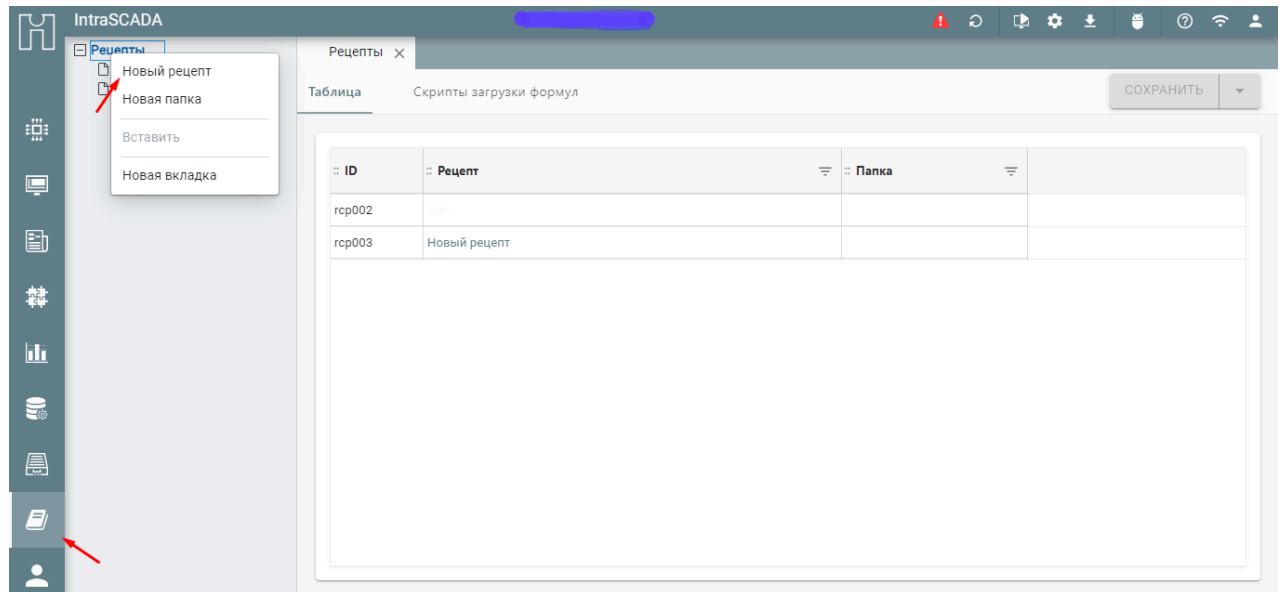
```

```js
module.exports = async function({ local, context, source }, core, debug) {
  await core.deleteFormula(local.formula_id);
  core.updateLocals({rcp_list:local.rcp_list, grid_id:local.grid_id})
};

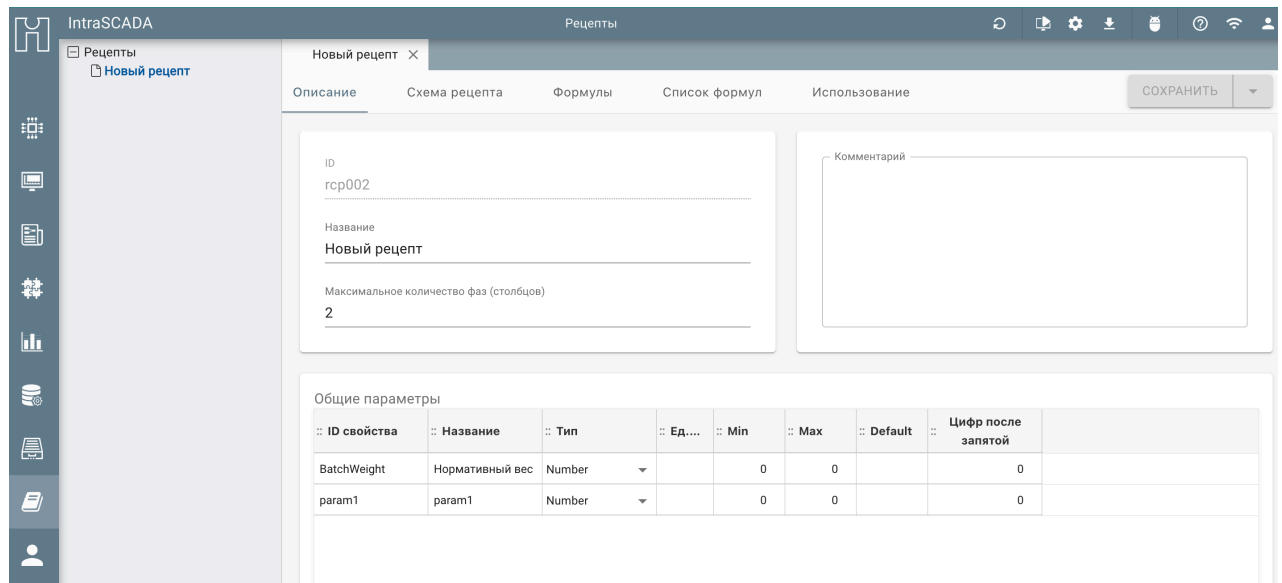
```

## Пример создания рецепта

- Создадим новый рецепт в разделе Рецепты



- Меняем название, указываем количество фаз(столбцов), добавляем общие параметры рецепта



По умолчанию создается параметр BatchWeight (вес замеса), который в дальнейшем будет учитываться при пересчете ингредиентов на другой вес.



- Заполняем схему рецепта

N строки	ID строки	Название	Ед...	Вид	Тип	Min	Max	Default	L n за
1	p1	Молоко		Ингредиент	Num...	0	0	1	
2	p2	Скорость миксера		Тех.параметр	Bool			2	
3	p3	Сироп		Ингредиент	Num...	0	0	0.5	
4	p4	Название		Ингредиент	String				

- Тип может быть:

- Number(Число),
- Bool(true или false),
- String(Строка),
- List(Список строк через разделитель), например OFF|ON|AUTO.

Списки используются для удобства выбора значения параметра пользователем.

Фактическим значением параметра является индекс (порядковый номер) выбранной опции

- Вид может принимать значение:

- Ингредиент
- Технологический параметр

Вид учитывается при пересчете на другой вес.

Если изменить вес замеса, параметры типа Ингредиент автоматически пропорционально пересчитаются, а Тех. параметры не изменятся.

- Создаём новую сетку в разделе Ресурсы -> Сетки.

Для того, чтобы вывести рецепт на визуализацию, будем использовать сетку (элемент Grid). В привязках данного визуального элемента будем выбирать не саму сетку из Ресурсов, а переменную клиента `grid_id`, у которой значение по умолчанию будет **vgird002**. Это нужно будет для того, чтобы при выборе в списке нового рецепта перезаполнять Grid с новыми данными.

- Создаем переменную клиента **grid\_id** со значением по умолчанию **vgird002**, которое соответствует id сетки
- В Скрипте заполнения Сетки пишем скрипт

```
module.exports = async function({ local, context, source }, core, debug) {
  const rid = 'rcp002'; // идентификатор схемы
  const schema = await core.getRecipeSchema(rid);
  const maxPhases = schema.phases;

  // Заполняем шапку таблицы
  const columns = [
    { title: 'Ингредиент', prop: 'nameWithMu', width: 128, readonly: true },
  ];
  for (let i = 1; i <= maxPhases; i++) {
    columns.push({ title: 'Фаза ' + i, prop: 'step' + i, width: 80 });
  }

  const fid = local.formula_id;
  // Получаем данные из таблицы по formula_id
  const data = await core.getFormulaGridData(fid, schema, columns);

  return {
    data,
    columns,
    droplists: schema.droplists
  };
};
```

Где 'rcp002' – ID нашего рецепта

– Создаем новый список в разделе Ресурсы → Списки. Ставим галочку на **\*\*Ис**

![recipe15.png](/images/recipe15.png)

– В Скрипте заполнения Списка пишем скрипт с помощью которого заполняем спи

```
```js
module.exports = async function({ local, context }, core, debug) {
  const records = await core.getRecords(
    'formulas',
    { sql: 'SELECT * from formulas WHERE rid="rcp002" ORDER BY id' }
  );

  const data = records.map(item => ({
    id: item.id,
    title: item.title
  }));

  return { data };
};
```

- Создаем переменную клиента **rcp\_list** со значением по умолчанию **vlst005**, которое соответствует id списка
- Создаем переменную клиента **rcp\_title** для вывода в Input названия текущего рецепта
- Создаем переменную клиента **formula\_id**, в которую будем записывать id рецепта, который выбираем в списке
- Создаем скрипт визуализации **Обновить грид** для обновления сетки при выборе нового рецепта из списка. В скрипте пишем следующее:

```
module.exports = async function({ local, context }, core, debug) {
  // Находим название текущего рецепта в базе
  const sqlStr = `SELECT * from formulas WHERE id=${local.formula_id} ORDER BY`;

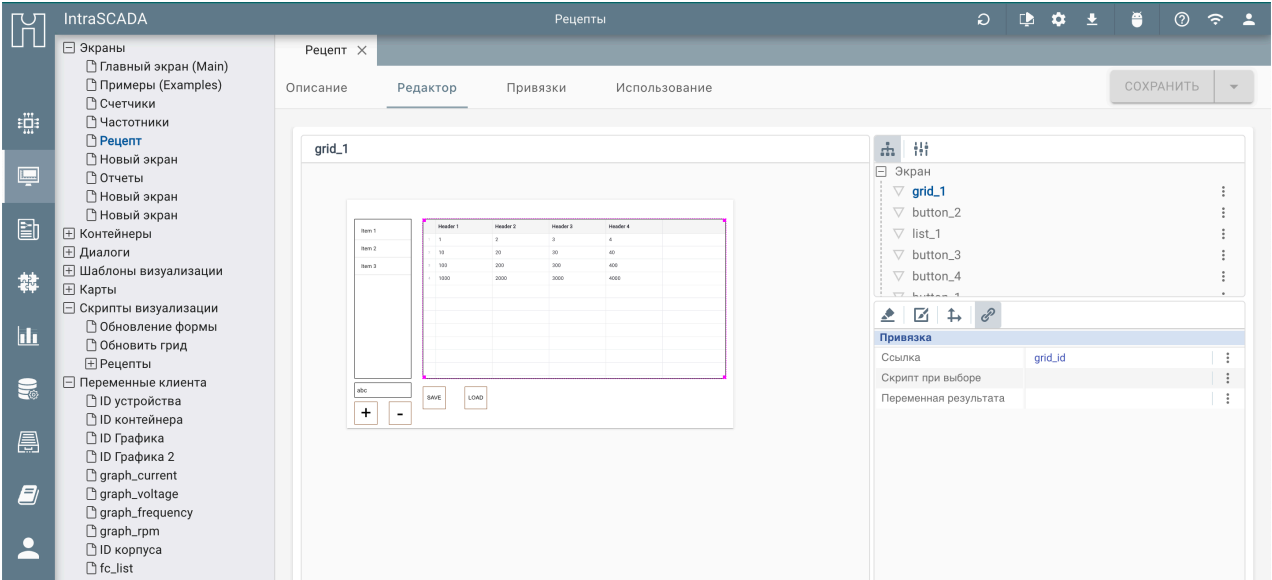
  const records = await core.getRecords(
    'formulas',
    { sql: sqlStr }
  );

  // Обновляем переменные клиента
  core.updateLocals({
    grid_id: local.grid_id,
    rcp_title: records[0].title
  });
};
```

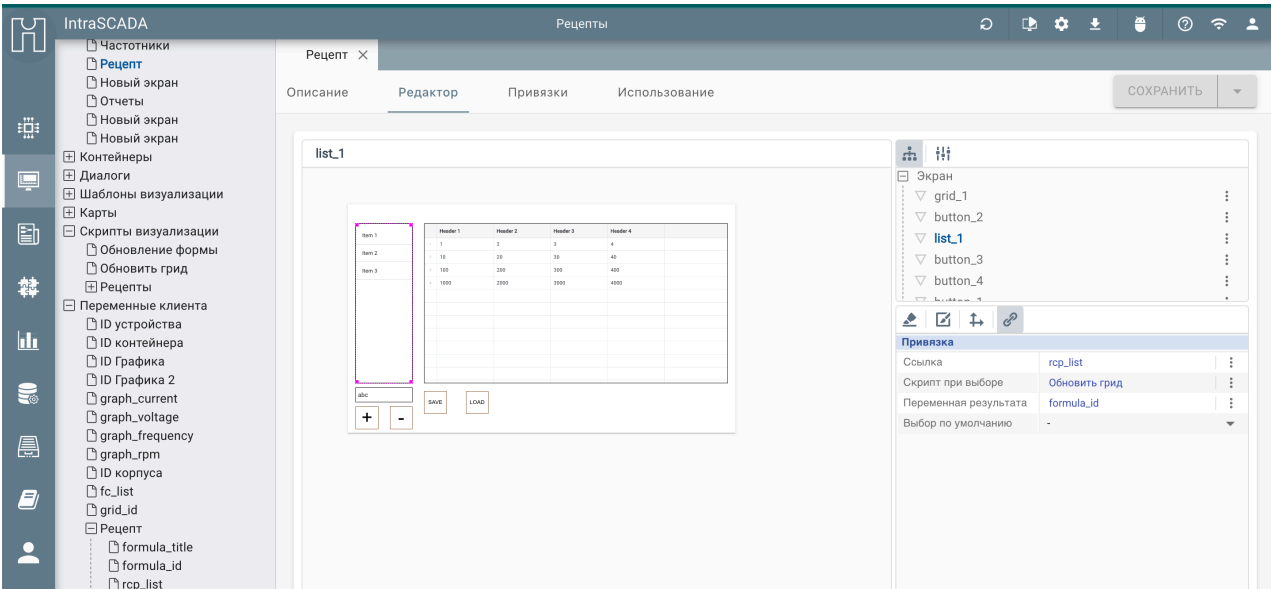
- Добавляем элементы Grid, List, 4 элемента Button и Input на экран.

[illegible]

- Элемент Grid привязываем к переменной клиента **grid\_id**, со значением по умолчанию **vgrid002**



- Элемент List привязываем к переменной клиента **rcp\_list**, со значением по умолчанию **vlist005** Переменную результата к **formula\_id**. Скрипт при выборе к скрипту визуализации **Обновить грид**



Скрипт обработки для нашей Сетки в Ресурсах будет следующий:

```
module.exports = async function({ local, context, source }, core, debug) {

  const rid = 'rcp002'; // идентификатор рецепта (схемы)
  const fid = local.formula_id;

  // context.command содержит команду setgridcell при редактировании ячейки,
  // setgridall при команде save
  return context.command == 'setgridall' ? save() : check();

  async function save() {
    const schema = await core.getRecipeSchema(rid);

    try {
      // Заголовочные и jhead – Сделать через inputs и локальные переменные
      const title = local.rcp_title;
      const active = 1;
      const phases = schema.phases;
      // Обязательно нужен Phases, остальные общие параметры опционально
      const jhead = {Phases: phases};
      // формирует из данных, полученных от сетки (source) для записи в БД
      const jrows = core.getJrowsFromGridData(source, schema, phases);

      //core.updateFormula(fid, {title, description, active, jhead, jrows })
      core.updateFormula(fid, {title, jhead, jrows })
      core.updateLocals({rcp_list:local.rcp_list})
      return { response: 1 };
    } catch (e) {
      return { response: 0, alert: e.message || 'Ошибка при сохранении форм'
    }
  }

  // При редактировании ячейки – здесь все как было
  async function check() {

  }

};
```

- Создаем устройство типа Рецепт и выбираем необходимый нам рецепт, в свойствах устройства добавляем общие параметры для рецепта:

- Элемент Button **Load** привяжем к скрипту визуализации **Загрузить рецепт в PLC**, который передаст параметры рецепта в матрицу устройства. Пример скрипта:

```
module.exports = async function({ local, context }, core, debug) {
  const fid = local.formula_id; // id формулы
  const did = 'd0235'; //id устройства рецепт
  const unit = 'modbus1'; //id плагина
  core.writeFormulaToPlugin(fid, did, unit, 0);
};
```

- Элемент Button **+** привяжем к скрипту визуализации **Добавить рецепт**, который добавит новый рецепт с параметрами по умолчанию, обновит Список, Сетку и установит курсор в списке на текущий рецепт. Пример скрипта:

```

module.exports = async function({ local, context }, core, debug) {
  const rid = 'rcp002'; // идентификатор схемы
  const title = local.rcp_title;
  const description = 'Самый новый рецепт';

  const id = await core.insertFormula(rid, {
    title,
    description,
    active: 1
  });

  core.updateLocals({
    rcp_list: local.rcp_list,
    formula_id: id,
    grid_id: local.grid_id
  });
};

```

- Элемент Button - привяжем к скрипту визуализации **Удалить рецепт**, который удалит выбранный рецепт и обновит Список, Сетку. Пример скрипта:

```

module.exports = async function({ local, context }, core, debug) {
  await core.deleteFormula(local.formula_id);
  core.updateLocals({rcp_list:local.rcp_list, grid_id:local.grid_id})
};

```

В результате у нас получился рецепт, заполненный с помощью элемента Grid, с возможностью сохранения изменений по кнопке **Save**, и с возможностью создания новой формулы с помощью элемента **+**, удаления рецепта по кнопке **-**, загрузка рецепта в ПЛК по кнопке **Load**



# Доступ

## Правила доступа

Правила доступа к системе определяются для **Группы пользователей**. Эти правила регламентируют как доступ к пользовательскому интерфейсу, так и к админке.

