Luxms BI

Data Boring. Руководство пользователя

ОПИСАНИЕ ИНТЕРФЕЙСА

ОПИСАНИЕ УЗЛОВ

ТАБЛИЦЫ РЕШЕНИЙ

КНИГА РЕЦЕПТОВ

2024-04-02

Оглавление

1 Интерфейс пользователя

1.1 Кратко о Luxms Data Boring

Luxms Data Boring — ELT/ETL инструмент визуального программирования класса dataflow programming. Предназначен для инженеров данных и аналитиков. Инструмент полезен для:

- формирования слоя «горячих» данных рядом с хранилищем;
- получения данных из разнородных источников и хранение их в одном месте;
- запуска задач обработки данных по расписанию;
- визуальной разработки ELT/ETL задач силами дата инженеров со знанием SQL без привлечения специалистов DevOps.

Инструмент оптимизирован под работу с Luxms BI, но может использоваться и самостоятельно для задач по обработке данных.

DataBoring не предназначен для тяжёлых вычислений. При решении задач обработки больших данных требуется использовать подход ELT и использовать вычислительные возможности СУБД.

1.1.1 Список выполняемых функций

Наличие средств для автоматизации корпоративных задач ELT/ETL:

- выгрузка данных из исходных систем-источников;
- преобразование и обогащение данных;
- валидация и очистка данных;
- загрузка в различные системы-приемники (JDBC, Kafka, Redis и др.);
- загрузка во внутреннее хранилище данных Luxms BI;
- возможность настройки расписания запуска алгоритмов ELT/ETL в визуальном интерфейсе;
- возможность просмотра ELT/ETL процессов в виде схемы в визуальном интерфейсе с отображением этапов и связей между ними;
- возможность просмотра результатов в визуальном интерфейсе, полученных на каждом из этапов ELT/ETL процессов.

1.1.2 Поддержка ETL и ELT в DataBoring

ETL – это последовательный процесс извлечения (**E**xtract), преобразования (**T**ransform) и загрузки (**L**oad) данных. Преобразование данных происходит после выгрузки их из источника на стороне DataBoring. Возможен запуск внешних скриптов для обработки данных. После преобразований, готовые данные загружаются в целевую систему хранения.

ELT – это последовательный процесс извлечения (**E**xtract), загрузки (**L**oad) и преобразования (**T**ransform) данных. При этом DataBoring только запускает команды на преобразование данных в целевой системе, выступая в качестве оркестратора.

В Data Boring возможны оба варианта.

При работе с Luxms BI мы рекомендуем реализовывать ELT-процессы, т.к. операция преобразования данных требует высоких вычислительных мощностей. Таким образом, при преобразовании данных во внутреннем хранилище Luxms BI, вы сможете добиться высокой скорости обработки данных.

Во время работы в BI системе фоном идёт выполнение заданий на загрузку и обработку данных в инструменте Data Boring.

1.1.3 Источники данных

- данные можно брать из JDBC-источника, например: PostgreSQL, ClickHouse, Oracle, Greenplum и др. При работе с платформой Luxms BI тип источника должен быть настроен в административной панели Luxms BI;
- загружать/выгружать данные из/в файлы: csv, xml, json, xslx, xls, txt и др.;
- HTTP сервисы: REST, SOAP и др.

1.1.4 Преобразование данных

При подходе ELT для преобразования данных используется вариант языка запросов, поддерживаемых источником. Чаще всего это язык SQL.

При подходе ETL используется встроенный в Data Boring язык программирования JavaScript и стандартные узлы для обработки данных.

1.1.5 Приёмники данных

Список систем-приёмников практически не ограничен.

Например:

- JDBC (PostgreSQL, ClickHouse, Oracle, Greenplum и др.);
- HTTP;
- MQTT;
- Kafka;
- Redis;
- TCP/UDP.

1.1.6 Используемые языки программирования

При настройке узлов в Data Boring, вы можете использовать язык SQL-запросов или JavaScript, в зависимости от назначения узла.

В то же время вы можете подключить сторонний скрипт, написанный на любом популярном языке программирования. Для этого вам необходимо разместить скрипт на сервере Data Boring, настроить среду для выполнения скрипта и создать поток с вызовом этого скрипта.

Также скрипты можно запускать на удалённых серверах с помощью SSH.

1.2 Окно редактора Luxms Data Boring

- Вверху – «шапка». На ней находится кнопка «Развернуть», кнопка главного меню «Гамбургер» (с тремя полосками), а также меню пользователя
- Слева – «палитра». В ней содержатся узлы, которые можно использовать в редакторе
- Посередине – «холст», главная рабочая область. В нём создаются потоки
- Справа – «боковая панель»

1.3 Палитра

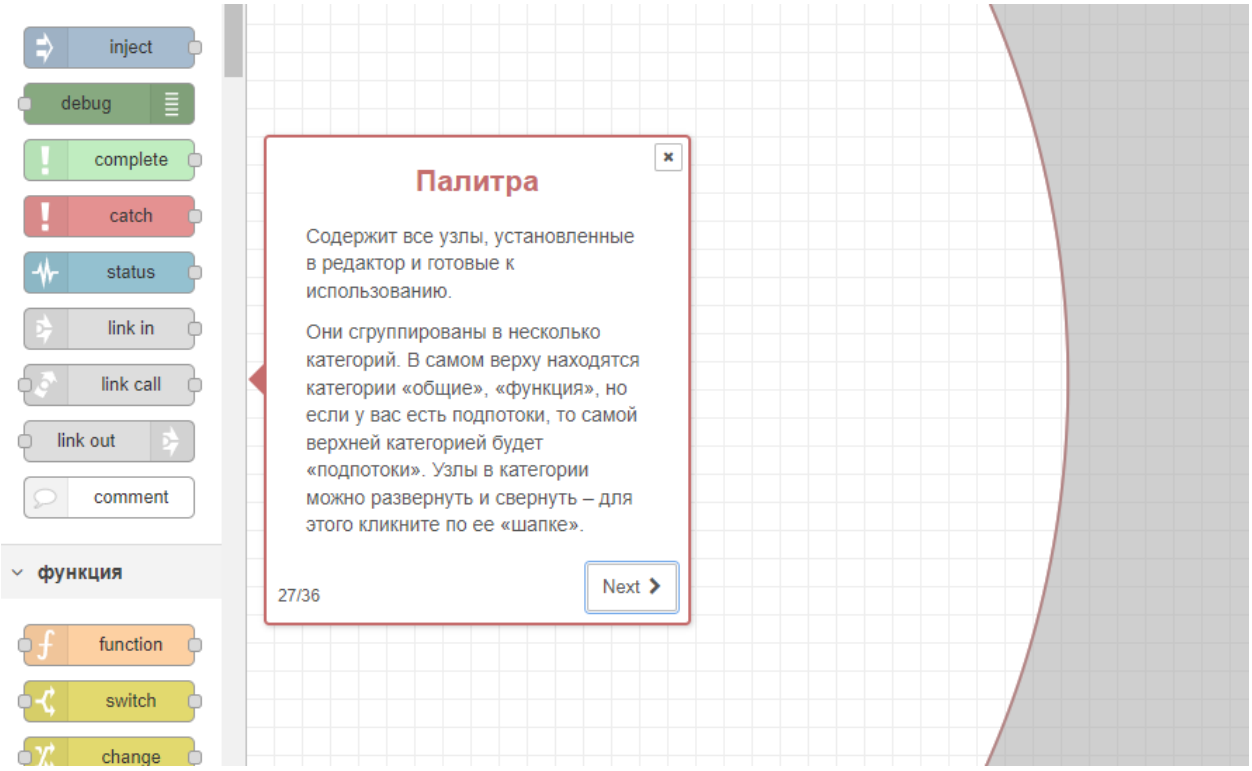


Рис. 1.1 **Палитра**

Палитра содержит все узлы, установленные в редактор и готовые к использованию.

Они сгруппированы в несколько категорий: - Общие; - Функция; - Сеть; - Последовательность; - Анализатор; - Хранилище; - Luxms BI; - Jupiter; - Redis; - Telegram.

В самом верху находятся категории «Общие», «Функция», но если у вас есть подпотоки, то самой верхней категорией будет «Подпотоки». Узлы в категории можно развернуть и свернуть – для этого щёлкните по ее «шапке».

В верхней части палитры есть поле для ввода текста, предназначенное для поиска нужных узлов.

Чтобы развернуть/свернуть все категории, воспользуйтесь двумя кнопками в самом низу «палитры» – они изображены в виде двойных стрелочек, направленных вниз и вверх.

1.4 Боковая панель

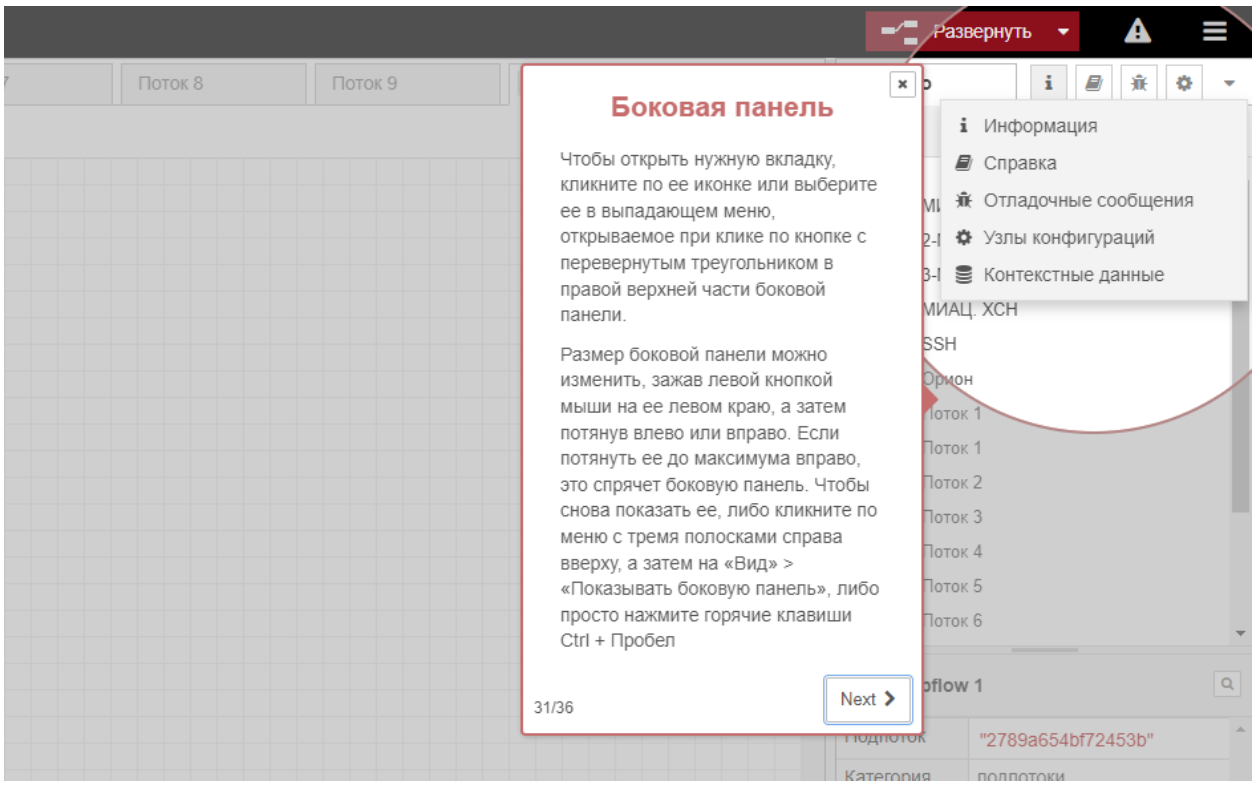


Рис. 1.2 **Боковая панель**

Содержит несколько полезных инструментов редактора:

- Вкладка «Информация» – содержит информацию об узле.
- Вкладка «Справка» – содержит вспомогательную справку для узла.
- Вкладка «Отладочные сообщения» – показывает сообщения, отправляемые узлами «отладка» (группа узлов «общие»).
- Вкладка «Узлы конфигурации» – для управления конфигурационными узлами.
- Вкладка «Контекстные данные» – показывает содержимое контекстов.

Чтобы открыть нужную вкладку, щёлкните по её иконке или выберите её в выпадающем меню, открываемое при щелчке по кнопке с перевёрнутым треугольником в правой верхней части боковой панели.

Размер боковой панели можно изменить, зажав левой кнопкой мыши на её левом краю, а затем потянув влево или вправо. Если потянуть её до максимума вправо, это спрячет боковую панель. Чтобы снова показать её, либо щёлкните по меню «Гамбургер» справа вверху, а затем на «Вид» > «Показывать боковую панель», либо просто нажмите горячие клавиши Ctrl + Пробел. Кроме того, убрать или развернуть боковую панель можно, кликнув по стрелке в середине её левой грани.

1.5 Холст и Потоки

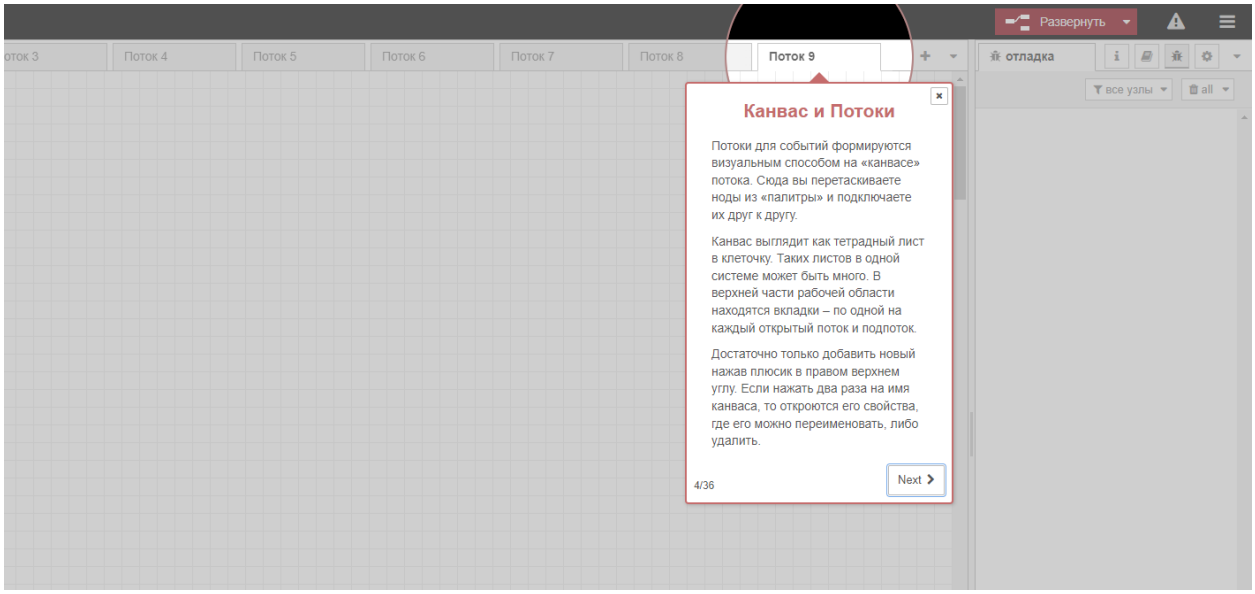


Рис. 1.3 **Холст и Потоки**

Потоки для событий формируются визуальным способом на «холсте» потока. Сюда вы перетаскиваете узлы из «палитры» и соединяете их между собой.

Холст выглядит как тетрадный лист в клеточку. Таких листов в одной системе может быть много. В верхней части холста находятся вкладки – по одной на каждый открытый поток и подпоток.

Достаточно только добавить новый поток, нажав плюсик в правом верхнем углу. Если нажать два раза на имя холста, то откроются его свойства, где его можно переименовать, либо удалить.

1.6 Инструменты для просмотра холста

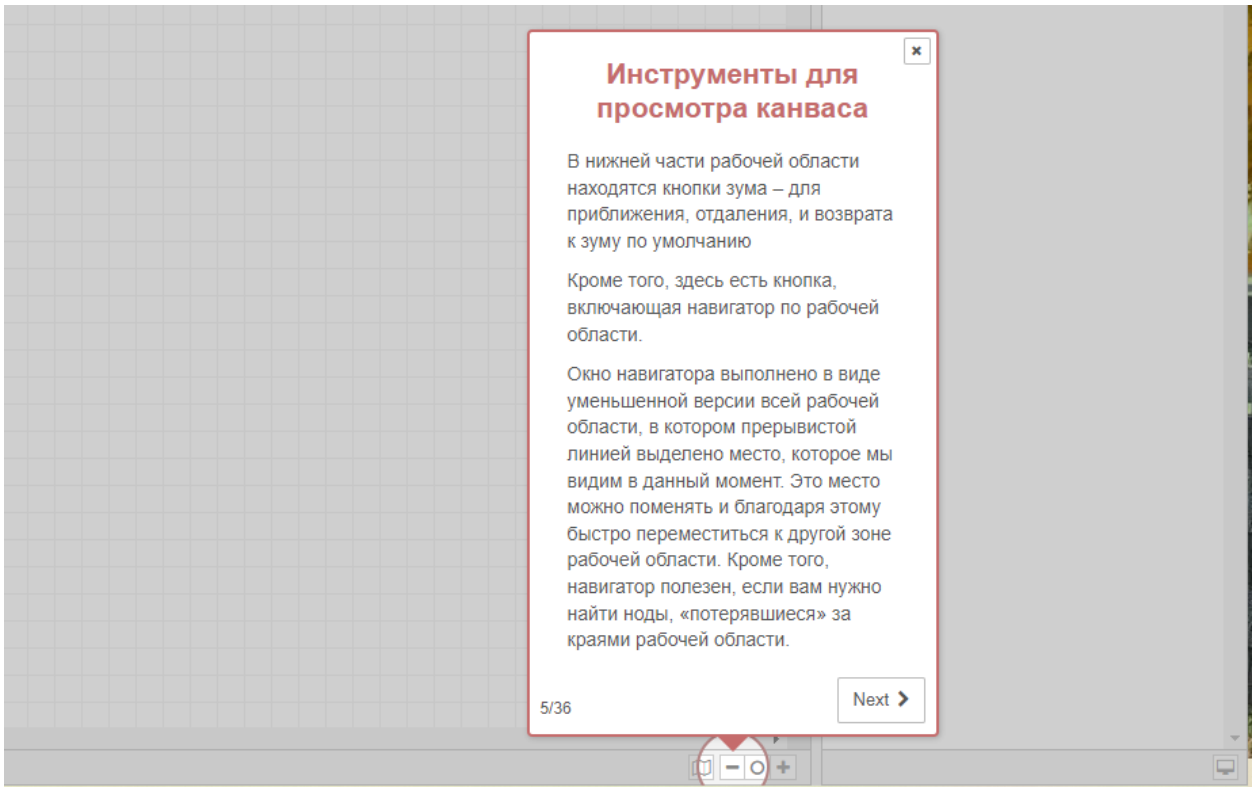


Рис. 1.4 **Инструменты для просмотра холста**

В нижней части холста находятся кнопки масштабирования – для приближения, отдаления, и возврата к масштабу по умолчанию

Кроме того, здесь есть кнопка, включающая навигатор по холсту.

Окно навигатора выполнено в виде уменьшенной версии всего холста, в котором прерывистой линией выделено место, которое мы видим в данный момент. Это место можно поменять и благодаря этому быстро переместиться к другой зоне холста. Кроме того, навигатор полезен, если вам нужно найти узлы, «потерявшиеся» за краями холста.

1.7 Настройка холста

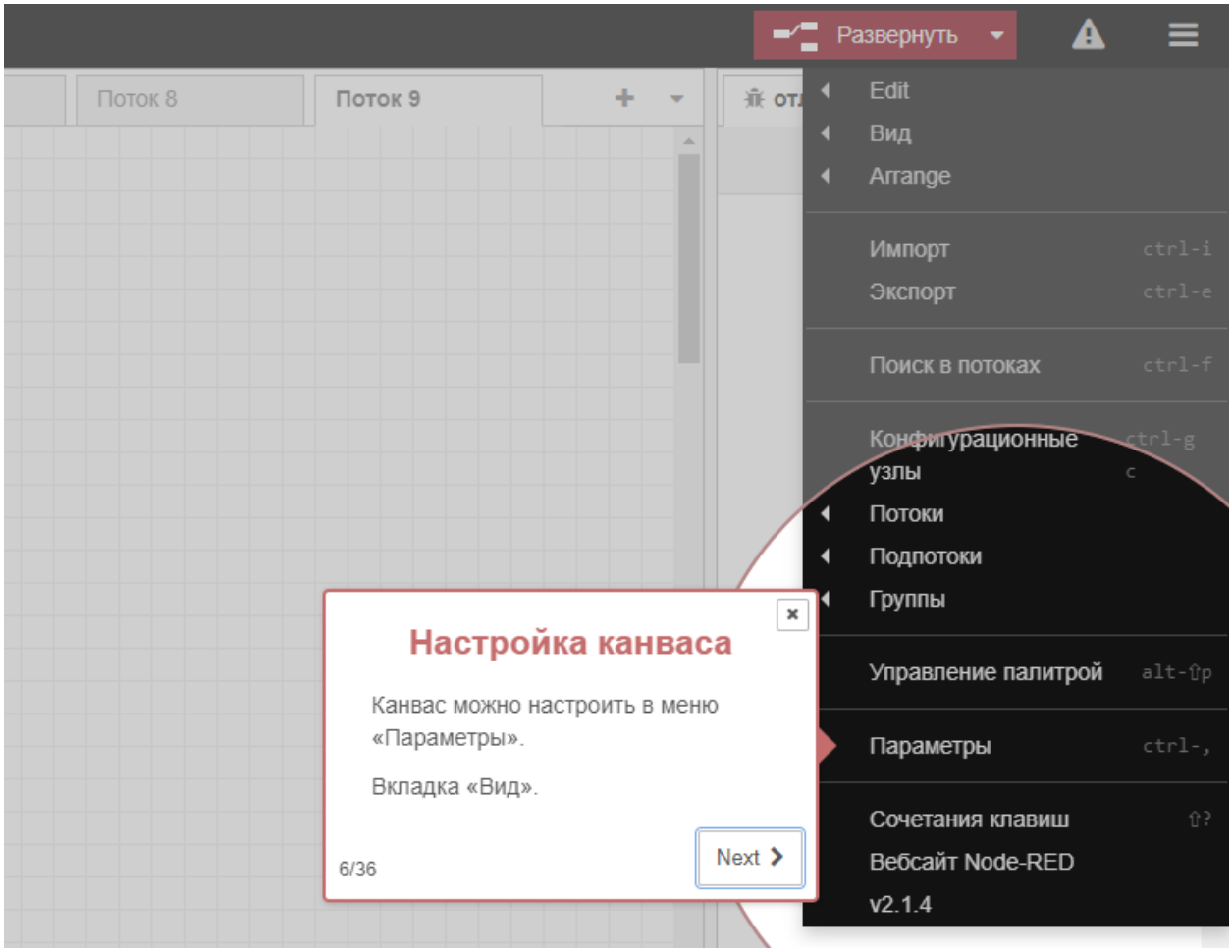


Рис. 1.5 **Настройка холста**

Холст можно настроить в меню «Параметры» во вкладке «Вид».

1.8 Добавление потока

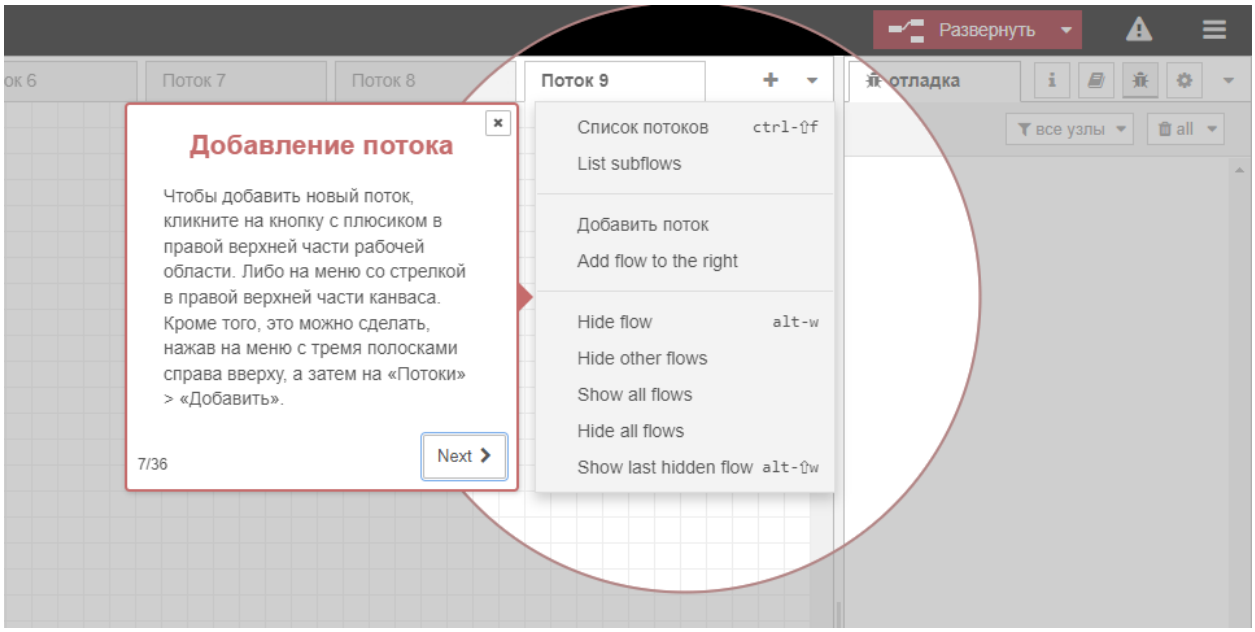


Рис. 1.6 **Добавление потока**

Чтобы добавить новый поток, щёлкните на кнопку с плюсиком в правой верхней части холста. Либо на меню со стрелкой в правой верхней части холста. Кроме того, это можно сделать, нажав на меню «Гамбургер» справа вверху, а затем на «Потоки» > «Добавить».

1.9 Редактирование свойств потока

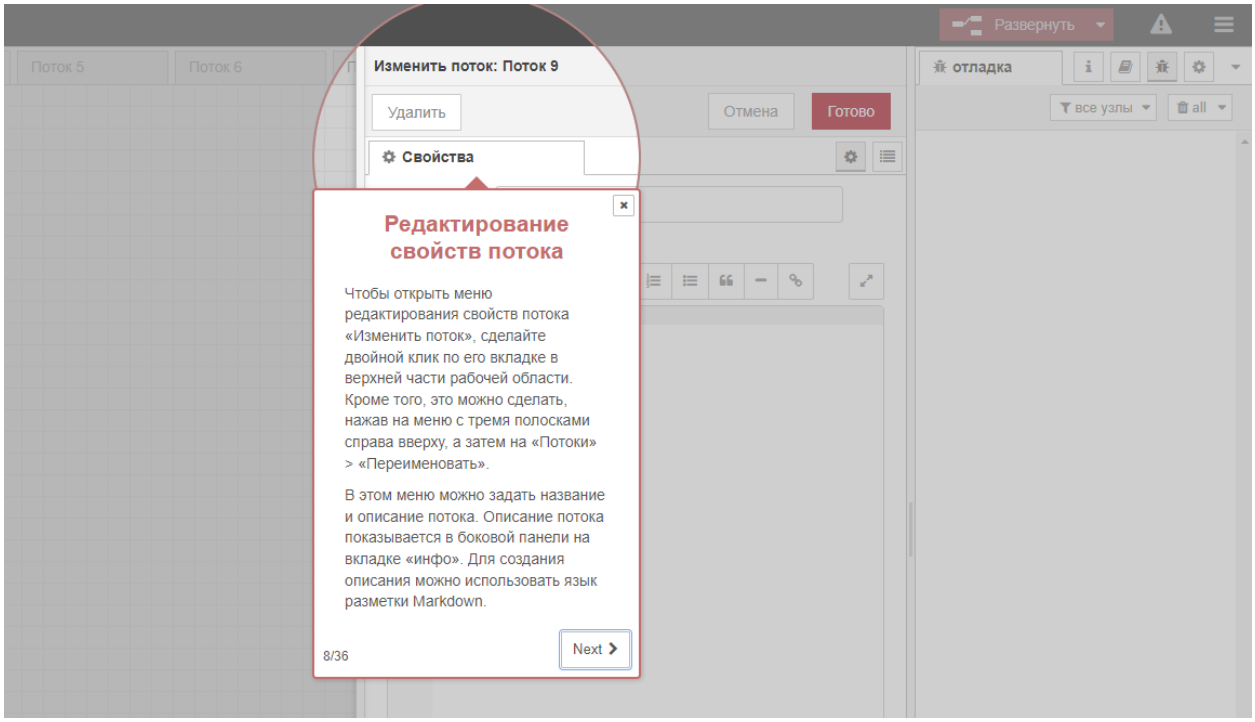


Рис. 1.7 **Редактирование свойств потока**

Чтобы открыть меню редактирования свойств потока «Изменить поток», сделайте двойной щелчок по его вкладке в верхней части холста. Кроме того, это можно сделать, нажав на меню «Гамбургер» справа вверху, а затем на «Потоки» > «Переименовать».

В этом меню можно задать название и описание потока. Описание потока показывается в боковой панели на вкладке «инфо». Для создания описания можно использовать язык разметки Markdown.

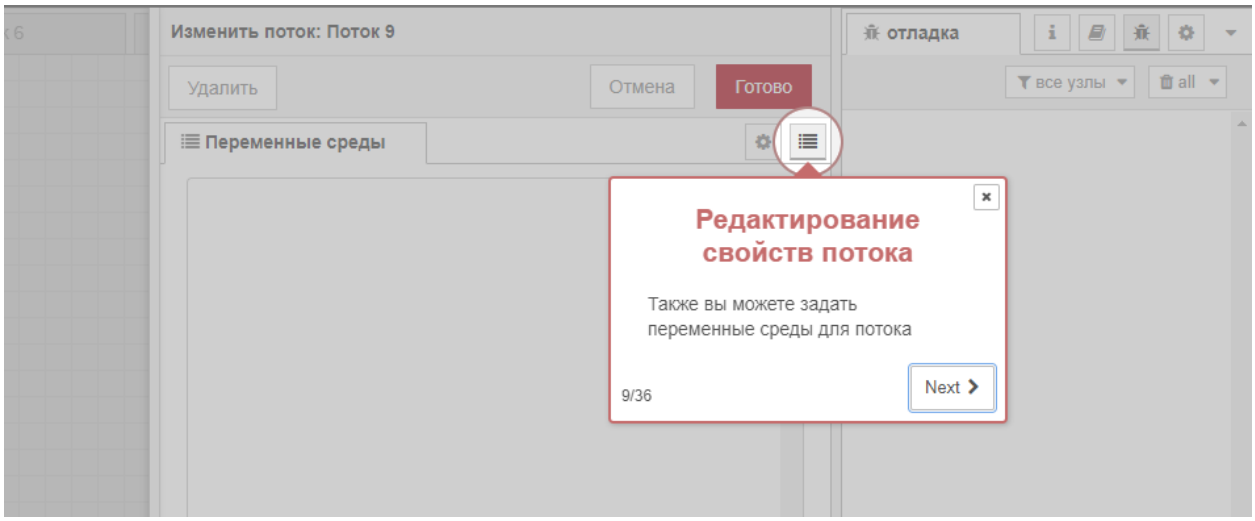


Рис. 1.8 **Переменные среды для потока**

Также вы можете задать переменные среды для потока.

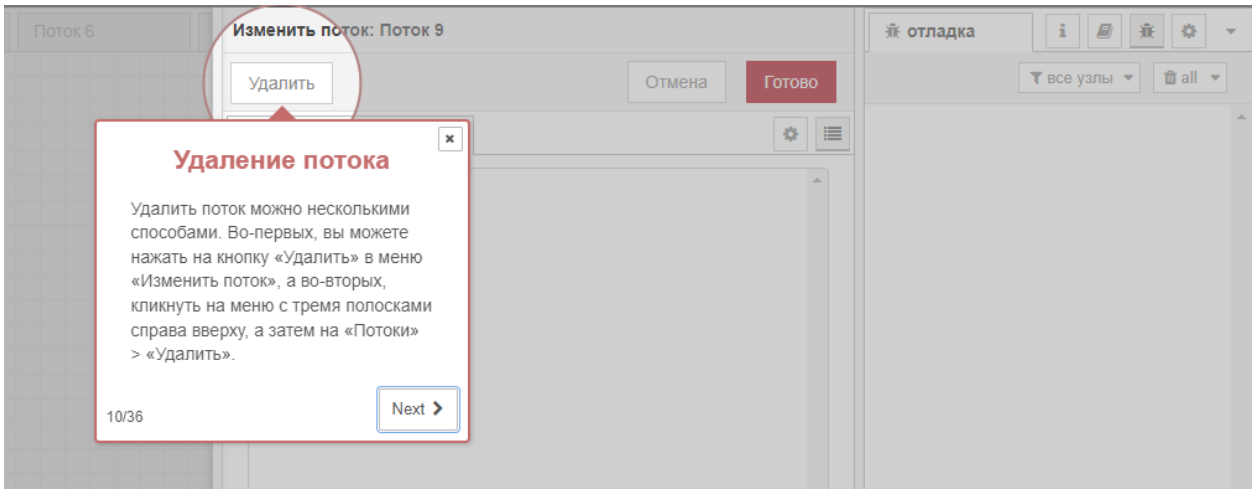


Рис. 1.9 **Удаление потока**

Удалить поток можно несколькими способами. Во-первых, вы можете нажать на кнопку «Удалить» в меню «Изменить поток», а во-вторых, щёлкнуть на меню «Гамбургер» справа вверху, а затем на «Потоки» > «Удалить».