По умолчанию в системе существует группа **Администраторы** с неограниченным доступом.

## Пользователи

По умолчанию в системе создается учетная запись **Admin**, включенная в группу **Администраторы**.

Для создания нового пользователя нажмите правой кнопкой мыши в дереве Пользователей.

Включите пользователя в группу, иначе у него не будет никаких прав и он не сможет войти в систему.

**Стартовый экран** задает экран при входе в пользовательский интерфейс.

Обратите внимание!

1. Если стартовый экран не задан, то после ввода пароля у пользователя будет пустой экран
2. Для админа также нужно задать стартовый экран для входа в пользовательский интерфейс
3. В группе пользователя должно быть разрешение на доступ к стартовому экрану.

Если разрешения нет, будет сообщение **Доступ запрещен**

# Учетная запись

Для создания нового пользователя нажмите правой кнопкой мыши в дереве Пользователей.

**Имя пользователя, Логин, Пароль** - обязательные поля.

**Стартовые экраны** задают экраны при входе в пользовательский интерфейс, в зависимости от устройства.

- Основной экран
- Экран для планшета
- Экран для телефона

Он может быть пустым, только если доступ к пользовательскому интерфейсу не планируется.

Например, создается учетная запись только для выполнения информирования.

**Заблокировать пользователя** - флаг временной блокировки пользователя без удаления его аккаунта.

**Эксперт** - флаг, дающий пользователю права на операции, требующие квалификации, например:

- работа в консоли
- установка дополнительных пакетов (библиотек)
- проброс портов в десктопном приложении

Вы можете добавлять пользователей в отдельные папки, если это удобно. Группировка по папкам никак не влияет на права доступа.

Права доступа определяются группой (группами), в которые пользователь входит.

Если пользователь не входит ни в одну группу, он не сможет войти в систему.

## Дефолтный пользователь с правами администратора

По умолчанию в системе создается учетная запись **Admin**, включенная в группу **Администраторы**, то есть имеющая неограниченные права.

Дефолтные настройки: Логин: admin, Пароль: 202020

Настоятельно рекомендуется изменить пароль после установки системы!

При желании, дефолтного пользователя можно удалить совсем. Но при этом нужно убедиться, что есть хотя бы один пользователь, имеющий доступ к админке!!!

## Виртуальная клавиатура

Добавлено v5.17.2

Добавлена виртуальная клавиатура для элемента Input, улучшающая удобство при вводе данных с пользовательского интерфейса.

Элемент Input описан [по ссылке](#).

Чтобы использовать виртуальную клавиатуру, нужно поставить галку в соответствующем поле.

Виртуальная клавиатура

Использовать виртуальную клавиатуру

☒

Только для сенсорных экранов

☐

Тема

Темная

Ширина клавиатуры (px)

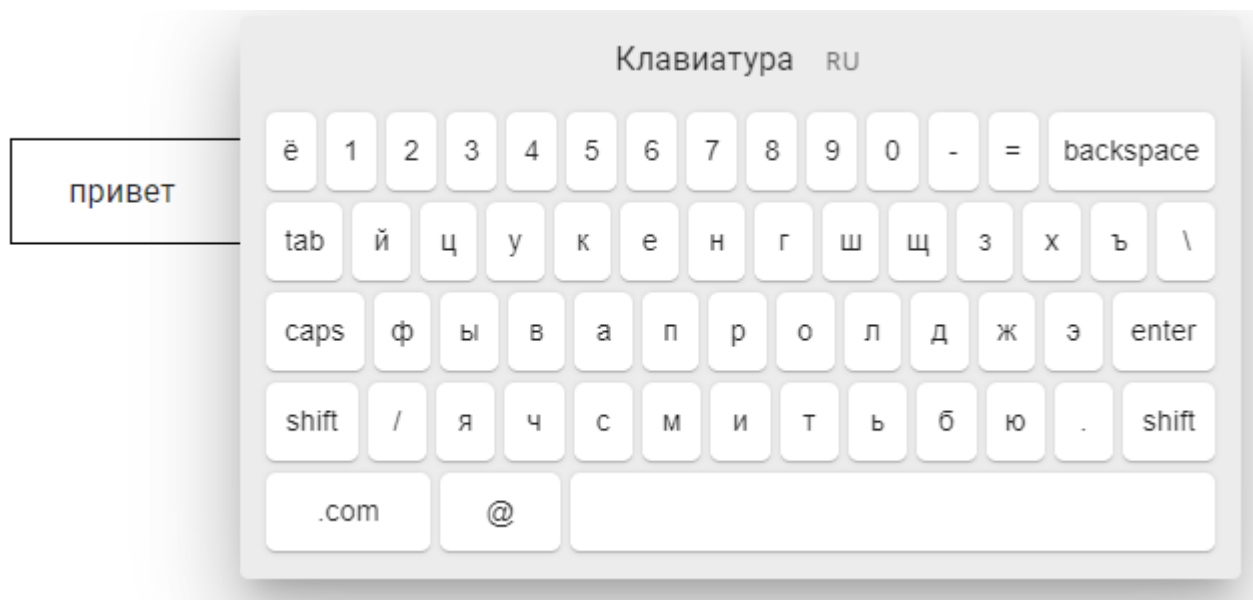
300

Ширина цифровой клавиатуры (px)

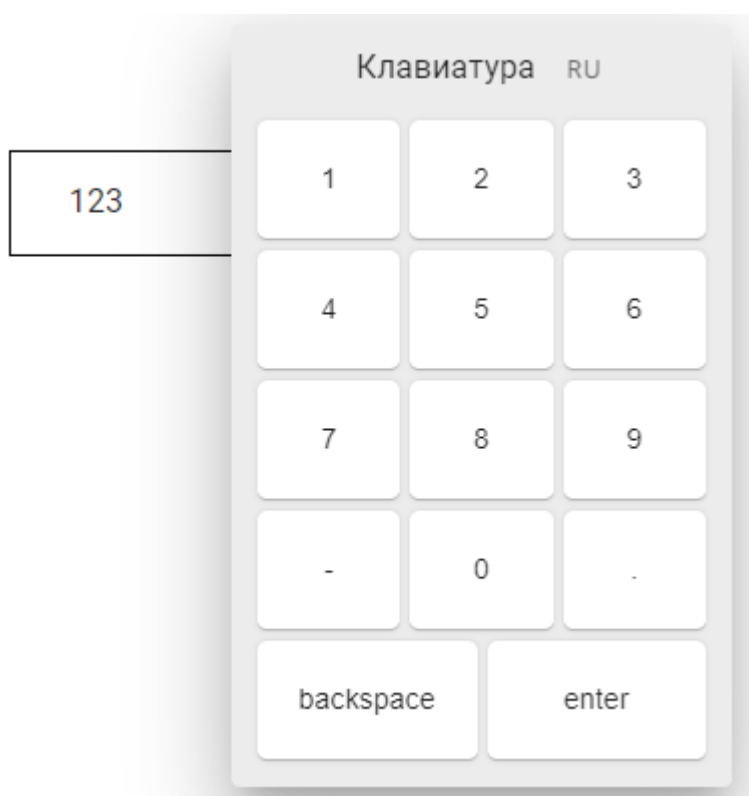
300

- Функция "Только для сенсорных экранов" означает, что клавиатура будет отображаться только на сенсорном экране.
- На выбор доступны 2 темы : тёмная и светлая.
- Функция "Ширина клавиатуры" изменяет ширину клавиатуры в пикселях при задании типа **строка(string)** элемента Input.
- Функция "Ширина цифровой клавиатуры" изменяет ширину клавиатуры в пикселях при задании типа **число(number)** элемента Input.

Текстовая клавиатура :



Цифровая клавиатура :



# Информирование

## Настройка

Создание каналов информирования для учетной записи пользователя.

Правой кнопкой мыши в области информирования добавить новую строку:

Admin ×

Учетная запись Информирование

СОХРАНИТЬ

Адреса для информирования

Тип	Адрес (номер)	Подпись	Отправлять сообщения	Тестовое сообщение
Add				

Pushnotification: зарегистрированные устройства

ID устройства	Модель	Отправлять сообщения	Тестовое сообщение	Токен

### 1. Выбрать канал информирования.

Канал информирования зависит от установленных плагинов информирования (e-mail, telegram). У любого пользователя может быть несколько каналов информирования.

Admin ×

Учетная запись Информирование

СОХРАНИТЬ

Адреса для информирования

Тип	Адрес (номер)	Подпись	Отправлять сообщения	Тестовое сообщение
email	test@myemail.com	My Signature	<input checked="" type="checkbox"/>	Отправить
telegram	213445777	My Signature	<input checked="" type="checkbox"/>	Отправить

Pushnotification: зарегистрированные устройства

ID устройства	Модель	Отправлять сообщения	Тестовое сообщение	Токен

Для E-Mail информирования в поле Адрес (номер) вводится почтовый адрес получателя сообщения.

Для Telegram информирования в поле Адрес (номер) вводится идентификационный номер пользователя Telegram

2. В поле "Подпись" вписать подпись для исходящего сообщения
3. Установить галку "Отправлять сообщения"

Для проверки можно нажать кнопку "Отправить" тестовое сообщение.

## Применение

### Информирование из сценариев

Отправка сообщения конкретному пользователю по его ID через E-Mail:

```
this.info('email', 'admin', 'Текст сообщения')
```

Отправка сообщения конкретному пользователю по его ID через Telegram:

```
this.info('telegram', 'admin', 'Текст сообщения')
```

# Группы пользователей

## Для чего нужны группы.

Группа определяет правила доступа:

- Доступ к экранам пользовательского интерфейса (ко всем или выборочно)
- Доступ к диалогам пользовательского интерфейса (ко всем или выборочно)
- Разрешение управлять устройствами (всеми или выборочно)
- Доступ в РМ (админку): полный доступ / запрет входа / доступ только к выбранным разделам

По умолчанию в системе существует группа **Администраторы**, дающая неограниченный доступ. Можно добавить любое количество групп.

Один пользователь может входить в несколько групп, разрешения при этом объединяются.

## Доступ к UI (Пользовательский интерфейс)

### Экраны

Для доступа к экранам пользовательского интерфейса предусмотрено 3 варианта:

- **Нет ограничений**
- **Только к разрешенным** - нужно добавить в список разрешенные экраны.  
Если в список разрешенных экранов ничего не добавлять, то не будет доступа ни к одному экрану.
- **Ко всем кроме запрещенных** - нужно добавить в список запрещенные экраны.  
Если в список запрещенных экранов ничего не добавлять, то будет доступ ко всем экранам (Вариант 1)

По умолчанию для экранов при создании новой группы выбран вариант **Только к разрешенным**, поэтому нужно добавить список разрешенных экранов либо изменить вариант доступа.

### Диалоги

Для доступа к диалогам (всплывающим окнам) предусмотрено 3 варианта:

- **Нет ограничений**
- **Только к разрешенным** - нужно добавить в список разрешенные диалоги.
- **Ко всем кроме запрещенных** - нужно добавить в список запрещенные диалоги.

По умолчанию для диалогов при создании новой группы выбран вариант **Нет ограничений**

## Правила управления устройствами

Применяются по принципу "Если не разрешено, то запрещено".

- если убрать галку **Разрешить управление всеми устройствами**, а в список групп устройств ничего не добавлять, получаем вариант **Только просмотр**

## Доступ к РМ (Project Manager)

На форме **Доступ к РМ** предусмотрено 3 варианта:

- **Полный доступ** без всяких ограничений
- **Нет доступа** - доступ запрещен на уровне входа в админку
- **Ограниченный доступ** - нужно выбрать разрешенные разделы

При **ограниченном доступе**, кроме списка разделов, есть опция **Разрешить редактирование**. Если редактирование не разрешено, изменения сохраняться не будут.

Доступ к этой форме возможен только для учетной записи, имеющей **Полный доступ** к РМ! Для остальных пользователей будет выведено сообщение: "Недостаточно прав"

Правила начинают работать сразу после сохранения. Никаких перезагрузок не требуется. Не запрещайте доступ к РМ для учетной записи, под которой вошли!

## Группа Администраторы

Группа с идентификатором admgrp является привилегированной группой.

По умолчанию она называется Администраторы. При необходимости, название группы можно изменить.

Группу с идентификатором admgrp удалить нельзя.

Пользователи этой группы имеют полный доступ к системе.

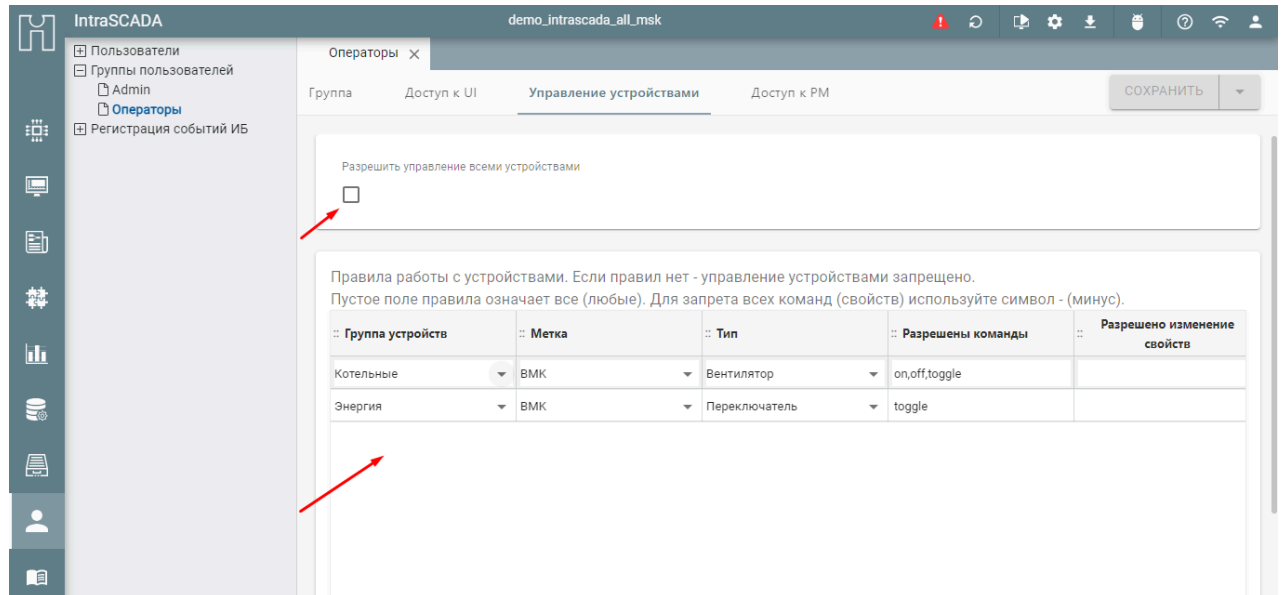
Если пользователю даны все привилегии для доступа к РМ, но он не входит в группу Администраторы, то он не сможет управлять группой Администраторы и пользователями, которые входят в группу Администраторы.

## Управление устройствами

Добавлена вкладка **Управление устройствами** для ограничения доступа к управлению устройствами

- Галка **Разрешить управление всеми устройствами** убирает ограничения на управление устройствами
- При снятии галки, появится окно **Правила работы с устройствами**, на котором вы можете ограничивать управление устройствами для отдельных групп пользователей.