1.10 Узлы

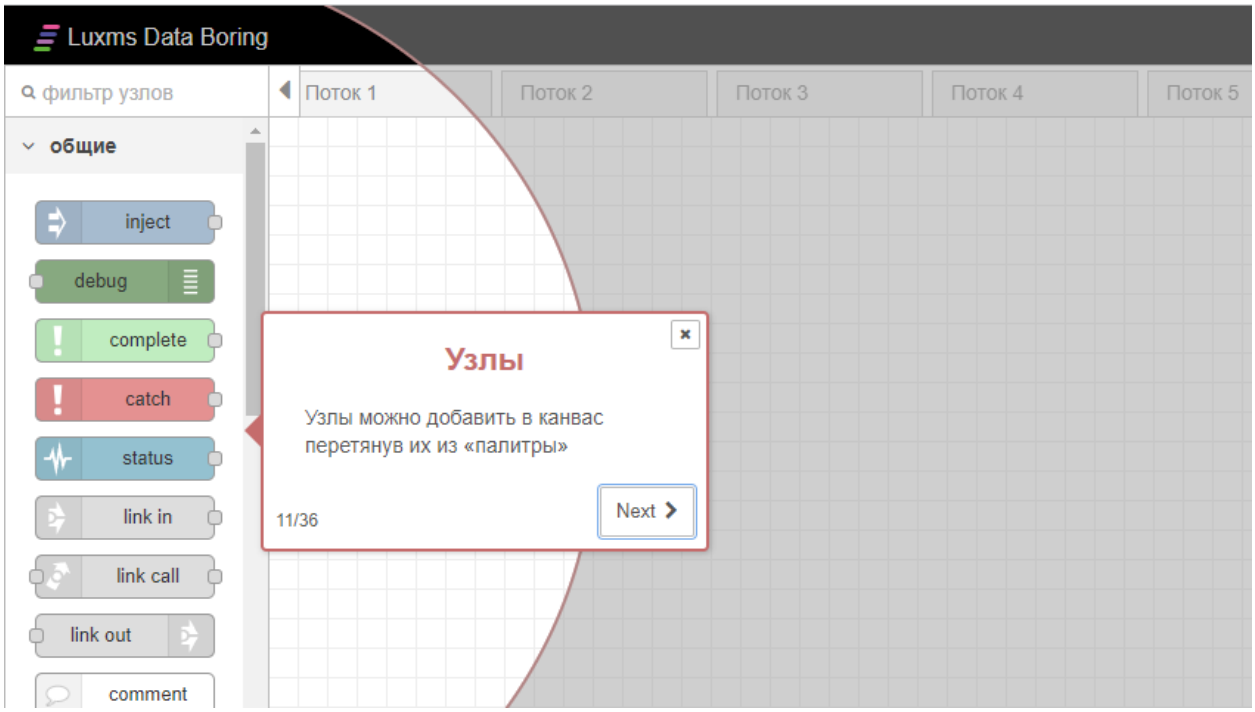


Рис. 1.10 **Узлы**

Узлы можно добавить на холст, перетянув их из «палитры»

1.11 Подробнее об узлах

Узлы соединяются друг с другом при помощи «связей», подключенных к портам. У узла может быть лишь один входной порт и несколько выходных портов. У порта может быть описание, которое появляется, если навести на него курсор мыши. Узлы могут сами задавать эти описания – например, узел «Переключатель» показывает правило, соответствующее выходному порту. Кроме того, вы сами можете задать описания для портов в меню редактирования узла.

У некоторых узлов внизу также показывается статусное сообщение и иконка. Они нужны для того, чтобы показать текущее состояние узла – к примеру, MQTT-узлы с помощью этих элементов показывают, подключены они или нет.

Если в узел были внесены какие-то изменения, развёртка которых ещё не была выполнена, то в верхней части узла будет показан синий кружок. Если в настройках узла есть ошибки, в его верхней части будет показан красный треугольник.

У некоторых узлов слева или справа есть кнопка, которая позволяет взаимодействовать с ним прямо в редакторе. Единственные стандартные узлы с кнопками – это «запуск» и «отладка».

1.12 Редактирование настроек узлов

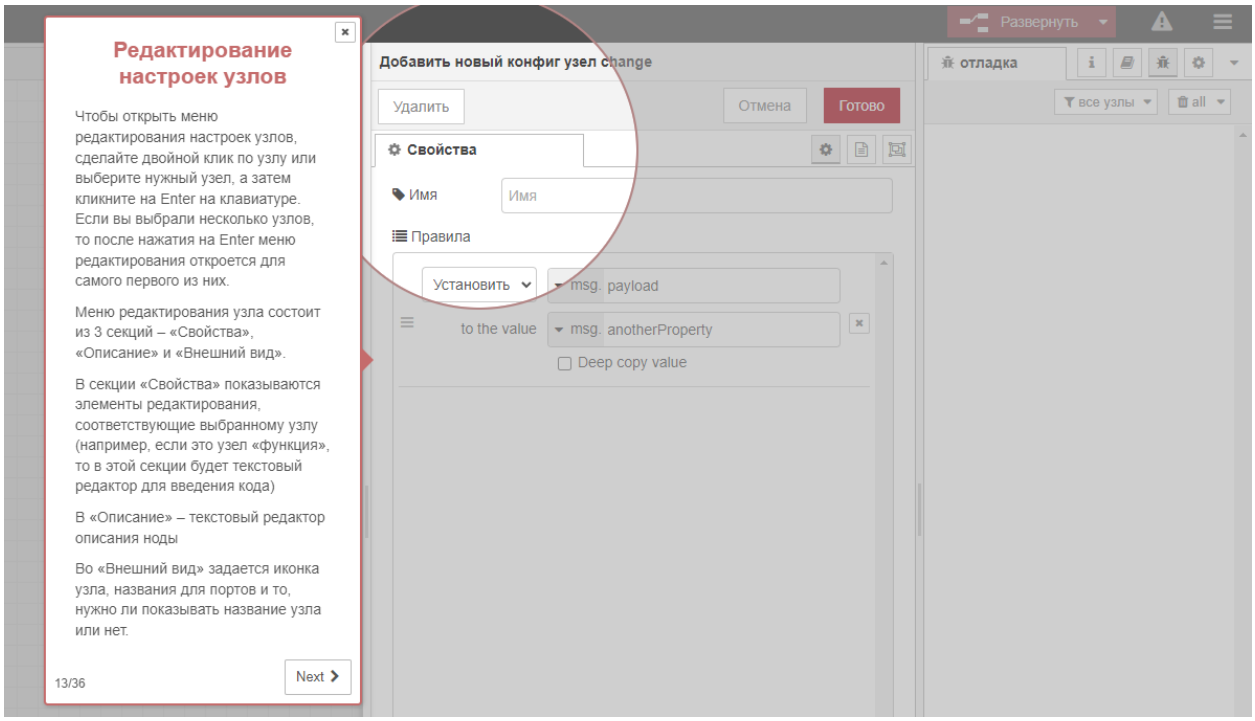


Рис. 1.11 **Редактирование настроек узлов**

Чтобы открыть меню редактирования настроек узлов, сделайте двойной щелчок по узлу или выберите нужный узел, а затем нажмите на Enter на клавиатуре. Если вы выбрали несколько узлов, то после нажатия на Enter меню редактирования откроется для самого первого из них.

Меню редактирования узла состоит из 3 секций – «Свойства», «Описание» и «Внешний вид».

В секции «Свойства» показываются элементы редактирования, соответствующие выбранному узлу (например, если это узел «функция», то в этой секции будет текстовый редактор для введения кода)

В «Описание» – текстовый редактор описания узла

Во «Внешний вид» задается иконка узла, названия для портов и то, нужно ли показывать название узла или нет.

1.13 Конфигурационные узлы

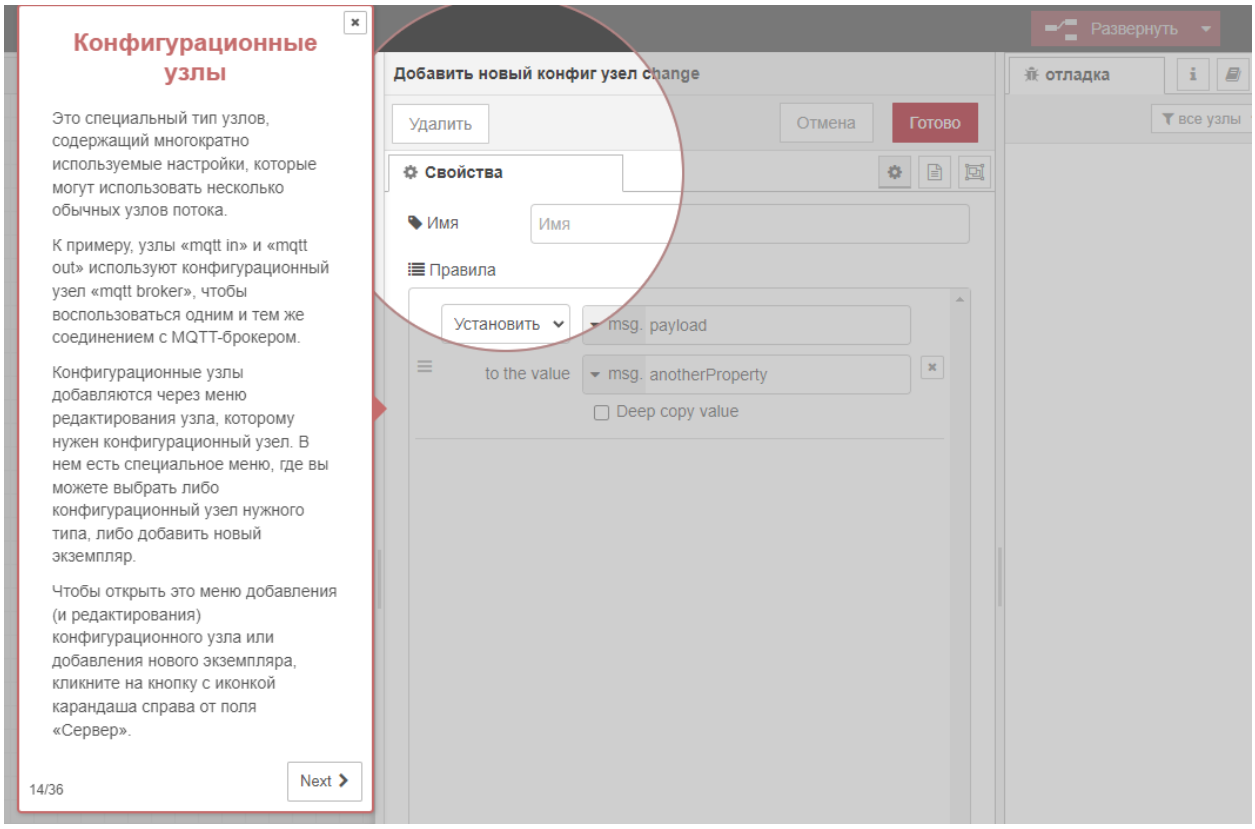


Рис. 1.12 **Конфигурационные узлы**

Это специальный тип узлов, содержащий многократно используемые настройки, которые могут использовать несколько обычных узлов потока.

К примеру, узлы «mqtt in» и «mqtt out» используют конфигурационный узел «mqtt broker», чтобы воспользоваться одним и тем же соединением с MQTT-брокером.

Конфигурационные узлы добавляются через меню редактирования узла, которому нужен конфигурационный узел. В нём есть специальное меню, где вы можете выбрать либо конфигурационный узел нужного типа, либо добавить новый экземпляр.

Чтобы открыть это меню добавления (и редактирования) конфигурационного узла или добавления нового экземпляра, щёлкните на кнопку с иконкой карандаша справа от поля конфигурации.

В меню добавления конфигурационного узла нет секции «Внешний вид», поскольку у конфигурационного узла нет ни иконки, ни портов. Впрочем, две другие секции – «Свойства» и «Описание» – по-прежнему присутствуют.

В левой нижней части этого меню показывается, сколько узлов используют этот конфигурационный узел, а в правой нижней части находится меню для выбора того, какими потоками будет использоваться этот конфигурационный узел. По умолчанию она будет использоваться всеми потоками, но через это меню можно задать использование этого узла каким-то одним потоком.

Для управления всеми конфигурационными узлами можно использовать вкладку «Узлы конфигураций» в боковой панели.

1.14 Подпотоки

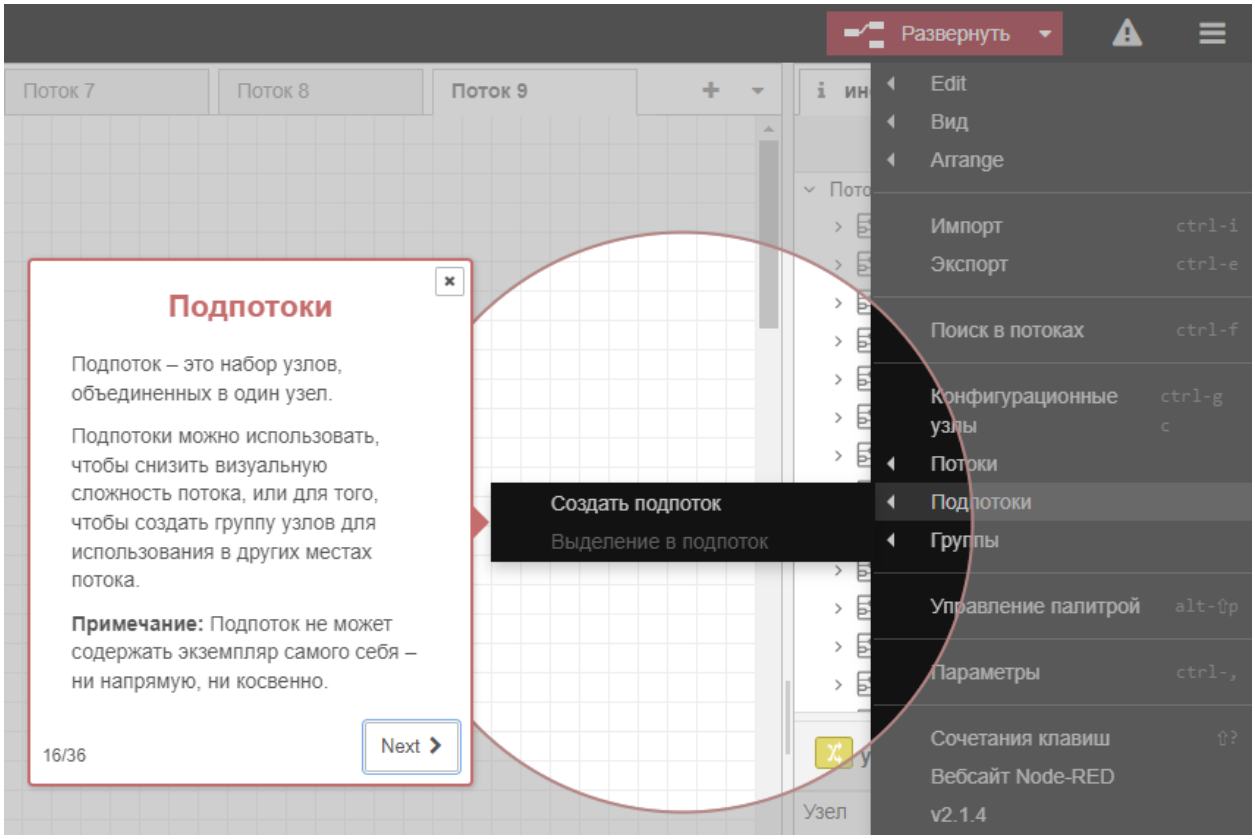


Рис. 1.13 **Создание подпотоков**

Подпоток – это набор узлов, объединённых в один узел.

Подпотоки можно использовать, чтобы снизить визуальную сложность потока, или для того, чтобы создать группу узлов для использования сразу в нескольких потоках.

> Подпоток не может содержать экземпляр самого себя – ни напрямую, ни косвенно.

Рис. 1.14 **Список подпотоков**

Когда вы создадите подпоток, он будет добавлен в палитру существующих узлов. Индивидуальные экземпляры подпотока могут быть добавлены на холст аналогично всем другим узлам.

1.15 Создание пустого подпотока

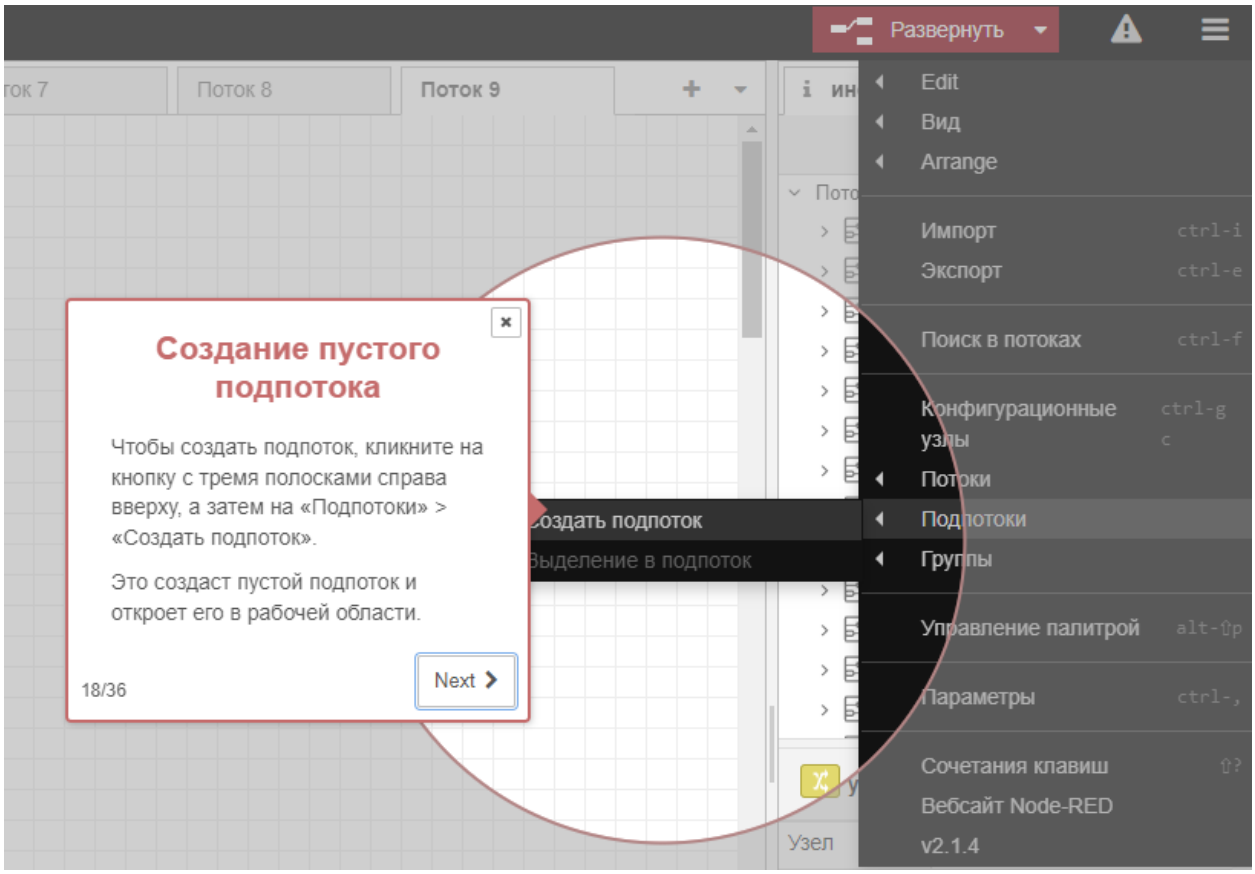


Рис. 1.15 **Создание пустого подпотока**

Чтобы создать подпоток, щёлкните на меню «Гамбургер» справа вверху, а затем на «Подпотоки» > «Создать подпоток». Это создаст пустой подпоток и откроет его в холсте.

1.16 Объединение узлов в подпоток

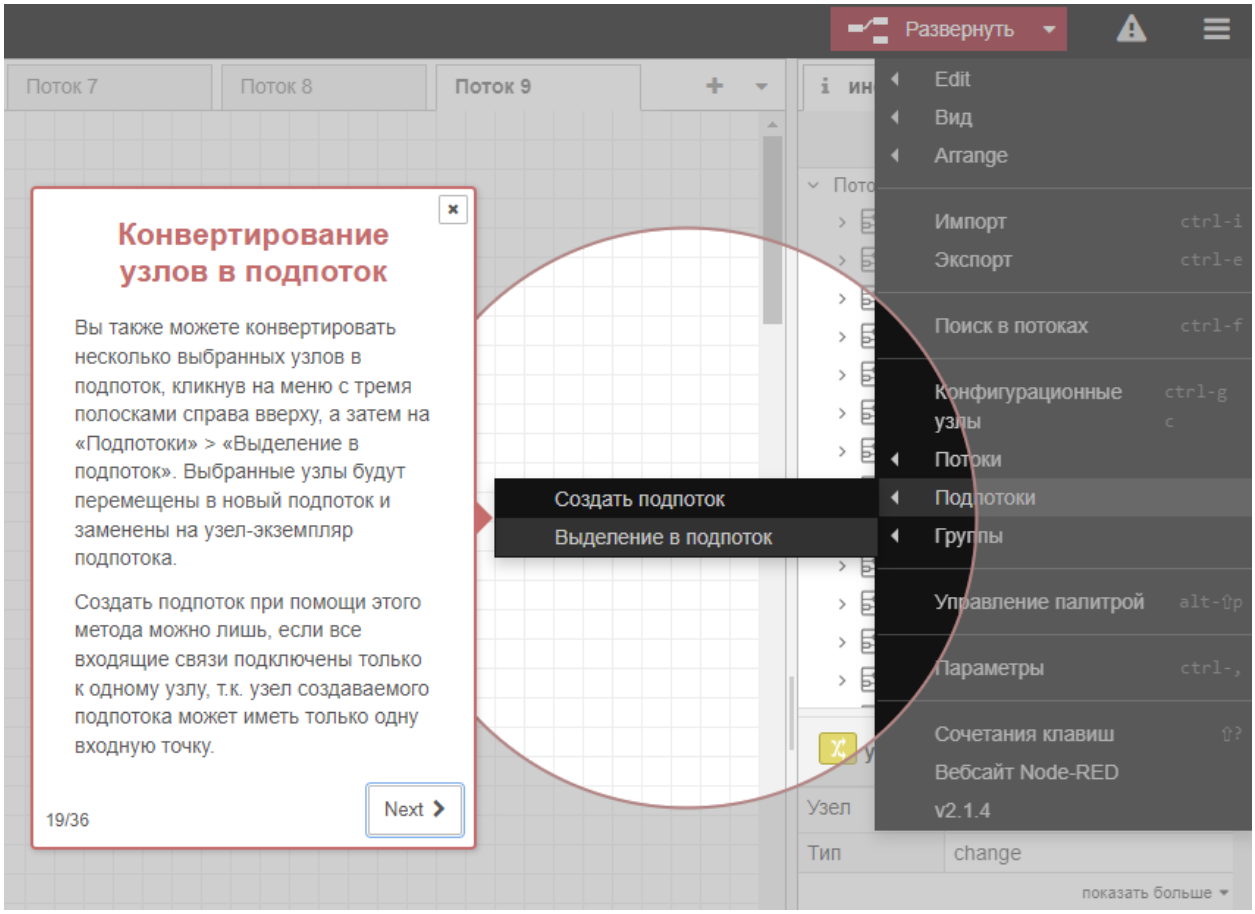


Рис. 1.16 **Объединение узлов в подпоток**

Вы также можете конвертировать несколько выбранных узлов в подпоток, щёлкнув на меню «Гамбургер» справа вверху, а затем на «Подпотоки» > «Выделение в подпоток». Выбранные узлы будут перемещены в новый подпоток и заменены на узел-экземпляр подпотока.

Создать подпоток при помощи этого метода можно лишь, если все входящие связи подключены только к одному узлу, т.к. узел создаваемого подпотока может иметь только одну входную точку.

1.17 Редактирование подпотока

Есть два способа открыть подпоток и начать редактировать его содержимое. Вам нужно либо дважды щёлкнуть мышкой на узел этого подпотока в «палитре», либо нажать на кнопку «Изменить шаблон подпотока» в меню редактирования узла подпотока.

В результате на холсте откроется новая вкладка для этого подпотока. В отличие от вкладок для обычных потоков, у вкладок для подпотоков справа есть крестик, с помощью которого их можно закрыть (спрятать).

1.17.1 Вход и выходы

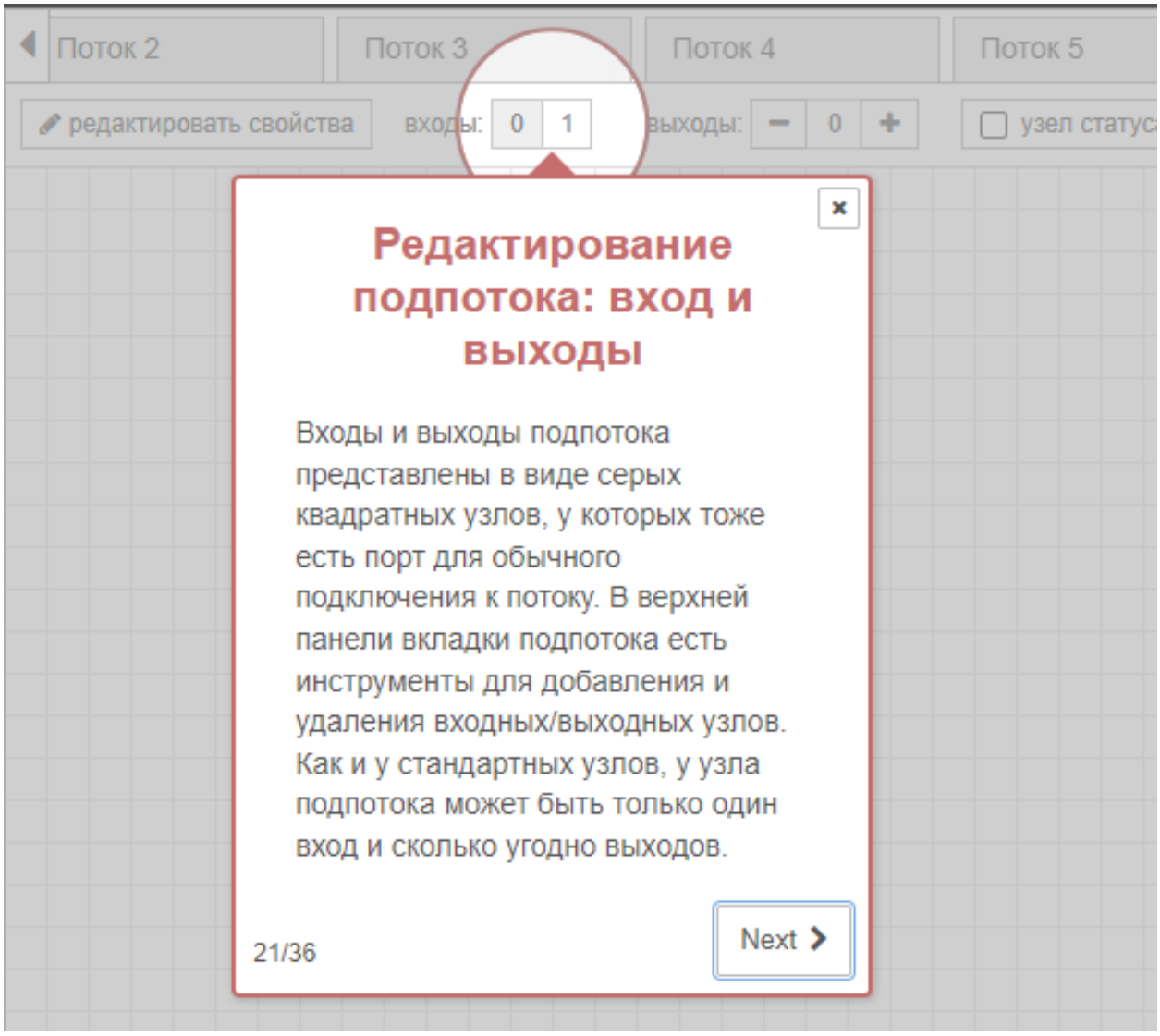


Рис. 1.17 **Вход и выходы подпотока**

Входы и выходы подпотока представлены в виде серых квадратных узлов, у которых тоже есть порт для обычного подключения к потоку. В верхней панели вкладки подпотока есть инструменты для добавления и удаления входных/выходных узлов. Как и у стандартных узлов, у узла подпотока может быть только один вход и сколько угодно выходов.

1.17.2 Свойства подпотока

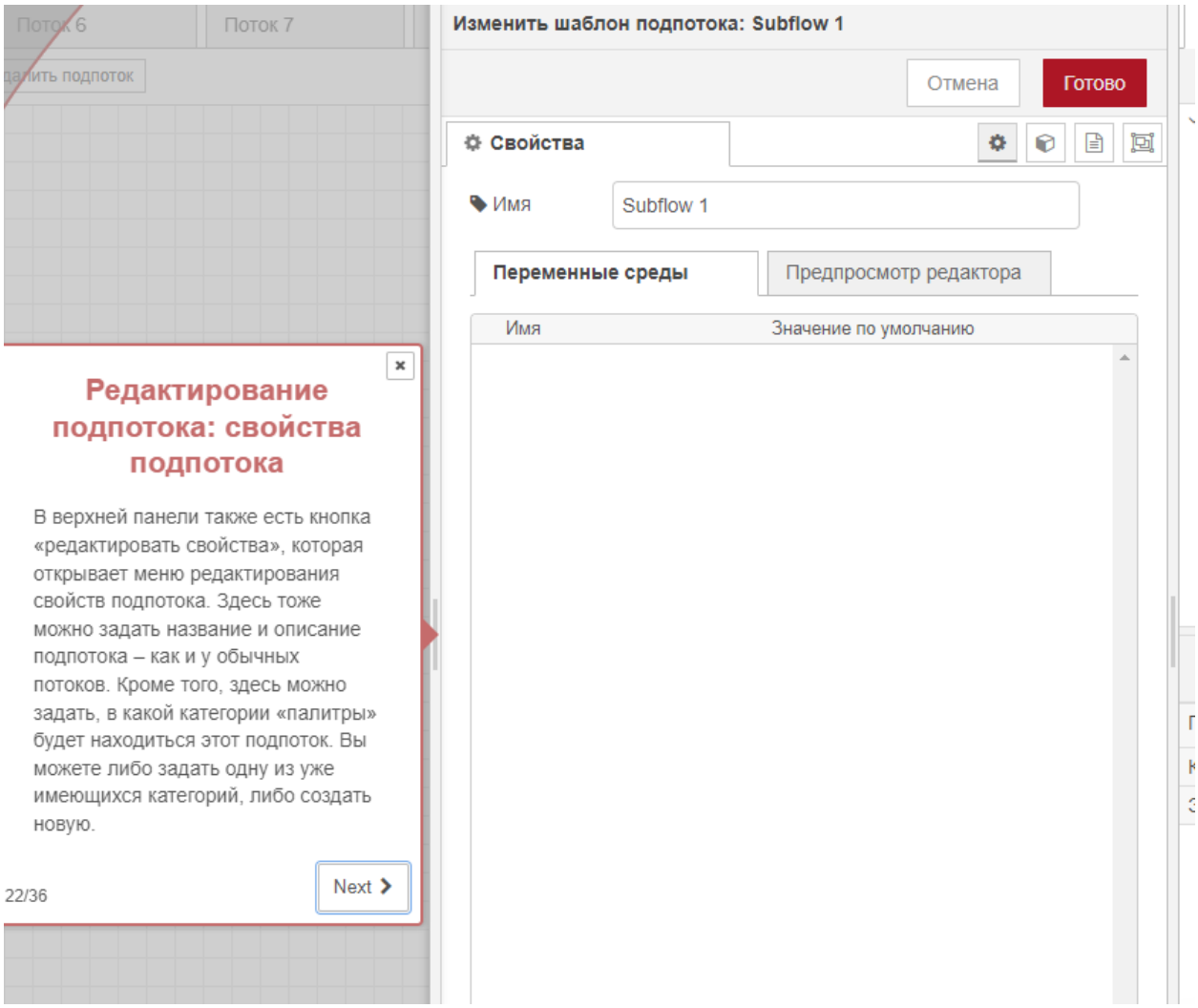


Рис. 1.18 **Свойства подпотока**

В верхней панели также есть кнопка «редактировать свойства», которая открывает меню редактирования свойств подпотока. Здесь тоже можно задать название и описание подпотока – как и у обычных потоков. Кроме того, здесь можно задать, в какой категории «палитры» будет находиться этот подпоток. Вы можете либо задать одну из уже имеющихся категорий, либо создать новую.

1.18 Удаление подпотока

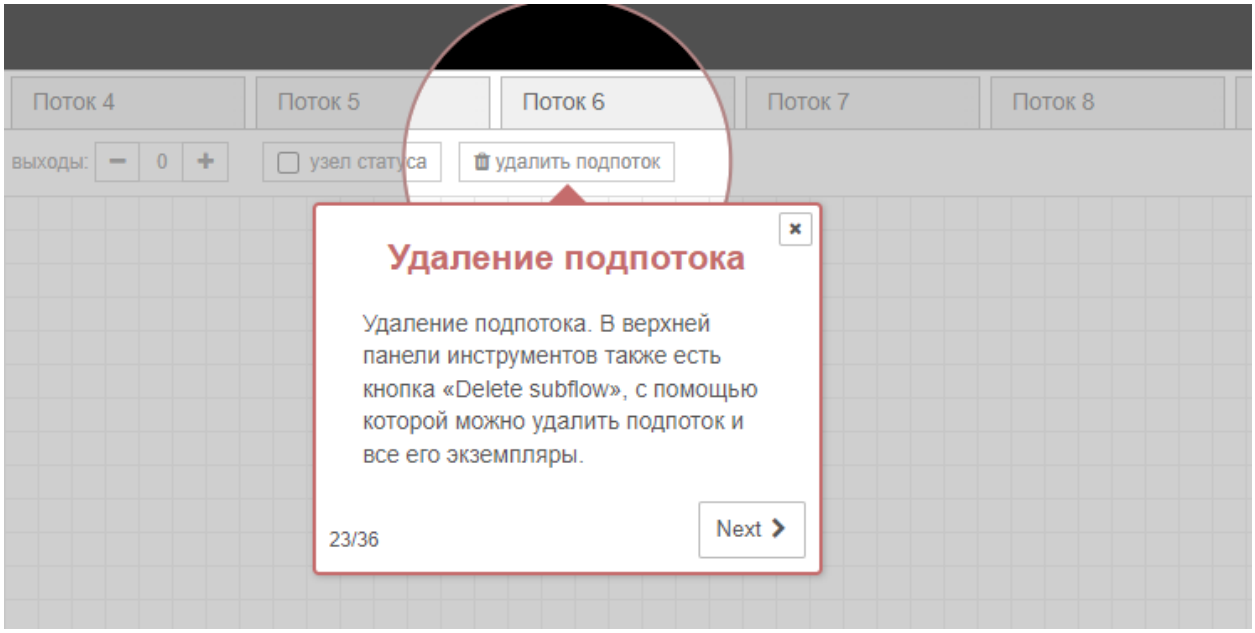


Рис. 1.19 **Удаление подпотока**

В верхней панели инструментов также есть кнопка «Удалить подпоток», с помощью которой можно удалить подпоток и все его экземпляры.