- Можно выбрать какие команды разрешены для данной группы пользователей
  - Возможно выбрать группу устройств, тип устройства
  - Какие команды разрешены
  - Изменение каких свойств разрешено

Если правил нет - управление устройствами запрещено. Пустое поле правила означает все (любые). Для запрета всех команд (свойств) используйте символ - (минус).

## Атрибут userAccess

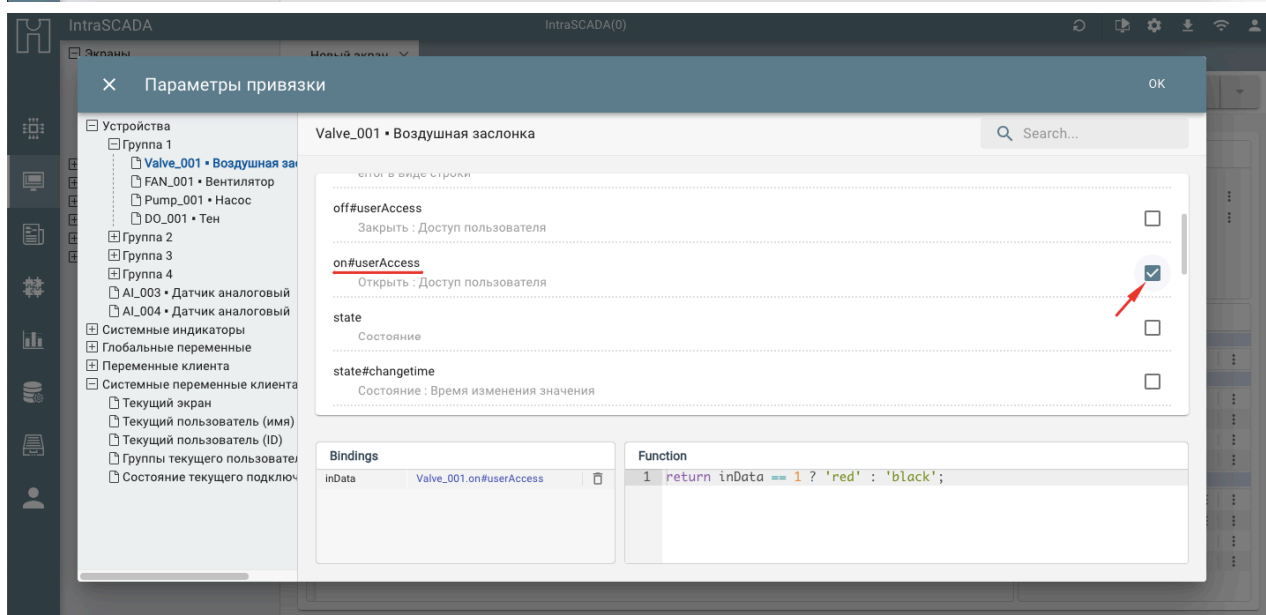
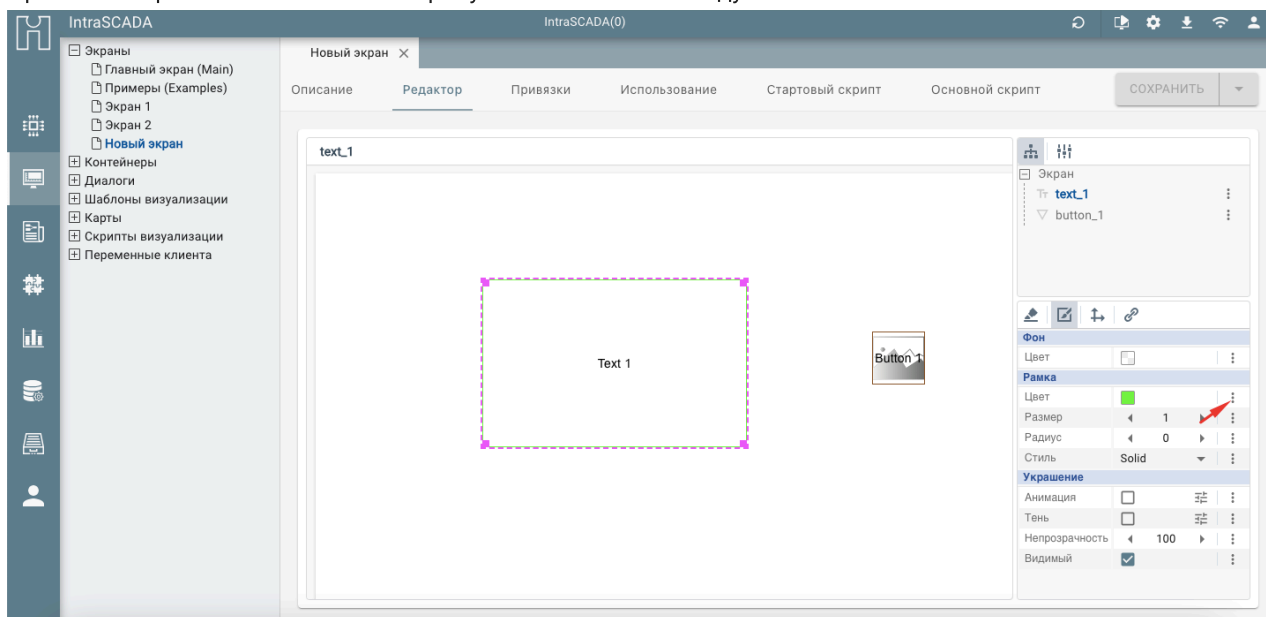
Добавлено: v5.17.64

Добавлен атрибут userAccess для свойств и команд устройств

Данный атрибут показывает, есть ли доступ к свойству или команде у текущего пользователя.

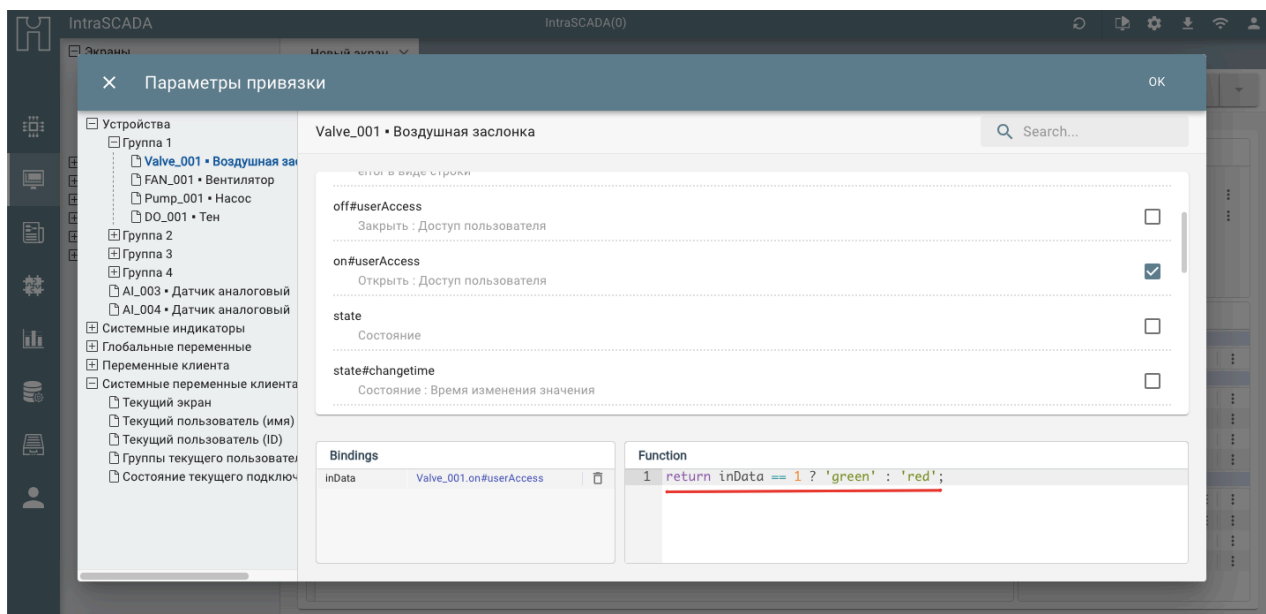
Пример использования:

- Привяжем к рамке элемента Text атрибут **on#userAccess** воздушной заслонки

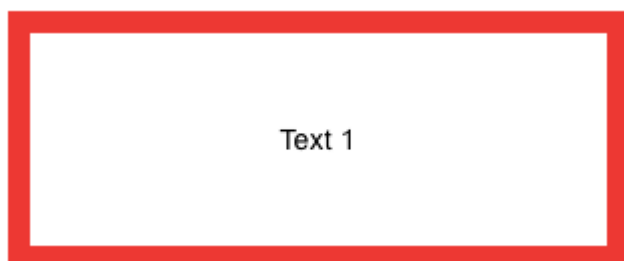


- В поле function пишем функцию

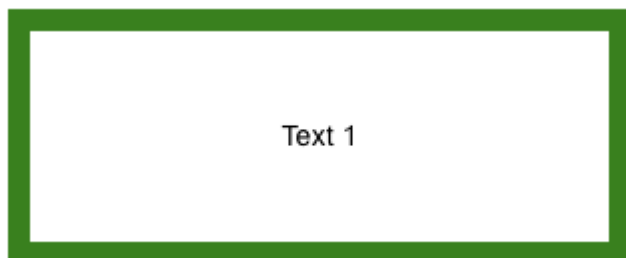
```
return inData == 1 ? 'green' : 'red';
```



- В результате, если доступ к данной команде у пользователя отсутствует - рамка будет красной:



- Если доступ разрешен - зеленой:



## Длина пароля

Добавлено: v5.17.56

По умолчанию минимальная длина пароля составляет:

- 4 символа в обычном режиме
- 8 символов в режиме [Высокий уровень информационной безопасности](#)

Можно задать минимальную длину пароля для группы.

Если задано значение менее 4, оно будет проигнорировано.

Для пользователя, входящего в несколько групп, длина пароля должна быть не менее максимальной из установленных в его группах.

Контроль длины пароля выполняется при вводе (изменении) пароля.

# Текущие подключения

Все текущие подключения к серверу можно увидеть в РМ разделе **Доступ**. Щелкните на узел **Пользователи** и перейдите на вкладку **Текущие подключения**. В таблице выводятся имя пользователя (учетной записи), а также дата и время входа в систему, клиентский IP и клиентское приложение.

Пользователи

Группы пользователей

Журналы ИБ

Пользователи

✕

Таблица

Текущие подключения

⌵	⌵	⌵	⌵
⌵	⌵	⌵	⌵
Operator	89....	02.07.23 22:18:12.028	webuser
Admin	89....	02.07.23 20:02:52.606	webadmin

# Интеграция с LDAP

Добавлено v5.17.29

Для интеграции с LDAP необходима установка плагина [LDAP client](#).

Интеграция обеспечивает вход в IntraSCADA с использованием учетных данных, хранящихся в вашем Active Directory / другом каталоге на основе LDAP.

LDAP позволяет:

- пользователям - использовать единую учетную запись для доступа к сервисам компании;
- администраторам
  - избежать необходимости отслеживания актуальности учётных данных в каждом отдельном сервисе,
  - оперативно удалять из системы пользователей, которым доступ к сервисам больше не предоставляется.

## Два типа учетных записей

IntraSCADA поддерживает ведение двух типов учетных записей:

- внутрисистемные учетные записи IntraSCADA
  - добавляются, редактируются, удаляются администратором системы в разделе Доступ
  - аутентификация происходит встроенными в IntraSCADA механизмами
- учетные записи LDAP
  - Загружаются с сервера LDAP операцией **Импорт пользователей с сервера LDAP**
  - аутентификация происходит через механизмы LDAP

## Загрузка данных

### Настройка плагина

Предварительно в плагине [LDAP client](#) следует настроить правила для импорта:

- пути и фильтры, чтобы в систему были загружены только нужные группы и учетные записи
- маппинг атрибутов групп и учетных записей

Эти настройки зависят от используемого LDAP сервера и структуры вашего LDAP каталога.

### Механизм интеграции

IntraSCADA через [LDAP client](#) импортирует данные с LDAP сервера, согласно правилам, созданным при настройке.

В разделах **Группы** и **Пользователи** создаются папки **LDAP**, в которые помещаются загружаемые записи. Загружаемые записи имеют пометку **Группа LDAP** или **Учетная запись LDAP**.

При повторном импорте происходит синхронизация групп и учетных записей IntraSCADA с пометкой **LDAP** с актуальными данными LDAP сервера :

- новые - добавляются;
- существующие - корректируются;
- более не существующие - удаляются.

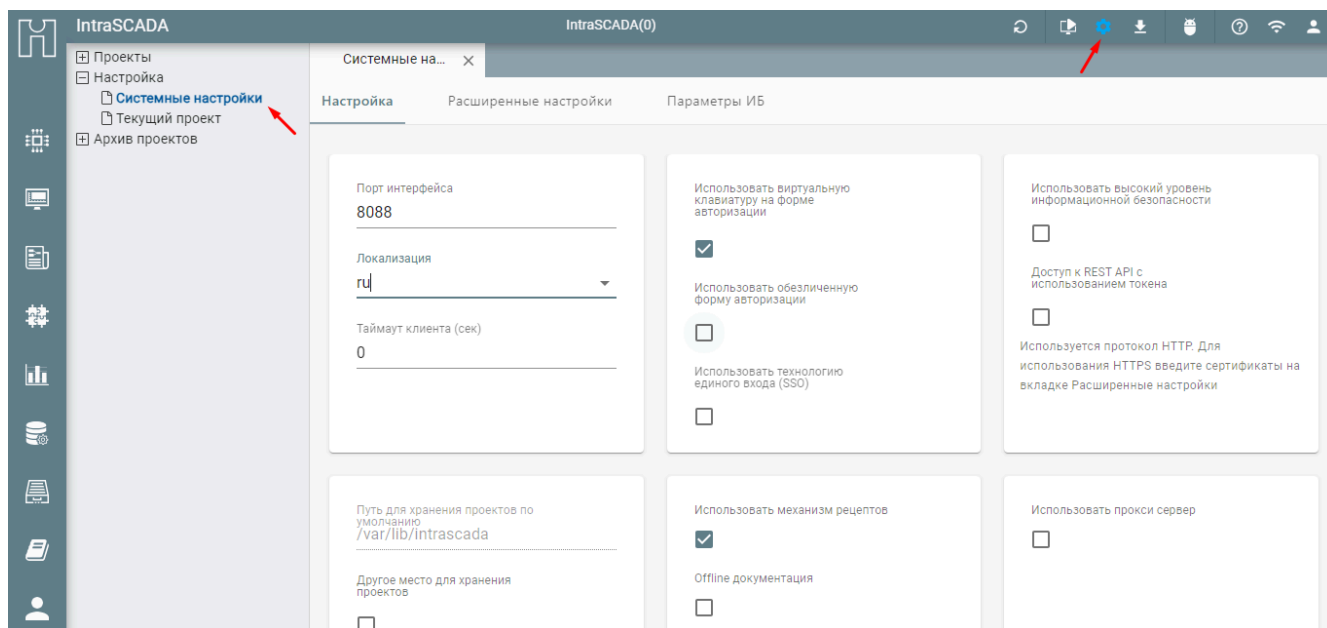
## Запуск процедуры

Процедура импорта может быть запущена:

- вручную в меню плагина [LDAP client](#)
  - **Импорт групп с сервера LDAP**
  - **Импорт пользователей с сервера LDAP**
- по расписанию - с заданным интервалом или в заданное время.  
Для этого нужно создать сценарий, который будет отправлять команды плагину [LDAP client](#).

Пример сценария **Синхронизация с LDAP** есть в разделе [LDAP client - Команды плагина](#).

## Системные настройки



Доступ к системным настройкам осуществляется по клику на шестеренку в правом верхнем углу дашборда.

На данной странице настраиваются параметры, общие для всех проектов.

Настройки хранятся в файле **config.json** в рабочей папке запущенной службы (сервера).

Сервер считывает настройки при старте.