1.19 Импорт и экспорт потоков

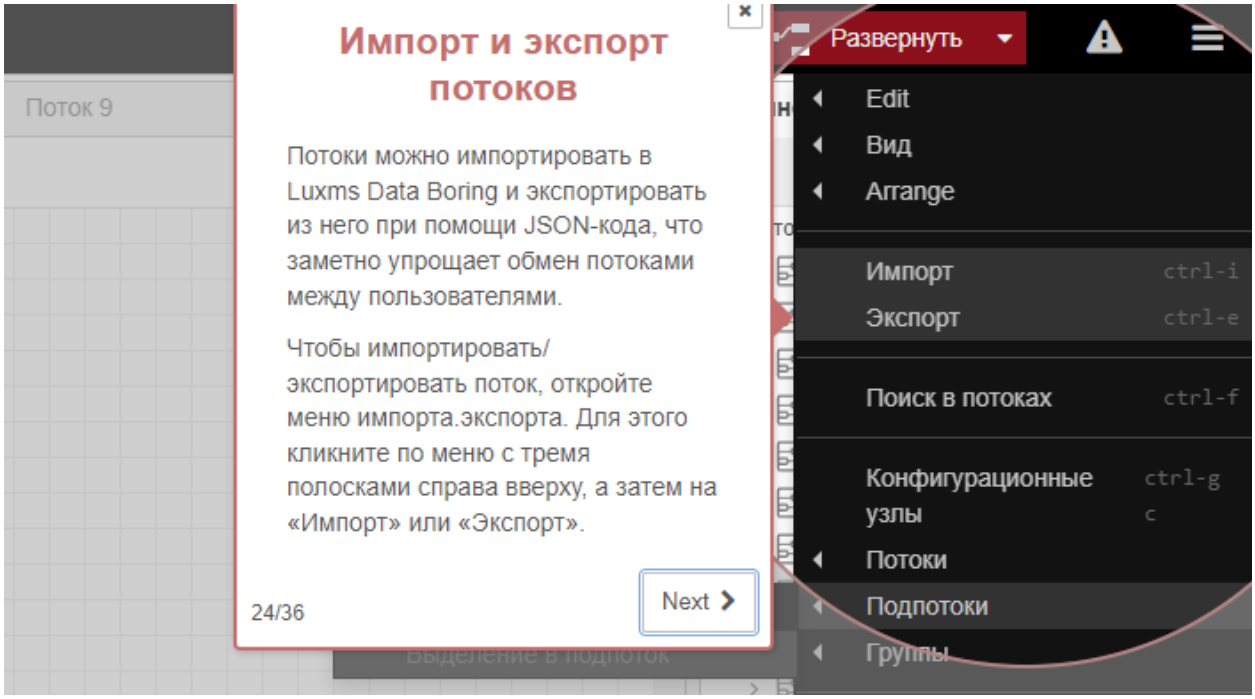


Рис. 1.20 **Импорт и экспорт потоков**

Потоки можно импортировать в Luxms Data Boring и экспортировать из него при помощи JSON-кода, что заметно упрощает обмен потоками между пользователями.

Чтобы импортировать/экспортировать поток, откройте меню импорта/экспорта. Для этого щёлкните по меню «Гамбургер» справа вверху, а затем на «Импорт» или «Экспорт».

1.19.1 Окно импорта

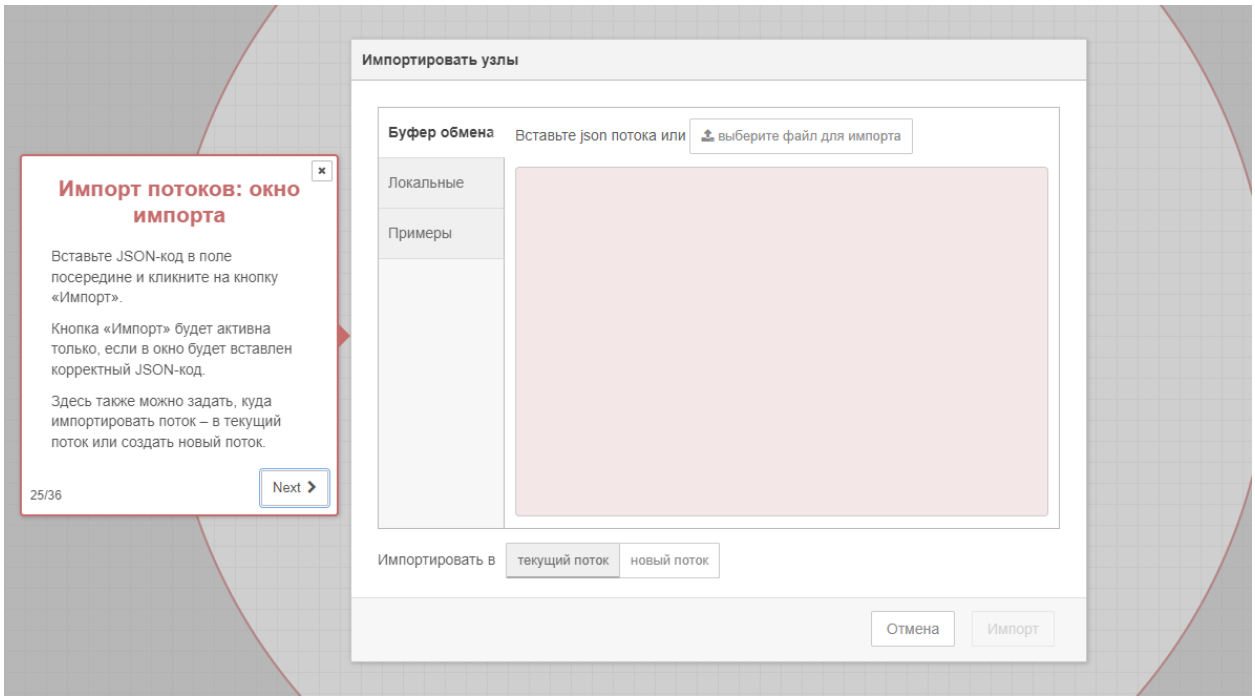


Рис. 1.21 **Окно импорта потока**

Вставьте JSON-код в поле посередине и щёлкните на кнопку «Импорт». Кнопка «Импорт» будет активна только, если в окно будет вставлен корректный JSON-код. Здесь также можно задать, куда импортировать поток – в текущий поток или создать новый поток.

1.19.2 Окно экспорта

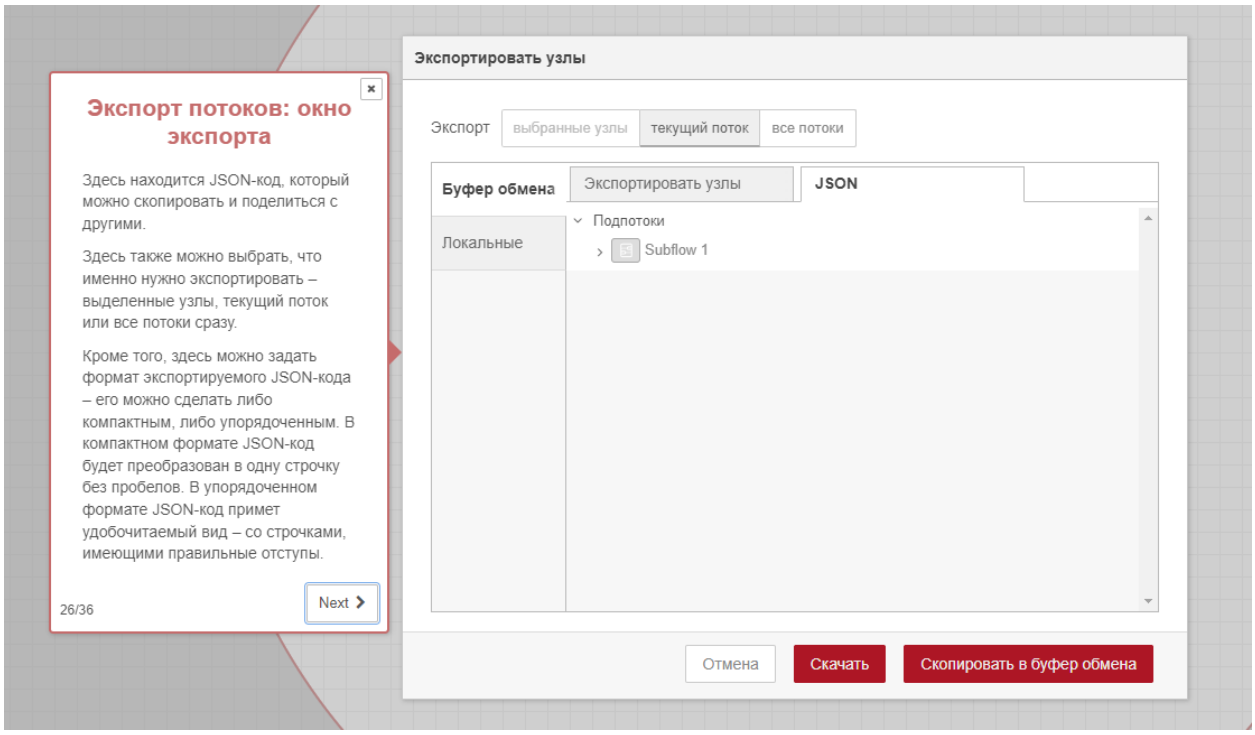


Рис. 1.22 **Окно экспорта потока**

Здесь находится JSON-код, который можно скопировать и поделиться с другими.

Здесь также можно выбрать, что именно нужно экспортировать – выделенные узлы, текущий поток или все потоки сразу.

Кроме того, здесь можно задать формат экспортируемого JSON-кода – его можно сделать либо компактным, либо упорядоченным. В компактном формате JSON-код будет преобразован в одну строчку без пробелов. В упорядоченном формате JSON-код примет удобочитаемый вид – со строчками, имеющими правильные отступы.

1.20 Вкладка «Информация»

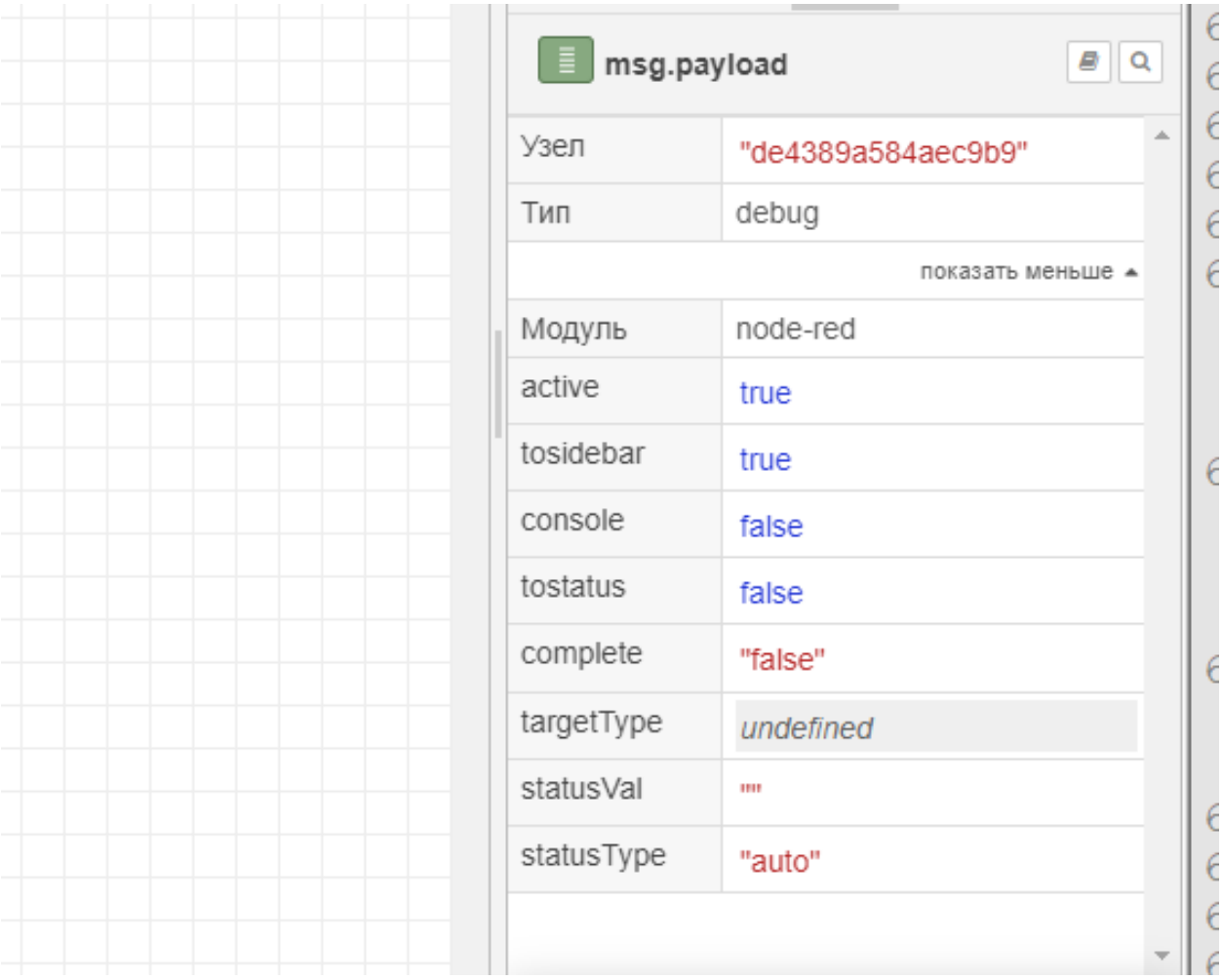


Рис. 1.23 **Вкладка «Информация»**

На этой вкладке показывается информация о свойствах узла или подпотока, выбранного в данный момент.

Если никакого узла не выбрано, в этой вкладке показывается описание текущего потока (его можно поменять в меню редактирования свойств потока). Чтобы открыть его, щёлкните на меню «Гамбургер» справа вверху, а затем на «Потоки» > «Переименовать».

1.21 Вкладка «Справка»

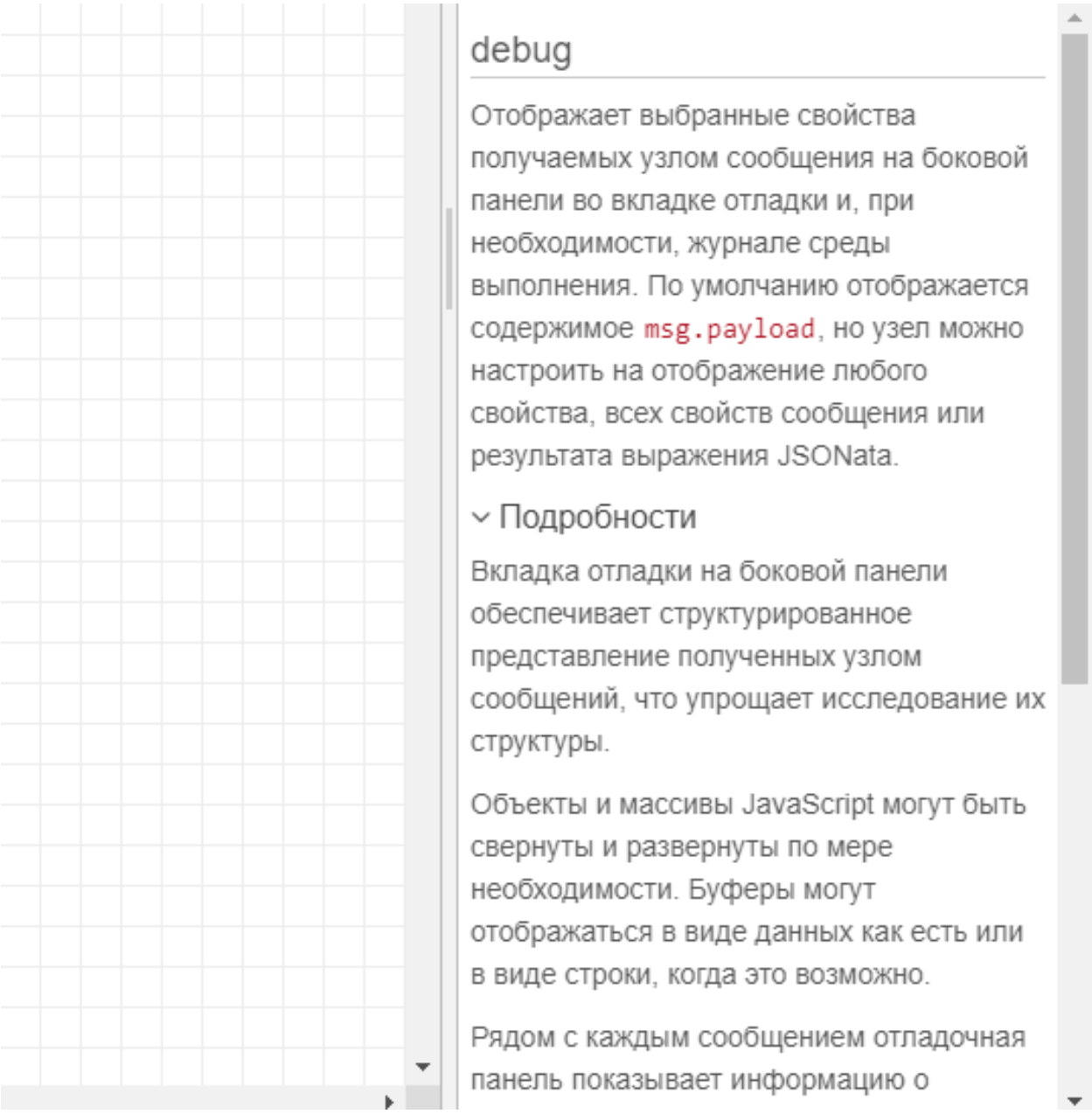


Рис. 1.24 **Вкладка «Справка»**

Эта вкладка содержит вспомогательную справку для узла.

1.22 Вкладка «Отладочные сообщения»

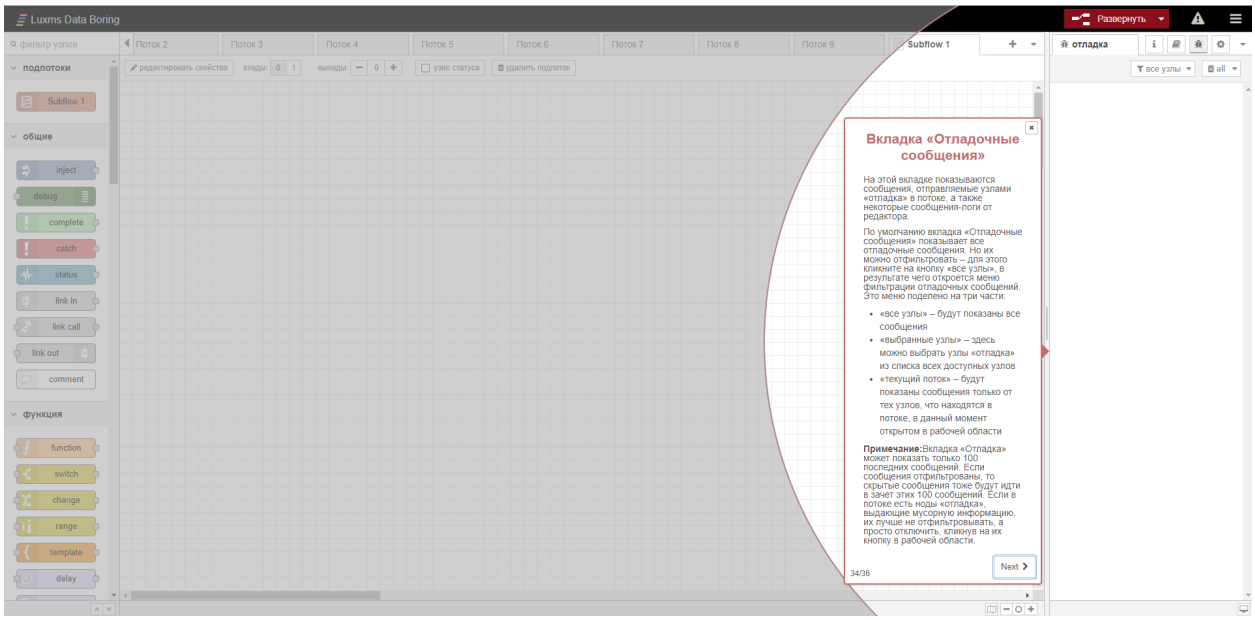


Рис. 1.25 **Вкладка «Отладочные сообщения»**

На этой вкладке показываются сообщения, отправляемые узлами «отладка» в потоке, а также некоторые сообщения от редактора.

По умолчанию вкладка «Отладочные сообщения» показывает все отладочные сообщения. Но их можно отфильтровать – для этого щёлкните на кнопку «все узлы», в результате чего откроется меню фильтрации отладочных сообщений. Это меню поделено на три части:

- «Все узлы» – будут показаны все сообщения.
- «Выбранные узлы» – здесь можно выбрать узлы «отладка» из списка всех доступных узлов.
- «Текущий поток» – будут показаны сообщения только от тех узлов, что находятся в потоке, в данный момент открытом в холсте.

> Вкладка «Отладка» может показать только 100 последних сообщений. Если сообщения отфильтрованы, то скрытые сообщения тоже будут идти в зачет этих 100 сообщений. Если в потоке есть узлы «отладка», выдающие мусорную информацию, их лучше не отфильтровывать, а просто отключить, щёлкнув на их кнопку в холсте.

Кроме того, содержимое вкладки «Отладка» можно в любой момент очистить, нажав на кнопку с иконкой мусорной корзины.

Если нажать на кнопку с иконкой монитора, которая находится в правой нижней части вкладки «Отладка», это откроет вкладку «Отладка» в новом отдельном окне браузера.

2 Описание узлов Data Boring

2.1 Группа узлов “общие”

2.1.1 Узел Запуск

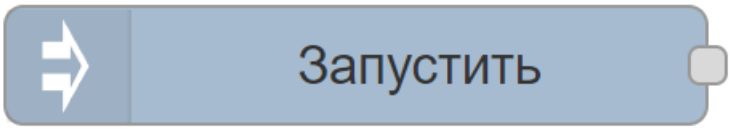


Рис. 2.1 **Узел Запуск в Палитре**

Рис. 2.2 **Узел Запуск на Холсте**

Вбрасывает сообщение в поток вручную или через равные промежутки времени. Данные в сообщении могут быть различных типов, включая строку, объект JavaScript или метку текущего времени.

Выводит:

- `payload` разные – настроенные данные сообщения.
- `topic` строка – необязательное свойство темы сообщения, которое можно настроить в узле.

Узел `Запуск` может инициировать выполнение потока с определенными данными (значение `payload`). Данные по умолчанию – это метка текущего времени в миллисекундах, прошедших с 1 января 1970 года.

Узел также поддерживает вывод строк, чисел, логических значений, объектов JavaScript или значений потоковых/глобальных контекстов.

По умолчанию узел запускается вручную при нажатии его кнопки в редакторе. Его также можно настроить на автоматический запуск через равные промежутки времени или по расписанию.

Он также может быть настроен однократный вброс сообщения при каждом (пере)запуске потоков.

Максимальный интервал, который можно указать, составляет около 596 часов / 24 дней. Однако если Вам нужны интервалы, превышающие один день, Вам следует рассмотреть возможность использования функций планировщика в узле, которые смогут корректно работать с перебоями электроэнергии и перезапусками.

> В параметрах “с интервалом в промежутке” и “в определенное время” используется стандартная система “cron”. Это означает, что “20 минут” будут в следующем часу, 20 минут спустя и 40 минут спустя - а не через 20 минут. Если нужен вброс сообщений каждые 20 минут - используйте параметр “с интервалом”.

> Чтобы включить многострочный текст в строковое значение, необходимо использовать узел `Функция` для формирования данных.

2.1.2 Узел Отладка

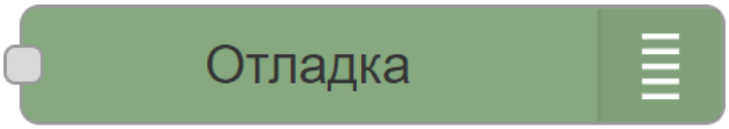


Рис. 2.3 **Узел Отладка в Палитре**

Рис. 2.4 **Узел Отладка на Холсте**

Отображает выбранные свойства получаемых узлом сообщений на боковой панели во вкладке отладки и, при необходимости, журнале среды выполнения. По умолчанию отображается содержимое `msg.payload`, но узел можно настроить на отображение любого свойства, всех свойств сообщения или результата выражения JSONata.

Вкладка отладки на боковой панели обеспечивает структурированное представление полученных узлом сообщений, что упрощает исследование их структуры.

Объекты и массивы JavaScript могут быть свернуты и развернуты по мере необходимости. Буферы могут отображаться в виде данных как есть или в виде строки, когда это возможно.

Рядом с каждым сообщением отладочная панель показывает информацию о времени получения сообщения, узле, который его отправил, и типе данных. Нажатие на идентификатор узла-источника покажет этот узел в рабочей области.

Кнопка на узле может использоваться для включения или отключения вывода информации о получаемых сообщениях. Рекомендуется отключать или удалять любые отладочные узлы, которые не используются.

Узел также может быть настроен на отправку всех сообщений в журнал выполнения или отправку короткого (32 символа) текста в статус под узлом.

2.1.3 Узел Завершение

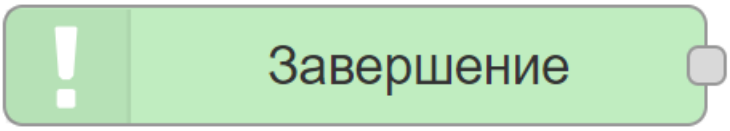


Рис. 2.5 **Узел Завершение в Палитре**

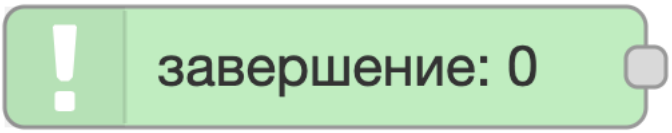


Рис. 2.6 **Узел Завершение на Холсте**

Инициирует запуск потока, когда другой узел завершает обработку сообщения.

Если узел сообщает среде выполнения о завершении обработки сообщения, этот узел можно использовать для запуска второго потока.

Например, его можно использовать вместе с узлом, у которого нет выходного порта, таким как узел Email, для продолжения потока.

Этот узел должен быть настроен на отслеживание событий выбранных узлов в потоке. В отличие от узла `Отлов ошибок`, у него нет режима `Обрабатывать все` для автоматического применения ко всем узлам в потоке.

Не все узлы могут инициировать запуск потока по завершению обработки сообщения. все зависит от того, была ли в них реализована поддержка этой функции, добавленной в Luxms Data Boring, или нет.

2.1.4 Узел Отлов ошибок

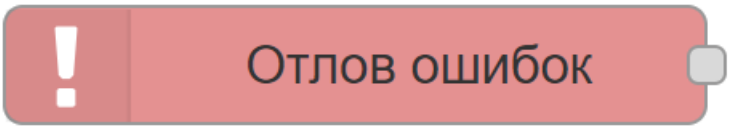


Рис. 2.7 **Узел Отлов ошибок в Палитре**

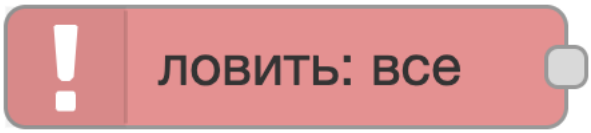


Рис. 2.8 **Узел Отлов ошибок на Холсте**

Ловит ошибки, выбрасываемые узлами на той же вкладке.

Выводит:

- `error.message` строка – сообщение об ошибке.
- `error.source.id` строка – идентификатор узла, выдавшего ошибку.
- `error.source.type` строка – тип узла, выдавшего ошибку.
- `error.source.name` строка – имя узла, выдавшего ошибку, если было настроено.

Если узел выбрасывает ошибку во время обработки сообщения, поток обычно останавливается. Этот узел можно использовать для отлова и обработки таких ошибок с помощью отдельного потока.

По умолчанию узел будет отлавливать ошибки, генерируемые любым узлом на той же вкладке. По желанию он может быть нацелен на определенные узлы или настроен на перехват только тех ошибок, которые не были перехвачены 'нацеленным' `Отлов ошибок` узлом.

Когда выдается ошибка, все соответствующие `Отлов ошибок` узлы получат сообщение.

Если ошибка выбрасывается в подпотоке, она будет обработана любыми `Отлов ошибок` узлами внутри подпотока. Если таковых нет, ошибка будет распространена до вкладки, на которой находится экземпляр подпотока.

Если сообщение уже имеет свойство `error`, оно копируется в `_error`.

2.1.5 Узел Статус

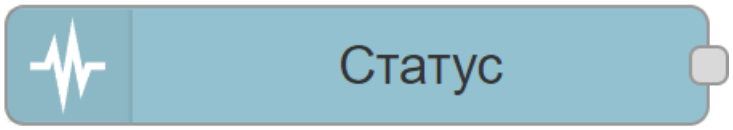


Рис. 2.9 **Узел Статус в Палитре**

Рис. 2.10 **Узел Статус на Холсте**

Инициирует запуск потока при изменении статуса других узлов на той же вкладке.

Выводит:

- `status.text` строка – текст статуса.
- `status.source.type` строка – тип узла, сообщившего о статусе.
- `status.source.id` строка – идентификатор узла, сообщившего о статусе.
- `status.source.name` строка – имя узла, сообщившего о статусе, если настроено.

Этот узел не создает `payload` данные.

По умолчанию узел сообщает о статусах всех узлов на той же вкладке в рабочей области. Его можно настроить на выборочное отслеживание статуса только отдельных узлов.

2.1.6 Узел Связь (вход)

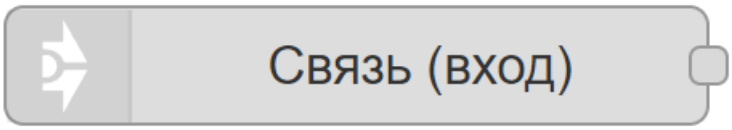


Рис. 2.11 **Узел Связь в Палитре**

Рис. 2.12 **Узел Связь на Холсте**

Создает виртуальный провод между потоками.

Этот узел может быть подключен к любому `Связь (выход)` узлу на любой вкладке. После подключения они ведут себя так, как если бы они были соединены вместе обычным проводом.

Связи между `Связь`-узлами отображаются, только когда выбран один из соединенных `Связь`-узлов. Если есть какие-либо провода, ведущие на другие вкладки, они отображаются в виде виртуального узла, по которому можно кликнуть, чтобы перейти на соответствующую вкладку.

> Провод не может вести внутрь подпотока или изнутри подпотока наружу.

2.1.7 Узел Задать связь

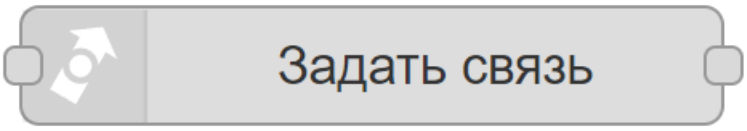


Рис. 2.13 **Узел Задать связь в Палитре**

Рис. 2.14 **Узел Задать связь на Холсте**

Вызывает поток, который начинается с узла `Связь (вход)` и передает ответ.

Этот узел может быть подключен к узлу `Связь (вход)`, который существует в любом потоке. Поток, подключенный к этому узлу, должен заканчиваться узлом `Связь (выход)`, настроенным в режиме `Вернуть к вызывающему узлу`.

Когда этот узел получает сообщение, оно передается к подключенному узлу `Связь (вход)`. Затем он ждет ответа, который затем отправляет.

Если в течение настроенного тайм-аута (по умолчанию 30 секунд) ответ не получен, узел регистрирует ошибку, которую можно перехватить с помощью узла `Отлов ошибок`.

2.1.8 Узел Связь (выход)

Рис. 2.15 **Узел Связь (выход) в Палитре**

Рис. 2.16 **Узел Связь (выход) на Холсте**

Создает виртуальный провод между потоками.

Узел может быть подключен к любому `Связь (вход)` узлу на любой вкладке. После подключения они ведут себя так, как если бы они были соединены вместе обычным проводом.

Связи между `Связь`-узлами отображаются, только когда выбран один из соединенных `Связь`-узлов. Если есть какие-либо провода, ведущие на другие вкладки, они отображаются в виде виртуального узла, по которому можно кликнуть, чтобы перейти на соответствующую вкладку.

> Провод не может вести внутрь подпотока или изнутри подпотока наружу.

2.1.9 Узел Комментарий

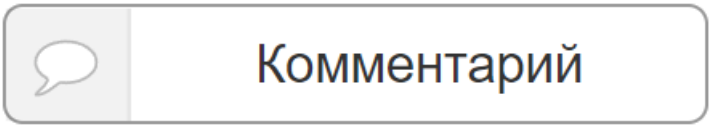


Рис. 2.17 **Узел Комментарий в Палитре**

Рис. 2.18 **Узел Комментарий на Холсте**

Узел, который можно использовать для добавления комментариев к вашим потокам.

Окно редактирования поддерживает синтаксис Markdown. Текст будет отображен на информационной вкладке боковой панели в секции `Описание`.

2.2 Группа узлов “функция”

2.2.1 Узел Функция

Рис. 2.19 **Узел Функция в Палитре**

Рис. 2.20 **Узел Функция на Холсте**

Исполняет JavaScript функцию (введенную в настройках, во вкладке `Функция`) для всех получаемых узлом сообщений.

Сообщения передаются в виде объекта JavaScript с именем msg.

Обычно у объекта есть свойство `msg.payload`, содержащее тело сообщения.

Ожидается, что функция вернет объект сообщения (или несколько объектов сообщения), которые будут отправлены следующим узлам в потоке, но может также ничего не возвращать, чтобы остановить поток.

Код настройки, выполняемый один раз при запуске сервера или при развертывании новой конфигурации потока, можно ввести во вкладке `Настройка`. Также во вкладке `Закрытие` можно ввести код очистки, выполняемый при остановке или повторном развертывании узла.

Если код настройки возвращает `Promise` объект, тогда обработка входящих сообщений узлом начнется после его завершения.

2.2.1.1 Отправка сообщений

Функция может либо вернуть сообщения, которые она хочет передать следующим узлам в потоке, либо вызвать `node.send` (сообщения).

Она может вернуть/отправить:

- Один объект сообщения - передается узлам, подключенным к первому порту выхода.
- Массив объектов сообщений - передается на узлы, подключенные к соответствующим портам выхода

> Код настройки выполняется во время инициализации узлов. Таким образом, если на вкладке настройки кода вызывается `node.send`, последующие узлы могут не получить это сообщение.

Если какой-либо элемент массива сам является массивом сообщений, тогда на соответствующий выход отправляется несколько сообщений.

Если возвращен `null`, либо сам по себе, либо как элемент массива, тогда сообщение не передается.

2.2.1.2 Ведение журнала и обработка ошибок

Для добавления информации в журнал или сообщения об ошибке доступны следующие функции:

- `node.log` (“Сообщение для журнала”)
- `node.warn` (“Предупреждение”)
- `node.error` (“Ошибка”)

Также узел `Отлов ошибок` может использоваться для обработки ошибок. Чтобы можно было ловить оповещения об ошибке, при вызове `node.error` передайте msg в качестве второго аргумента:

- `node.error` (“Ошибка”, `msg`);

2.2.1.3 Доступ к информации об узле

В функции можно обращаться к идентификатору и имени узла, используя следующие свойства:

- `node.id` - идентификатор узла
- `node.name` - имя узла

2.2.1.4 Использование переменных среды

Доступ к переменным среды можно получить с помощью:

- `env.get("MY_ENV_VAR")`.