## Общие настройки

- Порт интерфейса
- Локализация (язык системы)
- Таймаут клиента (рекомендуемое время от 10 до 30 сек)
  - Если таймаут установлен, сервер и клиент будут периодически пинговать друг друга для оперативного определения, что связь не нарушена. Через заданное время при отсутствии ответа сервер сбрасывает соединение и убирает клиента из таблицы [Текущих подключений](#).
  - Если таймаут установлен в 0, пинги не выполняются.

Установка таймаута позволяет:

- клиенту - оперативно определить потерю связи с сервером;
- серверу - закрыть подвисшие соединения при нештатном завершении сеанса со стороны клиента.

## Параметры формы авторизации

- Использование виртуальной клавиатуры на форме авторизации.  
Дает возможность включить виртуальную клавиатуру при авторизации в интерфейс пользователя





## Авторизация

Пользователь



Пароль



☐ Запомнить меня

ВХОД

- Использование обезличенной формы авторизации.  
При установке галочки заставка IntraSCADA и версия системы убираются при авторизации



## Авторизация

Пользователь



Пароль



☐ Запомнить меня

ВХОД

## Параметры безопасности

- [Использование высокого уровня информационной безопасности](#)
- [Доступ к REST API с использованием токена](#)
- Использование технологии единого входа (SSO)
  - SSO — технология, при которой пользователь входит в систему без использования формы авторизации.  
Требуется установка дополнительных плагинов (ldapclient, kerberos)

## Путь для хранения проектов

Вы можете указать свой путь для хранения проектов

## Подключение дополнительных возможностей

- [Использовать механизм рецептов](#)
- [Offline документация](#)

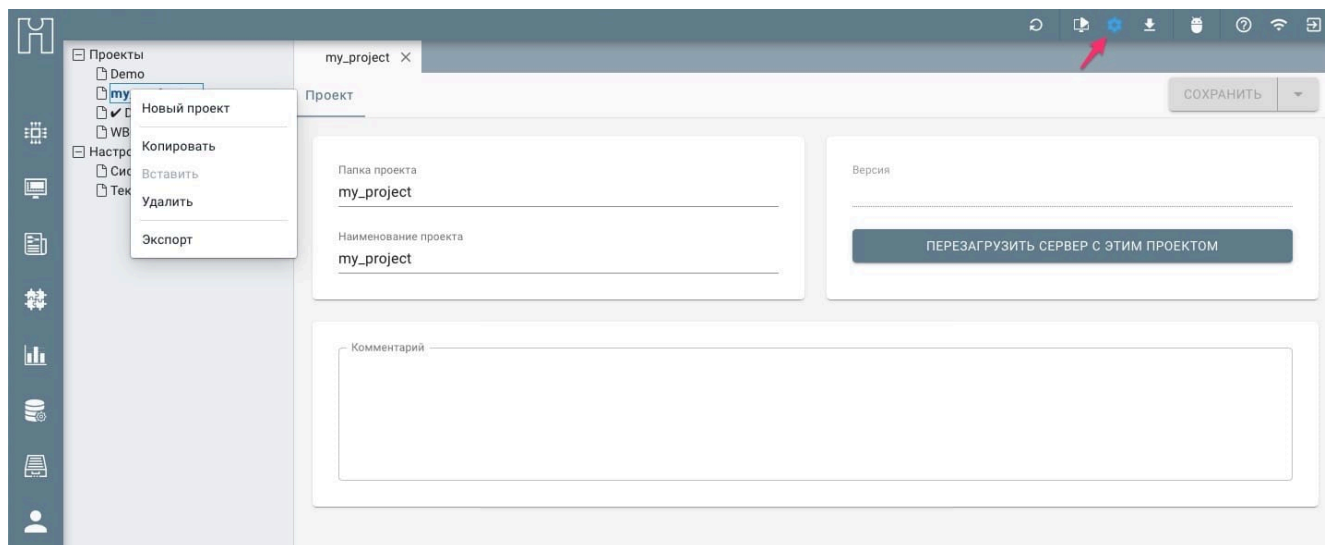
## Использование прокси сервера

Для использования прокси сервера установите галочку и настройте параметры прокси.

Обратите внимание: чтобы изменения вступили в силу, требуется перезагрузить сервер.

# Работа с проектами

Работа с проектами выполняется в разделе "Настройки сервера":



## Дерево проектов

Система позволяет иметь несколько проектов. Это может быть полезно для инсталляторов. Переключение между проектами выполняется выбором нужного проекта и нажатием кнопки "Перезагрузить сервер с этим проектом"

По правой кнопке мыши в дереве проектов доступны операции:

- Новый проект
  - Создание нового пустого проекта
- Копировать
- Вставить
  - Создание копии существующего проекта
- Удалить
  - Удаление проекта с сервера
- Экспорт
  - Сохранение проекта в виде архива на свой компьютер

## Загрузка проекта

Загрузка проекта выполняется кнопкой "Импорт" в верхней инструментальной панели: 

После загрузки проект появится в дереве проектов.

## Перенос проекта

Механизмами экспорта и импорта можно воспользоваться для переноса проекта на другой сервер. Для этого необходимо выполнить два действия:

1. Экспортировать проект с сервера на свой компьютер
2. Импортировать проект со своего компьютера на новый сервер кнопкой "Импорт".

Если вы используете базу данных SQLite, то для переноса исторических данных нужно скопировать две папки с данными для графиков и журналы, **db** и **logdb** соответственно. При этом нужно либо переключиться на другой проект, либо отстоявить сервис IntraScada, чтобы система закрыла файлы БД.

## Версия системы и версия проекта

В наших продуктах используется семантическое версионирование. Версия системы состоит из трех чисел, разделяемых точкой:

<номер МАЖОРНОЙ версии>.<номер МИНОРНОЙ версии>.<номер ПАТЧА>

Данная документация содержит описание пятой версии системы, то есть мажорная версия равна 5.

Изменение минорной версии системы обычно сопровождается необходимостью выполнить какое-либо преобразование проекта.

Преобразование выполняется автоматически при активизации проекта, если произошло обновление системы (или был загружен проект, который работал с более старой версией системы).

После преобразования проекта в файле project.json фиксируется его новая версия.

Поскольку при изменении последней цифры версии системы (номер ПАТЧА) никакого изменения проекта не происходит, номер версии проекта состоит из двух чисел, соответствующих мажорной и минорной версии.

Проекты совместимы с системой **снизу вверх**, то есть всегда возможна активизация проекта более младшей версии, причем можно загрузить проект с минорной версией, отстающей на несколько шагов.

Например, можно загрузить проект v5.9 в систему v5.16.12.

В таких случаях сервер применит последовательно все преобразования, чтобы в результате обновить проект до v5.16.

Но сверху вниз это не работает!! Не следует выполнять запуск обновленного проекта с более старой версией системы.

## Архив проектов

*Добавлено v5.16.16*

Начиная с v5.16.16, при преобразовании проекта, связанном с повышением версии, исходный проект автоматически сохраняется на сервере в Архиве проектов.

Если по какой-либо причине был выполнен откат системы, вы сможете легко восстановить проект. Для этого в

дереве **Архив проектов** выберите нужный архив и нажмите кнопку **Восстановить проект из архива**. Сервер сообщит, в какую папку дерева проектов был восстановлен проект.

Для архивных проектов доступны операции **Экспорт** и **Удалить**.

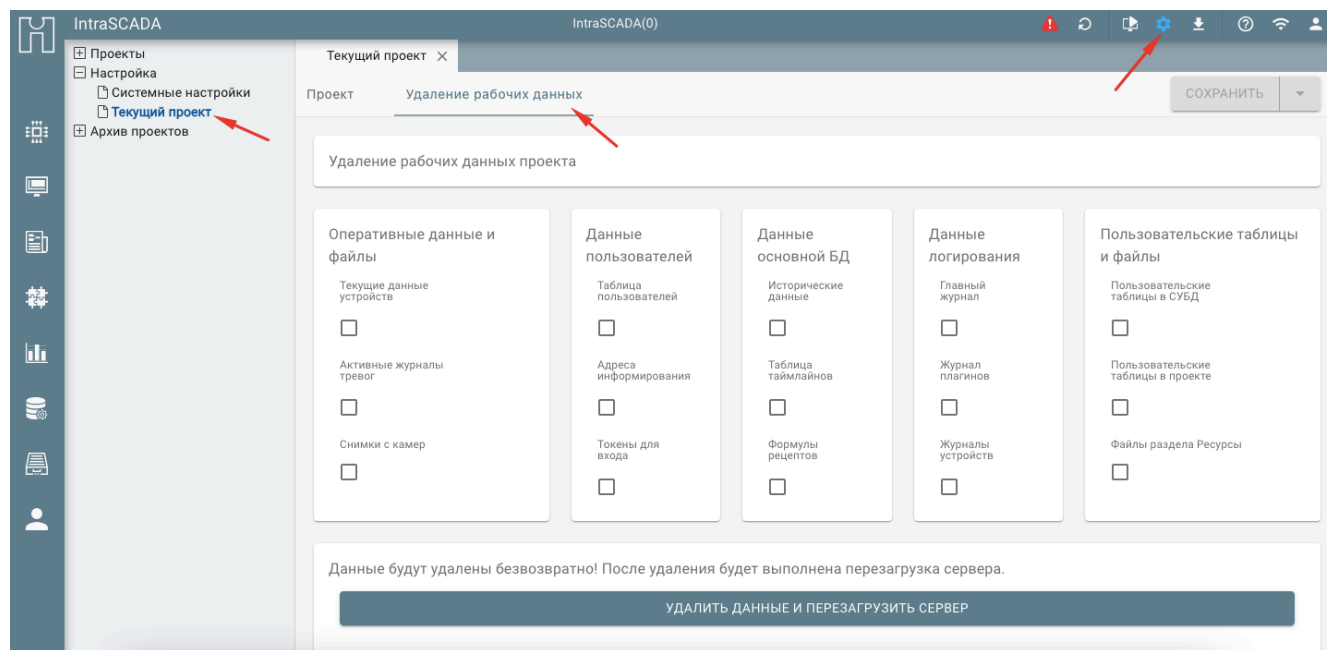
Можно также создать архивную копию любого проекта из дерева проектов вручную.

## Удаление рабочих данных проекта

Эта процедура предназначена для:

- Полной или частичной очистки данных при тестировании, миграции или устранении ошибок.
- Оптимизации хранилища без потери критически важной информации.

Важно: Удаление данных необратимо. Перед выполнением создайте резервные копии через раздел "Проекты > Экспорт". После удаления (полного или частичного) сервер перезагрузится для применения изменений. Доступ к разделу требует роли администратора. Раздел расположен в административной панели: "Настройки сервера > Настройка > Текущий проект > Удаление рабочих данных". В разделе поддерживается выборочное удаление: вы можете отметить только нужные категории данных с помощью чекбоксов, вместо удаления всего сразу. Это позволяет гибко управлять очисткой, минимизируя риски.



## Удаляемые элементы

Данные сгруппированы по категориям. В интерфейсе раздела каждая категория и подкатегория отображается как отдельный пункт для выбора. При выборочном удалении система удалит только отмеченные элементы.

Оперативные данные и файлы:

- Текущие данные устройств
- Активные журналы тревог
- Снимки с камер

Данные пользователей:

- Таблица пользователей

- Адреса информирования
- Токены для входа

Данные основной БД:

- Исторические данные
- Таблица таймлайнов
- Формулы рецептов

Данные логирования:

- Главный журнал
- Журнал плагинов
- Журналы устройств

Пользовательские таблицы и файлы:

- Пользовательские таблицы в СУБД
- Пользовательские таблицы в проекте
- Файлы раздела "Ресурсы"

## Шаги выполнения процедуры

Подготовка:

- Войдите как администратор.
- Экспортируйте проект или отдельные данные для резервной копии.

Запуск удаления:

- Перейдите в "Настройки > Текущий проект > Удаление рабочих данных".
- Отметьте чекбоксами нужные категории/элементы для удаления (для полного — отметьте все).
- Прочитайте предупреждение: "Данные будут удалены безвозвратно! После удаления будет выполнена перезагрузка сервера."
- Нажмите "Удалить" и подтвердите.

## Механизм резервирования серверов

Предусматривает одновременную работу двух серверов IntraSCADA с одним проектом.

При этом один из серверов работает в режиме MASTER как обычный сервер IntraSCADA (обслуживает клиентов, все плагины запущены), а второй в режиме SLAVE (общается только с MASTER-ом, получает оперативные данные и синхронизирует проект).

Для организации резервирования необходимы лицензии на оба сервера.

### Синхронизация проекта

При резервировании выполняется синхронизация проекта и оперативных данных.

Проект, запущенный на MASTER-е, будет автоматически синхронизироваться на SLAVE-е.

### Синхронизация БД

Механизм не предполагает автоматическую синхронизацию БД (исторические данные и журналы, пользовательские таблицы в СУБД).

Необходимо настроить репликацию с учетом схемы размещения серверов СУБД.

Механизм резервирования не работает для конфигураций, использующих встроенную СУБД SQLite.

### Резервирование на уровне клиентов

При использовании десктопного приложения **IntraSCADA Client** предусмотрена возможность бесшовной работы с резервируемыми серверами.

При потере связи с одним из серверов приложение автоматически выполнит подключение к другому.

На клиенте нужно **настроить раздел Резервирование**: прописать IP адреса, временные параметры и приоритет подключения.

### Переключение MASTER/SLAVE

На старте сервер, использующий механизм резервирования, выполняет поиск дублера.

В ситуации, когда оба сервера работоспособны и перезагружаются одновременно, MASTER-ом станет тот, кто быстрее пройдет процедуру поиска дублера.

Второй сервер запустится в режиме SLAVE:

- получит от MASTER-а путь к рабочему проекту
- синхронизирует или закачает проект полностью, если его еще нет на втором сервере
- будет получать оперативные данные от MASTER-а
- автоматически начнет работать в режиме MASTER, если MASTER перестанет отвечать.

Таким образом, любой из серверов может работать в режиме MASTER.

Чтобы при прочих равных условиях один из хостов имел приоритет как MASTER, при [настройке](#) установите для него меньшее **Число запросов на поиск мастера**.

## Логирование в режиме SLAVE

Обратите внимание, web-интерфейс сервера в режиме SLAVE становится не доступен.

Логи пишутся в тот же файл, что и в режиме MASTER (/var/log/intrascada/ih.log)

Пример лога сервера в режиме SLAVE:

```
10.08 10:43:15.947 INFO: Server has started in Master/Slave mode.  
    Config file /etc/intrascada/config.json  
10.08 10:43:16.956 INFO: SEEK_MASTER result: MASTER_OK probe = 0  
10.08 10:43:16.957 WARN: SEEK_MASTER result: MASTER_OK  
10.08 10:43:16.960 INFO: WS message {  
    "currentProject": "/var/lib/intrascada/projects/project_1723707196979"  
}
```

В ответ на запрос SEEK\_MASTER сервер получает ответ MASTER\_OK и путь к текущему проекту.

## Алгоритм при потере связи между серверами

При потере связи между серверами каждый может запуститься как MASTER.

При восстановлении связи в ситуации двух мастеров сервис, имеющий более позднее время старта, выполнит перезагрузку, чтобы стартовать как SLAVE.



# Настройка резервирования

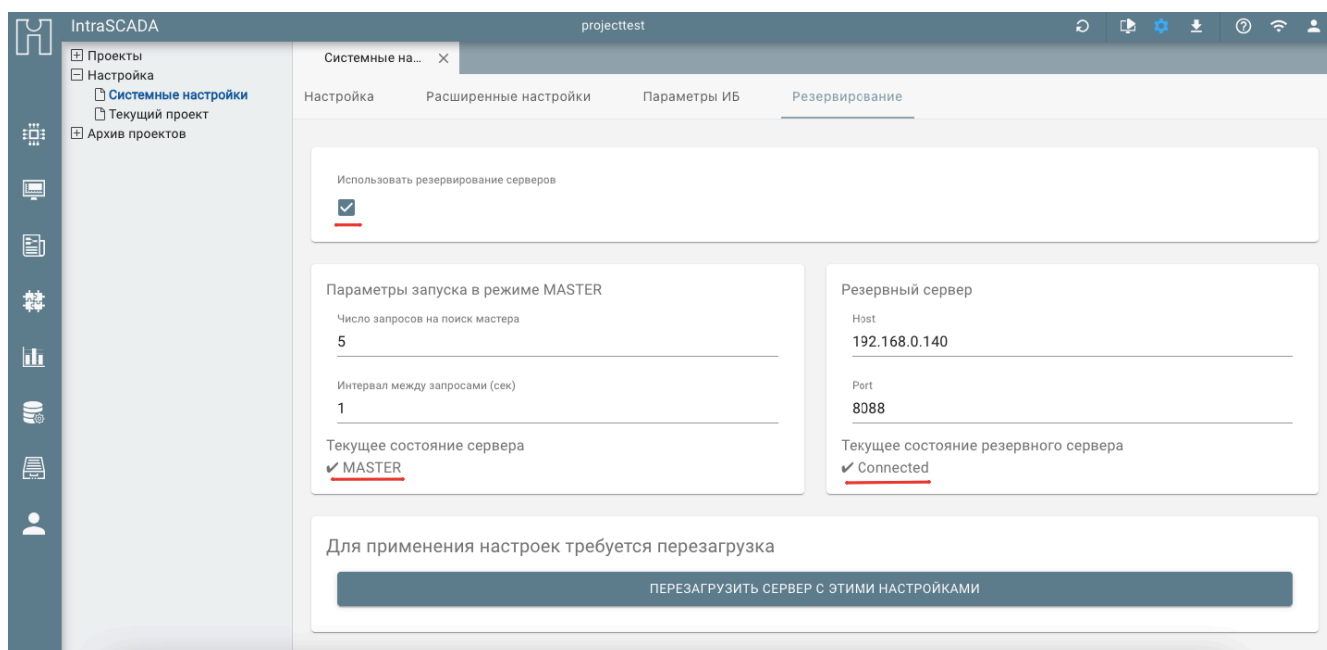
Добавлено в v5.17.44

## Системные настройки

Настройка механизма резервирования выполняется в разделе **Системные настройки** на вкладке **Резервирование**. После выбора опции **Использовать резервирование серверов** будут доступны плашки для ввода параметров.

Обязательные опции - адрес и порт резервного сервера. Порт сервера тот же, что и для клиентских подключений (по умолчанию 8088).

Порт используется для TCP и UDP подключений.



Эти настройки нужно сделать на обоих хостах.

## Пример запуска проекта на двух серверах (Master/Slave)

1. На обеих машинах установить IntraSCADA одинаковых версий.
2. На Master развернуть проект. Указать ip адрес/порт слейва. Перезагрузиться.
3. На Slave указать ip адрес/порт мастера. Перезагрузиться.
4. На Master в параметрах резервирования появится наличие соединения со Slave (Connected).
5. Веб интерфейс Slave теперь будет не доступен.
6. Если надо переключиться на Slave, то необходимо остановить службу intrascada на Master (sudo service intrascada stop). Slave через время, которое было установлено в настройках резервирования станет Master и запуститься веб интерфейс.

## Дополнительные зависимости

Для синхронизации проекта используется стандартная утилита **rsync**.

## rsync для Windows

Для Windows пакет **rsync** (реализация **cygwin**) включен в комплект поставки (папка Program Files/IntraSCADA/tools).

## rsync для Linux

Новые дистрибутивы Linux обычно содержат **rsync**. Проверьте наличие пакета, если нет - установите пакет rsync стандартным для ОС способом.

Больше никаких настроек делать не надо. IntraSCADA использует протокол rsync поверх TCP, порт 8873.

- SLAVE запускает rsync для удаленного копирования с MASTER-а
- MASTER запускает rsync в режиме службы (отображается в системных процессах как rsyncdaemon).  
Для процесса доступен лог

## Конфигурационный файл

Конфигурационный файл для службы создается автоматически:

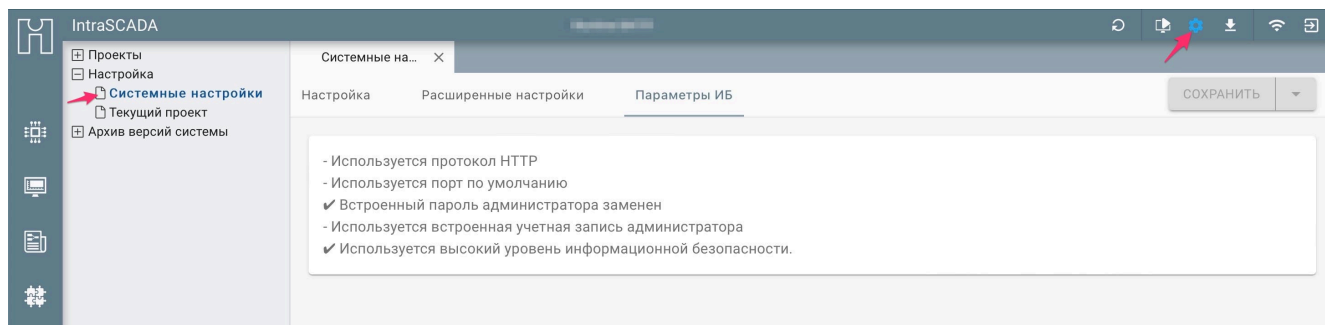
- Windows: C:\ProgramData\IntraSCADA\intrascada\rsyncd.conf
- Linux: /var/lib/intrascada/rsyncd.conf

При необходимости вы можете отредактировать файл rsyncd.conf, например, разрешить вход только с определенного IP/хоста (hosts allow). Не следует изменять имя модуля [intrascada] и путь к папке с проектами path

```
[intrascada]
path = /var/lib/intrascada/projects
read only = false
hosts allow = *
list = true
```

# Сводная информация по ИБ

Сводные параметры информационной безопасности приведены на вкладке "Параметры ИБ" в системных настройках:



Символ слева для каждого параметра означает:

- Установлен параметр по умолчанию
- ✓ Параметр по умолчанию изменен для реализации функции безопасности

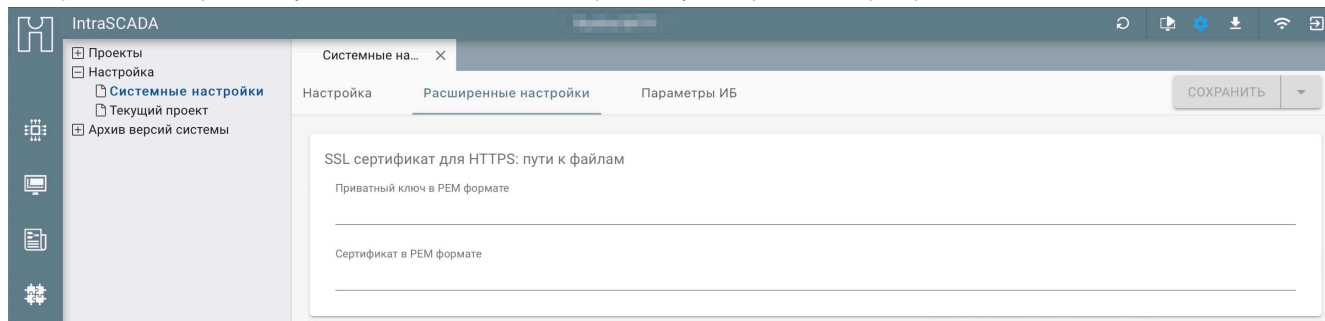
## Протокол HTTP/HTTPS

Веб интерфейс системы может работать по протоколам HTTP и HTTPS

HTTP — протокол прикладного уровня передачи данных

HTTPS — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности

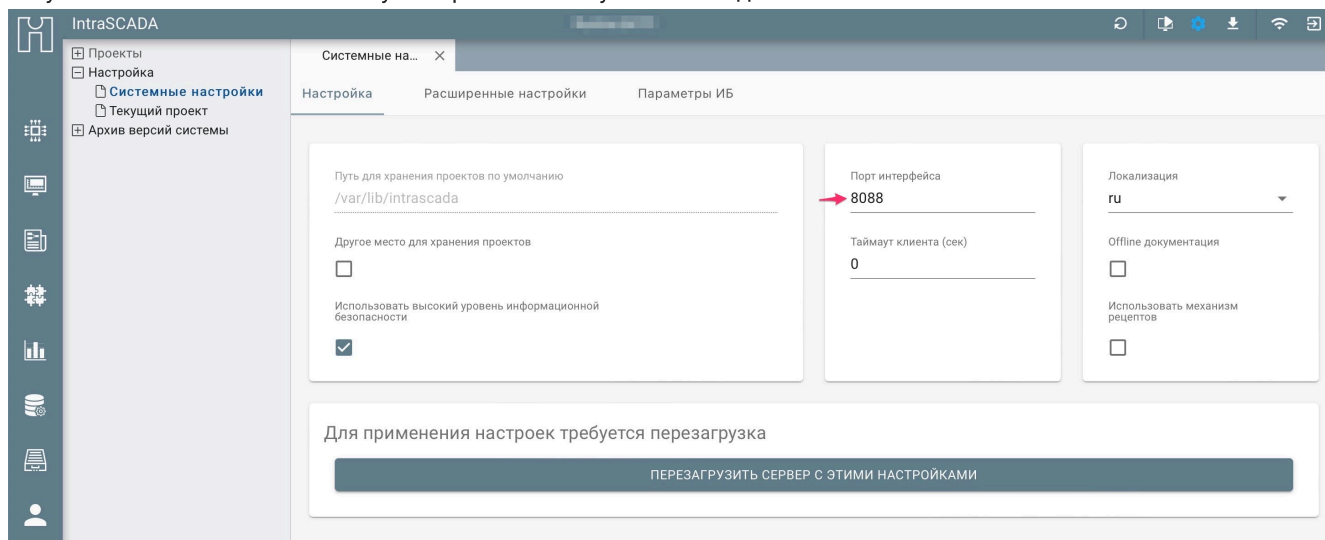
Для работы по протоколу HTTPS необходимо настроить пути к файлам сертификатов:



[Инструкция по генерации самоподписных сертификатов](#)

## Порт по умолчанию

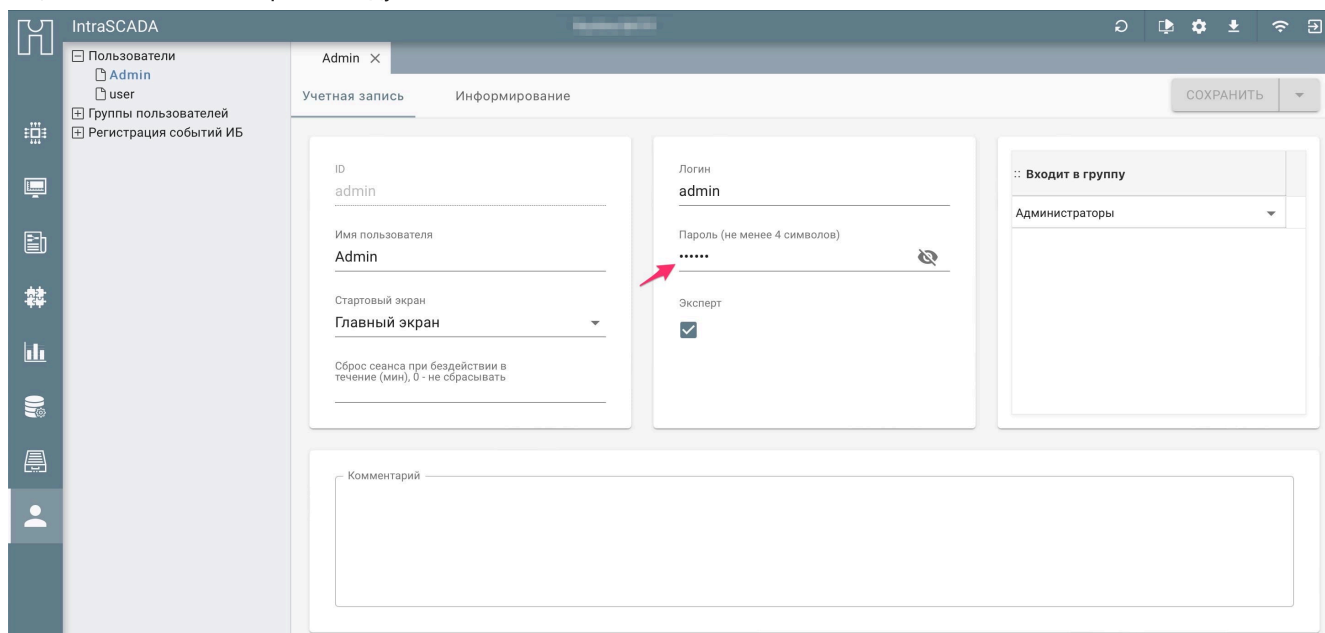
По умолчанию система использует порт 8088. В случае необходимости его можно изменить:



## Пароль администратора

По умолчанию для администратора используется пароль 202020.

В целях безопасности рекомендуется изменить его:

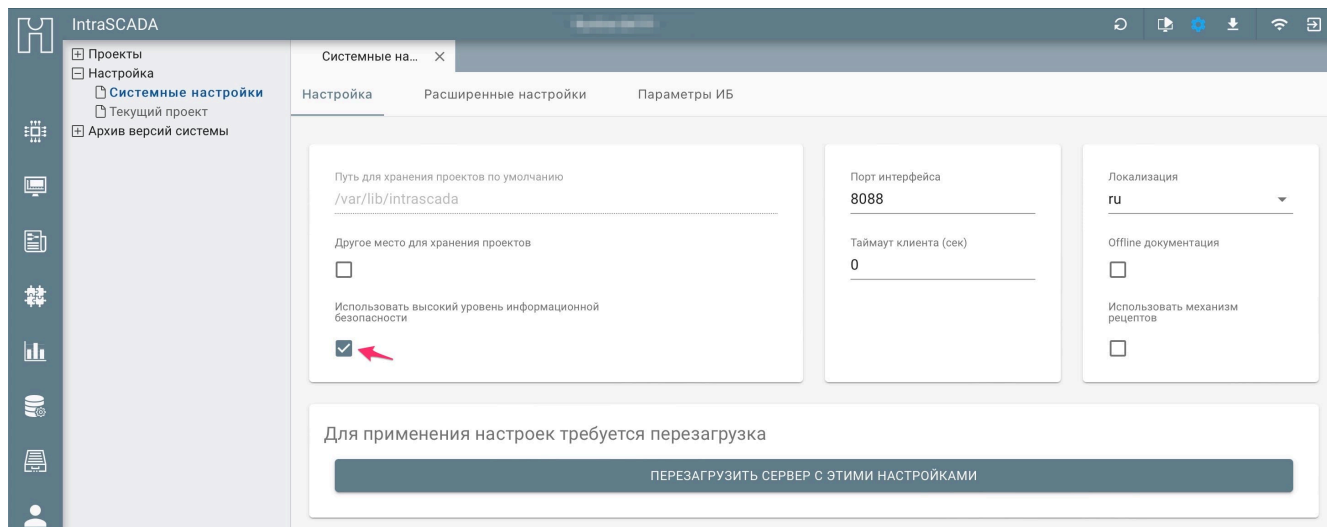


## Встроенная учетная запись администратора

По умолчанию в системе есть встроенная учетная запись администратора.

В целях безопасности ее можно удалить. Перед удалением необходимо создать для администратора новую учетную запись.

## Высокий уровень информационной безопасности



После установки галки и перезагрузки система переходит в режим **"Высокого уровня информационной безопасности"**

В этом режиме блокируется выход к онлайн ресурсам и доступ по P2P, предъявляются более высокие требования к паролям, разрешается только один вход для одной учетной записи.

## Для доступа к REST API используется токен

При установке в **Системных настройках** галки **Доступ к REST API с использованием токена** перед запуском обработчика сервер будет проверять наличие токена в заголовке запроса.

Описание механизма получения и использования токена описано в [REST API -> Доступ](#)

## Шифрование паролей

IntraSCADA выполняет хранение и передачу паролей только в зашифрованном виде (SHA256/AES256).

# Порты по умолчанию

В своей работе IntraSCADA использует следующие порты по умолчанию:

Порт	Назначение	Протокол	Примечание
8088	Порт для входа в пользовательский и административный интерфейсы	HTTP и HTTPS	v
502	Для плагина Modbus/TCP Server	Modbus/TCP	o
1883	Для плагина MQTT Server	MQTT	o
8883	Для плагина MQTT Server	MQTTS	o

Примечание:

v - порт открыт всегда

o - порт необходим для работы соответствующего плагина

Все адреса портов можно изменить в настройках системы или плагина.

# Регистрация событий ИБ

Система выполняет регистрацию событий информационной безопасности.