2.2.2 Узел Переключатель

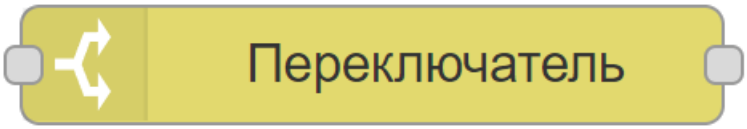


Рис. 2.21 **Узел Переключатель в Палитре**

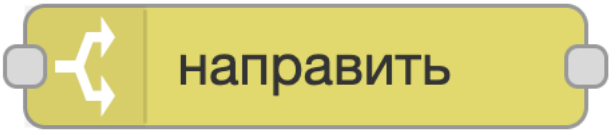


Рис. 2.22 **Узел Переключатель на Холсте**

Направляет сообщения по разным ветвям потока, в завсимости от значений свойств сообщения или его позиции в последовательности.

Когда приходит сообщение, узел выполняет проверку соответствия сообщения каждому из установленных правил и перенаправляет сообщение на соответствующие выходы.

При желании узел может быть настроен на прекращение проверки правил после того как найдено первое подходящее.

Правила могут применяться к свойству сообщения, свойству потокового или глобального контекста, переменной среды или результату выражения JSONata.

2.2.2.1 Правила

1. Правила категории `value rules` сравнивают значение указанного свойства сообщения
2. Правила категории `sequence rules` могут применяться для последовательностей сообщений, таких как сгенерированные узлом `Разделить`
3. Выражение JSONata исполняется для сообщения и считается подходящим, если оно возвращает истинное значение
4. Правило иначе используется для сообщений, не подходящих ни под одно из предыдущих правил.

> Правила равно *true/false* и равно *null* проводят строгое сравнение с этими типами. Они не конвертируются между типами.

Правила пустое и не пустое могут быть использованы для проверки длины строк, массивов и буферов, или количества свойств, которые содержит объект. Эти правила не подходят, если тестируемое свойство содержит значение логического типа, `null` или `undefined`.

2.2.2.2 Обработка последовательностей сообщений

По умолчанию узел не изменяет свойство `msg.parts` у сообщений, являющихся частью последовательности.

Можно включить параметр **пересоздавать последовательности сообщений** для создания новых последовательностей сообщений для каждого соответствующего правила. В этом режиме узел будет помещать в буфер всю входящую последовательность перед отправкой новых последовательностей. Настройка `nodeMessageBufferMaxLength` может быть использована для ограничения количества сообщений в буфере.

2.2.3 Узел Замена

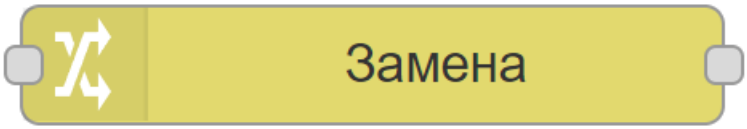


Рис. 2.23 **Узел Замена в Палитре**

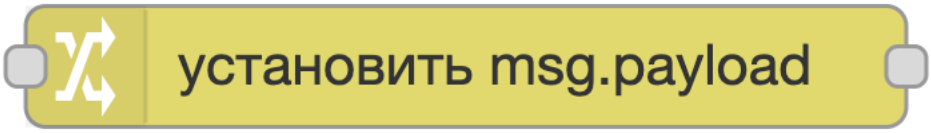


Рис. 2.24 **Узел Замена на Холсте**

Устанавливает, изменяет, удаляет или перемещает свойства сообщения, контекста потока или глобального контекста.

В узле можно указать несколько операций, которые будут применяться в том порядке, в котором они заданы.

Доступные операции:

- `Установить` – устанавливает свойство. Значение может быть различных типов или может быть взято из существующего свойства сообщения или контекста.
- `Изменить` – ищет и заменяет части текста в свойстве сообщения. Если используется регулярное выражение, настройка `"заменить на"` может включать группы захвата, например `$1`. При полном совпадении заменяет только тип.
- `Удалить` – удаляет свойство.
- `Переместить` – перемещает или переименовывает свойство.

Тип свойства `"выражение"` использует язык запросов и выражений JSONata.

2.2.4 Узел Диапазон

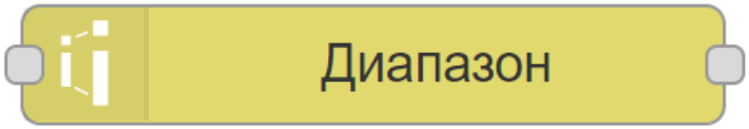


Рис. 2.25 **Узел Диапазон в Палитре**

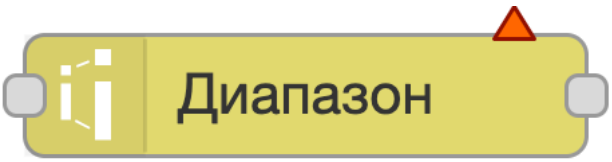


Рис. 2.26 **Узел Диапазон на Холсте**

Соотносит значение одного числового диапазона с другим числовым диапазоном.

Принимает:

- `payload` число – данные должны быть числом. Данные другого типа будут преобразованы в число и отклонены, если это не удастся.

Выводит:

- `payload` число – значение, соотнесенное с новым диапазоном.

Этот узел будет линейно масштабировать полученное значение. По умолчанию результат не ограничен диапазоном, определенным в узле.

Масштабировать и ограничить целевым диапазоном означает, что результат никогда не будет выходить за пределы диапазона, указанного в целевом диапазоне.

Масштабировать и обернуть в целевой диапазон означает, что результат будет обернут вокруг целевого диапазона:

Например, если входные 0 - 10 сопоставляются с 0 - 100:

режим	вход	выход
масштабировать	2	20
ограничить	2	20
обернуть	2	20
масштабировать	12	120
ограничить	12	100
обернуть	12	20
масштабировать	18	180
ограничить	18	100
обернуть	18	80

Рис. 2.27 **Целевой диапазон**

2.2.5 Узел Шаблон

Рис. 2.28 **Узел Шаблон в Палитре**

Рис. 2.29 **Узел Шаблон на Холсте**

Генерирует текст по установленному шаблону и присваивает его выбранному свойству сообщения.

Принимает:

- `msg`, объект – объект сообщения `msg`, содержащий информацию для заполнения шаблона.
- `шаблон`, строка – шаблон текста для заполнения данными из сообщения `msg`. Если шаблон не введен на панели редактирования, тогда он может быть установлен через свойство `msg.template`.

Выводит:

- `msg`, объект – сообщение msg, у которого выбранному свойству присвоено тексктовое значение, полученное после заполнения шаблона данными входящего сообщения.

По умолчанию используется формат `mustache`. При необходимости это можно отключить.

Например, когда шаблон:

```
Привет, {{`payload`.name}}. Сегодня {{date}}.
```

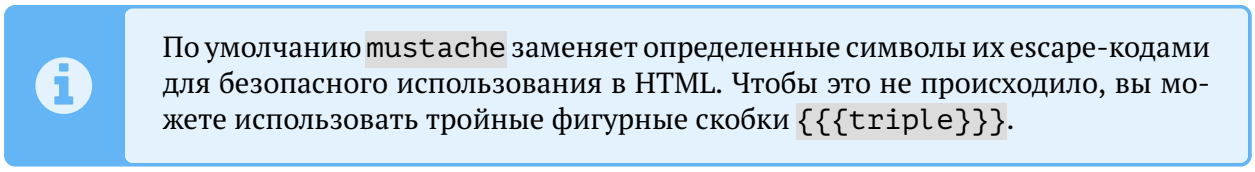
получает сообщение msg, содержащее:

```
{
  date: "понедельник",
  `payload`: {
    name: "Иван"
  }
}
```

Выбранному свойству будет присвоен текст:

```
Привет, Иван. Сегодня понедельник.
```

Можно использовать свойство из контекста потока или глобального контекста. Просто используйте `{{flow.name}}` или `{{global.name}}`, или для постоянного хранилища `store` используйте `{{flow[store].name}}` или `{{global[store].name}}`.



По умолчанию `mustache` заменяет определенные символы их escape-кодами для безопасного использования в HTML. Чтобы это не происходило, вы можете использовать тройные фигурные скобки `{{{triple}}}`.

2.2.6 Узел Задержка

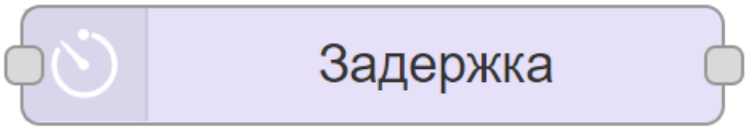


Рис. 2.30 **Узел Задержка в Палитре**

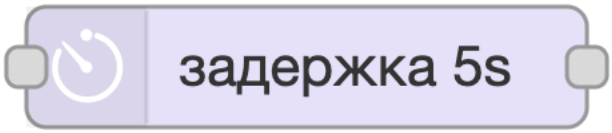


Рис. 2.31 **Узел Задержка на Холсте**

Задерживает каждое сообщение, проходящее через узел, или ограничивает скорость, с которой они могут проходить.

Принимает:

- `delay`, число – устанавливает задержку в миллисекундах, которая будет применена к сообщению. Этот параметр применяется только в том случае, если узел настроен так, чтобы разрешать сообщению переопределять установленный интервал задержки по умолчанию.
- `reset` – если в полученном сообщении этому свойству присвоено какое-либо значение, все ожидающие сообщения, удерживаемые узлом, сбрасываются без дальнейшей отправки.
- `flush` – если в полученном сообщении этому свойству присвоено какое-либо значение, все ожидающие сообщения, удерживаемые узлом, немедленно отправляются далее.

Когда узел настроен на задержку сообщений, интервал задержки может быть фиксированным значением, случайным значением в пределах диапазона или динамически установленным для каждого сообщения. Каждое сообщение задерживается независимо от любых других сообщений, основываясь на времени его прибытия в узел.

Когда узел настроен на ограничение скорости сообщений, их доставка распределяется по установленному периоду времени. Статус показывает количество сообщений, находящихся в данный момент в очереди. При необходимости узел может быть настроен на отбрасывание промежуточных сообщений по мере их поступления.

Ограничение скорости может применяться ко всем сообщениям или группам сообщений в соответствии с их значением темы `msg.topic`. При группировании промежуточные сообщения автоматически отбрасываются. В каждом интервале времени узел может либо выпустить самое последнее сообщение для всех тем, либо выпустить самое последнее сообщение для следующей темы.

2.2.7 Узел Триггер

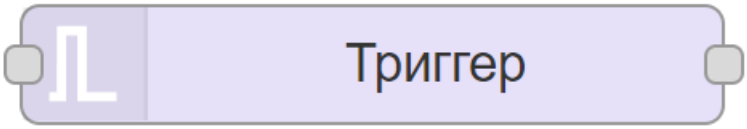


Рис. 2.32 **Узел Триггер в Палитре**

Рис. 2.33 **Узел Триггер на Холсте**

При срабатывании может отправить сообщение, а затем дополнительно второе сообщение, если не было сделано продление или сброс.

Принимает:

- `delay`, число – устанавливает задержку в миллисекундах, которая будет применяться к сообщению. Этот параметр применяется только в том случае, если узел настроен так, чтобы сообщение могло отменять настроенный интервал задержки по умолчанию.
- `reset` – при получении сообщения с этим свойством, любой тайм-аут или повтор, находящиеся в обработке в текущий момент, будут сброшены, а сообщение не сработает.

Этот узел можно использовать для создания тайм-аута внутри потока. По умолчанию, когда узел получает сообщение, он отправляет сообщение с `payload`, равным `1`. Затем он ждет 250 мс, прежде чем отправить второе сообщение с `payload` равным `0`. Это можно использовать, например, для мигания светодиодом, подключеным к выходу *GPIO* у *Raspberry Pi*.

Данные каждого отправленного сообщения могут быть настроены на различные значения, включая возможность ничего не отправлять. Например, если установить первоначальное сообщение на ничего и выбрать опцию продления таймера с каждым новым сообщением, узел будет действовать как сторожевой таймер, отправляя сообщение только в том случае, если ничего не получено в течение установленного интервала.

Если установлен тип строка, узел поддерживает синтаксис шаблона `mustache`.

Задержка между отправкой сообщений может быть изменена с помощью `msg.delay`, если эта опция включена в узле. Значение должно быть указано в миллисекундах.

Если узел получает сообщение со свойством `reset` или `payload`, который совпадает с настроенным в узле, любой тайм-аут или повтор, находящийся в обработке в текущий момент, будет сброшен, и сообщение не сработает.

Узел может быть настроен на повторную отправку сообщения с регулярным интервалом, пока не будет произведен сброс отправкой на него сообщения.

При желании узел может быть настроен на обработку сообщений, как если бы они были отдельными потоками, используя свойство `msg` для идентификации каждого потока. По умолчанию `msg.topic`.

Статус показывает - активен ли узел в данный момент. Если работает в многопоточном режиме, статус показывает количество удерживаемых потоков.

2.2.8 Узел Выполнить

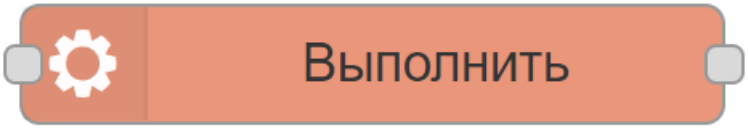


Рис. 2.34 **Узел Выполнить в Палитре**

Рис. 2.35 **Узел Выполнить на Холсте**

Запускает системную команду и возвращает ее вывод.

Узел может быть настроен либо на ожидание завершения выполнения команды, либо на отправку выходных данных по мере их генерации в ходе выполнения.

Выполняемая команда может быть установлена в настройках узла или полученным сообщением.

Принимает:

- `payload` строка – будет добавлено к выполняемой команде, если узел настроен так делать.
- `kill` строка – тип сигнала уничтожения для отправки существующему процессу узла `Выполнить`.
- `pid` число, строка – идентификатор существующего процесса узла `Выполнить` для уничтожения.

Выводит:

1. Стандартный вывод:

- `payload` строка – стандартный вывод команды.
- `rc` объект – только в `exec` режиме, копия объекта кода возврата (также доступна по 3-му порту).

2. Стандартный вывод ошибок:

- `payload` строка – стандартная ошибка команды.
- `rc` объект – только в `exec` режиме, копия объекта кода возврата (также доступна по 3-му порту).

3. Код возврата

- `payload` объект – объект, содержащий код возврата и, возможно, свойства `message`, `signal`.

По умолчанию используется системный вызов `exec`, который вызывает команду, ожидает ее завершения и возвращает результат. Например, успешная команда должна иметь код возврата `{code: 0}`.

При желании вместо этого можно использовать `spawn`, который возвращает выходные данные из `stdout` и `stderr` по ходу выполнения команды, обычно по одной строке за раз. После завершения он возвращает объект на 3-й порт. Например, успешная команда должна вернуть `{code: 0}`.

Ошибки могут возвращать на 3-й порт дополнительную информацию в `msg.payload`, такую как строка `message`, строка `signal`.

Выполняемая команда настраивается в узле, с возможностью добавления к ней `msg.payload` и дополнительного набора параметров.

Команды или параметры с пробелами должны быть заключены в кавычки - `"Это один параметр"`.

Возвращаемый `payload` обычно представляет собой строку, пока не обнаружены символы, отличные от `UTF8`, в этом случае возвращаемое значение будет иметь тип буфер.

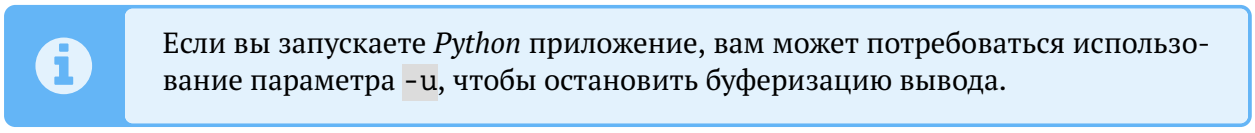
Значок статуса узла и `PID` будут видны, когда узел активен. Изменения в статусе можно отслеживать узлом `Статус`.

2.2.8.1 Уничтожения процессов

Отправка `msg.kill` уничтожит один активный процесс. `msg.kill` должен быть строкой, содержащей тип передаваемого сигнала, например, `SIGINT`, `SIGQUIT` или `SIGHUP`. По умолчанию будет передан `SIGTERM`, если задана пустая строка.

Если узлом запущено более одного процесса, тогда в `msg.pid` также должно быть установлено значение `PID` процесса для уничтожения.

Если в поле `Тайм-аут` введено значение, тогда, если процесс не завершится по истечении указанного количества секунд, он будет автоматически уничтожен.



Если вы запускаете *Python* приложение, вам может потребоваться использование параметра `-u`, чтобы остановить буферизацию вывода.

2.2.9 Узел Фильтр

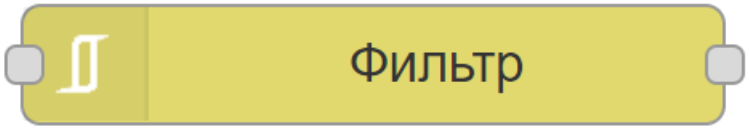


Рис. 2.36 **Узел Фильтр в Палитре**

Рис. 2.37 **Узел Фильтр на Холсте**

Report by Exception (RBE) узел - передает данные только в том случае, если `payload` изменился. Он также может блокировать или игнорировать при изменениях `payload` на определённую величину (Dead- и Narrowband режимы).

Принимает:

- `payload` – числа, строки и простые объекты. Другие режимы должны предоставлять данные, приводимые к числу.
- `topic` – если указано, функция будет работать для каждого `topic`. Это свойство может быть установлено конфигурацией.
- reset – если установлено, очищает сохраненное значение для указанного `msg.topic` или всех `topic`, если `msg.topic` не указан.

Выводит:

- `payload` – при срабатывании выход будет таким же, как и вход.

В режиме RBE этот узел будет блокироваться до тех пор, пока значение `msg.payload` (или выбранное Свойство) не будет отличаться от предыдущего. При необходимости он может игнорировать начальное значение, чтобы ничего не отправлять при запуске.

Режим Deadband будет блокировать входящее значение, пока его изменение не будет больше или больше-равно, чем ± ширина запрещенной зоны от предыдущего значения.

Narrowband режим блокирует входящее значение, если его изменение больше или больше-равно, чем ± ширина запрещенной зоны от предыдущего значения. Это полезно для игнорирования выбросов, например.

Оба режима «Deadband» и «Narrowband» требуют, чтобы входящее значение содержало значение, приводимое к числу, и оба также поддерживают `%` - условие срабатывает только в том случае, когда вход отличается более чем на x% от исходного значения.

И Deadband, и Narrowband позволяют сравнивать либо предыдущее допустимое выходное значение, таким образом игнорируя любые значения, выходящие за пределы диапазона, либо предыдущее входное значение, которое сбрасывает заданное значение, тем самым позволяя постепенное смещение (deadband) или поэтапное изменение (narrowband).

Это работает для каждого сообщения `msg.topic`, хотя при желании его можно изменить на другое `Свойство`. Это означает, что один rbe-узел может одновременно обрабатывать несколько разных `topics`.

2.2.10 Узел loop

Рис. 2.38 **Узел loop в Палитре**

Рис. 2.39 **Узел loop на Холсте**

Создание циклов — распространенный метод программирования. Узел `loop` обеспечивает их поддержку в `Databoring`. Можно выбрать один из трех видов создания циклов:

- Цикл с фиксированным количеством повторений;
- Цикл, основанный на условии;
- Цикл, повторяющийся на итерируемом перечислении.

2.2.10.1 Общее описание

Можно управлять циклом, отправив на вход `msg.command`. Если он содержит строку `break`, цикл будет завершен немедленно. Если он содержит строку перезапуска, цикл начнется снова с первого значения. У узла есть два выхода: конец цикла и шаг цикла. Первый выход используется когда цикл заканчивается, то есть в конце. Второй выход следует подключить к входу потока, необходимого для повторения, для создания цикла.

Узлы в повторяемом потоке получают объект msg.loop со следующим параметром:

- index - счетчик увеличиваемый на 1 с 0.

Может содержаться и большее количество свойств, но они зависят от типа цикла, они будут описаны ниже. Также можно выбрать одно из свойств, которое будет помещено в msg.payload, для этого используйте параметр `Loop Payload` в форме редактора.

В конце вывода цикла вы получите объект `msg.loop` со следующими свойствами:

- `break` - истина, если цикл получил команду остановки;
- `timeout` - true, если время ожидания истекло;
- `pass.total` — общее количество проходов по циклу, включая все перезапуски;

- `pass.last` — количество проходов по циклу с момента последнего рестарта или старта;
- `restarts` - количество перезапусков цикла.

А также в конце вывода цикла будет выведен `msg.payload`. Содержащееся значение внутри сообщения - опционально. Можно выбрать между выводом последнего payload из цикла и исходного `payload`, который узел получил на входе в начале, для этого выбора используйте `End Payload` в форме редактора.

Можно ограничить время циклов для всех трех типов. Для этого используйте `Time Limit` в форме редактора или `msg.limit` для передачи “на лету”. Время указывается в миллисекундах.

2.2.10.2 Тип цикла «Fixed Count» (Фиксированное значение)

Даннный тип цикла используется, когда необходимо произвести фиксированное количество повторений.

- Поле `Count` указывает количество проходов через цикл (повторений), его так же можно оставить пустым, при этом используя `msg.count` на лету;
- Поле `Initial Value` определяет значение, от которого счетчик должен вести отсчет, его так же можно оставить пустым, при этом используя `msg.initial` на лету;
- В поле `Step Value` указывается шаг приращения или уменьшения для счетчика.

Узлы в повторяемом потоке, получат объект msg.loop со следующими дополнительными свойствами:

- `value` - значение счетчика;
- `count` - запрошенное количество проходов через цикл;
- `initial` - значение, с которого начался счетчик;
- `step` - значение шага, используемое для увеличения или уменьшения счетчика.

2.2.10.3 Тип цикла «Condition» (Условие)

Используйте этот тип цикла, когда вам нужен цикл, основанный на каком-либо условии, для этого необходимо определить условие в поле `Condition`. Можно выбрать один из трех языков:

- `JavaScript` — условие, написанное в виде кода на языке `JavaScript`. Оценка состояния будет использовать результат последней части. Цикл будет продолжаться, если это правда, иначе цикл завершится. Например:

```
msg.payload !== "done"
roses = global.get("roses"); global.set("roses", ++roses); roses <= 100
```

Код `JavaScript` выполняется в песочнице с ограниченной средой, аналогичной функциональному узлу.

- `JSONata` — условие в виде выражения `JSONata`. Цикл продолжится, если выражение оценивается как истинное, в противном случае цикл завершится. Например:

```
msg.payload in [1,2,3,5,7,13,21]
$globalContext("status") or msg.size > 10
```

- `Regex` — регулярное выражение, используемое с функцией test(), применяемой к `msg.payload`. Цикл будет продолжаться, если он сообщение подходит по условию, в противном случае цикл завершится. Например:

```
file[0-9].txt
^[a-zA-Z0-9]+: .*
```

Следующие три примера выполняют одну и ту же работу (проверяют, заканчивается ли строка символом «n») на разных языках с разной скоростью:

`msg.payload.substr(-1) != "n"` `$substring(msg.payload, -1) != "n"` `.*[ñ]`

В данном случае `JavaScript` является самым быстрым из вариантов, а `Regex` — самым медленным, но с более сложными условиями все может измениться. При этом, не все условия можно легко или вообще возможно описать одним определенным языком. Например, регулярное выражение применимо не только для сравнения строк, но также это лучший выбор для поиска сложных шаблонов.

`msg.loop` не имеет дополнительных свойств для этого типа цикла.

2.2.10.4 Тип цикла «Enumeration» (Перечисление)

Используйте этот тип цикла, когда вы хотите выполнить итерацию по некоторому перечислению. Тип объекта должен быть итерируемым, например: массив, список, словарь, множество или строка.

Вам необходимо указать перечисление в поле Enumeration как свойство сообщения, потока или глобального контекста. Также можно указать его напрямую как `JSON` или обычную строку. Узлы в повторяемом потоке, получат объект `msg.loop` со следующими дополнительными свойствами:

- `value` - повторяющееся значение элемента;
- `key` - итерируемый ключ элемента данных (напр. для словаря);
- `enumeration` - объект, по которому он повторяется.

2.3 Группа узлов “сеть”

2.3.1 Узел mqtt (вход)

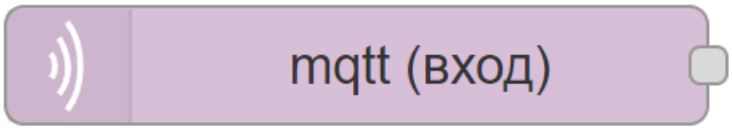


Рис. 2.40 **Узел mqtt (вход) в Палитре**

Рис. 2.41 **Узел mqtt (вход) на Холсте**

Подключается к MQTT брокеру и подписывается на сообщения указанной темы.

Выводит:

- `payload` строка | буфер – строка, если не обнаружено как двоичный буфер.
- `topic` строка – тема MQTT, использует / в качестве разделителя иерархии.
- `qos` число – `0`: приходит не более одного раза, `1`: приходит не менее одного раза, `2`: приходит только один раз.
- `retain` логич.тип – значение `true` указывает, что сообщение было сохранено и может быть устаревшим.

Тема подписки может включать подстановочные знаки MQTT, `+` для одного уровня, `#` для нескольких уровней.

Для этого узла требуется соединение с брокером MQTT. Это настраивается нажатием на значок карандаша.

Несколько MQTT узлов (`вход` или `выход`) могут совместно использовать одно и то же соединение с брокером, если это необходимо.

2.3.2 Узел mqtt (выход)

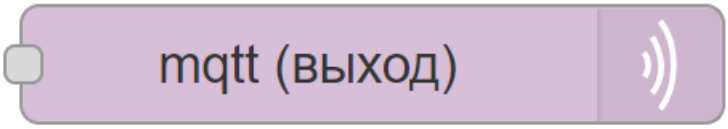


Рис. 2.42 **Узел mqtt (выход) в Палитре**

Рис. 2.43 **Узел mqtt (выход) на Холсте**

Подключается к MQTT брокеру и публикует сообщения.

Принимает:

- `payload` строка | буфер – данные для публикации. Если это свойство не установлено, сообщение не будет отправлено. Чтобы отправить пустое сообщение, установите для этого свойства пустую строку.
- `topic` строка –тема MQTT для публикации.
- `qos` число – `0`: приходит не более одного раза, `1`: приходит не менее одного раза, `2`: приходит только один раз. По умолчанию `0`.
- `retain` логич.тип – установите значение `true`, чтобы сохранить сообщение в брокере. По умолчанию `false`.

Свойство `msg.payload` используется в качестве данных опубликованного сообщения. Если оно содержит объект, то он будет преобразован в строку JSON перед отправкой. Если оно содержит двоичный буфер, сообщение будет опубликовано как есть.

Используемая тема может быть настроена в узле или, если оставить ее пустой, может быть установлена с помощью `msg.topic`.

Аналогично, значения `QoS` и сохранения могут быть сконфигурированы в узле или, если они оставлены пустыми, установлены с помощью `msg.qos` и `msg.retain` соответственно. Чтобы удалить ранее сохраненную тему из брокера, отправьте пустое сообщение в эту тему с установленным флагом сохранения.

Для этого узла требуется соединение с брокером MQTT. Это настраивается нажатием на значок карандаша.

Несколько узлов MQTT (`вход` или `выход`) могут совместно использовать одно и то же соединение с брокером, если это необходимо.

2.3.3 Узел http вход

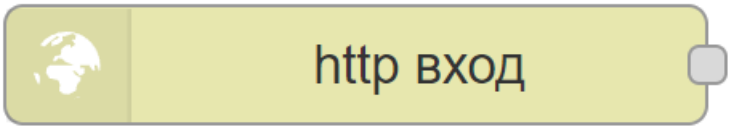


Рис. 2.44 **Узел http вход в Палитре**

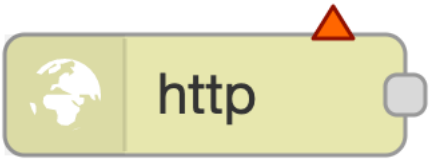


Рис. 2.45 **Узел http вход на Холсте**

Создает конечную точку HTTP для веб-службы.

Выводит:

- `payload` – для `GET`-запроса содержит объект с любыми параметрами строки запроса. В противном случае содержит тело HTTP-запроса.
- `req` объект – объект HTTP-запроса. Этот объект содержит несколько свойств, которые предоставляют информацию о запросе:

1. `body` - тело входящего запроса. Формат будет зависеть от запроса.
2. `headers` - объект, содержащий заголовки HTTP-запроса.
3. `query` - объект, содержащий любые параметры строки запроса.
4. `params` - объект, содержащий любые параметры маршрута.
5. `cookies` - объект, содержащий куки запроса.
6. `files` - если включено в узле, объект, содержащий любые файлы, загруженные как часть `POST`-запроса.

- `res` объект – объект ответа HTTP. Это свойство не должно использоваться напрямую. Для ответа на входящий HTTP запрос используется узел ответа HTTP Response. Это свойство должно оставаться в сообщении вплоть до передачи узлу ответа.

По настроенному пути узел будет принимать запросы определенного типа. Путь может быть указан конкретно, например `/user`, или включать именованные параметры, которые принимают любое значение, например `/user/:name`. Когда используются именованные параметры, их фактическое значение в запросе будет доступно в `msg.req.params`.

Для запросов, которые содержат тело, такие как `POST` или `PUT`, содержимое запроса доступно в `msg.payload`.

Если тип содержимого запроса может быть определен, тело будет преобразовано в любой подходящий тип. Например, application/json будет преобразован в объект JavaScript.

Этот узел не отправляет никакого ответа на запрос. Поток должен включать узел ответа HTTP Response для завершения запроса.

2.3.4 Узел http ответ

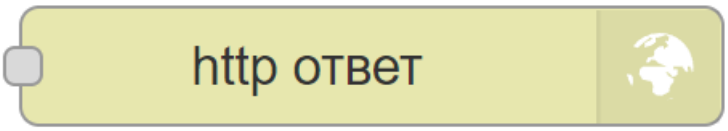


Рис. 2.46 **Узел http ответ в Палитре**

Рис. 2.47 **Узел http ответ на Холсте**

Отправляет ответ на запрос, полученный от узла `HTTP Ввод`.

Принимает:

- `payload` строка – тело ответа.
- `statusCode` число – если установлено, то используется как код состояния ответа. По умолчанию: `200`.
- `headers` объект – если установлено, предоставляет заголовки HTTP для включения в ответ.
- `cookies` объект – если установлено, может использоваться для записи или удаления куков.

Свойства `statusCode` и `headers` также могут быть установлены внутри самого узла, настройками `код состояния` и `заголовки` соответственно. Если свойство установлено в узле, оно не может быть переопределено соответствующим свойством сообщения.

2.3.4.1 Обработка куков

Свойство `cookies` должно быть объектом пар имя/значение. Значением может быть либо строка для установки значения куки с параметрами по умолчанию, либо это может быть объект параметров.

В следующем примере устанавливаются два куки - один с именем `name` и значением `nick`, другой с именем `session` и значением `1234` и сроком действия в 15 минут.

```
msg.cookies = {
    name: 'nick',
    session: {
        value: '1234',
        maxAge: 900000
    }
}
```

Допустимые параметры включают в себя:

- `domain` - (Строка) доменное имя для куки
- `expires` - (Дата) срок годности по Гринвичу. Если не указан или установлен в `0`, создает сессионный куки
- `maxAge` - (Строка) дата истечения срока действия относительно текущего времени в миллисекундах
- `path` - (Строка) путь для куки. По умолчанию `/`
- `value` - (Строка) значение, используемое для куки

Чтобы удалить куки, установите для его `value` значение `null`.

2.3.5 Узел http запрос

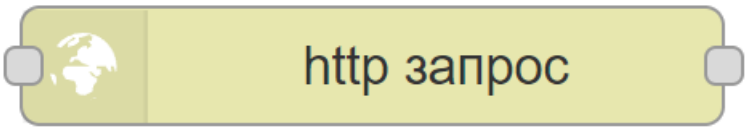


Рис. 2.48 **Узел http запрос в Палитре**

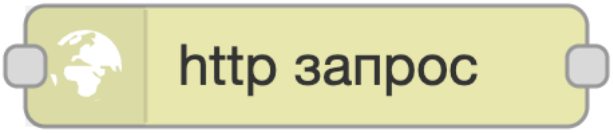


Рис. 2.49 **Узел http запрос на Холсте**

Отправляет HTTP-запрос и возвращает ответ.

Принимает:

- `url` строка – если ‘URL’ не установлен в узле, это необязательное свойство устанавливает URL запроса.
- `method` строка – если ‘метод’ не установлен в узле, это необязательное свойство устанавливает HTTP-метод запроса. Должно быть `GET`, `PUT`, `POST`, `PATCH` или `DELETE`.
- `headers` объект – устанавливает заголовки HTTP запроса.
- `cookies` объект – если установлено, может использоваться для отправки куки с запросом.
- `payload` – отправляется как тело запроса.
- `rejectUnauthorized` – если установлено значение `false`, разрешает отправлять запросы на сайты https, которые используют самозаверенные сертификаты.
- `followRedirects` – если установлено значение `false`, запрещает переадресацию (HTTP 301). По умолчанию `true`.
- `requestTimeout` – если задано положительное число миллисекунд, будет переопределен глобально заданный параметр `httpRequestTimeout`.

Выводит:

- `payload` строка | объект | буфер – тело ответа. Узел можно настроить так, чтобы он возвращал тело в виде строки, пытался проанализировать его как строку JSON или оставлял его в виде двоичного буфера.

- `statusCode` число – код статуса ответа или код ошибки, если запрос не может быть выполнен.
- `headers` объект – объект, содержащий заголовки ответа.
- `responseUrl` строка – если при обработке запроса произошли перенаправления, это свойство является окончательным перенаправленным URL-адресом. В противном случае, URL исходного запроса.
- `responseCookies` объект – если ответ содержит куки, это свойство является объектом пар имя/значение для каждого из них.
- `redirectList` массив – если запрос был перенаправлен один или несколько раз, накопленная информация будет добавлена в это свойство. `location` - это следующий пункт назначения перенаправления. `cookies` - это куки, возвращаемые из источника перенаправления.

При настройке внутри узла свойство URL может содержать `mustache`-теги. Это позволяет построить URL-адрес, используя значения входящего сообщения. Например, если для URL-адреса задано значение `example.com/{{{topic}}}`, то автоматически будет вставлено значение `msg.topic`. Использование `{{{...}}}` предотвращает HTML-кодирование таких символов, как `/` `&` и т.д.

Узел может также автоматически кодировать `msg.payload` как параметры для `GET`-запроса. В этом случае `sg.payload` должен быть объектом.

> При работе через прокси-сервер следует либо установить стандартную переменную среды `http_proxy=...` и перезапустить Luxms Databoring, либо использовать конфигурацию прокси-сервера. Если была установлена конфигурация прокси, то она имеет приоритет над переменной среды.

Использование нескольких узлов HTTP-запросов Чтобы использовать несколько таких узлов в одном потоке, нужно позаботиться о свойстве `msg.headers`. Первый узел установит это свойство с заголовками ответа. Затем следующий узел будет использовать эти заголовки для своего запроса - обычно это неправильный подход. Если свойство `msg.headers` остается неизменным между узлами, оно будет проигнорировано вторым узлом. Чтобы установить свои заголовки, сначала необходимо удалить `msg.headers` или сбросить это свойство до пустого объекта: `{}`.

2.3.5.1 Обработка куки

Свойство `cookies`, передаваемое узлу, должно быть объектом из пар имя/значение. Значением может быть либо строка для установки значения куки, либо это может быть объект с единственным свойством `value`.

Все куки, возвращаемые запросом, передаются обратно в свойстве `responseCookies`.

Обработка типов контента Если `msg.payload` является объектом, узел автоматически установит тип содержимого запроса в application/json и закодирует тело соответствующим образом.

Чтобы закодировать запрос как данные формы, для `msg.headers` `["content-type"]` должно быть установлено `application/x-www-form-urlencoded`.

Загрузка файла Чтобы выполнить загрузку файла, для `msg.headers` `["content-type"]` должно быть установлено значение `multipart/form-data`, а `msg.payload` переданный на узел должен быть объектом со следующей структурой:

```
{
  "КЛЮЧ": {
      "value": СОДЕРЖИМОЕ_ФАЙЛА,
      "options": {
          "filename": "ИМЯ_ФАЙЛА"
      }
  }
}
```

На местах `КЛЮЧ`, `СОДЕРЖИМОЕ_ФАЙЛА` и `ИМЯ_ФАЙЛА` должны быть установлены соответствующие значения.

2.3.6 Узел Веб-сокет (вход)

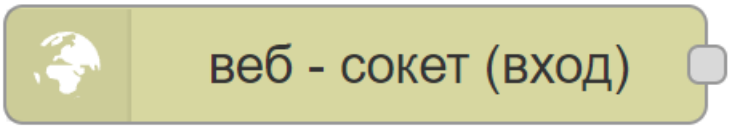


Рис. 2.50 **Узел Веб-сокет (вход) в Палитре**

Рис. 2.51 **Узел Веб-сокет (вход) на Холсте**

Входящий узел WebSocket.

По умолчанию данные, полученные из WebSocket, будут в `msg.payload`. Сокет можно настроить так, чтобы он ожидал правильно сформированную строку JSON. В этом случае он будет анализировать JSON и отправлять полученный объект как полное сообщение.

2.3.7 Узел Веб-сокет (выход)

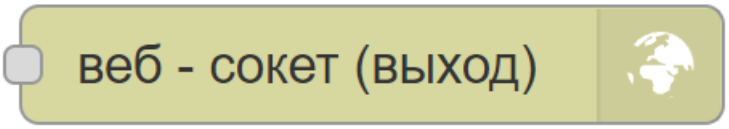


Рис. 2.52 **Узел Веб-сокет (выход) в Палитре**

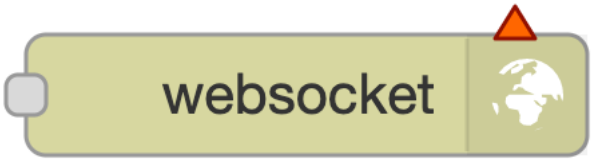


Рис. 2.53 **Узел Веб-сокет (выход) на Холсте**

Исходящий узел WebSocket.

По умолчанию `msg.payload` будет отправлено через WebSocket. Сокет может быть сконфигурирован для кодирования всего объекта msg в виде строки JSON и отправки его через WebSocket.

Если сообщение, поступающее на этот узел, было начато в потоке узлом `Веб-сокет (вход)`, сообщение будет отправлено обратно клиенту, который запустил поток. В противном случае сообщение будет передано всем подключенным клиентам.

Если Вы хотите передать сообщение, которое началось с узла `Веб-сокет (вход)`, всем подключенным клиентам, тогда вам нужно удалить свойство `msg._session` перед тем как передавать его этому узлу.

2.3.8 Узел tcp (вход)

Рис. 2.54 **Узел tcp (вход) в Палитре**

Рис. 2.55 **Узел tcp (вход) на Холсте**

Предоставляет выбор из нескольких входящих TCP-узлов. Можно либо подключаться к удаленному TCP-порту, либо принимать входящие подключения.

> В некоторых системах вам могут потребоваться права root или администратора для доступа к портам ниже 1024.

2.3.9 Узел tcp (выход)

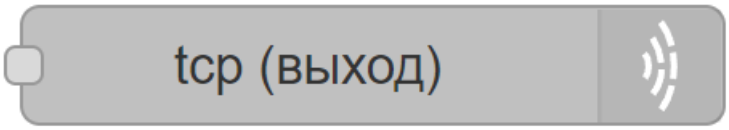


Рис. 2.56 **Узел tcp (выход) в Палитре**

Рис. 2.57 **Узел tcp (выход) на Холсте**

Предоставляет выбор из нескольких исходящих TCP-узлов. Может подключаться к удаленному TCP-порту, принимать входящие подключения или отвечать на сообщения, полученные от узла `tcp (вход)`.

Отправляются только данные `msg.payload`.

Если `msg.payload` является строкой, содержащей Base64-кодировку двоичных данных, опция Base64-декодирования приведет к ее преобразованию обратно в двоичный файл перед отправкой.

Если `msg._session` отсутствует, данные отправляется всем подключенным клиентам.

> В некоторых системах вам могут потребоваться права root или администратора для доступа к портам ниже 1024.

2.3.10 Узел tcp (запрос)

Рис. 2.58 **Узел tcp (запрос) в Палитре**

Рис. 2.59 **Узел tcp (запрос) на Холсте**

Узел TCP-запроса - отправляет `msg.payload` на tcp-порт сервера и ожидает ответа.

Подключается, отправляет `запрос` и читает `ответ`. Он может либо подсчитать количество возвращенных символов в фиксированный буфер, сопоставить указанный символ перед возвратом, дождаться фиксированного времени ожидания от первого ответа и затем вернуться, сидеть и дожидаться данных, или отправить, а затем немедленно закрыть соединение, не дожидаясь ответа.

Ответ будет возвращен в `msg.payload` в виде буфера, поэтому вам может понадобиться применить к нему `.toString()`, если нужно преобразование в строку.

Если вы оставите tcp-хост или порт пустыми, они должны быть установлены с помощью свойств `msg.host` и `msg.port` в каждом сообщении, отправляемом узлу.

2.3.11 Узел udp (вход)

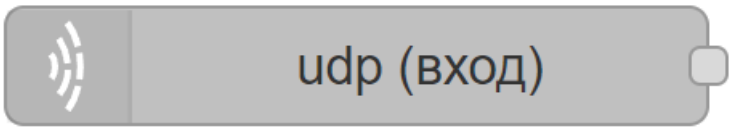


Рис. 2.60 **Узел udp (вход) в Палитре**

Рис. 2.61 **Узел udp (вход) на Холсте**

Входящий UDP-узел, который создает `msg.payload`, содержащий буфер, строку или base64-строку. Поддерживает многоадресную рассылку.

Он также предоставляет `msg.ip` и `msg.port` для IP-адреса и порта, с которого было получено сообщение.

> В некоторых системах вам могут потребоваться права root или администратора для доступа к портам ниже 1024 и/или широковещательной рассылки.

2.3.12 Узел udp (выход)

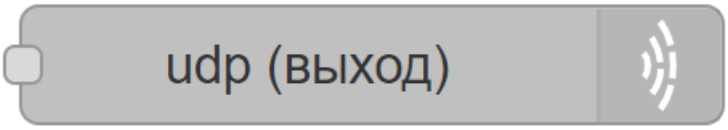


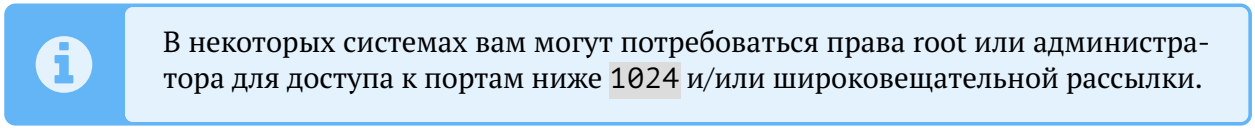
Рис. 2.62 **Узел udp (выход) в Палитре**

Рис. 2.63 **Узел udp (выход) на Холсте**

Этот узел отправляет `msg.payload` на назначенный UDP-хост и порт. Поддерживает многоадресную рассылку.

Вы также можете использовать `msg.ip` и `msg.port` для установки значений пункта назначения, но статически настроенные значения имеют приоритет.

Если Вы выберете широковещательную рассылку, то либо задайте в качестве адреса локальный широковещательный IP-адрес, либо попробуйте `255.255.255.255`, который является глобальным широковещательным адресом.



В некоторых системах вам могут потребоваться права root или администратора для доступа к портам ниже `1024` и/или широковещательной рассылки.

2.4 Группа узлов “последовательность”

2.4.1 Узел Разделить

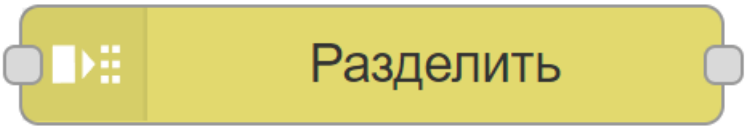


Рис. 2.64 **Узел Разделить в Палитре**

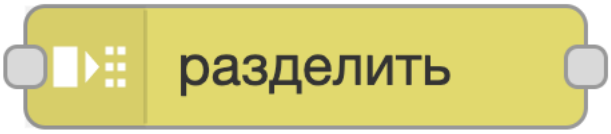


Рис. 2.65 **Узел Разделить на Холсте**

Разбивает сообщение на последовательность сообщений.

Принимает:

- `payload` объект | строка | массив | буфер

Поведение узла определяется типом `msg.payload`:

- строка/буфер – сообщение разделяется с помощью указанного символа (по умолчанию: `\n`), последовательности буфера или по фиксированной длине.
- массив – сообщение разбивается на отдельные элементы массива или массивы фиксированной длины.
- объект – сообщение отправляется для каждой пары ключ/значение объекта.

Выводит:

- `parts` объект

Это свойство содержит информацию о том, как сообщение было отделено от исходного сообщения. При передаче на узел join последовательность может быть собрана обратно в одно сообщение.

Объект содержит следующие свойства:

- `id` – идентификатор группы сообщений
- `index` – позиция в группе
- `count` – если известно, общее количество сообщений в группе. Смотрите `потоковый режим` ниже.
- `type` – тип сообщения: строка = `string`; массив = `array`; объект = `object`; буфер = `buffer`
- `ch` – для строки или буфера данные, использованные для разделения сообщения, в виде строки или массива байтов
- `key` – для объекта - ключ свойства, из которого было создано это сообщение. Узел также может быть сконфигурирован для копирования этого значения в другие свойства сообщения, такие как `msg.topic`.
- `len` – длина каждого сообщения при разделении с использованием значения фиксированной длины.

Этот узел облегчает создание потока, который выполняет общие действия над последовательностью сообщений, перед тем как с помощью узла join объединить последовательность обратно в одно сообщение.

Он использует свойство `msg.parts` для отслеживания отдельных частей последовательности.

2.4.1.1 Потоковый режим

Узел также может использоваться для переформатирования потока сообщений. Например, последовательное устройство, которое отправляет завершенные новой строкой команды, может доставлять одно сообщение с частичной командой в конце. В `потоковом режиме` этот узел будет разбивать сообщение и отправлять каждый завершенный сегмент. Если в конце есть частичный сегмент, узел удержит его и добавит к следующему полученному сообщению.

При работе в этом режиме узел не будет устанавливать свойство `msg.parts.count`, так как он не знает, сколько сообщений ожидать в потоке. Это означает, что его нельзя использовать вместе с узлом join в его автоматическом режиме.

2.4.2 Узел Соединить

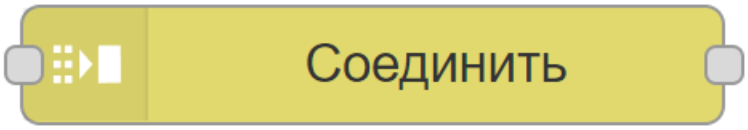


Рис. 2.66 **Узел Соединить в Палитре**

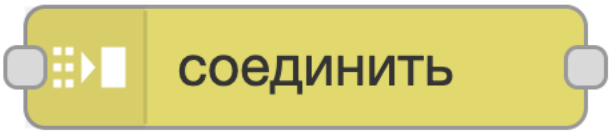


Рис. 2.67 **Узел Соединить на Холсте**

Объединяет последовательности сообщений в одно сообщение.

Доступны три режима:

- автоматический – при использовании с узлом `Разделить` он автоматически объединит сообщения, чтобы восстановить структуру сообщения, которая была до разделения.
- ручной – объединяет последовательности сообщений различными способами.
- агрегация последовательности – применяет JSONata-выражение ко всем сообщениям в последовательности, чтобы свести его к одному сообщению.

Принимает:

- `parts` объект

Чтобы автоматически соединить последовательность сообщений, у них всех должно быть установлено это свойство. Узел `Разделить` генерирует это свойство, но его можно создать вручную. Оно имеет следующие свойства:

1. `id` - идентификатор группы сообщений
2. `index` - позиция в группе
3. `count` - общее количество сообщений в группе
4. `type` - тип сообщения: строка = `string`; массив = `array`; объект = `object`; буфер = `buffer`
5. `ch` - для строки или буфера данные, использованные для разделения сообщения, в виде строки или массива байтов
6. `key` - для объекта - ключ свойства, из которого было создано это сообщение
7. `len` - длина каждого сообщения при разделении с использованием значения фиксированной длины

- complete

Если установлено, узел добавит `payload`, а затем отправит выходное сообщение в своем текущем состоянии. Если вы не хотите добавлять `payload`, удалите его из `msg`.

2.4.2.1 Автоматический режим

В автоматическом режиме используется свойство `parts` входящих сообщений, чтобы определить способ объединения последовательности. Это позволяет автоматически выполнять обратное действие для узла `Разделить`.

2.4.2.2 Ручной режим

Когда узел настроен на соединение в ручном режиме, он может объединять последовательности сообщений в различные результаты:

- строка или буфер - создается путем объединения выбранного свойства каждого сообщения с указанными символами соединения или буфером.
- массив - создается путем добавления каждого выбранного свойства или всего сообщения в выходной массив.
- объект ключей/значений - создается с использованием свойства каждого сообщения для определения ключа, под которым хранится требуемое значение.
- слитый объект - создается путем объединения свойства каждого сообщения в один объект.

Другие свойства исходящего сообщения берутся из последнего сообщения, полученного перед отправкой результата.

Свойство кол-во может быть установлено для количества сообщений, которое должно быть получено перед генерацией выходного сообщения. Для выходных данных объекта, когда это число достигнуто, узел может быть настроен на отправку сообщения для каждого последующего полученного сообщения.

Свойство время (сек) может быть установлено, чтобы инициировать отправку нового сообщения с использованием всего, что было получено до сих пор.

Если сообщение получено с установленным свойством `msg.complete`, выходное сообщение завершается и отправляется. Это сбрасывает любой подсчет частей.

Если сообщение получено с установленным свойством `msg.reset`, частично завершенное сообщение удаляется и не отправляется. Это сбрасывает любой подсчет частей.

2.4.2.3 Режим агрегации последовательности

Когда настроено объединение в режиме агрегирования, выражение применяется к каждому сообщению в последовательности, и результат накапливается для создания одного сообщения.

1. Начальное значение

Начальное значение накопленного значения `$A`.

2. Агрегирующее выражение

Выражение JSONata, которое вызывается для каждого сообщения в последовательности. Результат передается следующему вызову выражения как накопленное значение. В выражении могут использоваться следующие специальные переменные:

- `$A`: накопленное значение,

- `$I`: индекс сообщения в последовательности,
- `$N`: количество сообщений в последовательности.

3. Исправляющее выражение

Необязательное выражение JSONata, которое применяется после того, как агрегирующее выражение было применено ко всем сообщениям в последовательности. В выражении могут использоваться следующие специальные переменные:

- `$A`: накопленное значение,
- `$N`: количество сообщений в последовательности.

По умолчанию агрегирующее выражение применяется по порядку, от первого до последнего сообщения последовательности. При желании его можно применять в обратном порядке.

Пример: следующие параметры, при получении последовательности числовых значений, рассчитывают среднее значение:

- Агрегирующее выражение: `$A+payload`
- Начальное значение: `0`
- Исправляющее выражение: `$A / $N`

2.4.2.4 Хранение сообщений

Этот узел будет буферизировать сообщения внутри, чтобы работать между последовательностями. Параметр `nodeMessageBufferMaxLength` можно использовать для ограничения количества сообщений, которые узел будут буферизовать.

2.4.3 Узел Сортировать

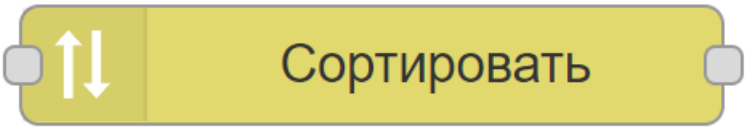


Рис. 2.68 **Узел Сортировать в Палитре**

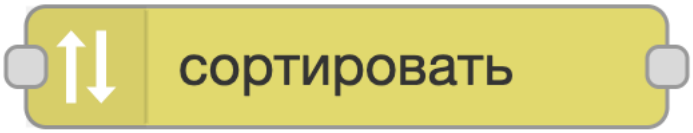


Рис. 2.69 **Узел Сортировать на Холсте**

Функция, которая сортирует свойство сообщения или последовательность сообщений.

Когда узел настроен сортировать свойство сообщения, он сортирует данные массива в указанном свойстве сообщения.

Когда узел настроен сортировать последовательность сообщений, он будет переупорядочивать сообщения.

Порядок сортировки может быть:

- Восходящий.
- Нисходящий.

Для чисел, числовое упорядочивание можно указать с помощью флажка.

Когда выбрана сортировка свойства, ключом сортировки может быть значение элемента или выражение JSONata. Когда выбрана сортировка последовательности сообщений, ключом сортировка может быть свойство сообщения или выражение JSONata.

При сортировке последовательности сообщений узел сортировки полагается на полученные сообщения, чтобы установить `msg.parts`. Узел `Разделить` генерирует это свойство, но его можно создать вручную. Оно имеет следующие свойства:

- `id` - идентификатор группы сообщений.
- `index` - позиция в группе.
- `count` - общее количество сообщений в группе.

> Этот узел буферизирует сообщения внутри для своей работы. Чтобы предотвратить непредвиденное использование памяти, можно указать максимальное количество хранимых сообщений. По умолчанию количество сообщений не ограничено.

Свойство `nodeMessageBufferMaxLength` устанавливается в settings.js.

2.4.4 Узел Группировать

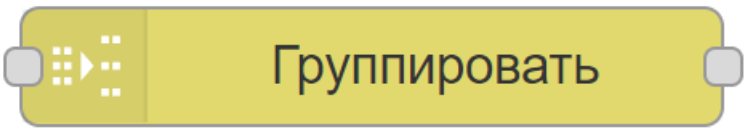


Рис. 2.70 **Узел Группировать в Палитре**

Рис. 2.71 **Узел Группировать на Холсте**

Создает последовательности сообщений на основе различных правил.

Существует три режима создания последовательностей сообщений:

1. Количество сообщений - группирует сообщения в последовательности заданной длины. Параметр совпадения указывает, сколько сообщений в конце одной последовательности следует повторить в начале следующей последовательности.

2. Интервал времени - группирует сообщения, поступающие в указанный интервал. Если в течение интервала сообщения не поступают, узел может при желании отправлять пустое сообщение.
3. Объединение последовательностей - создает последовательность сообщений путем объединения входящих последовательностей. Каждое сообщение должно иметь свойство msg.`topic` и свойство `msg.parts`, определяющие его последовательность. Узел настраивается со списком значений `topic` для идентификации порядка объединения последовательностей.

2.4.4.1 Хранение сообщений

Этот узел будет буферизировать сообщения внутри, чтобы работать с последовательностями. Параметр `nodeMessageBufferMaxLength` можно использовать для ограничения количества сообщений, которые узел будут буферизовать.

Если сообщение получено с установленным свойством msg.reset, буферизованные сообщения удаляются и не отправляются.

2.5 Группа узлов “анализатор”

2.5.1 Узел csv

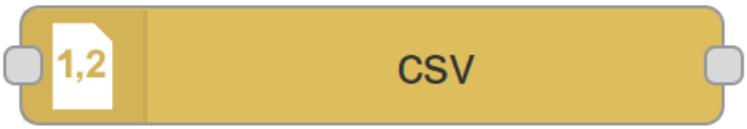


Рис. 2.72 **Узел csv в Палитре**

Рис. 2.73 **Узел csv на Холсте**

Выполняет преобразования между строкой в CSV формате и ее представлением в JavaScript-объекте, в любом направлении.

Принимает:

- `payload` объект | массив | строка – JavaScript объект, массив или CSV-строка.

Выводит:

- `payload` объект | массив | строка

1. Если на входе значение строкового типа, узел попытается проанализировать его как CSV и создает объект JavaScript из пар ключ/значение для каждой строки. Затем узел либо отправит сообщение для каждой строки или одно сообщение, содержащее массив объектов.
2. Если на входе JavaScript объект, узел попробует построить CSV-строку.
3. Если на входе массив простых значений, узел построит однострочную CSV-строку.
4. Если на входе массив массивов или массив объектов, создается многострочная CSV-строка.

Шаблон столбцов может содержать упорядоченный список имен столбцов. При преобразовании CSV в объект имена столбцов будут использоваться в качестве имен свойств. Кроме того, имена столбцов могут быть взяты из первой строки CSV.

При преобразовании в CSV шаблон столбцов используется для определения того, какие свойства извлекать из объекта и в каком порядке.

Если шаблон пуст, то узел может использовать простой список свойств, разделенных запятыми, предоставленных в `msg.columns`, чтобы определить, что извлечь. Если его нет, то все свойства объекта выводятся в том порядке, в котором они были найдены в первой строке.

Если входные данные являются массивом, то шаблон столбцов используется только для необязательного генерирования строки с заголовками столбцов.

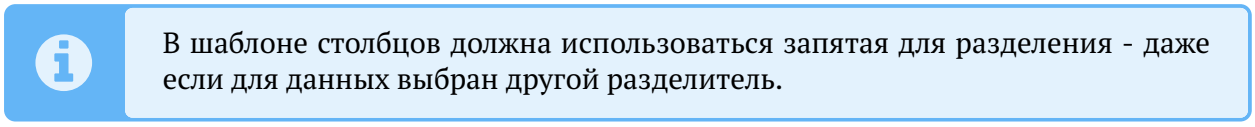
Если выбрана опция `'разбирать числовые значения`, строковые числовые значения будут возвращаться в виде чисел. К примеру, среднее значение в CSV-строке `1,"1,5",2`.

Если выбрана опция `включать пустые строковые значения`, пустые строки будут возвращениы в результате. К примеру, среднее значение в CSV-строке `"1","",3`.

Если выбрана опция `включить null-значения`, `null`-значения будут возвращениы в результате. К примеру, среднее значение в CSV-строке `"1",,3`.

Узел может принимать входные данные, состоящие из нескольких частей, при условии, что свойство `parts` установлено правильно, например, из узла `Запись в файл` или узла `Разделить`.

При выводе нескольких сообщений они будут иметь свойство parts и формировать полную последовательность сообщений.



В шаблоне столбцов должна использоваться запятая для разделения - даже если для данных выбран другой разделитель.

2.5.2 Узел html

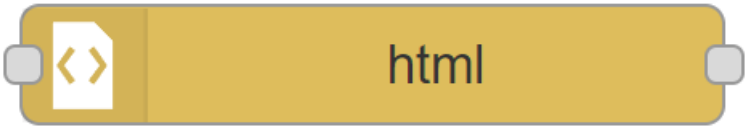


Рис. 2.74 **Узел html в Палитре**

Рис. 2.75 **Узел html на Холсте**

Извлекает элементы из HTML-документа, хранящегося в `msg.payload`, с помощью CSS-селектора.

Принимает:

- `payload` строка – HTML-строка, из которой извлекать элементы.
- `select` строка – если селектор не настроен на панели редактирования узла, то он может быть установлен как свойство `msg`.

Выход:

- `payload` массив | строка – результатом может быть либо одно сообщение, в котором `payload` содержит массив найденных элементов, либо несколько сообщений, каждое из которых содержит найденный элемент. Если отправлено несколько сообщений, у них также будет установлен `parts`.

Этот узел поддерживает комбинацию селекторов CSS и jQuery.

2.5.3 Узел json

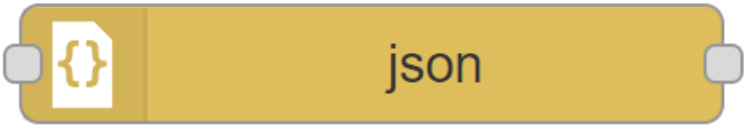


Рис. 2.76 **Узел json на Холсте**

Рис. 2.77 **Узел json на Холсте**

Выполняет преобразования между строкой в JSON формате и ее представлением в JavaScript-объекте, в любом направлении.

Принимает:

- `payload` объект | строка – JavaScript объект или JSON-строка.
- `schema` объект – необязательный объект JSON-схемы для проверки данных. Свойство будет удалено перед отправкой msg следующему узлу.

Выводит:

- `payload` объект | строка
  - Если вход является JSON-строкой, узел пытается проанализировать ее как JavaScript объект.
  - Если вход является JavaScript объектом, узел создает JSON-строку. Строка может быть при желании отформатирована.
- `schemaError` массив – если проверка JSON-схемы завершится неудачно, узлом `Отлов ошибок` можно получить свойство `schemaError`, содержащее массив ошибок.

По умолчанию узел работает с `msg.payload`, но его можно настроить для преобразования любого свойства сообщения.

Узел также может быть сконфигурирован для обеспечения конкретной кодировки вместо переключения между ними. Это можно использовать, например, при работе с узлом `HTTP Вход`, чтобы гарантировать, что данные `payload` всегда будут являться объектом, даже если входящий запрос неправильно установил свой тип содержимого для узла `HTTP Вход`, чтобы выполнить преобразование.

Если узел настроен так, чтобы свойство кодировалось как строка, и он получает строку, дальнейшие проверки этого свойства выполняться не будут. Он не будет проверять, является ли строка допустимым JSON, и не будет переформатировать ее, если выбрана опция форматирования.

Подробнее о JSON-схеме Вы можете узнать в спецификации здесь.

2.5.4 Узел xml

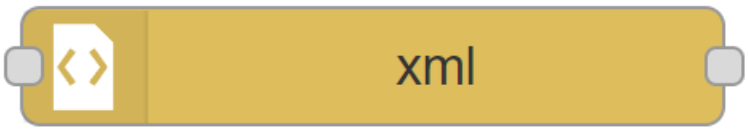


Рис. 2.78 **Узел xml в Палитре**

Рис. 2.79 **Узел xml на Холсте**

Выполняет преобразования между строкой в XML формате и ее представлением в JavaScript-объекте, в любом направлении.

Принимает:

- `payload` объект | строка – JavaScript объект или XML-строка.
- `options` объект – это необязательное свойство может использоваться для передачи любых параметров, поддерживаемых библиотекой, которая используется для преобразования в/из XML. См. документацию xml2js для получения дополнительной информации.

Выводит:

- `payload` объект | строка
  - Если на входе значение строкового типа, узел пытается проанализировать его как XML и создает объект JavaScript.
  - Если на входе JavaScript объект, узел пытается построить XML-строку.

При преобразовании между XML и объектом любые XML-атрибуты по умолчанию добавляются как свойство с именем `$`. Любое текстовое содержимое добавляется как свойство с именем `_`. Эти имена свойств могут быть указаны в конфигурации узла.

Например, следующий XML:

```
<p class="tag">Hello World</p>
```

будет преобразован в:

```
{
  "p": {
    "$": {
      "class": "tag"
    },
    "_": "Hello World"
  }
}
```

2.5.5 Узел yaml

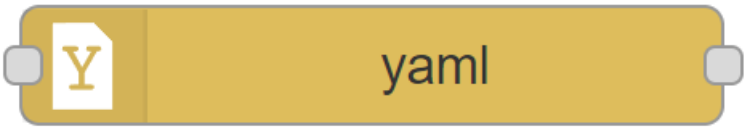


Рис. 2.80 **Узел yaml в Палитре**

Рис. 2.81 **Узел yaml на Холсте**

Выполняет преобразования между строкой в YAML формате и ее представлением в JavaScript-объекте, в любом направлении.

Принимает:

- `payload` объект | строка – JavaScript объект или YAML-строка.

Выводит:

- `payload` объект | строка
  - Если на входе YAML-строка, узел пытается проанализировать ее как JavaScript объект.
  - Если на входе JavaScript объект, узел создает YAML-строку.

2.6 Группа узлов “хранилище”

2.6.1 Узел Запись в файл

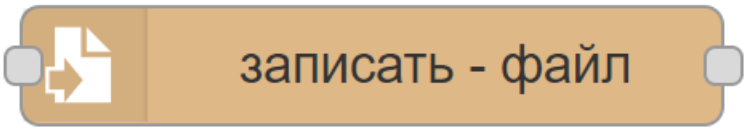


Рис. 2.82 **Узел Запись в файл в Палитре**

Рис. 2.83 **Узел Запись в файл на Холсте**

Записывает `msg.payload` в файл, либо добавлением в конец, либо заменой существующего контента. Кроме того, им можно удалять файл.

Принимает:

- `filename` строка – если не настроено в узле, это необязательное свойство устанавливает имя файла, который будет обновлен.

Выход: по завершении записи входное сообщение отправляется на выходной порт.

Данные каждого сообщения будут добавлены в конец файла, при желании добавляя символ новой строки `\n` между каждым.

Если используется `msg.filename`, файл будет закрыт после каждой записи. Для лучшей производительности используйте фиксированное имя файла.

Он может быть настроен на перезапись всего файла, а не на добавление. Например, при записи двоичных данных (типа изображения) в файл, следует использовать эту опцию и отключить опцию добавления новой строки.

Кодировка данных, записанных в файл, может быть выбрана в списке кодировок.

Кроме того, этот узел может быть настроен на удаление файла.

2.6.2 Узел Чтение из файла

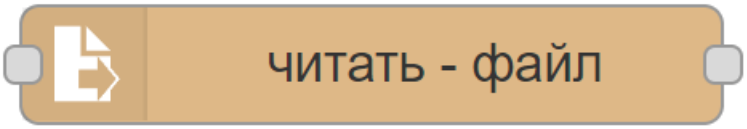


Рис. 2.84 **Узел Чтение из файла в Палитре**

Рис. 2.85 **Узел Чтение из файла на Холсте**

Читает содержимое файла как строку или двоичный буфер.

Принимает:

- `filename` строка – это свойство устанавливает имя файла для чтения, если оно не задано в настройках узла.

Выводит:

- `payload` строка | буфер – содержимое файла в виде строки или двоичного буфера.
- `filename` строка – если не настроено в узле, это необязательное свойство устанавливает имя файла для чтения.

Имя файла должно быть абсолютным путем к файлу, иначе оно будет путем относительно рабочего каталога процесса Luxms Databoring.

В Windows может быть необходимо кодировать разделители пути двойной косой чертой, например: `\\Users\\myUser`.

При желании текстовый файл можно разбить на строки, выводя по одному сообщению для каждой строки, или двоичный файл разбить на более мелкие фрагменты буфера - размер блока зависит от операционной системы, но обычно составляет 64 КБ (Linux/Mac) или 41 КБ (Windows).

При разбиении на несколько сообщений каждое сообщение будет иметь свойство parts, формирующее полную последовательность сообщений.

Кодировка входных данных может быть выбрана из списка кодировок, если выходной формат - строка.

2.6.3 Узел Наблюдение

Рис. 2.86 **Узел Наблюдение в Палитре**

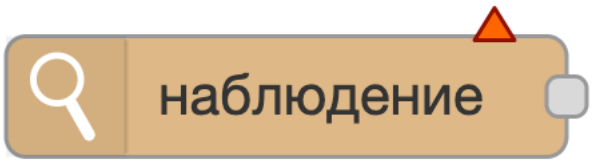


Рис. 2.87 **Узел Наблюдение на Холсте**

Наблюдает за изменениями директории или файла.

Вы можете ввести список разделенных запятыми директорий и/или файлов. Вам нужно взять в кавычки `"..."` те из них, в которых есть пробелы.

В Windows вы должны использовать двойную обратную косую черту `\\` вместо одной в именах директорий.

Полное имя фактически измененного файла помещается в `msg.payload` и msg.filename, а строковая версия списка наблюдения возвращается в `msg.topic`.

`.file` содержит только краткое имя файла, который изменился. `msg.type` содержит тип измененной единицы, обычно это `file` для файла или `directory` для директории, тогда как `msg.size` содержит размер файла в байтах.

В Linux все является файлами, и может быть под наблюдением.

> Наблюдаемые директория или файл должны существовать. Если файл или директория будут удалены, они могут перестать отслеживаться, даже если они затем будут созданы заново.

2.7 Группа узлов luxmsbi

2.7.1 Узел Задать регистр

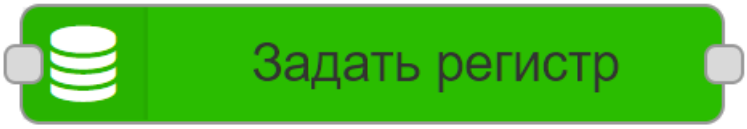


Рис. 2.88 **Узел Задать регистр в Палитре**

Рис. 2.89 **Узел Задать регистр на Холсте**

Данный узел позволяет установить для текущего потока определённый контекст использования данных, а также определить начальную дату для выборки данных.

> Поддержка узла прекращена после обновления Data Boring до версии 9.2.4. Не рекомендуется к использованию.

Принимает:

- Узел принимает управляющий сигнал.

Выводит:

- Узел задает глобальный контекст (настройки), из значений которых формируется доступный набор колонок в узлах типа `medrouting-step`.

Параметры:

- `DataSource` строка – заданный источник данных, в котором будет проводиться выборка данных.
- `Start Date` строка – задаёт фильтр для выборки данных, перед их передачей на первый шаг.
- Глобальный контекст потока – типы данных и компонент, которые будут доступны текущему потоку

2.7.2 Узел Задать таблицу условий

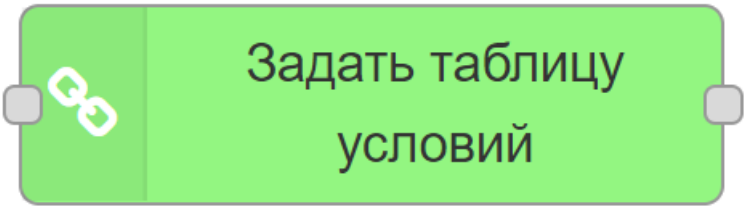


Рис. 2.90 **Узел Задать таблицу условий в Палитре**

Рис. 2.91 **Узел Задать таблицу условий на Холсте**

Задает тип шага и условия фильтрации для формирования данных. Узел использует глобальный заданный контекст данных, на основе этого формирует доступные таблицы и тип узла.

Данный узел в графическом режиме позволяет пользователю создать таблицы условий и фильтрации которые будут подавать на выход заданные метки и результат фильтрации.