Предусмотрено два опциональных варианта регистрации: в СУБД и в лог.

При необходимости можно использовать оба варианта.

Настройка выполняется на вкладке Доступ - Регистрация событий ИБ.

## Регистрируемые события

- Вход в систему/ Выход из системы
- Неуспешная попытка входа
- Действия с учетными записями пользователей:
  - добавление учетной записи
  - изменение параметров учетной записи
  - изменение пароля
  - добавление пользователя в группу
  - исключение пользователя из группы
  - удаление учетной записи
- Действия с группами пользователей:
  - добавление группы
  - изменение параметров группы (настройки доступа к UI и PM)
  - удаление группы

Также фиксируются важные системные события - старт/стоп сервера, изменение проекта, ...

## Регистрация событий ИБ в СУБД

Система IntraSCADA использует СУБД для логирования системных и прикладных событий.

Выбор СУБД (SQLite, PostgreSQL, MySQL) для конкретного проекта выполняется в разделе [База данных](#). Для событий ИБ существует отдельная таблица.

Период хранения событий определяется настройкой, старые записи удаляются.

Сохранение событий ИБ в СУБД позволяет создавать журналы ИБ и выводить их на экранах проекта с real-time обновлением.

## Интерфейс для работы с данными

Механизм создания интерфейсов для работы с данными ИБ совпадает с основными механизмами журналирования, принятыми в системе - на базе регистрируемых данных с применением фильтров может быть построено множество журналов.

Журналы формируются в разделе Доступ - Журналы ИБ.

При создании журнала можно задать фильтры и набор столбцов для отображения на интерфейсе.

Далее журналы привязываются к виджету Журнал и размещаются на экранах, доступ к которым регулируется

ролевыми настройками доступа. При необходимости может быть создана группа, имеющая эксклюзивный доступ к журналам.

Дата	Текст сообщения	Субь...	Объект	Изменения
02.07.2023 20:35:15	Вход в систему	admin		
02.07.2023 20:35:14	Штатное завершение сеанса. Начало сеанса 02.0...	admin		
02.07.2023 20:35:11	Пользователь добавлен в группу Супергруппа 1	admin	Operator	
02.07.2023 20:34:12	Вход в систему	admin		
02.07.2023 20:34:11	Штатное завершение сеанса. Начало сеанса 02.0...	admin		
02.07.2023 20:34:09	Изменена группа	admin	Супергруппа 1	Доступ к РМ: "Полный доступ" => "Ограниченный доступ"

## Регистрация событий ИБ в логе

Система предоставляет опциональную возможность регистрировать события ИБ в лог.

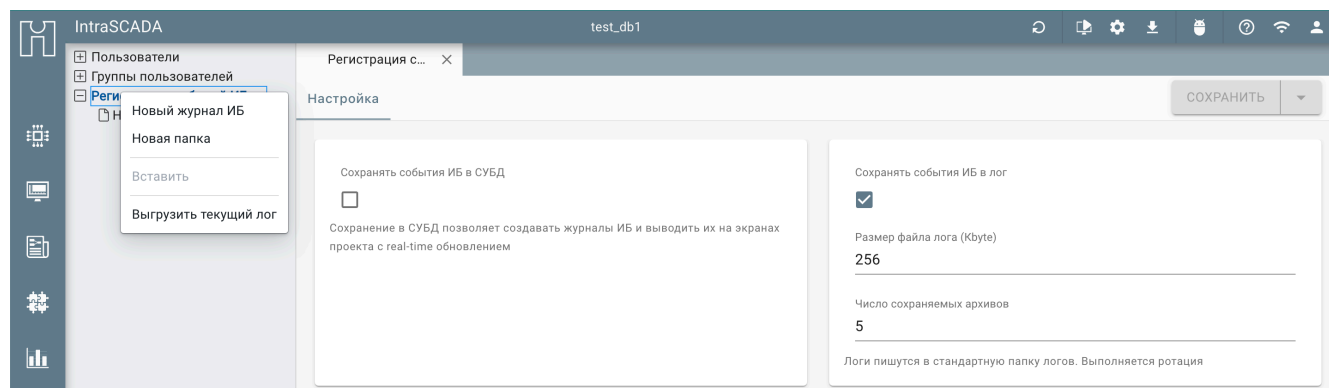
Лог isec.log пишется в стандартную папку логов системы (по умолчанию для систем на базе Linux папка /var/log/intrascada) Можно задать размер лога в килобайтах. При достижении заданного размера создается новый файл, а предыдущий переименовывается.

Выполняется ротация: по умолчанию хранится 5 архивных файлов, старые удаляются встроенными механизмами системы.

## Выгрузка логов ИБ

Логи ИБ можно выгрузить с помощью встроенного механизма выгрузки логов, либо получить к ним доступ через сервисы FTP, SCP, HTTP операционной системы. По умолчанию для систем на базе Linux папка логов ИБ находится по пути /var/log/intrascada.

Для того, чтобы выгрузить логи через интерфейс системы, необходимо получить доступ к интерфейсу разработчика и перейти в раздел **Доступ**. В появившемся дереве, необходимо нажать правой кнопкой мыши на папку **Регистрация событий ИБ** и в выпадающем меню выбрать пункт **Выгрузить текущий лог**.





# Типовые инциденты

## 1. Попытка пользовательского входа с неверным паролем.

При входе в систему требуется ввести логин и пароль.

Если пользователь в течении пяти минут вводит три раза неверный пароль, доступ пользователя блокируется.

Разблокировать аккаунт может только администратор с соответствующими полномочиями.

## 2. Попытка административного входа с неверным паролем.

Если пользователь с правами администратора в течении пяти минут вводит три раза неверный пароль, его доступ в систему блокируется.

Разблокировка администратора возможна только с помощью перезагрузки системы.

# Резервное копирование проекта

Система IntraSCADA обеспечивает различные варианты резервного копирования.

При резервном копировании создается zip архивный файл с расширением ihpack. Этот файл содержит все элементы проекта.

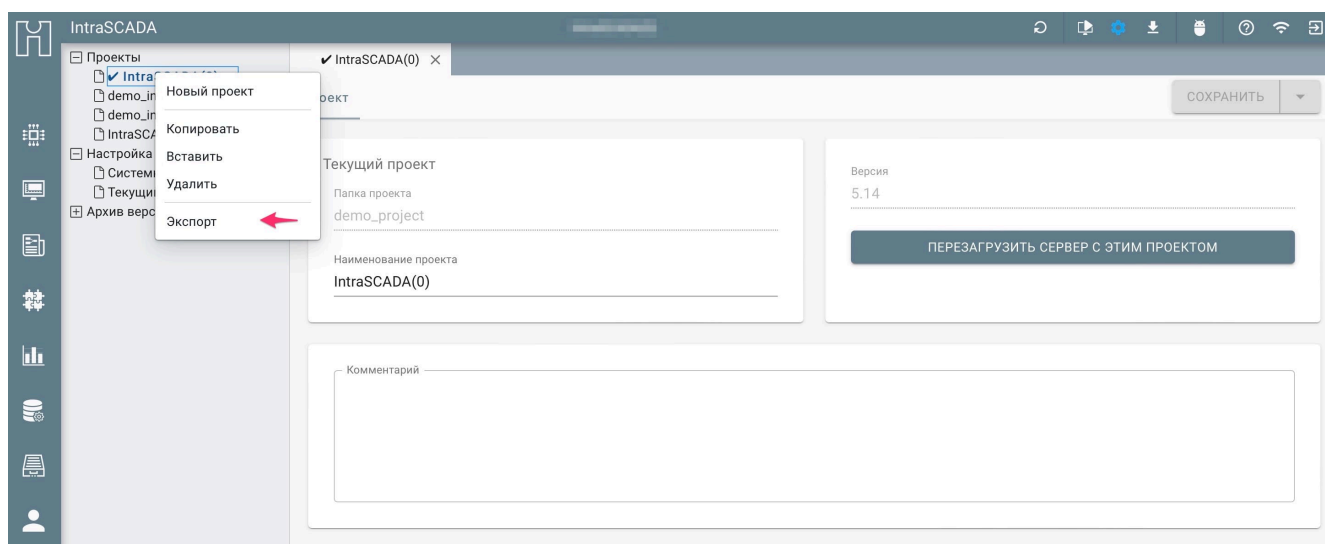
Следующие компоненты принадлежат самой системе IntraSCADA и не включаются в архив:

- Плагины
- База данных с историческими данными
- Конфигурационный файл с настройками порта
- Файлы лицензий

## Вариант 1

Резервное копирование на свой локальный компьютер

### Создание резервной копии

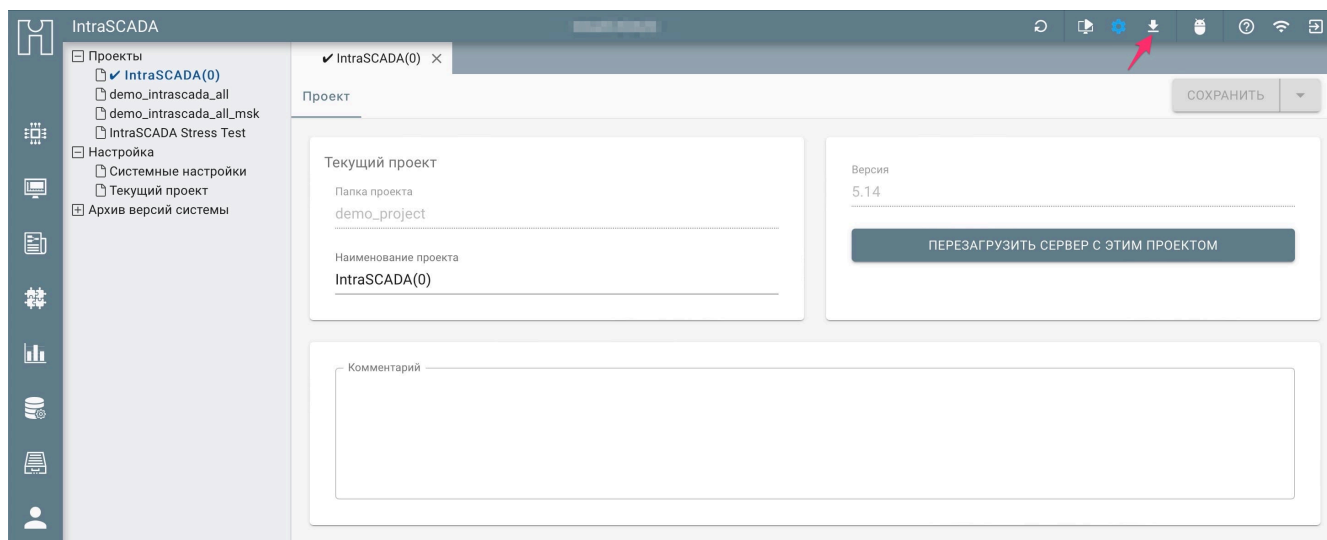


Для создания резервной копии проекта:

1. Правой кнопкой мыши на названии проекта выбрать "Экспорт"

На ваш компьютер будет загружен архивный zip файл с расширением ihpack.

### Восстановление резервной копии

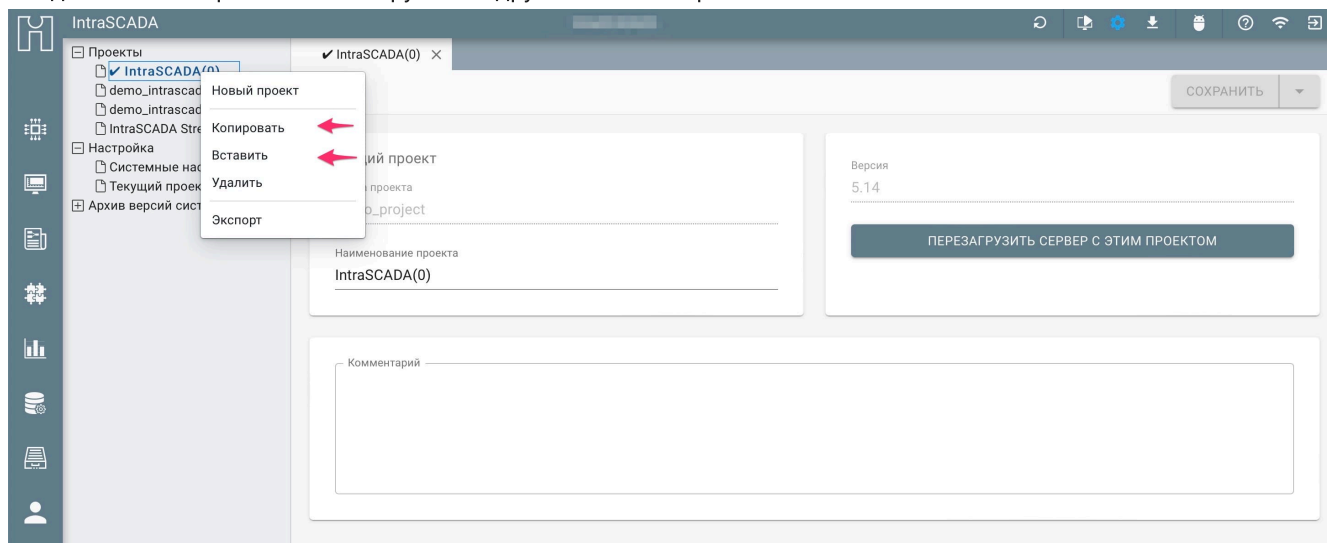


Для восстановления резервной копии проекта:

1. Нажать кнопку "Импорт" и загрузить архивный файл проекта с расширением ihpasc
2. В дереве проектов встать на загруженный проект и нажать кнопку "Перезагрузить сервер с этим проектом"

## Вариант 2

Создание копии проекта без выгрузки на другой компьютер.



Для копирования проекта на компьютере с системой:

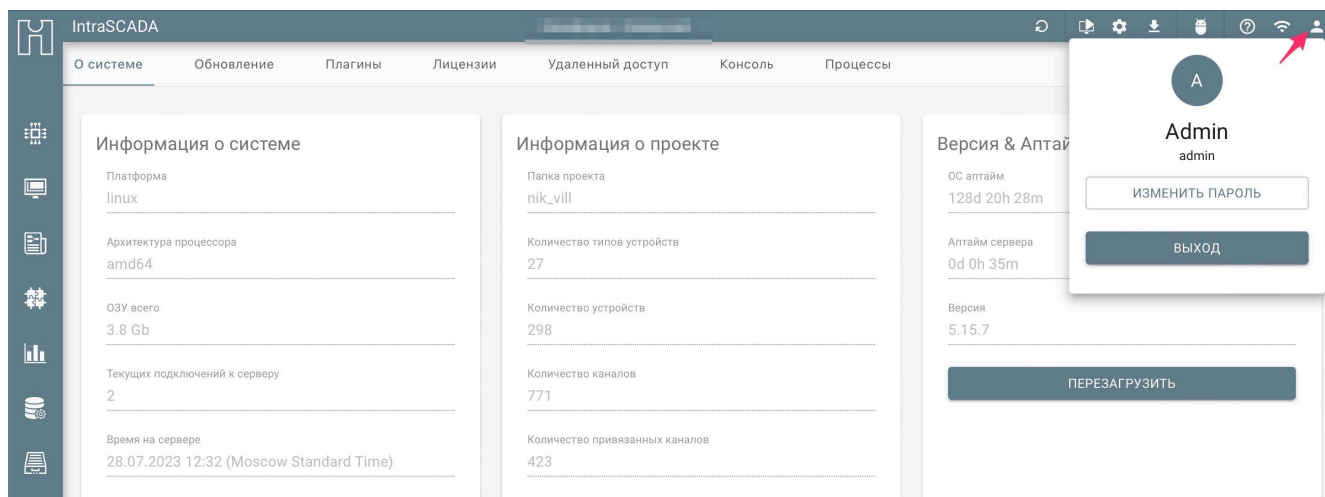
1. Правой кнопкой мыши на названии проекта выбрать "Копировать"
2. Правой кнопкой мыши на названии проекта выбрать "Вставить"

Будет создана копия проекта.