Принимает:

- `dataSourceId` строка – источник данных для преобразования и отображения.
- `startDate` строка – задаёт фильтр для выборки данных.
- `prevNodeId` число|строка – идентификатор существующего узла, с которого был передан сигнал.
- `flowId` число|строка – идентификатор текущего потока, к которому применяется правило сравнения и фильтрации

Выводит:

- Текущий статус операции и результирующее свойство `result` со значением `red` или `green`, а также набором меток заданными через запятую.

2.7.3 Узел Группировать данные для импортера

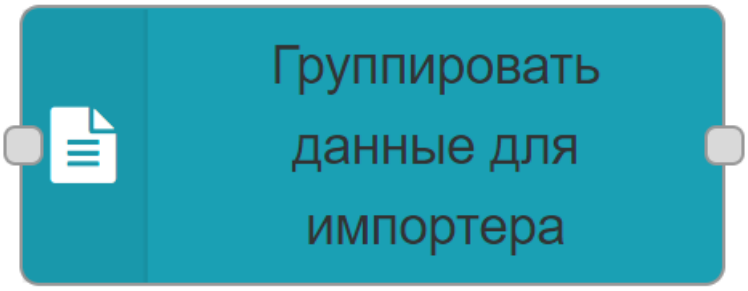


Рис. 2.92 **Узел Группировать данные для импортера в Палитре**

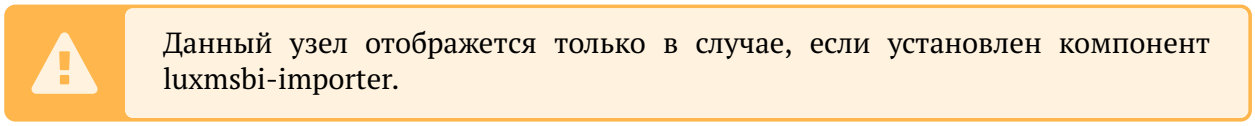
Рис. 2.93 **Узел Группировать данные для импортера на Холсте**

Узел запроса данных за один период времени, а также трансформации/агрегации для импортера. Создает последовательности сообщений для Luxms Importer на основе различных правил.

Принимает:

- `Name` строка – задает свойство имени узла на холсте.
- `Batch ID` строка – начальный ID пакета для импортера.
- `Batch` строка – дополнительный параметр для Luxms Importer.

Хранение сообщений: Этот узел будет буферизировать сообщения внутри, чтобы работать с последовательностями. Глобальный параметр `nodeMessageBufferMaxLength` можно использовать для ограничения количества сообщений, которые узел будут буферизовать.



Данный узел отображается только в случае, если установлен компонент luxmsbi-importer.

2.7.4 Узел Планировщик задач

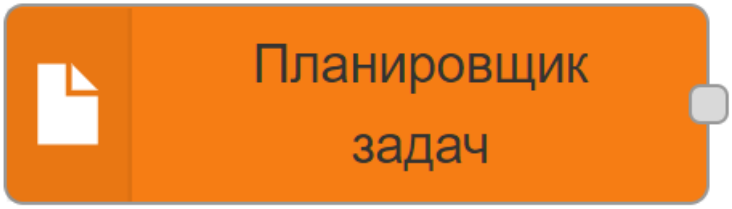


Рис. 2.94 **Узел Планировщик задач в Палитре**

Рис. 2.95 **Узел Планировщик задач на Холсте**

Данный узел позволяет планировать запуск выполнения задач, используя полный синтаксис `crontab`. Не имеет входов. На выходе будет подан стартовый сигнал на следующий в указанный период времени и с заданным интервалом и промежутком.

Принимает:

- `Pattern` строка – период запуска задается согласно синтаксису CRON.

2.7.5 Узел SQL источник

Рис. 2.96 **Узел SQL источник в Палитре**

Рис. 2.97 **Узел SQL источник на Холсте**

Узел запроса данных с произвольным sql-запросом. Данный узел выполняет запрос в заданном `Datasource` с помощью RSocket и с указанными параметрами, статус выполнения запроса будет отображен в статусе узла

Принимает:

- `msg.payload.query` – строку sql-запроса, если не установлен в самом узле.

Выводит:

- Результат выполнения sql-запроса сохраняется в объекте `msg`.

Параметры:

- `Имя` строка - наименование узла на холсте.
- `Источник данных` дропдаун – заданный источник данных, для которого будет выполнен запрос.
- `Размер чанка` число - результаты запроса к бд собираются в группы указанного размера.

Причечание. Результаты выдаются массивом в окно отладчика, иначе выдаются в `payload` по одному

Editor:

- `Строка` - тело запроса который должен быть выполнен при инициализации узла.

Пример запроса:

```
select 1 as num, 'one' as text_num
```

Результат запроса, выведенный в окно отладки:

```
{
  "_msgid": "ad90d997b8ae0ac4",
  "payload": {
    "num": 1,
    "text_num": "one"
  },
```

```
8    "topic": "",
9    "end_of_stream": true
10 }
```

2.7.6 Узел Загрузка в датасет

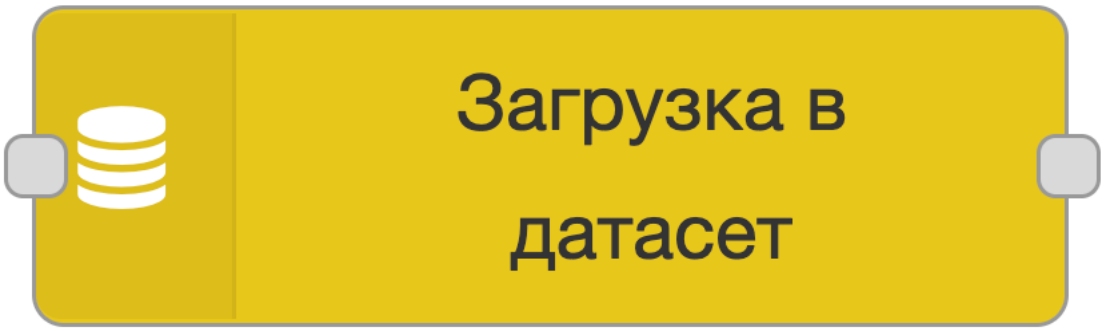


Рис. 2.98 **Узел Загрузка в датасет в Палитре**

Рис. 2.99 **Узел Загрузка в датасет на Холсте**

Узел загрузки данных в заданный датасет, подготовленных предыдущим узлом для Luxms BI Importer. Узел не имеет выхода, на вход принимает один параметр.

Параметры:

- `Name` строка – наименование узла на холсте.
- `Dataset ID` строка – ID начального набора данных / операции для Luxms BI Importer.

Принимает:

- `payload` строка – сформированные параметры для загрузки данных в датасет через Luxms BI Importer.

Данный узел отображается только в случае, если установлен компонент luxmsbi-importer.

2.7.7 Узел Отладка

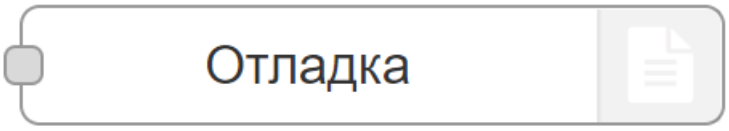


Рис. 2.100 **Узел Отладка в Палитре**

Рис. 2.101 **Узел Отладка на Холсте**

Отображает отладочные сообщения и свойства Luxms BI Importer получаемые, указанным узлом на боковой панели во вкладке отладки и, при необходимости, журнале среды выполнения.

Вкладка отладки на боковой панели обеспечивает структурированное представление полученных узлом сообщений, что упрощает исследование их структуры.

Объекты и массивы JavaScript могут быть свернуты и развернуты по мере необходимости. Буферы могут отображаться в виде данных как есть или в виде строки, когда это возможно.

Рядом с каждым сообщением отладочная панель показывает информацию о времени получения сообщения, узле, который его отправил, и типе данных. Нажатие на идентификатор узла-источника покажет этот узел в рабочей области.

Узел не имеет выходов.

Данный узел имеет только один вход, который инициализирует запрос отладочных сообщений в Luxms BI Importer, и параметр `Name`, который задает имя узла на холсте.

> Данный узел отображется только в случае, если установлен компонент luxmsbi-importer.

2.7.8 Узел SQL запрос в файл

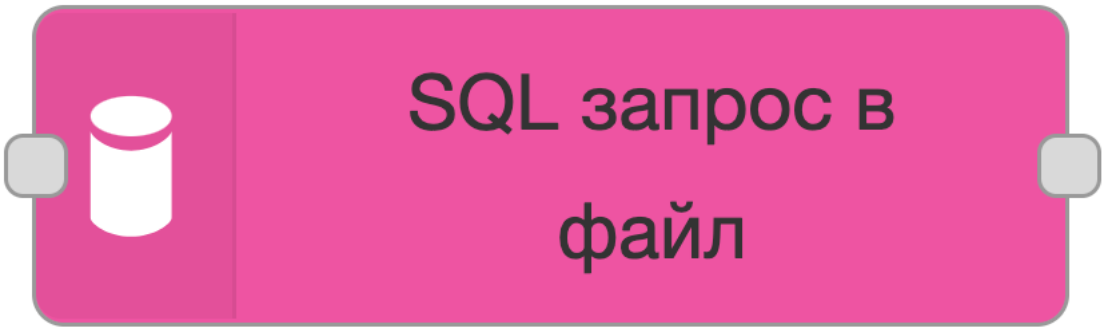


Рис. 2.102 **Узел SQL запрос в файл в Палитре**

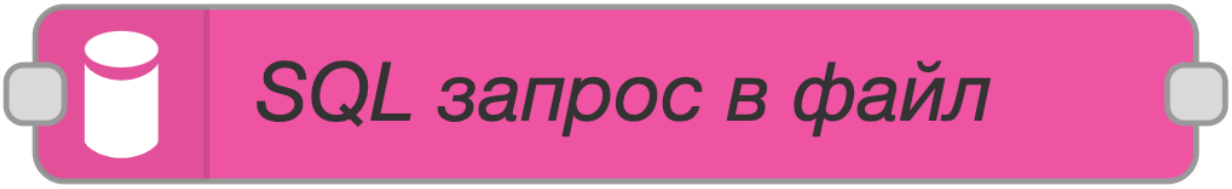


Рис. 2.103 **Узел SQL запрос в файл на Холсте**

Выполнение произвольного sql-запроса в заданном источнике данных с указанными параметрами с сохранением результата в файл формата `.csv` на сервере.

> Поддержка узла прекращена после обновления Data Boring до версии 9.2.4. Не рекомендуется к использованию. Рекомендуется использование узла Datatransfer (Перенос данных).

Принимает:

- `msg.payload.query` - код запроса, если не установлен в самом узле.

Выводит:

- Результат выполнения запроса или выборки заданной пользователем.

Параметры:

- `Имя` строка - наименование узла на холсте.
- `Источник данных` дропдаун – заданный источник данных, для которого будет выполнен запрос.
- `CSV с заголовками` чекбокс - по-умолчанию, если не стоит галочка, производится сохранение результата запроса без заголовка
- `CSV разделитель` дропдаун - меню выбора разделителей для `.csv` файла. На выбор доступны:
  1. `tab`
  2. `|`
  3. `,`
  4. `;`
- `Сжатый` чекбокс - если стоит галочка, производится сохранение результата запроса в один из самых популярных алгоритмов сжатия, который позволяет уменьшить размер файла. Файл сохраняется в формате `.gz`.
- `Попытки` число - общее число попыток исполнения. Если указать `1` (как и по умолчанию), то повторных попыток не будет.
- `Задержка (мс)` число - задержка между повторами при возникновении исключения, если параметр не указан - по-умолчанию `5000` мс.

Пример структуры кода в узле `функция` исходящий на вход узлу `SQL запрос в файл`

```
msg.payload = {};
```

```
msg.payload.query = 'select 1 as num, \'one\' as string';
return msg;
```

Editor:

- `Строка` - тело запроса который должен быть выполнен при инициализации узла.

2.7.9 Узел Выбрать SQL-таблицу

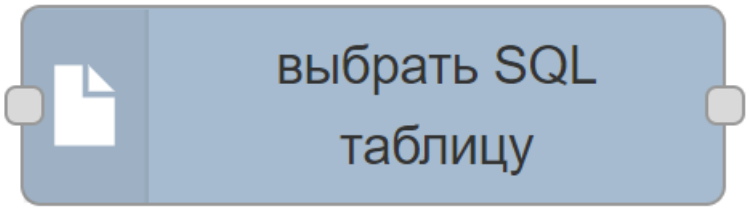


Рис. 2.104 **Узел Выбрать SQL-таблицу в Палитре**

Рис. 2.105 **Узел Выбрать SQL-таблицу на Холсте**

Выгружает дамп таблиц в .csv файл. Существует два режима работы узла: `SQL` и `Table`.

> Поддержка узла прекращена после обновления Data Boring до версии 9.2.4. Не рекомендуется к использованию. Рекомендуется использование узла Datatransfer (Перенос данных).

SQL Позволяет выгрузить данные с произвольного sql-запроса, сформировать их в файл с возможностью дальнейшей пакетной обработки. При ошибке в SQL запросе файл все равно будет выгружен, но пустой – сообщение об ошибке может быть выведено в узле `Отладка`.

Table Позволяет выбрать нужную таблицу в графическом режиме, последовательно будет предложено выбрать из текущей структуры БД, необходимые пользователю данные.

Принимает:

- Управляющий сигнал для начала выгрузки или сформированный SQL запрос.

Выводит:

- На выходе получает ссылку на сформированный файл выгрузки в CSV формате с разделителем `,`.

2.7.10 Узел Импорт файлов

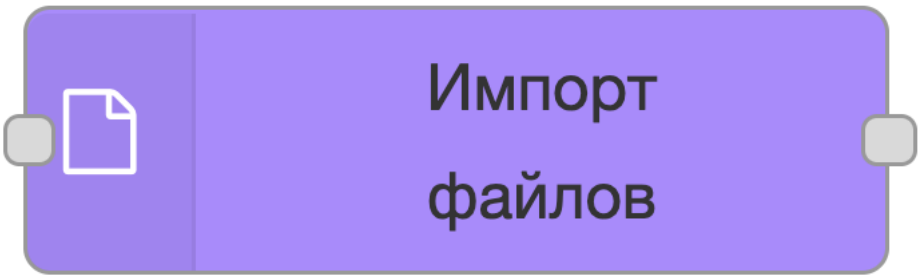


Рис. 2.106 **Узел Импорт файлов в Палитре**

Рис. 2.107 **Узел Импорт файлов на Холсте**

Данный узел позволяет произвести импорт `.xlsx`, `.csv`, `.dbf` файлов в таблицу баз:

- PostgreSQL
- ClickHouse
- MsSQL
- Oracle (экспериментальный)

При загрузке файлов типа `.xslx`, `.dbf` узел автоматически создает таблицу. При загрузке `.csv` файла необходимо заранее создать таблицу, в которую загружаются данные.

Поддержка узла прекращена после обновления Data Boring до версии 9.2.4. Не рекомендуется к использованию. Рекомендуется использование узла Datatransfer (Перенос данных).

Принимает:

- `msg.dataSourceFilePath` строка – путь к файлу-источнику на сервере.
- `msg.headerRow` число - номер строки заголовка.
- `msg.firstDataRow` число - номер первой строки с данными.
- `msg.sheetNumbers` число/строка - список страниц.
- `msg.shemaName` строка – имя схемы.
- `msg.tableName` строка – имя таблицы.

При загрузке `.xlsx` файла узел автоматически приводит заголовок к нижнему регистру, транслитерирует, а также к дублируемым записям добавляет постфикс

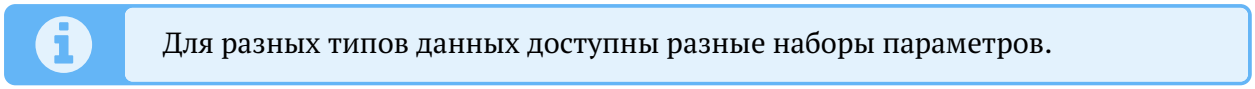
Выводит: - Результатом выполнения запроса является имя схемы, в которую была произведена запись, и таблица.

Узел имеет следующие параметры:

- `Имя` строка – имя узла, отображаемое на холсте.
- `Параметры из потока` чекбокс
- `Идентификатор получателя` строка - ранее заданный источник данных в административной панели.
- `Путь к файлу-источнику` строка
- `Номер строки заголовка` число
- `Номер первой строки с данными` число
- `Список номеров страниц` число/строка
- `Имя схемы` строка
- `Имя таблицы` строка
- `Mode` дропдаун - имеет три состояния, которые можно выбрать:

Mode	Описание
Truncate	удалить содержимое таблицы
Append	добавить к содержимому таблицы
Drop	полностью удалить таблицу

- `Преобразовывать типы` чекбокс - по-умолчанию, если не стоит галочка, производится загрузка без преобразования в формат столцбов типа `string/text`.
- `Удалить файл по завершению` чекбокс - по-умолчанию, если не указана галочка, файл не удаляется с сервера.



Для разных типов данных доступны разные наборы параметров.

2.7.11 Узел Импорт CSV

Рис. 2.108 **Узел Импорт CSV в Палитре**

Рис. 2.109 **Узел Импорт CSV на Холсте**

Данный узел позволяет произвести импорт `.csv` файла в таблицу.

Поддержка узла прекращена после обновления Data Boring до версии 9.2.4. Не рекомендуется к использованию. Рекомендуется использование узла Datatransfer (Перенос данных).

Принимает:

- `msg.payload.table` строка – имя таблицы, если не установлен в самом узле.
- `msg.payload.sink` строка - путь файла csv, хранящийся на сервере. Свойство может быть передано на вход узлу, определенным в узле `функция`:

```
msg.payload = {};

msg.payload.sink = 'file://'+flow.get("path");
msg.payload.table = flow.get("table")
return msg;
```

Выводит:

- Данный узел передает далее текущий статус операции импорта файла

Узел имеет следующие параметры:

- `Имя` строка – имя узла, отображаемое на холсте.
- `Источник данных` меню дропдаун – заданный источник данных, откуда будет произведен импорт.
- `Таблица` строка – наименование таблицы, в которую будет произведен импорт.
- `Удалить файл по завершении` чекбокс - по-умолчанию, если не указана галочка, файл не удаляется с сервера.
- `CSV с заголовками` чекбокс - по-умолчанию, если не стоит галочка, загружется `.csv` файл без заголовка
- `CSV разделитель` дропдаун - меню выбора разделителей для `.csv` файла. На выбор доступны:

  1. `tab`
  2. `|`
  3. `,`
  4. `;`

Файл удаляется всегда, даже при возникновении исключения.

2.7.12 Узел Kafka потребитель сообщений в Data Boring

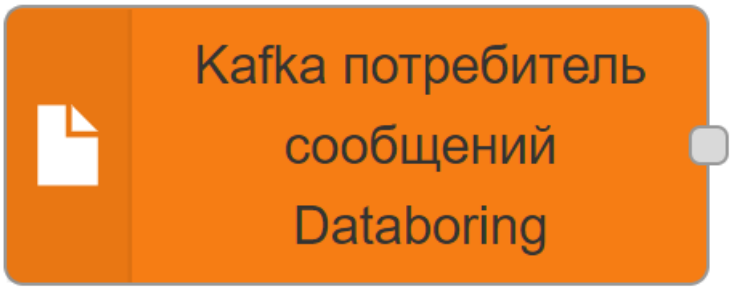


Рис. 2.110 **Узел Kafka потребитель сообщений в Data Boring в Палитре**

Рис. 2.111 **Узел Kafka потребитель сообщений в Data Boring на Холсте**

Этот узел можно использовать для работы с сообщениями Kafka. Узел-потребитель сообщений.

Потребитель тематических сообщений kafka, которые генерируются в сообщениях Data Boring. Предоставляет типы базового и высокого уровня.

2.7.13 Узел Kafka производитель сообщений в Data Boring

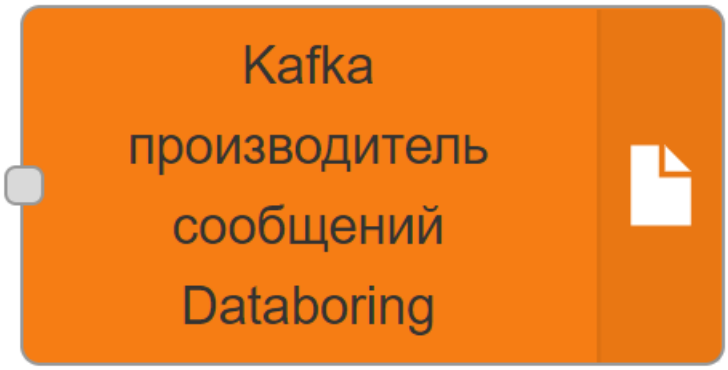


Рис. 2.112 **Узел Kafka производитель сообщений в Data Boring в Палитре**

Рис. 2.113 **Узел Kafka производитель сообщений в Data Boring на Холсте**

Этот узел можно использовать для работы с сообщениями Kafka. Узел-производитель сообщений.

Преобразует сообщение из Data Boring в тематические сообщения Kafka. Предоставляет типы базового и высокого уровня.

2.7.14 Узел Регистрация потока

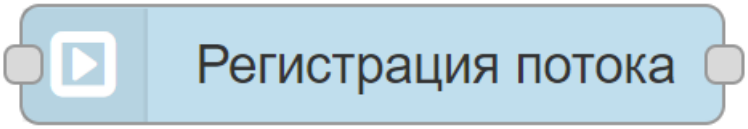


Рис. 2.114 **Узел Регистрация потока в Палитре**

Рис. 2.115 **Узел Регистрация потока на Холсте**

Узел регистрации статуса потока, для отслеживания старта, завершения и аварийного завершения потока.

Параметры:

- `Имя` строка.

Наименоваение узла на холсте:

- `Фаза` строка. Задает триггер регистрации события начала, окончания или аварийного завершения потока

Фаза:

- `Старт потока`. Регистрирует событие оркестратора при старте потока.
- `Завершение потока`. Регистрирует событие оркестратора при завершении потока.
- `Ошибка в потоке`. Регистрирует событие оркестратора при появлении ошибки во время выполнения потока.

Поддержка узла прекращена после обновления Data Boring до версии 9.2.4. Не рекомендуется к использованию.

2.7.15 Узел Период

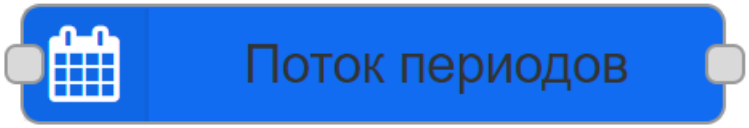


Рис. 2.116 **Узел Период в Палитре**

Рис. 2.117 **Узел Период на Холсте**

Создает поток “периодов” для импортера данных в датасеты LuxmsBI.

Узел имеет несколько выходов, каждый выход отвечает за свой тип периода – `ГОД`, `МЕСЯЦ` и так далее. Набор типов можно выбирать в конфигурации узла. В поток будут отправлены сообщения с полем `periodId`, который затем используется в узлах типа `Группировать данные для импортера`.

Параметры:

- `fromDate` – дата начала генерации периодов (включительно). Может быть переопределено во входящем сообщении в формате ISO8601.
- `toDate` – дата окончания генерации периодов (исключительно). Может быть переопределено во входящем сообщении в формате ISO8601.
- `duration` – длительность интервала генерации в формате ISO 8601 duration, если указано – параметр `toDate` игнорируется. Может принимать отрицательные значения, в таком случае генерация происходит от `fromDate` в прошлое, `fromDate` при этом исключается как последнее значение. Может быть переопределено во входящем сообщении.

Пример: Входящее сообщение `{fromDate:'2000-01-01', duration:'P10Y'}` генерирует десять периодов типа `ГОД` с 2000 по 2019 год включительно.

> Данный узел отображается только в случае, если установлен компонент luxmsbi-importer.

2.7.16 Узел Выполнить произвольный запрос на сервере

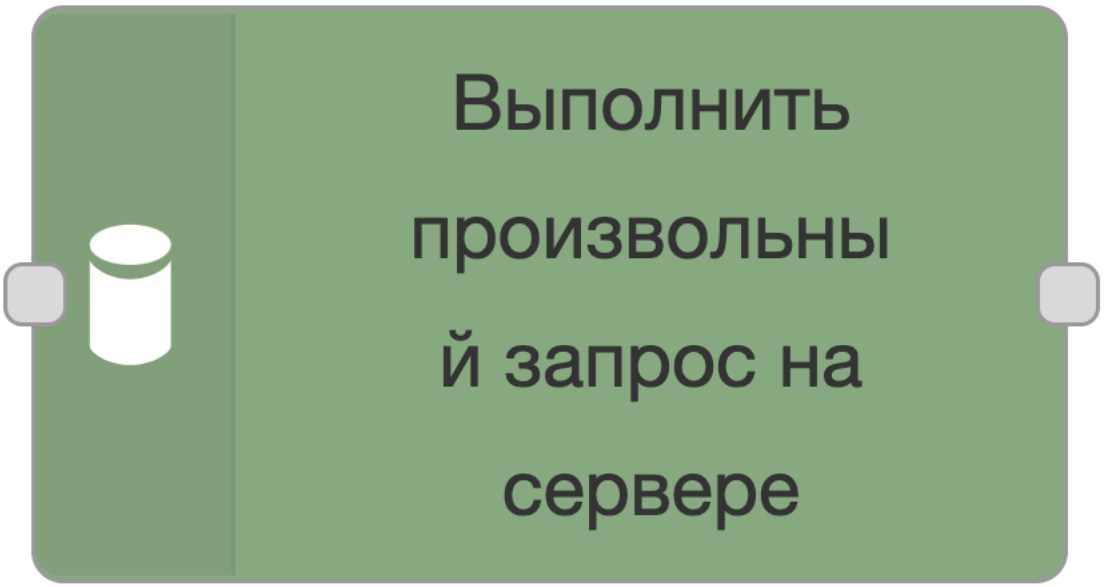


Рис. 2.118 **Узел Выполнить произвольный запрос на сервере в Палитре**

Рис. 2.119 **Узел Выполнить произвольный запрос на сервере на Холсте**

Выполнение произвольного sql-запроса в заданном источнике данных с указанными параметрами

Принимает:

- `msg.payload.query` – строку sql-запроса, если не установлен в самом узле.

Выводит:

- Результат выполнения запроса или выборки, заданной пользователем.

Параметры:

- `Имя` строка – наименование узла на холсте.
- `Источник данных` строка – заданный источник данных, для которого будет выполнен запрос.
- `Ошибку в результат` чекбокс - если задан, то код ошибки сохраняется в объект `msg`. Узел в таком случае отображает сообщение `warn` желтой меткой под узлом и поток обработки не прерывается.
- `Editor` строка – тело запроса, который должен быть выполнен при инициализации узла.

Пример запроса:

```
INSERT INTO d_date
(dt, plan, fakt)
SELECT t.day::date as dt, random() * 100 + 44 AS plan,  random() * 100 AS fakt
FROM   generate_series(timestamp '2022-05-01'
                     , timestamp '2022-10-30'
                     , interval  '1 day') AS t(day)
order by t.day::date;
```

Результат запроса, выведенный в окно отладки:

```
{
"_msgid": "b0312aba0ab8f088",
"payload": {
  "result": 183,
  "counts": [
    183
  ],
  "time": 9
},
"topic": ""
}
```

2.7.17 Узел SAP RFC

Рис. 2.120 **Узел SAP RFC в Палитре**

Рис. 2.121 **Узел SAP RFC на Холсте**

Использует `RFC` протокол для соединения с системами `SAP` и получения данных.

Принимает:

- `connectorIDстрока` – имя источника данных

- `functionNameстрока` - название удаленного метода, который будет вызван из `SAP`.
- `datagateUrlстрока` - адрес datagate в формате `http://host:port/path` (например, `http://localhost:8200/sap`). Необязательный параметр.
- `parametersJSON` - параметры запроса, отправятся в поле `body`.

Пример запроса:

```
{
 "BUSINESSPARTNER" : 8
}
```

Выводит:

Результат выполнения запроса в формате `JSON`.

2.7.18 Узел SOAP

Рис. 2.122 **Узел SOAP в Палитре**

Рис. 2.123 **Узел SOAP на Холсте**

Узел SOAP позволяет пользователям отправлять SOAP-запрос.

2.7.19 Узел Перенос данных

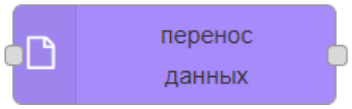


Рис. 2.124 **Узел Перенос данных в палитре**

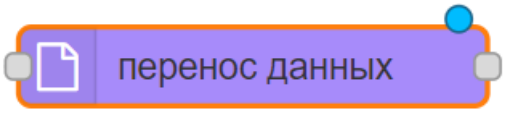


Рис. 2.125 **Узел Перенос данных на Холсте**

Типы источников данных:

- `SQL` - результат выполнения SELECT-запроса к БД
- `XLS` - содержимое Excel-файла. Поддерживаются форматы `xls` и `xlsx`
- `CSV` - содержимое CSV-файла
- `DBF` - содержимое DBF-файла
- `QVD` - содержимое QVD-файла
- `Avro` - содержимое Avro-файла
- `Parquet` - содержимое Parquet-файла

Типы получателей данных:

- `SQL` - данные сохраняются в БД (в одну или несколько таблиц)
- `CSV` - данные сохраняются в файл в формате CSV

Общие входные параметры:

- `Имя (name)` Название узла в потоке
- `Повторы (retryCount)` Общее количество попыток выполнения
- `Задержка \(мс\) (retryDelay)` Задержка между попытками в мс
- `Тип источника (sourceType)` Доступные типы `SQL`, `XLS`, `CSV`, `DBF`, `QVD`, `Avro`, `Parquet`
- `Тип получателя (destinationType)` Доступные типы `SQL`, `CSV`

Входные параметры для `SQL`-источника:

- `ИД соединения (sourceSqlConnIdent)` ИД настроенного источника данных. Например, `luxmsbi` или `ch`
- `Запрос (sourceSqlQuery)` Текст SELECT-запроса для получения данных

Входные параметры для `XLS`-источника:

- `Путь к файлу-источнику (sourceXlsFilePath)` Путь к Excel-файлу, из которого будут переноситься данные
- `Номер строки заголовка (sourceXlsHeaderRow)` Если указано, определяет номера строк на листах Excel-файла, в которых находятся заголовки
- `Номер первой строки с данными (sourceXlsFirstDataRow)` Если указано, определяет номера строк на листах Excel-файла, с которых начинаются данные
- `Список номеров страниц \(1,3...\) (sourceXlsSheetNumbers)` Если указано, определяет номера загружаемых листов из Excel-файла
- `Потоковая обработка (sourceXlsStreamProcessing)` Если включено, файл-источник загружается потоковым обработчиком. Подходит для больших файлов, где есть только данные и не надо вычислять формулы и т.п. Доступна только для файлов в формате `xlsx`
- `Удалить файл по завершении (sourceXlsDropFileOnComplete)` Если включено, файл-источник будет удалён после переноса данных

Входные параметры для `CSV`-источника:

- `Кодировка файла (sourceCsvFileEncoding)` Доступные кодировки - `utf-8`, `cp866`, `cp1251`
- `Путь к файлу-источнику (sourceCsvFilePath)` Путь к CSV-файлу, из которого будут переноситься данные
- `CSV с заголовками (sourceCsvWithNames)` Если включено, первая строка должна содержать заголовки
- `CSV разделитель (sourceCsvDelimiter)` Доступные разделители - `tab`, `pipe`, `comma`, `semicolon`
- `Сжатый (sourceCsvZipped)` Если включено, файл-источник должен быть сжат. Формат сжатия - `gzip`
- `Удалить файл по завершении (sourceCsvDropFileOnComplete)` Если включено, файл-источник будет удалён после переноса данных

Входные параметры для `DBF`-источника:

- `Кодировка файла (sourceDbfFileEncoding)` Доступные кодировки - `utf-8`, `cp866`, `cp1251`
- `Путь к файлу-источнику (sourceDbfFilePath)` Путь к DBF-файлу, из которого будут переноситься данные
- `Удалить файл по завершении (sourceDbfDropFileOnComplete)` Если включено, файл-источник будет удалён после переноса данных

Входные параметры для `QVD`-источника:

- `Путь к файлу-источнику (sourceQvdFilePath)` Путь к QVD-файлу, из которого будут переноситься данные
- `Удалить файл по завершении (sourceQvdDropFileOnComplete)` Если включено, файл-источник будет удалён после переноса данных

Входные параметры для `Avro`-источника:

- `Путь к файлу-источнику (sourceAvroFilePath)` Путь к Avro-файлу, из которого будут переноситься данные
- `Удалить файл по завершении (sourceAvroDropFileOnComplete)` Если включено, файл-источник будет удалён после переноса данных

Входные параметры для `Parquet`-источника:

- `Путь к файлу-источнику (sourceParquetFilePath)` Путь к Parquet-файлу, из которого будут переноситься данные
- `Удалить файл по завершении (sourceParquetDropFileOnComplete)` Если включено, файл-источник будет удалён после переноса данных

Входные параметры для `SQL`-получателя:

- `ИД соединения (destinationSqlConnIdent)` ИД настроенного источника данных. Например, `luxmsbi` или `ch`
- `Имя схемы (destinationSqlSchemaName)` Имя целевой схемы в БД. Если не указано, будет использоваться имя `custom`

- `Имя таблицы (destinationSqlTableName)` Имя целевой таблицы в БД
- `Режим переноса (destinationSqlMode)` Доступные режимы - `truncate`, `append`, `drop`
- `Преобразовывать типы (destinationSqlTryToCast)` Если включено, колонки строкового типа будут, при возможности, преобразованы в другие типы
- `Размер чанка (destinationSqlChunkSize)` Задаёт количество строк в чанке, которыми будут переноситься данные

Входные параметры для `CSV`-получателя:

- `CSV с заголовками (destinationCsvWithNames)` Если включено, первой строкой будет добавлена строка с заголовками
- `CSV разделитель (destinationCsvDelimiter)` Доступные разделители - `tab`, `pipe`, `comma`, `semicolon`
- `Сжатый (destinationCsvZipped)` Если включено, файл-получатель будет сжат. Формат сжатия - `gzip`

Выходные параметры для `SQL`-получателя:

Выходное сообщение в поле `payload` содержит объект: - `msg.payload.schema` - имя схемы в БД, в которой находятся таблицы с полученными данными - `msg.payload.tables` - список имён таблиц в БД, в которые переносились данные

Выходные параметры для `CSV`-получателя:

- `msg.payload.sink` - URI получателя
- `msg.payload.count` - Количество строк с данными

CSV-разделители:

Разделитель	Описание
`tab`	Символ табуляции
`pipe`	Вертикальная черта
`comma`	Запятая
`semicolon`	Точка с запятой

Режимы переноса данных:

Режим	Описание
`truncate`	Предварительно удалить содержимое целевой таблицы
`append`	Добавить к содержимому целевой таблицы
`drop`	Предварительно полностью удалить целевую таблицу

В версии **9.0 Luxms Data Boring** появилось распределенное документное хранилище, работающее по протоколу **NATS**. Это позволяет удобно передавать сообщения между различными приложениями. Если схема не указана при указании пути к файлу, то по умолчанию приложение будет работать с хранилищем **NATS**, что может привести к нежелательным последствиям для **Luxms Data Boring**. По умолчанию **NATS** использует схему `dds://` перед путем к файлу. Если необходимо работать с локальными данными, всегда явно указывайте схему. Например: `file:///tmp/excel_one.xlsx`.

2.7.20 Узел Ждать всех

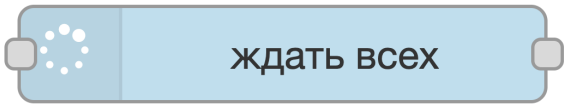


Рис. 2.126 **Узел Ждать всех в Палитре**

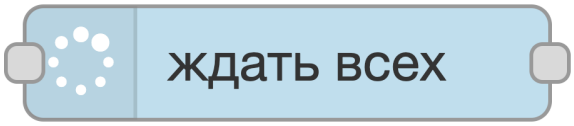


Рис. 2.127 **Узел Ждать всех на Холсте**

Узел ждёт входящих сообщений от всех присоединённых на вход узлов, собирает их в массив `msg.payload` и затем передает этот массив для дальнейшей обработки или передачи.

Количество ожидаемых сообщений должно соответствовать количеству присоединенных связей

2.7.21 Узел Отправить письмо

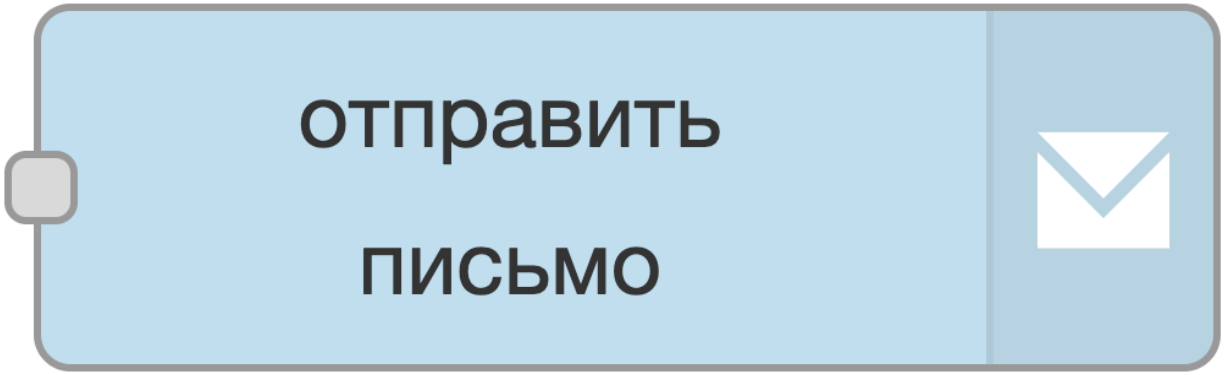


Рис. 2.128 **Узел Отправить письмо в Палитре**

Рис. 2.129 **Узел Отправить письмо на Холсте**

Узел позволяет отправить электронное письмо со вложением.

Принимает:

- `from` - адрес отправителя.
- `fromName` - имя отправителя.
- `subject` - тема письма.
- `to` - адрес получателя.
- `cc` - адреса получателей, которым отправятся копии письма, указываются через запятую.
- `body` - текстовое содержимое письма.
- `attachment` - файлы, приложенные к письму.

> Для работы узла `Отправить письмо` необходимо настроить работу с почтовым сервером в конфигурационном файле `application.properties` сервиса `luxmsbi-appserver` на каждом сервере. Необходимые настройки описаны по данной ссылке.

2.7.22 Узел SSH EXEC

Рис. 2.130 **Узел Отправить письмо в Палитре**

Рис. 2.131 **Узел Отправить письмо на Холсте**

Узел для работы с SSH соединениями. Позволяет установить SSH соединение и выполнить команду на удаленном сервере.

Принимает:

- `Command` - команда, которая будет выполнена в терминале на удаленном сервере.
- `Host` - Хост удаленного сервера.
- `Port` - порт на удаленном сервере, который прослушивается для SSH подключения.
- `MethodAuth` - Метод авторизация по паролю или по приватному ключу - `password` или `private key`.
- `Username` - имя пользователя на удаленном сервере.
- `Password` - пароль пользователя на удаленном сервере.
- `TimeOut` - максимально время ожидания ответа при попытке подключения или выполнения команды.
- `Keys` - приватный ключ для авторизации на удаленном сервере с сохранением исходного форматирования, не передается в `msg`.
- `Passphrase` - парольная фраза, которая была указана при генерации ключа.

2.8 Группа узлов jupyter

2.8.1 Узел jupyter-get

Рис. 2.132 **Узел jupyter-get в Палитре**

Рис. 2.133 **Узел Ждать всех на Холсте**

Узел позволяет получать файл с сервера Jupyter.

Входные параметры:

- `Имя (строка)` - наименование узла в потоке

- `Конфигурация Jupyter (luxmsbi-jupyter-config)` - настройки для подключения к серверу Jupyter
- `Путь к файлу (строка)` - файл, который будет передан с сервера Jupyter. Путь задаётся от корня рабочей папки на сервере Jupyter

Выходные параметры:

- `msg.payload (объект)` - объект с информацией о полученном файле. Содержимое файла находится в поле `content`

2.8.2 Узел jupyter-put

Рис. 2.134 **Узел jupyter-get в Палитре**

Рис. 2.135 **Узел Ждать всех на Холсте**

Узел позволяет загружать файлы на сервер Jupyter.

Входные параметры:

- `Имя (строка)` - наименование узла в потоке
- `Конфигурация Jupyter (luxmsbi-jupyter-config)` - настройки для подключения к серверу Jupyter
- `Путь к файлу (строка)` - файл, в который будет загружено содержимое поля `msg.payload`. Путь задаётся от корня рабочей папки на сервере Jupyter
- `Тип файла (строка)` - возможные значения - `notebook` и `file`. Если передаётся `notebook`, то `Формат файла` должен быть `json`
- `Формат файла (json)` - возможные значения - `json`, `text`, `base64`

Выходные параметры:

- `msg.payload (объект)` - объект с информацией о загруженном на сервер файле

2.8.3 Узел jupyter-run

Рис. 2.136 **Узел jupyter-get в Палитре**

Рис. 2.137 **Узел Ждать всех на Холсте**

Узел позволяет запускать на выполнение код на сервере Jupyter.

Входные параметры:

- `Имя (строка)` - наименование узла в потоке
- `Конфигурация Jupyter (luxmsbi-jupyter-config)` - настройки для подключения к серверу Jupyter
- `Выполнить ноутбук (чекбокс)` - выбор источника кода для выполнения
- `Путь к ноутбуку (строка)` если выбрано `Выполнить ноутбук`, в поле задаётся имя файла с ноутбуком. Путь задаётся от корня рабочей папки на сервере Jupyter
- `Вернуть Stdout (чекбокс)` - если стоит галочка, в `msg.payload.content` возвращается вывод в `Stdout`
- `Вернуть файл (строка)` - если не выбрано `Вернуть Stdout`, в поле можно указать файл, чьё содержимое вернётся в `msg.payload.content`. Путь задаётся от корня рабочей папки на сервере Jupyter
- `Питон-маджики (текст)` - если не выбрано `Выполнить ноутбук`, в поле задаётся код питон-маджиков для выполнения
- `Питон-скрипт (текст)` - если не выбрано `Выполнить ноутбук`, в поле задаётся код питон-скрипта для выполнения

Выходные параметры:

- `msg.payload.content (строка)` - если выбрано `Вернуть Stdout`, в данном поле возвращается вывод в `Stdout`. Иначе, если указан путь в поле `Вернуть файл`, в данном поле возвращается содержимое файла по этому пути

3 Таблицы решений

3.1 Интерфейс пользователя

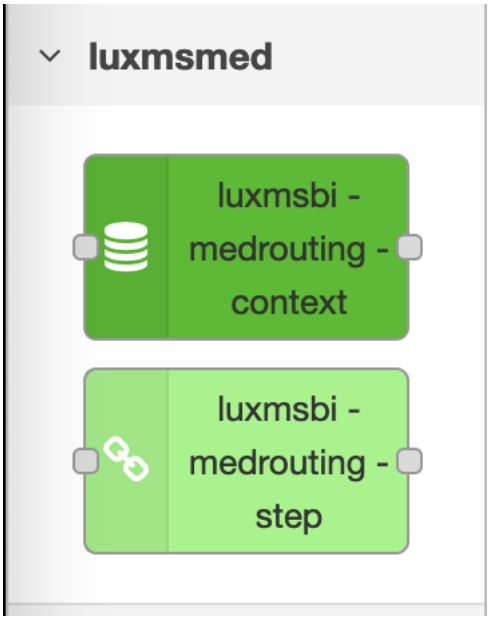


Рис. 3.1 **Узлы для работы с настроечными таблицами**

3.1.1 luxmsbi-medrouting-context

Этот узел должен идти первым. В нём задаётся подключение к источнику данных для всех последующих *решающих таблиц*.

3.1.2 luxmsbi-medrouting-step

Этот узел предоставляет интерфейс к *решающим таблицам*.

3.1.3 Пример потока

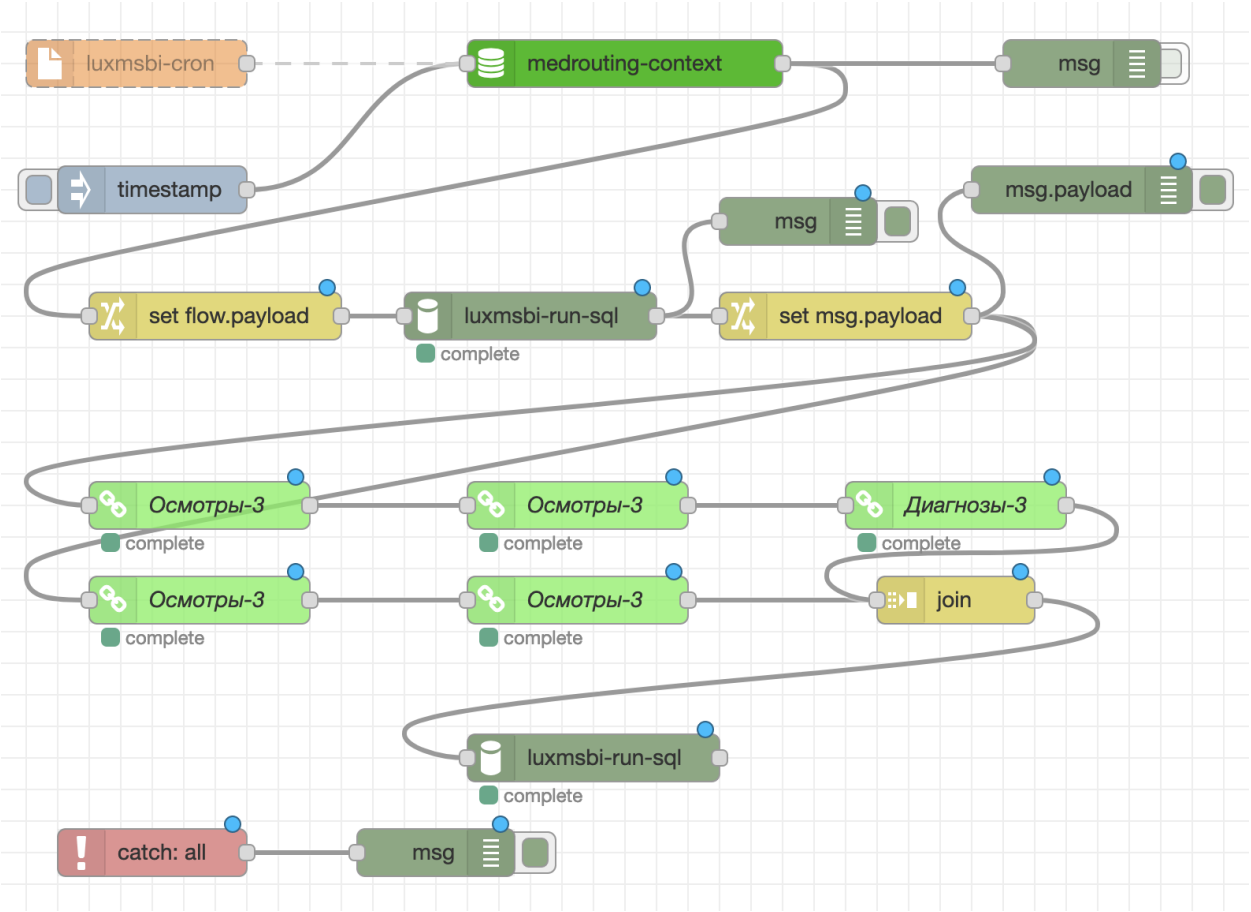


Рис. 3.2 **Пример потока с решающими таблицами**

3.1.4 Подготовка данных

Для подготовки начальных данных, создания необходимых таблиц и прочих похожих задач можно пользоваться обычными инструментами Data Boring. Например, можно воспользоваться узлом `luxmsbi-run-sql`. Этот узел поддерживает шаблонизацию с помощью специального синтаксиса. Переменные из среды Data Boring можно подставлять в тело запроса. Для этого используйте конструкцию: `${var_name}`.

Пример:

```
create table if not exists databoring."flow_${msg.payload.flowId}_all"
(
  patient_id      uuid,
  event_date      date,
  patient_sex     char(1),
  patient_age     int,
  lpu_code        text,
  visit_case_type int,
  vitalparam_code text,
```

```
11    mpstr_value     text,
12    mpint_value     int,
13    vitalspr_code   text,
14    mkb             text
15  );

17  truncate table databoring."flow_${msg.payload.flowId}_all";

19  insert into databoring."flow_${msg.payload.flowId}_all"
20  select
21    k."identityCode"::uuid as patient_id,
22    v."visitDate" as event_date,
23    'F' as patient_sex,
24    date_part('year', age(k."dateTaken", k."datR")) as patient_age,
25    l.code as lpu_code,
26    v."caseTypeId" as visit_case_type,
27    vp.code as vitalparam_code,
28    mstr.value as mpstr_value,
29    mval.value as mpint_value,
30    vspr.code as vitalspr_code,
31    ml.mkb
32  from reg_pregnancy.karta k
33  join reg_pregnancy.visit v on v."kartaId" = k.id
34  join reg_pregnancy.lpufull l on l."lpuId" = v."lpuId"
35  left join reg_pregnancy.mkblist ml on ml."visitId" = v.id
36  left join reg_pregnancy.meddocument md on md."visitId" = v.id
37  left join reg_pregnancy.meddocquantity mdq on mdq."docId" = md.id
38  left join reg_pregnancy.vitalparam vp on vp.id = mdq."paramId"
39  left join reg_pregnancy.medicalparamspr mspr on mspr."quantityId" = mdq.id
40  left join reg_pregnancy.medicalparamstring mstr on mstr."quantityId" = mdq.id
41  left join reg_pregnancy.medicalparamvalue mval on mval."quantityId" = mdq.id
42  left join reg_pregnancy.vitalspr vspr on vspr.id = mspr.value
43  where k."dateTaken" between '${msg.payload.startDate}' and current_date;


46  create table if not exists databoring."flow_${msg.payload.flowId}"
47  (
48    patient_id        uuid not null,
49    route_node_ident  text not null,
50    node_id           text not null,
51    event_date        date,
52    patient_sex       char(1),
53    patient_age       int,
54    lpu_code          text,
55    step_no           int,
56    step_result       text,
57    reason text
58  );

60  create table if not exists databoring."flow_${msg.payload.flowId}_s"
61  as table databoring."flow_${msg.payload.flowId}" with no data;

63  truncate table databoring."flow_${msg.payload.flowId}_s";
```

3.1.5 Переменные среды выполнения Data Boring

Переменная	Описание
msg.payload.flowId	id потока (уникален для сервера Data Boring)
msg.payload.startDate	Дата, выбранная в контексте medrouting-context

3.2 Описание настроечных таблиц в БД

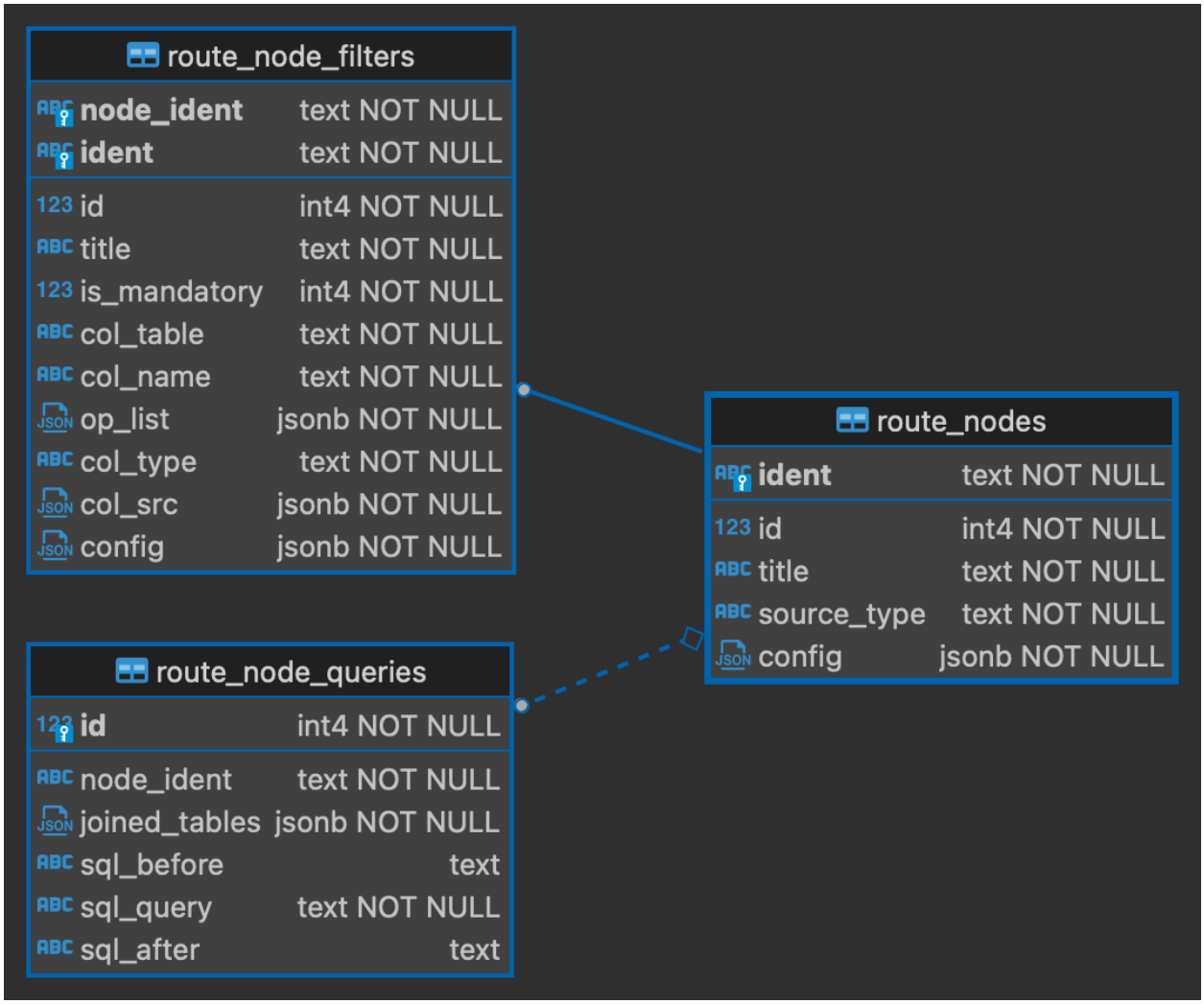


Рис. 3.3 **Схема настроечных таблиц**

3.2.1 databoring.route_nodes

В этой таблице хранятся типы *решающих таблиц*, которые будут доступны в интерфейсе Data Boring.

Таблица 3.2 **Струкутра таблицы `databoring.route_nodes`**

Столбец	Тип	Описание
id	integer	заполняется автоматически
ident	text	текстовая метка-ключ
title	text	название для интерфейса
source_type	text	название регистра (`reg_platform` или `reg_pregnancy`)
config	jsonb	системные настройки

Таблица 3.3 **Пример заполнения `databoring.route_nodes`**

id	ident	title	source_type	config
1	iam-reg-pregnancy-observation	Осмотры	reg_pregnancy	{}
2	iam-reg-pregnancy-diagnosys	Диагнозы	reg_pregnancy	{}
3	iam-reg-pregnancy-observation-2	Осмотры-2	reg_pregnancy	{}
4	iam-reg-pregnancy-diagnosys-2	Диагнозы-2	reg_pregnancy	{}
5	iam-reg-pregnancy-observation-3	Осмотры-3	reg_pregnancy	{}
6	iam-reg-pregnancy-diagnosys-3	Диагнозы-3	reg_pregnancy	{}

3.2.2 databoring.route_node_filters

Таблица 3.4 **Струкутра таблицы `databoring.route_node_filters`**

Столбец	Тип	Описание
id	integer	заполняется автоматически
node_ident	text	привязка к `route_nodes`
ident	text	текстовая метка-ключ
title	text	название для интерфейса
is_mandatory	integer	является ли обязательным
col_table	text	таблица в БД, с которой связано это условие
col_name	text	столбец в БД, с которым связано это условие
op_list	jsonb	список допустимых операций сравнения
col_type	text	тип столбца в БД
col_src	jsonb	служебное поле
config	jsonb	системные настройки

Пример заполнения поля `op_list`: `["=", ">", ">=", "<=", "<", "!=", "in", "between"]`

Таблица 3.5 **Пример заполнения `databoring.route_node_filters`.**

id	node_ident	ident	title	is_mandatory	col_table	col_name
20	iam-reg-pregnancy-diagnosys-2	lpu_code	OID MO	1		lpu_code
21	iam-reg-pregnancy-diagnosys-2	patient_age	Возраст пациентки	1		patient_age
22	iam-reg-pregnancy-diagnosys-2	vitalparam_code	Код результата осмотра	0		vitalparam_code
23	iam-reg-pregnancy-diagnosys-2	mkb	Код МКБ	0		mkb
24	iam-reg-pregnancy-diagnosys-2	_prev_reason	Причина	0	prev_step	reason

3.2.3 databoring.route_node_queries

Таблица 3.6 **Струкутра таблицы `databoring.route_node_queries`**

Столбец	Тип	Описание
id	integer	заполняется автоматически
node_ident	text	привязка к `route_nodes`
joined_tables	jsonb	Список столбцов, которые используются в решающей таблице. используется для выбора SQL шаблона.
sql_before	text	Запрос, выполняемый перед основым
sql_query	text	Основной запрос
sql_after	text	Запрос, выполняемые после основного

4 Книга рецептов

4.1 Инициализация свойств для потока

`inject` отправляет сообщение в поток вручную или через равные промежутки времени. Данные в сообщении могут быть различных типов, включая строку, числа, логические значения, объекты JavaScript, значения потоковых/глобальных контекстов или метку текущего времени, которая является текущим временем в миллисекундах, прошедших с 1 января 1970 года.

Создадим простейший поток с использованием `inject`:

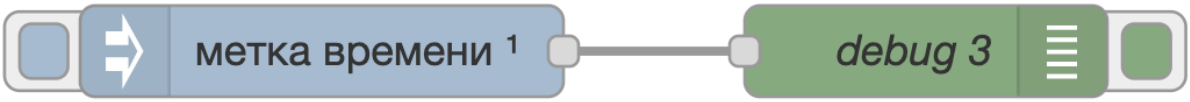


Рис. 4.1 **Поток с inject**

Рассмотрим пример с включенными сообщениями строки, числа, логического значения и метки времени при ежедневном повторении в 12:00:

Изменить узел inject

Удалить Отмена Готово

Свойства

Имя Имя

msg. topic = az

msg. payload = метка времени

msg. number = az 12345

msg. boolean = az true

msg. array = {} [-9,10,-5,1,0,1000]

+ добавить inject now

Отправить через 0.1 сек, затем

Повторять в определенное время

в 12:00

по пн вт ср чт пт сб вс

Рис. 4.2 **Свойства inject**

В результате запуска узла получим следующий ответ:

По умолчанию узел запускается вручную по нажатии на кнопку в редакторе. Его также можно настроить на автоматический запуск с интервалом через равные промежутки времени или по расписанию в определённые дни и в определённое время.

Код потока Initializing properties for a stream

4.2 Планировщик задач

Данный узел предназначен для управления периодическим запуском потока. Предоставляет более глубокие возможности, нежели `inject`. В шаблон узла можно вписывать любую конфигурацию `cron`.

Документация cron

Примеры конфигураций использования cron

Рис. 4.3 **Поток с планировщиком задач**

В данном примере используем данную конфигурацию: `16 8,10,12,14,16,18 * * *` - которая означает, что данный узел отдаст команду на старт потока в 16 минут в 9,11,13,15,17,19 часов ежедневно и ежемесячно.

В данном примере используем простой запрос 1 из базы.

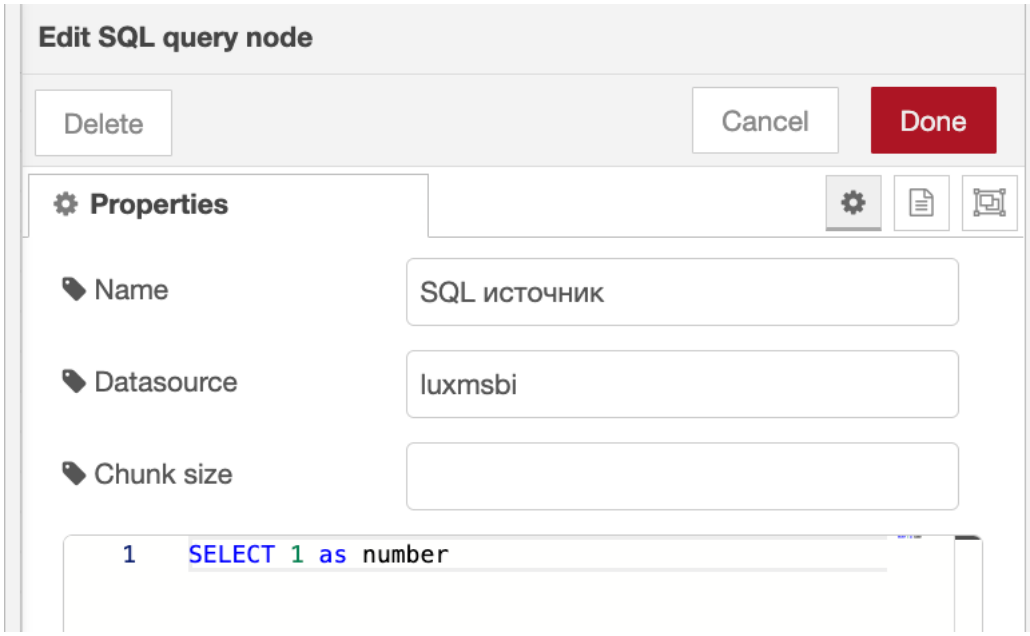


Рис. 4.4 **Запрос в источник**

В результате получаем следующий ответ:

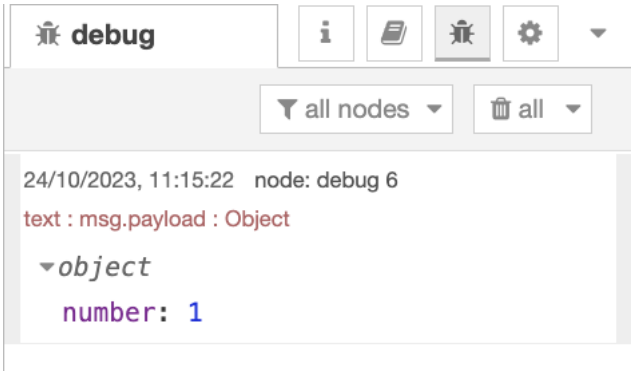


Рис. 4.5 **Результат работы узла `Планировщик задач`**

> В связи с некоторыми ограничениями, которые на данный момент не разрешены, сам запрос может запускаться до 2 минут позже выбранной минуты, но при этом периодичность запуска остаётся точной, то есть при данной конфигурации запрос может отратработать в 9:18 (9:17:29 в нашем случае), а не в 9:16 ровно, но при этом следующий запрос так же отработает в 11:18 (11:17:29).

Код потока Task Scheduler

4.3 Отслеживание изменений в папке

Узел `watch` предназначен для отслеживания изменений действий с определенным файлом, может являться триггером для запуска потока. В данном примере рассмотрим простейший поток для общего понимания работы узла.

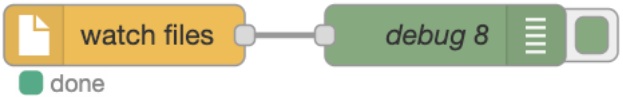


Рис. 4.6 **Поток с `watch`**

“Нацелим” наш `watch` на файл `Data.xslx`, находящийся в папке `/tmp/`:

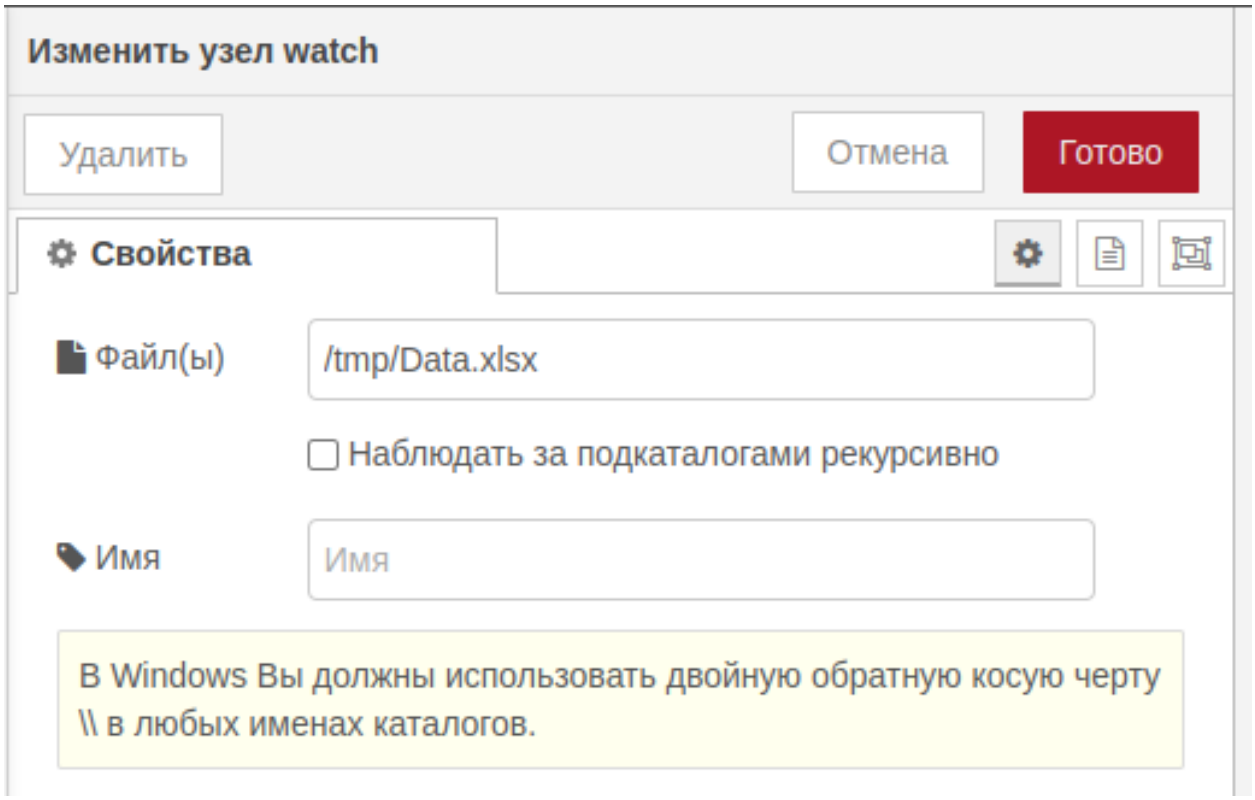


Рис. 4.7 **Параметры `watch`**

При любом изменении файла (удаление, создание, обновление) мы будем получать сообщение от данного узла.

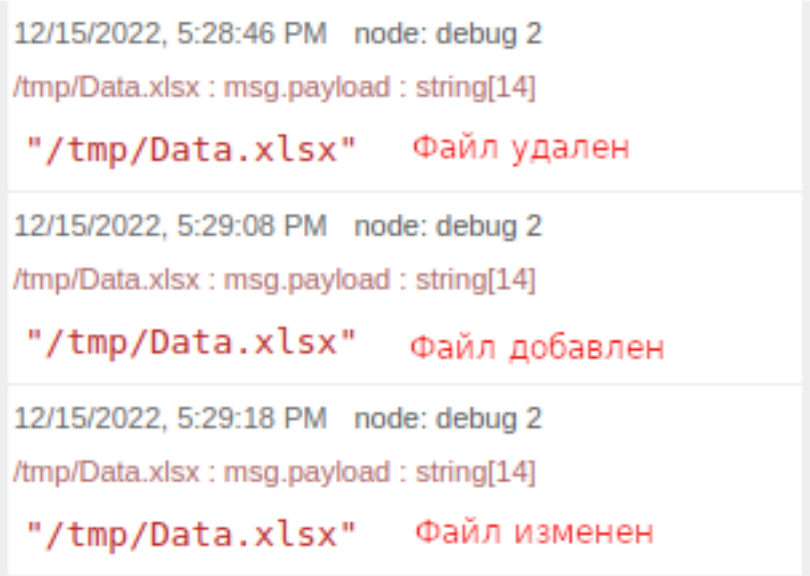


Рис. 4.8 **Результат работы `watch`**

Внедрение узла `watch` очень удобно при перезаписи какого-либо файла, являющегося источником данных или справочником. То есть, можно после данного узла поставить узлы `Импорт файлов` - который прогрузит файл в базу, а затем `SQL источник`, который запустит соответствующие скрипты, для обновления данных в кубах. В результате получим поток, который самостоятельно обновит данные при появлении обновлений.

При этом нужно помнить, что при удалении файла данный узел так же подаст сигнал на исполнение потока, в связи с чем стоит производить дополнительные проверки, чтобы отсутствие файла не привело к потере отображаемых данных.

Код потока Tracking changes in a folder

4.4 SQL источник / Выполнить произвольный запрос на сервере

Узлы `SQL источник` и `Выполнить произвольный запрос на сервере` выполняют схожую функцию - запросы к базе данных, подключенной к данному контуру. Однако у них есть одно существенное отличие - `SQL источник` обязательно ожидает какой-то ответ после запроса, в то время как `Выполнить произвольный запрос на сервере` наоборот, работает только когда запрос ничего не возвращает (например процедуры или функции возвращающие `void`).

Рассмотрим несколько примеров на основании простейшего потока:

Рис. 4.9 **Поток с использованием `SQL источник`**

В запрос узла пропишем запрос `SELECT 1;`.