# Смена пароля

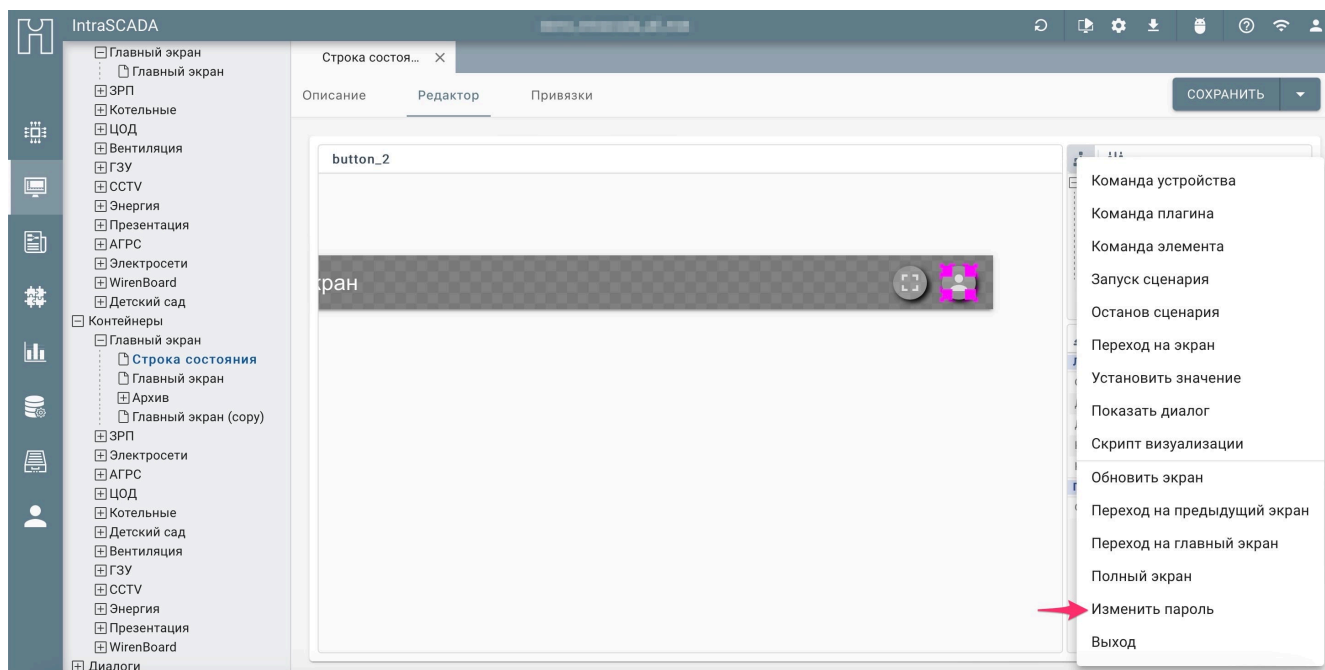
## В интерфейсе разработчика (Project Manager)

Для смены пароля в интерфейсе разработчика необходимо нажать кнопку в правом верхнем углу и изменить пароль:



## В пользовательском интерфейсе

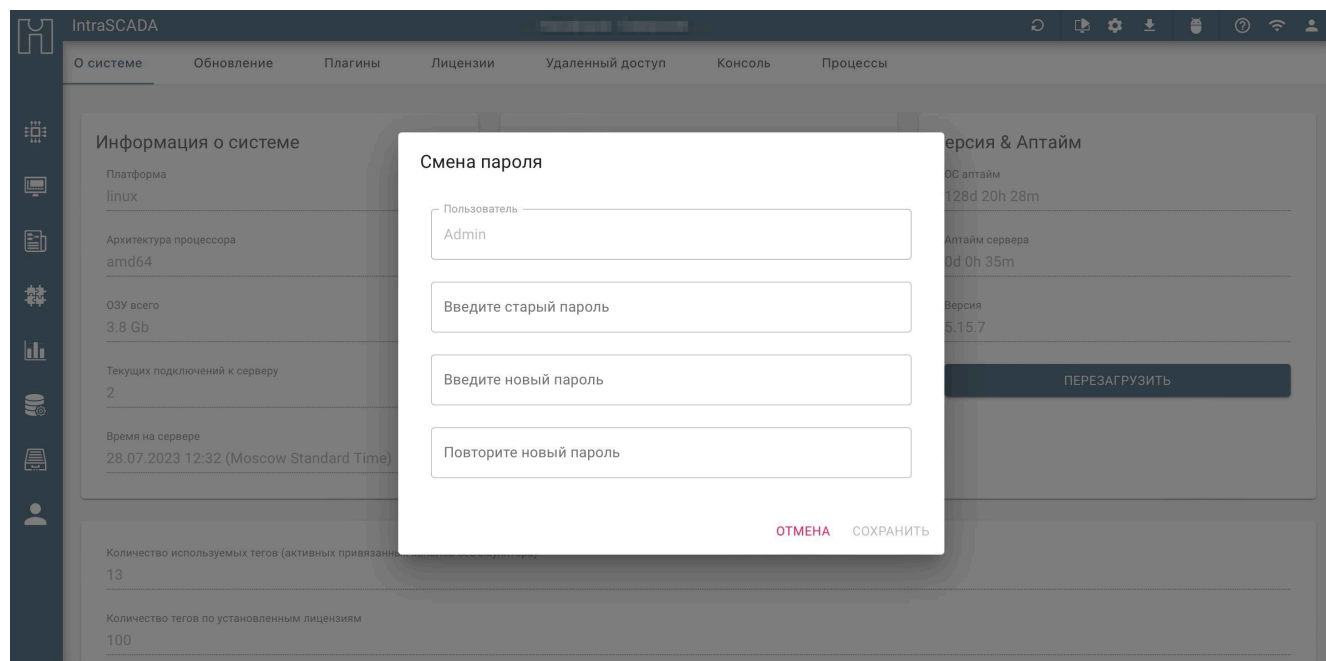
Для смены пароля из пользовательского интерфейса необходимо в любом месте интерфейса разместить кнопку и привязать к ней команду: Изменить пароль:



Примечание. Сменить пароль в пользовательском интерфейсе можно комбинацией клавиш Shift + Alt +

## Окно смены пароля

При нажатии кнопок или комбинации клавиш для смены пароля вызывается диалоговое окно:

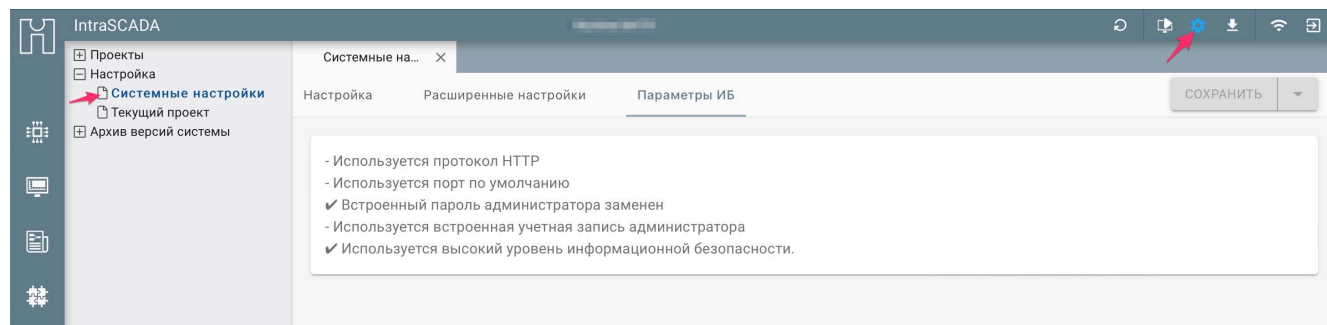


## Аудит ИБ

Текущие параметры ИБ можно получить через web-интерфейс или http(s) запрос.

### Web-интерфейс

В административном интерфейсе в **Системных настройках** на вкладке "Параметры ИБ":



Символ слева для каждого параметра означает:

- Установлен параметр по умолчанию

✓ Параметр по умолчанию изменен для реализации функции безопасности

### REST API

Эту же информацию можно получить с использованием REST API.

Данные настроек хранятся в таблицах, их можно считать, используя функции ядра системы.

IntraSCADA предоставляет возможность создавать собственные эндпоинты и обработчики для запросов [REST API](#).

Используя этот механизм, можно обеспечить получение параметров ИБ по запросу.

Ниже приведен обработчик, возвращающий JSON объект, содержащий текущие параметры ИБ:

```

/**
 * Обработчик запроса /audit – endpoint можно назвать произвольно
 * Виртуальная таблица 'isecaudit' состоит из одной записи и содержит параметры
 * Возвращается первая запись таблицы в виде JSON объекта
 */
module.exports = async (req, res, holder, debug) => {
  try {
    const data = await holder.getVirttable('isecaudit');
    res.json(data[0]);
    // Только для того, чтобы увидеть результат в консоли при отладке
    debug(JSON.stringify(data[0]));
  } catch (e) {
    res.json({ error: 1, message: e.message });
    debug(e.message)
  }
};

```

В полученном объекте только параметры со значением **true** удовлетворяют требованиям ИБ (имеют ✓ на web-интерфейсе).

```

{
  "usehttps": false,
  "useApiAccessToken": false,
  "strictIS": true,
  "notDefaultHTTPPort": true,
  "notDefaultAdmin": true,
  "notDefaultAdminPassword": true,
  "isecwritedb": true,
  "isecwritelog": true
}

```

ID параметра	При значении true
usehttps	Используется протокол HTTPS
useApiAccessToken	Используется токен для доступа к REST API
strictIS	Используется высокий уровень ИБ на уровне функций системы
notDefaultHTTPPort	Порт для доступа к интерфейсу по умолчанию заменен
notDefaultAdminPassword	Пароль встроенной учетной записи администратора по умолчанию заменен
notDefaultAdmin	Встроенная учетная запись администратора удалена
isecwritedb	Журнал ИБ пишется в БД
isecwritedb	Журнал ИБ пишется в ротлируемые лог файлы

Полное описание параметров дано в [Сводной информации по ИБ](#)



# Десктопное приложение IntraSCADA Client

## Назначение

Клиентское приложение для работы с системой IntraSCADA

Использование десктопного приложения по сравнению с веб-версией предоставляет:

### Для пользователей

- **Учет стандартов предприятия**

Иногда на предприятии в целях безопасности запрещено использование каких-либо браузеров.

- **Удобство использования**

Десктопным приложением пользоваться удобнее. Не нужно помнить адреса и сохранять страницы браузера. Приложение будет сразу открываться с нужным проектом.

- **Полноэкранный режим**

Есть возможность запускать приложение в полноэкранном режиме и скрывать системную панель.

- **Мультиэкранный режим**

Возможность открывать разные экраны(мнемосхемы) интерфейса на разных мониторах в разных пропорциях. С помощью данной технологии можно организовывать огромные видео стены.

- **Стабильность, независимость от браузера и его версии**

Мы стараемся проверять функционал системы IntraSCADA на разных браузерах. Тем не менее, бывают нарушения корректной работы пользовательских интерфейсов в связи с обновлением версии браузера. Использование десктопного приложения гарантирует стабильную работу пользователей системы.

- **Технология единого входа SSO**

Если на сервере IntraSCADA и на сервере LDAP настроена технология Kerberos, то приложение будет автоматически подключаться к серверу IntraSCADA под тем пользователем под которым авторизовались в операционной системе.

- **Запуск сторонних приложений**

Возможность запускать сторонние приложения на клиенте, например Калькулятор, Блокнот и т.д.

- **Удаленный доступ**

В приложение встроены механизмы удаленного доступа (P2P) к серверам IntraSCADA. Проект может быть размещен на удаленном сервере, это работает прозрачно для пользователя.

### Для инсталляторов

- **Многооконный интерфейс**

Можно одновременно работать с интерфейсом разработчика и пользовательским интерфейсом.

- **Список серверов**

Можно одновременно работать с несколькими серверами IntraSCADA, в том числе удаленными. Список и настройки серверов сохраняются, все проекты доступны сразу внутри приложения.

- **Проброс портов**

Благодаря встроенному механизму проброса портов, есть возможность управлять не только сервером с системой IntraSCADA, но и любыми другими устройствами и сервисами в локальной сети сервера. Например, вы можете удаленно, не выезжая на объект, перезалить прошивку контроллера.

## Установка

Приложение можно установить на компьютеры с операционными системами Linux x64/arm64 (deb,rpm), macOS arm64, Windows x64

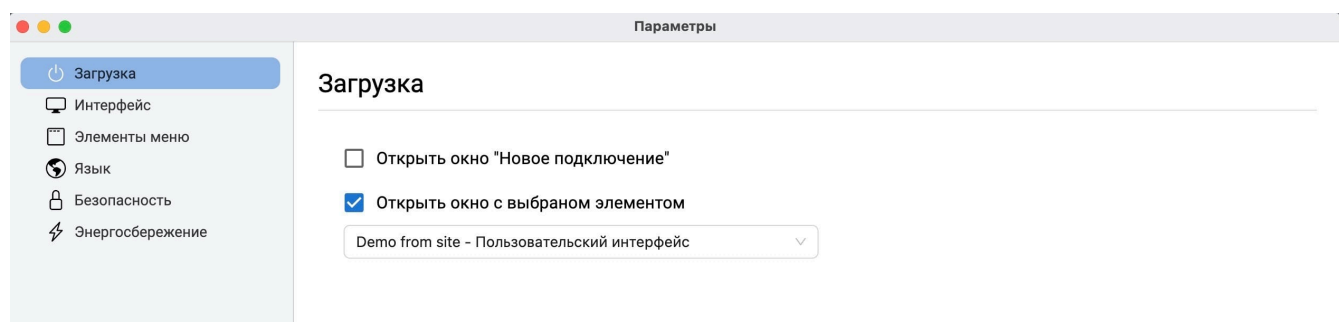
При установке на Ubuntu 24 необходимо в терминале выполнить команду и перезагрузить ОС:

```
sudo bash -c 'echo -e "abi <abi/4.0>,\n\
include <tunables/global>\n\n\
profile intrascada_client /opt/IntraSCADA\\ client/intrascada_client flags=
users,\n\
include if exists <local/intrascada_client>\n\
}" > /etc/apparmor.d/intrascada_client'

sudo service apparmor reload
```

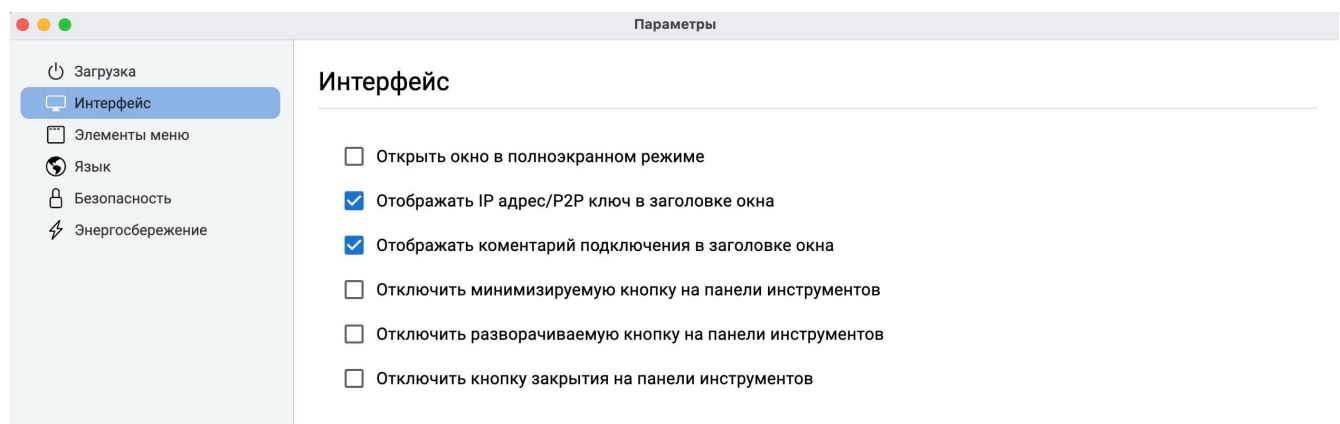
## Настройки

### Загрузка



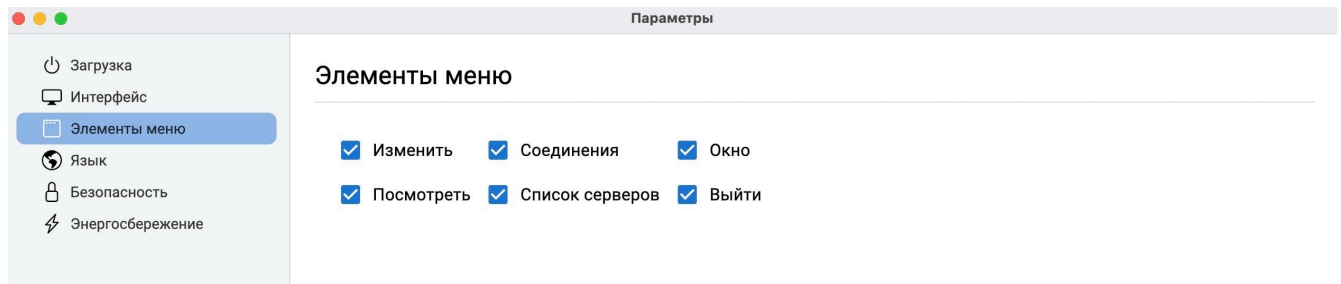
При загрузке приложения открыть новое окно или окно с выбранным адресом сервера.

### Интерфейс



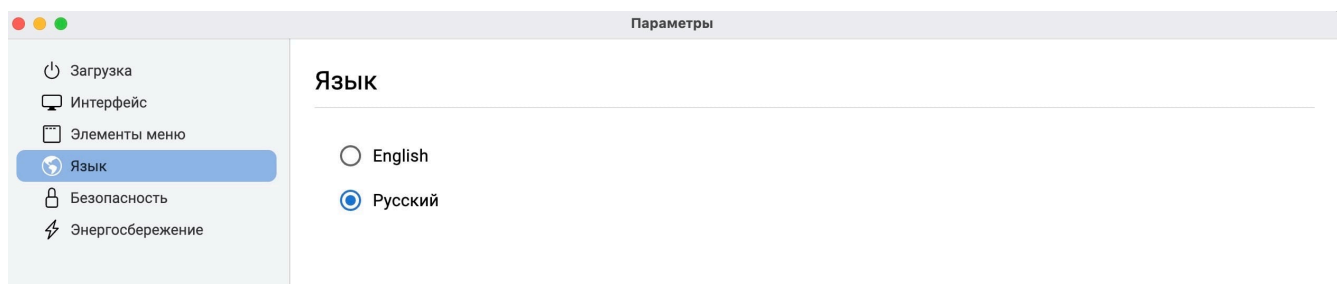
Для выхода из полноэкранного режима можно воспользоваться быстрой клавишей F11.

## Элементы меню



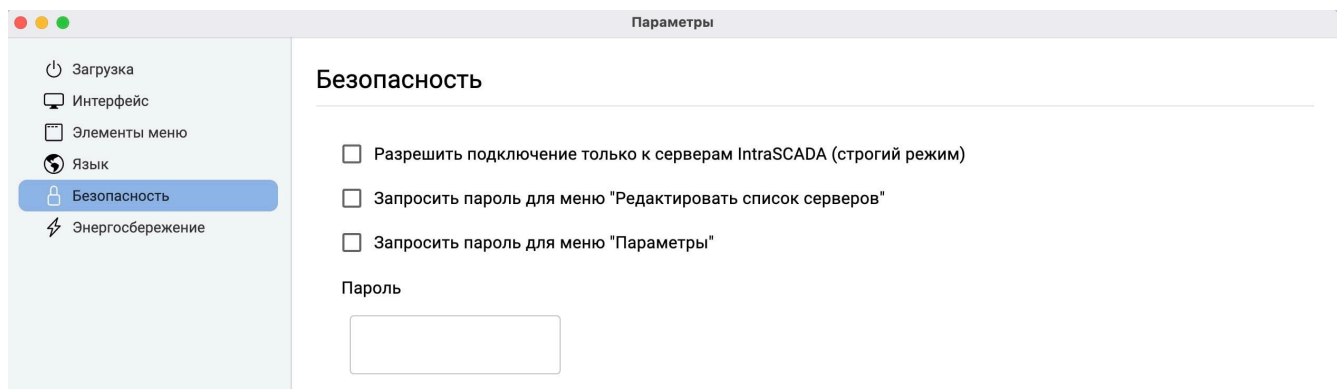
Скрыть/показать элементы меню

## Язык



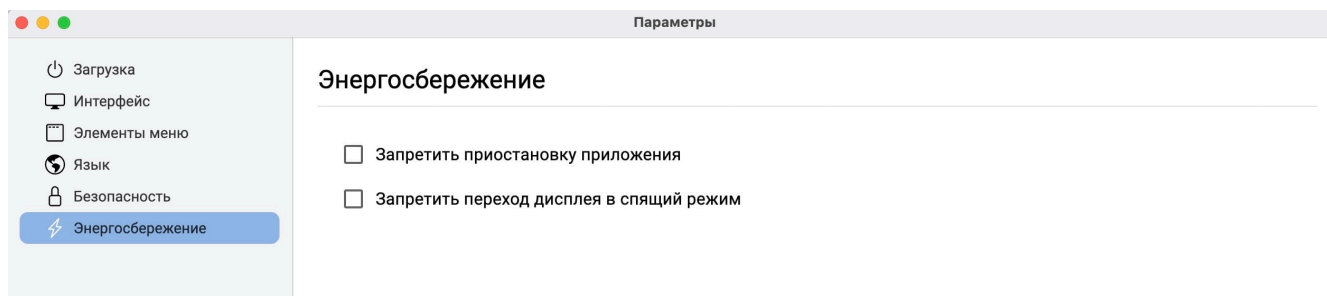
Выбрать язык интерфейса приложения

## Безопасность



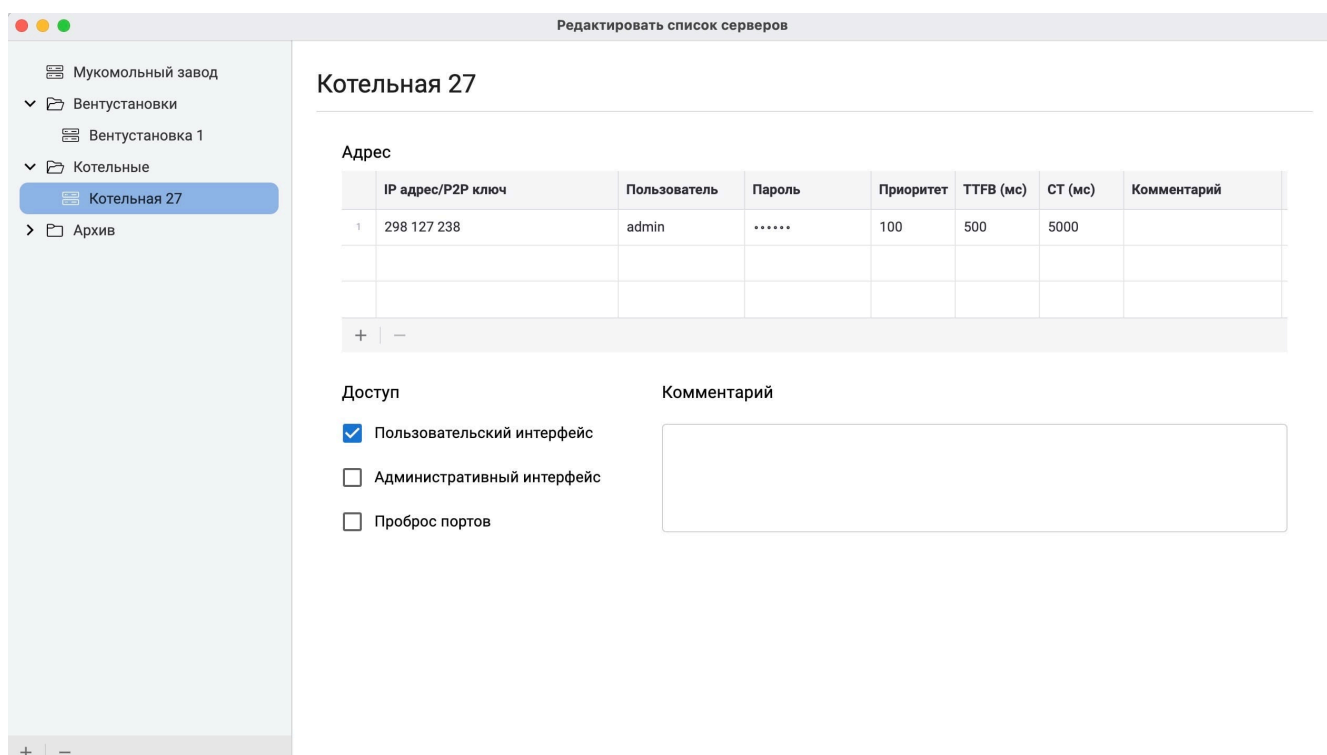
- Разрешить подключение только к серверам IntraSCADA (строгий режим)  
Разрешается подключение только к серверам IntraSCADA
- Запросить пароль  
При входе в настройки параметров и списка серверов запрашивать пароль

## Энергосбережение



- Исключает приложение из плана энергосберегающего режима операционной системы

## Редактирование списка серверов



В левой части отображается список серверов в виде дерева.

Этот список актуален для инсталляторов систем автоматизации. Для конечных пользователей в этом списке вероятнее всего будет только одна запись с одним сервером.

В левом нижнем углу расположены две кнопки для добавления или удаления папок дерева и серверов. Методом drag&drop можно перемещать папки и записи.

## Параметры подключения

### • IP адрес / P2P ключ

Адрес сервера или P2P ключ для подключения к удаленному серверу.

Примеры:

**192.168.0.245:8088** -для доступа в локальной сети,

**http://176.145.21.40:46089** -для доступа к серверу с выделенным IP адресом

**https://176.145.21.40:46089** -для доступа к серверу с выделенным IP адресом по протоколу https

**298 127 238** -для удаленного доступа через P2P

- **Пользователь**

Имя пользователя на сервере IntraSCADA

- **Пароль**

Пароль для входа пользователя на сервер IntraSCADA

- **Приоритет**

Приоритет входа на сервер. Используется в случае резервирования серверов.

- **TTFB (Time To First Byte)**

Время до получения первого байта в миллисекундах при подключении к серверу. Если в течении этого времени не будет получен ответ от сервера, будет выполнено переподключение.

- **CT (Connection Timeout)**

Время получения ответа от сервера при подключенном сервере. В случае потери связи с сервером по истечении этого времени будет выполнено переподключение.

## Доступ

- Разрешить/Запретить доступ в пользовательский и административный интерфейс (PM).
- Разрешить/Запретить проброс портов.

## Проброс портов

Редактировать список серверов

Мукомольный завод

- Вентустановки
- Котельные
  - Котельная 27**
  - Архив

### Котельная 27

Адрес

	IP адрес/P2P ключ	Пользователь	Пароль	Приоритет	TTFB (мс)	CT (мс)	Комментарий
1	298 127 238	admin	*****	100	500	5000	

+ | -

Доступ

☒ Пользовательский интерфейс

☒ Административный интерфейс

☒ Проброс портов

Комментарий

Список проброса портов

	Название	Локальный порт	Удаленный хост	Удаленный порт	Комментарий
1	Контроллер Wago	2455	192.168.7.178	2455	Для Codesys
2	Контроллер Wago	9980	192.168.7.178	80	

Допустим, в локальной сети предприятия, где установлен сервер IntraSCADA есть PLC контроллер. Требуется удаленный доступ к PLC для возможности перезалить прошивку контроллера или проверить его работу из программы Codesys. Для этого необходимо пробросить порт 2455. Кроме этого можно пробросить 80 порт контроллера для доступа в веб интерфейс контроллера.

Для запуска проброса портов зайти в Список серверов/Выбрать сервер/Проброс портов и нажать кнопку "Запустить":



После выполнения соединений можно переходить к настройкам контроллера.

Для входа на контроллер (PLC) запускаем на своем локальном компьютере программу Codesys. В настройках доступа к контроллеру прописываем локальный адрес вашего компьютера: 127.0.0.1

Для входа в веб интерфейс контроллера запускаем браузер и в адресной строке вписываем адрес локальный адрес вашего компьютера и порт (127.0.0.1:9980).

Таким образом, заходя на своем компьютере по локальному адресу 127.0.0.1 с соответствующим портом, вы попадаете на контроллер, который может находиться за тысячи километров от вас.

Аналогично можно настроить доступ к любым устройствам в той сети, где установлен сервер IntraSCADA

## Мультиэкран

В данном разделе можно настроить на какой монитор будет выводиться тот или иной экран(мнемосхема).

Если указать просто монитор(0) и id экрана(I025) и поставить галочку **Полный экран**, то мнемосхема откроется на весь экран, а если указать конкретные координаты экрана и его ширину и высоту, то можно собрать целую видео стену из нескольких десятков мнемосхем с возможностью разворачивать любую из них на полный экран и обратно.

Режим **Без рамки** позволяет выводить изображение без рамки окна операционной системы, тем самым позволяя вам сделать монолитное изображение из мнемосхем

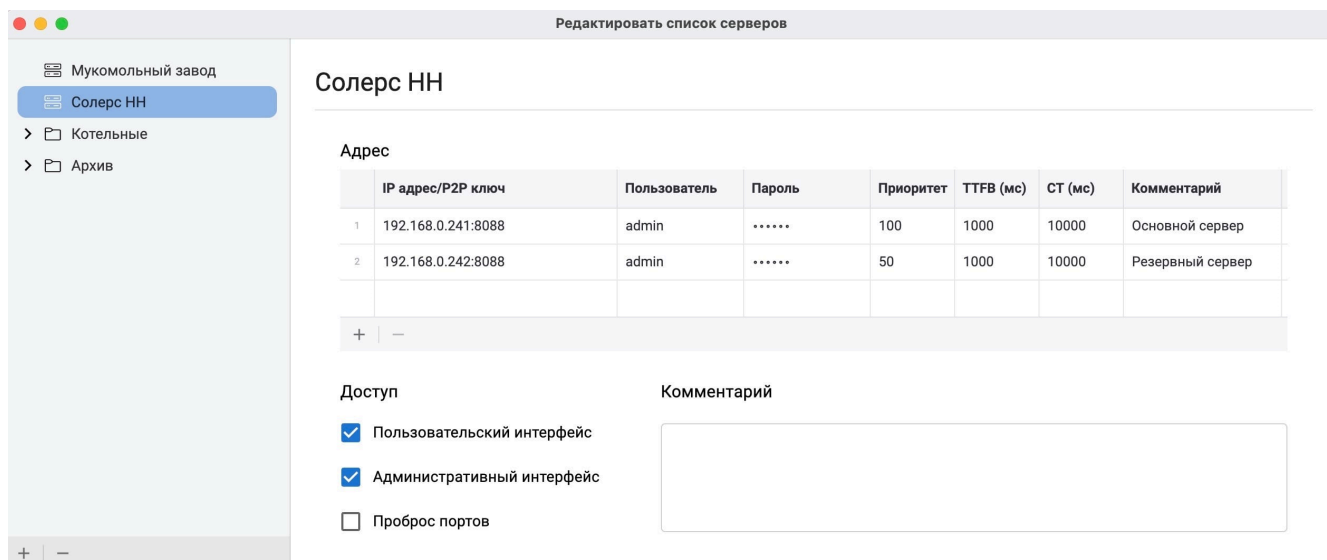
☒ Мультиэкран

### Список экранов

	Дисплей	ID экрана	x	y	w	h	Без рамки	Полный экран	Комментарий
1	0	I025	0	0	500	500	<input type="checkbox"/>	<input type="checkbox"/>	
2	0	I025	0	0	0	0	<input type="checkbox"/>	<input type="checkbox"/>	

+ | -

## Резервирование серверов



Предусмотрена возможность бесшовной работы с несколькими серверами.

В примере выше показана настройка для двух серверов: основного и резервного.

Для основного сервера выставлен приоритет 100, для резервного 50.

При первом включении приложение выполнит попытку подключения к основному серверу. Если основной сервер не доступен, будет выполнено подключение к резервному серверу. Аналогичная ситуация и в процессе работы. При потере связи с одним из серверов, приложение автоматически выполнит подключение к другому.

### Полезные ссылки

Официальный сайт:	<a href="http://intrascada.ru">intrascada.ru</a>
Онлайн документация:	<a href="http://docs.intrascada.ru">docs.intrascada.ru</a>
Личный кабинет пользователя:	<a href="http://lk.intrascada.ru">lk.intrascada.ru</a>