В результате получаем следующий ответ:

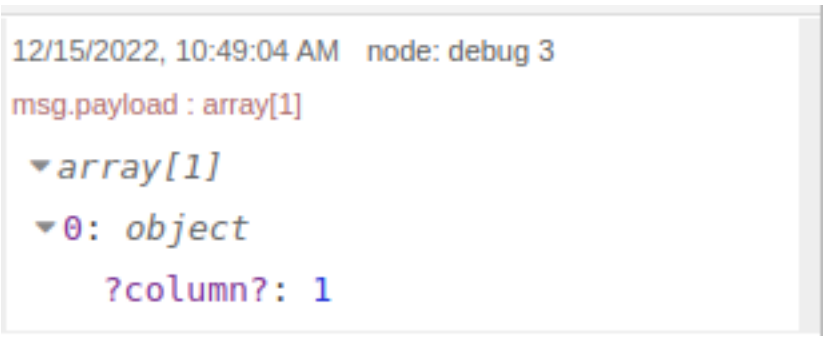


Рис. 4.10 **Результат исполнения `SQL источник` запроса 1**

Также можно передавать свойства, определённые в узле `function`, на вход узлу `SQL источник`.

```
msg.number = 1;
return msg;
```

В самом узле `SQL источник` в запросе необходимо указать, какое свойство принимается на вход.

```
select ${msg.number} as nm
```

Код потока SQL source 1

Либо в самом узле `function` формировать запрос, который пойдет на вход узлу `SQL источник`

```
let number = 1;
msg.payload = {};
msg.payload.query = 'select ' + number + ' as nm';

return msg;
```

Код потока SQL source 2

Так же, при многострочном запросе есть возможность прописать количество сообщений, объединенных в массив, например, в данном случае 10 сообщений (строк запроса) будут объединены в один массив, остальные 6 строк - в другой массив:

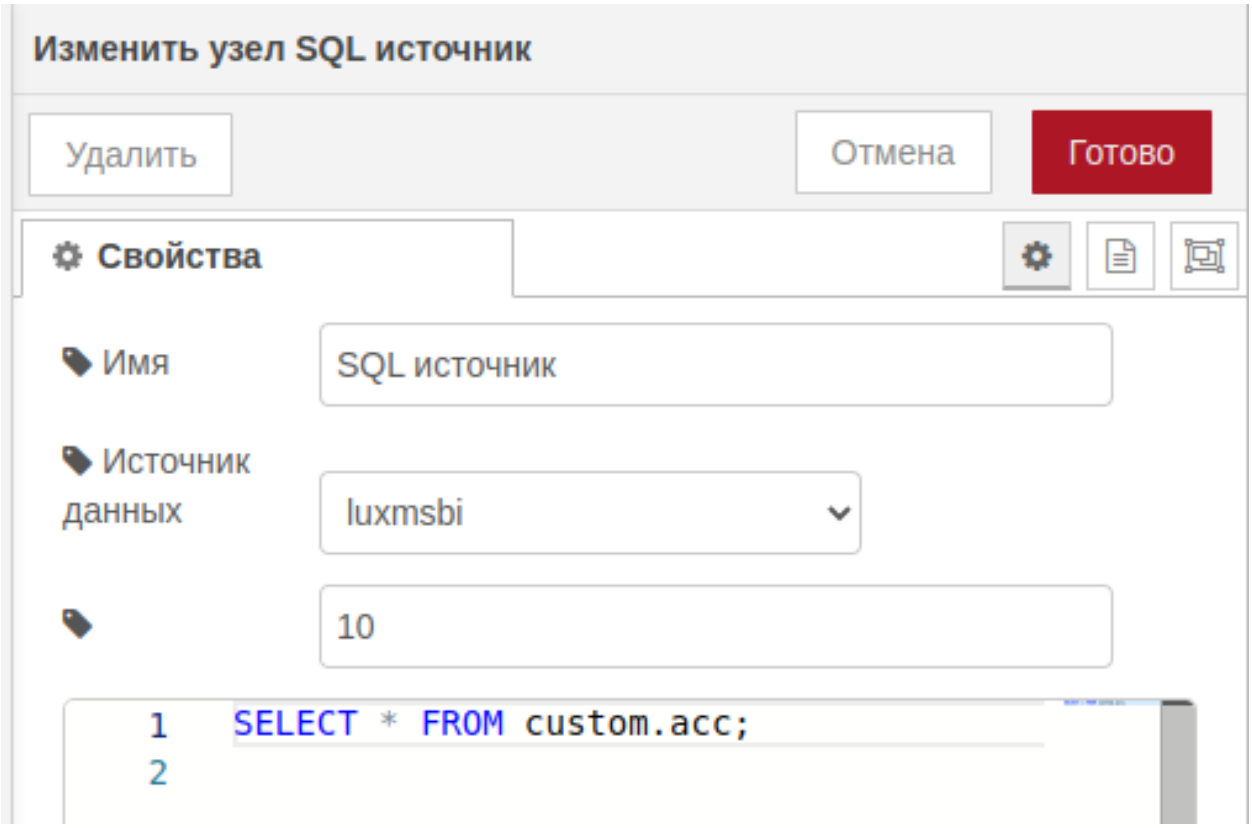


Рис. 4.11 **Запрос с объединением 10 сообщений в массив**

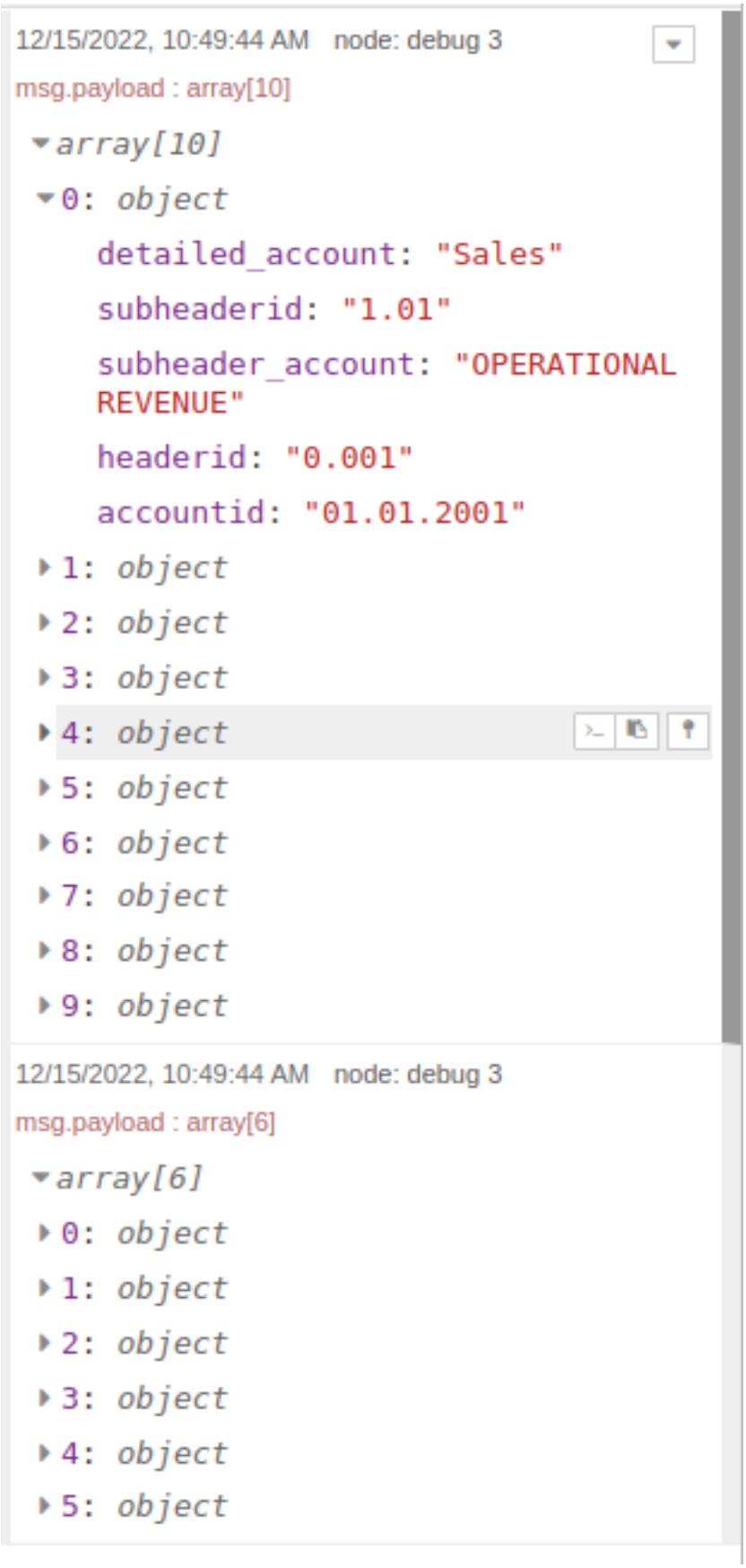


Рис. 4.12 **Результат исполнения `SQL источник` с массивами**

При этом, если мы попробуем в запрос прописать функцию которая ничего не возвращает,

например:

```
CREATE PROCEDURE custom.create_table_foo()
AS $$
  CREATE TABLE custom.foo ( id int )
$$ LANGUAGE sql;
```

При запросе данной процедуры (`CALL custom.create_table_foo();`) получим следующий ответ:

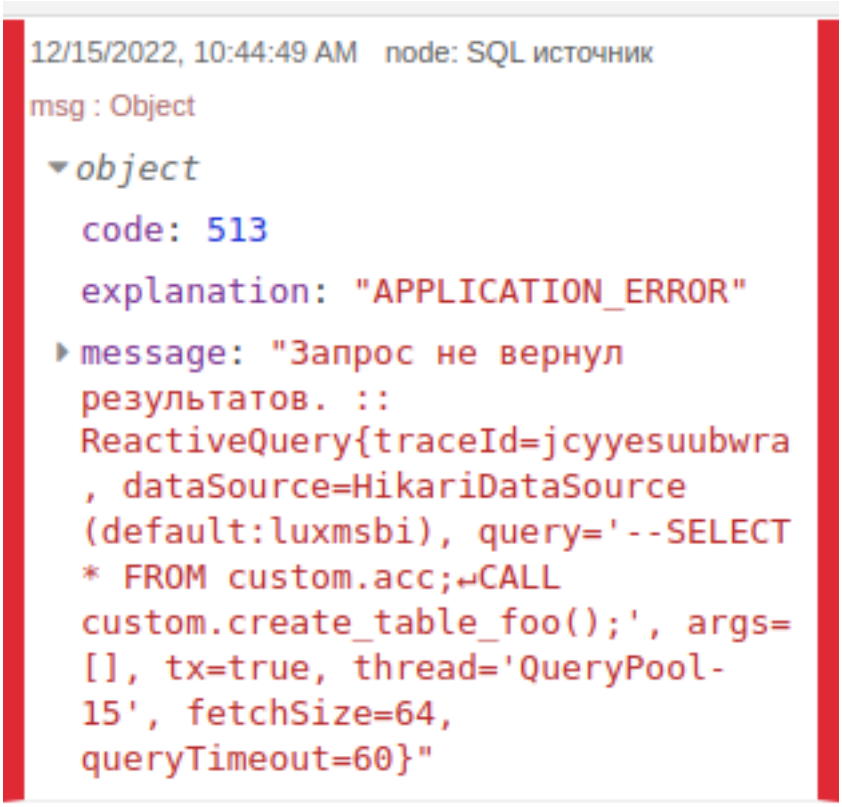


Рис. 4.13 **Результат исполнения `SQL источник` при запросе без возврата**

При этом, при запуске того же запроса с помощью узла `Выполнить произвольный запрос на сервере`:

Рис. 4.14 **Поток с использованием `Выполнить произвольный запрос на сервере`**

Получим следующий ответ:

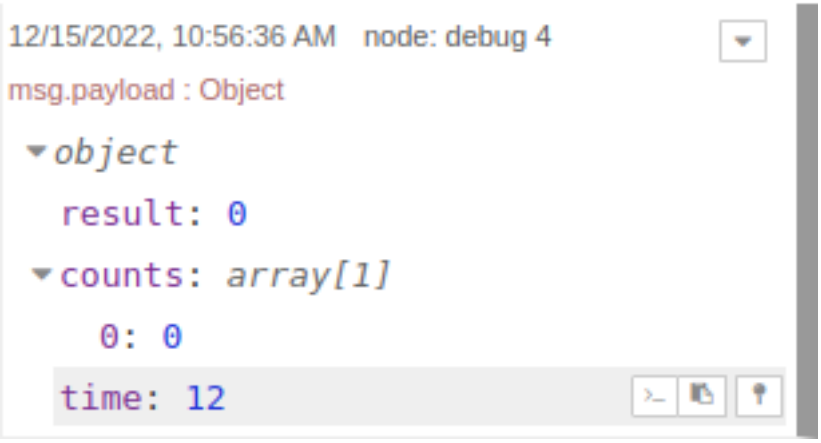


Рис. 4.15 **Результат успешного исполнения `Выполнить произвольный запрос на сервере`**

А при запросе `SELECT 1` получим ошибку:

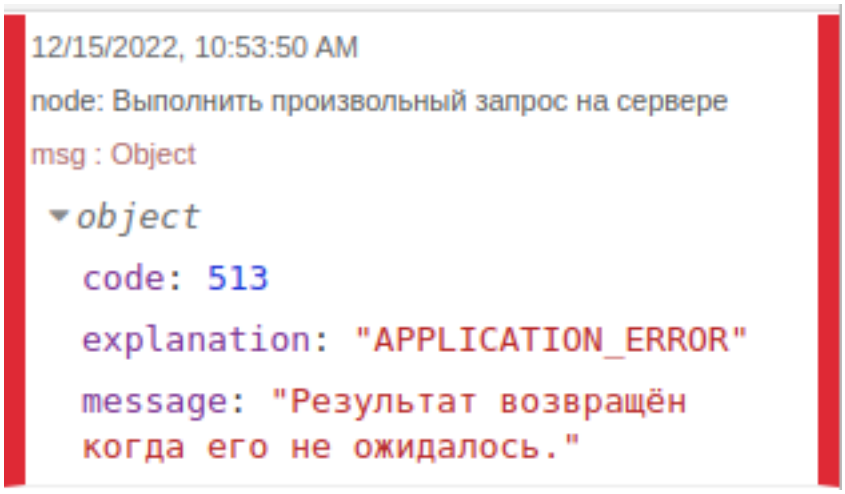


Рис. 4.16 **Результат исполнения `Выполнить произвольный запрос на сервере` запроса 1**

Код потока SQL source 3

4.5 Перенос данных (sql/xlsx/csv/dbf/qvd/avro/parquet)

Узел `Перенос данных` дает возможность загрузить в базу данных различные данные из файлов форматов:

1. Excel (xls/xlsx);
2. csv;
3. dbf;
4. qvd;
5. avro;
6. parquet.

В данном примере рассмотрим возможность прогрузки файла в формате `.xslx`. Для этого создадим простейший поток.

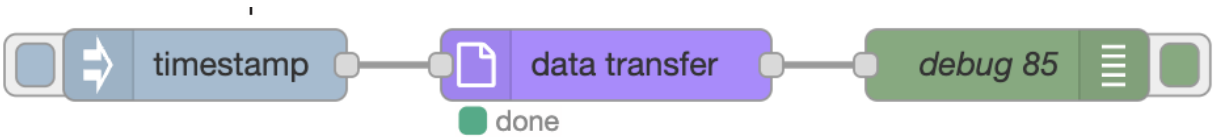


Рис. 4.17 **Поток с переносом данных из файла xlsx**

Сам узел имеет множество различных параметров, каждый из которых разберём отдельно:

Edit data transfer node

Delete Cancel Done

Properties

Name: Name

Retry count: 1

Retry delay (ms): 5000

Source

Source type: SQL XLS CSV DBF QVD Avro Parquet

Data source file path: file:///tmp/data.xlsx

Sheets to import: 1, 2

Header rows: 1, 1

First data rows: 2, 2

Stream processing: ☐

Drop file on complete: ☐

Destination

Destination type: SQL CSV

Connection ID: luxmsbi

Schema name: custom

Table name: data, sex

Mode: Append

Convert types: ☐

Chunk size: 1000

Рис. 4.18 **Параметры узла “Перенос данных”**

1. Параметры из потока дают возможность принимать конфигурацию данного узла из данных потока (например заданных ранее узлом function);
2. Тип данных - выбор типа загружаемого файла (excel/csv/dbf/qvd/avro/parquet);
3. Идентификатор получателя - наименование id источника данных, стандартный источник - `luxmsbi` (можно посмотреть в административной панели во вкладке `Источник данных`);
4. Путь к файлу - абсолютный путь к файлу, расположенному на том же сервере, что и база данных;
5. Номер строки заголовка - номер строки заголовка данных, по стандарту - 1 строка, можно выбрать любую (если заголовка нет, можно оставить поле пустым);
6. Номер первой строки с данными - аналогично строке заголовка, можно выбрать с какой строки будут начинаться данные (в случае с 1 строкой с заголовком -> 2 строка будет с данными);
7. Список номеров страниц - можно выбрать только необходимые листы из файла excel, для прогрузки;
8. Имя схемы - наименование схемы в базе данных, в которую требуется прогрузить данные;
9. Имя таблицы - можно задать имя таблицы, в которую прогрузятся данные из файла, иначе наименование таблицы будет соответствовать наименованию листа excel с транслитерацией (`лист = list`), (для многолистовой прогрузки рекомендуется оставить пустым, в таком случае все листы прогрузятся в таблицы с наименованиями, соответствующими наименованиям листов файла);
10. Mode - тип загрузки данных:

    - Drop - удаление таблицы с соответствующим наименованием и замена на новосозданную;
    - Append - добавление данных без какого-либо удаления;
    - Truncate - отчистка таблицы с соответствующим наименованием и загрузка данных в неё.

11. Преобразовывать типы - можно преобразовать типы данных в подходящие по логике, то есть чтобы числовые значения были в таблице в столбце с типом данных `int`, дата в формате `date` и т.д. В связи с множеством различных вариаций форматов, после прогрузки желательно скорректировать, при необходимости, форматы. Иначе можно прогрузить данные в формате ‘text’, то есть как есть, без каких-либо изменений;
12. Удалить файл по завершении - удалить исходный файл после прогрузки для освобождения места на диске.

В результате, после выполнения узла получаем следующие данные:

Рис. 4.19 **Результат работы узла `Перенос данных`**

- Наименование схемы, в которую были прогружены данные;
- Наименования таблиц, в которые были прогружены данные;
- Количество строк, которые были загружены в базу.

Код потока Importing files

4.6 http in / http response

С помощью узлов `http in` и `http response`, а так же любого html в узле `шаблон` можно создать веб-страницу по пути `.../databoring/...`. Рассмотрим простейший пример:

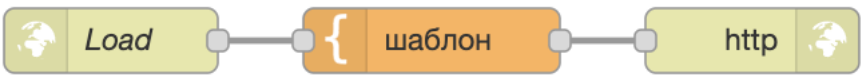


Рис. 4.20 **Поток с html**

В узле `http in` (Load) можно указать следующие параметры:

- Метод запроса:

  1. GET
  2. POST
  3. PUT
  4. DELETE
  5. PATCH

- URL страницы, который будет расположен по пути `.../databoring/`;

- Имя узла.

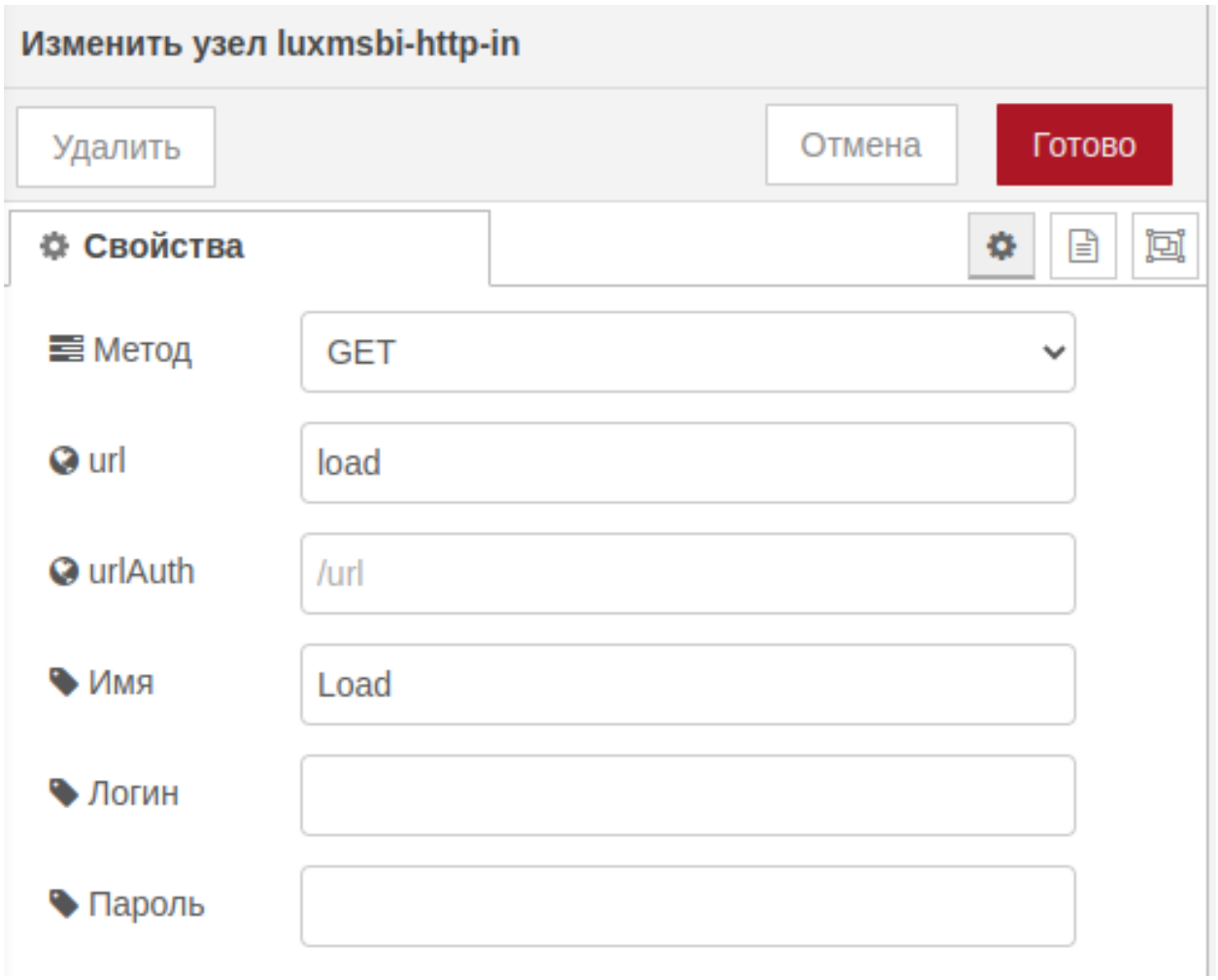


Рис. 4.21 **Узел `http in`**

В узел `шаблон` пропишем html следующего содержания:

```
<!DOCTYPE html>
<html>
        <head>
      <title>Заголовок</title>
    </head>
    <body>
        Какой-то текст
    </body>
</html>
```

В результате получаем страницу следующего содержания:

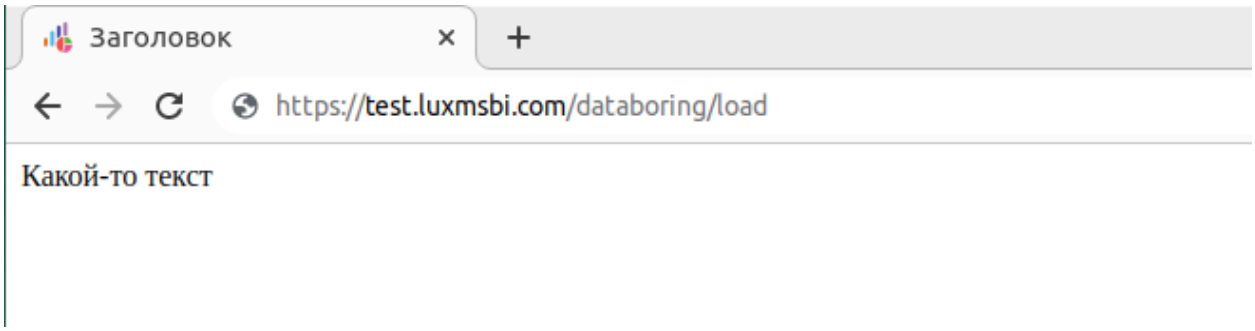


Рис. 4.22 **Веб-страница с простейшим HTML**

Так же можно собирать более сложные страницы, содержащие css, javascript, логику, переходы на другие страницы и т.д. Например рассмотрим следующий поток:

Рис. 4.23 **Поток с использованием css**

В узел `шаблон` (css) пропишем css страницы:

```
div {
height: 200px;
position: absolute;
width: 480px;
height: 200px;
top: 40%;
left: 40%;

/* background dash */

background: #FFFFFF;
box-shadow: 0px 10px 20px rgba(79, 79, 155, 0.05), 0px 2px 6px rgba(79, 79, 155,
   0.05), 0px 0px 1px rgba(79, 79, 155, 0.05);
border-radius: 8px;

}

p {
    font-family: Roboto, Arial;
    text-align:  center;
    font-size: large;
}
text-info{
position: absolute;
width: 438px;
height: 225px;
top: 5%;
left: 5%;
}
```

```
30 body{
31     background-color: #F2F2F8;
32    }

34 /* CSS */
35 .button-y {
36   background-color: #5FB138;
37   border-radius: 8px;
38   border-style: none;
39   box-sizing: border-box;
40   color: #FFFFFF;
41   cursor: pointer;
42   display: inline-block;
43   font-family: Roboto, Arial;
44   font-size: 16px;
45   font-weight: 600;
46   width: 216px;
47   height: 48px;
48   line-height: 20px;
49   list-style: none;
50   margin: 0;
51   outline: none;
52   padding: 14px 16px;
53   position: relative;
54   text-align: center;
55   text-decoration: none;
56   transition: color 100ms;
57   vertical-align: baseline;
58   user-select: none;
59   position: absolute;
60   top: 40%;
61   left: 25%;
62   -webkit-user-select: none;
63   touch-action: manipulation;
64 }

66 .button-y:hover {
67     opacity: 0.7;
68 },
69 .button-y:focus {
70   background-color: #F082AC;
71 }

73 copyright {
74   display:block;
75   margin-top: 100px;
76   text-align: center;
77   font-family: Helvetica, Arial, sans-serif;
78   font-size: 12px;
79   font-weight: bold;
80   text-transform: uppercase;
81   position: absolute;
```

```
  top: 80%;
  left: 44%;
}
copyright a{
  text-decoration: none;
  color: #EE4E44;
}
```

А в узел шаблон пропишем html следующего содержания:

```
<!DOCTYPE html>
<html>
       <head>
     <title>Вгрузка завершена</title>
    <style>
{{{css}}}
</style>
   </head>
   <body>
       <div>
           <text-info>


      <p>Данные вгружены</p> <hr>
    <a href="/databoring/start" class="button-y">В начало</a>
                 </text-info>
      </div>

       <copyright>
      <a class="fusion-no-lightbox" href="https://luxmsbi.com/" target=↩
  "_self"><img src="https://luxmsbi.com/wp-↩
  content/uploads/2020/12/LogoBI_colour_2-e1617718587530.png" data-orig-src=↩
  "https://luxmsbi.com/wp-content/uploads/2020/12/LogoBI_colour_2-↩
  e1617718587530.png" class="img-responsive wp-image-4839 lazyloaded" width=↩
  "256" height="45"></a>
  </a>
</copyright>

   </body>
</html>
```

В результате получим страницу следующего вида:

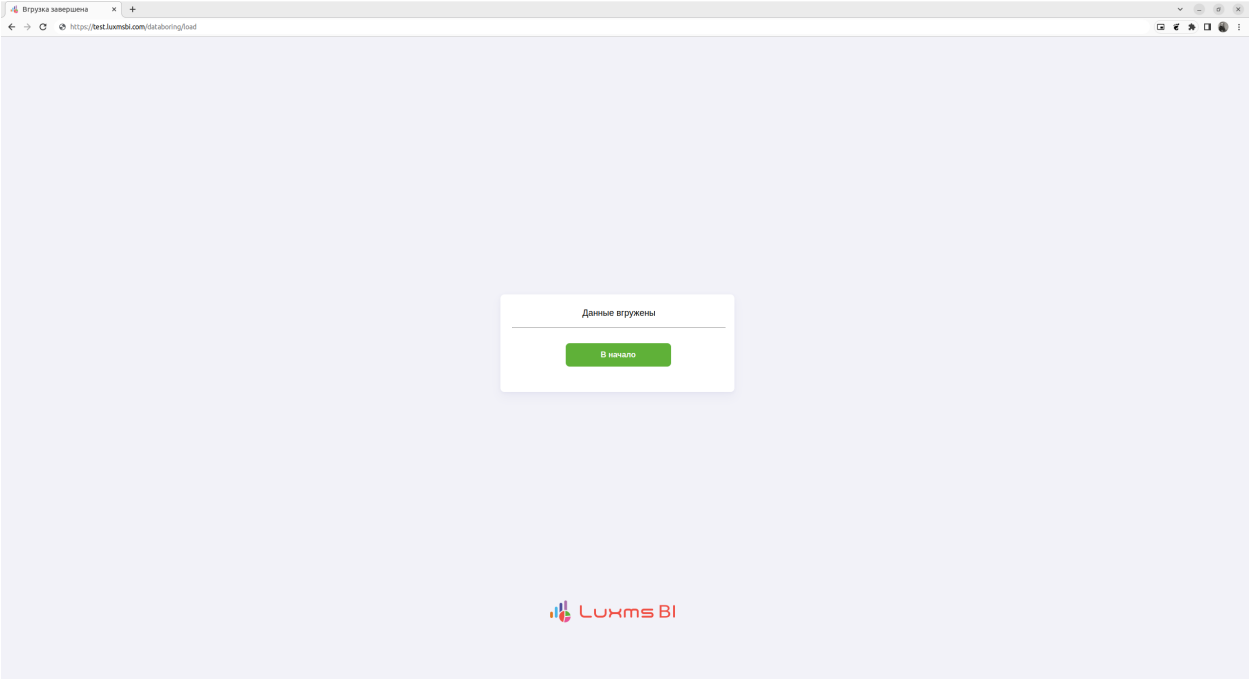


Рис. 4.24 **Веб-страница с css**

Данная страница так же выводит статический текст, но при этом ведет на страницу `start` при нажатии на кнопку. Более сложные потоки могут содержать большее количество переходов, динамически изменяемый текст (например полученный ранее из базы с помощью узла `SQL источник`) и т.д.

Код потока http in / http response с использованием css

4.7 Запуск службы SOAP

Пример потока, который вызывает метод `Multiply` службы `SOAP` wsdl и возвращает результат операции. Измените значения параметров `A` и `B` в узле `функция`.

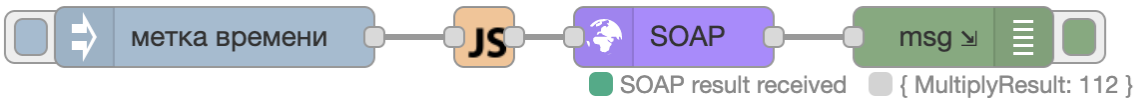


Рис. 4.25 **Поток отправки SOAP-запроса**

Код, установленный в узле `функция`, и поданный на вход узлу `SOAP`

```
var newmsg={
  options:{},
  headers:{},
```

```
5   payload:{"intA":2, "intB":56}
6 };
7 return newmsg;
```

Поля, которые были установлены в свойствах узла `SOAP`

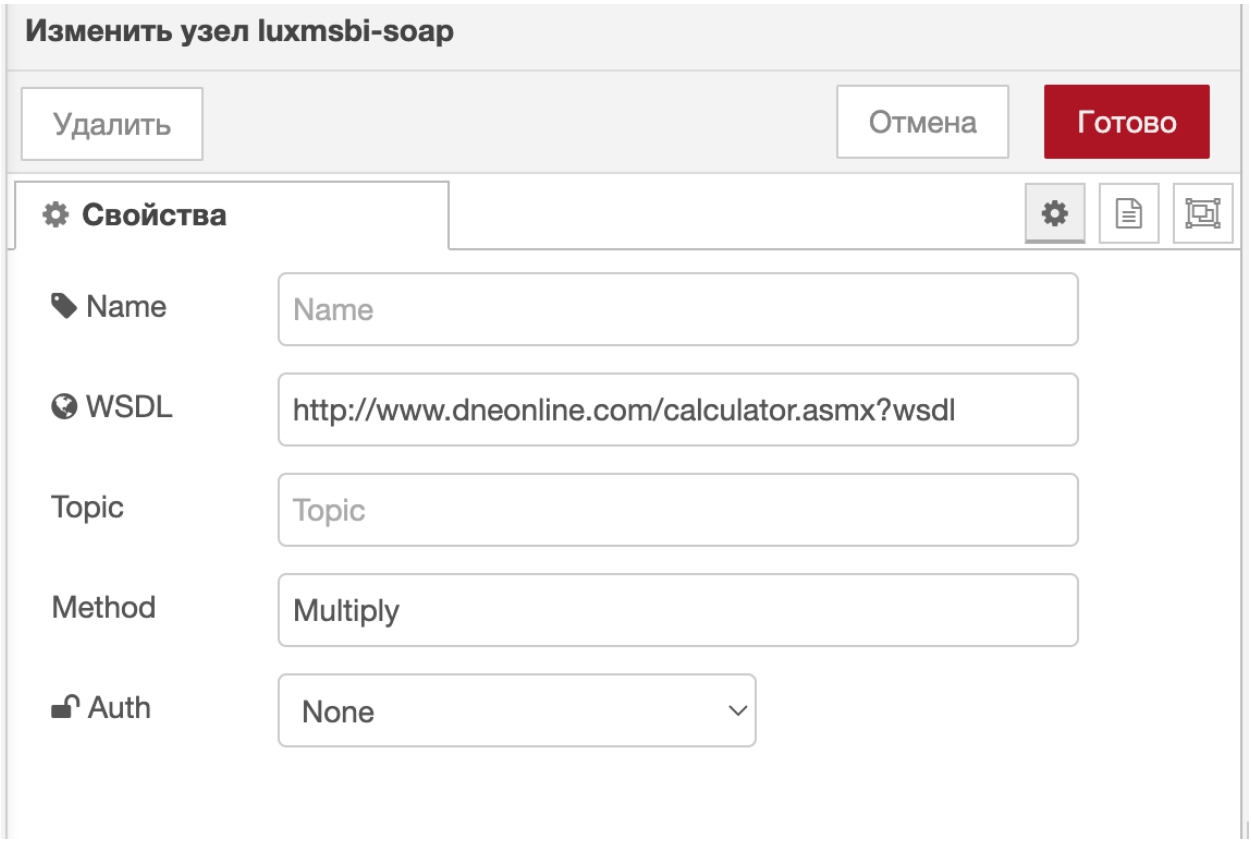


Рис. 4.26 **Свойства узла SOAP**

Результат выполненного потока, отображенного в окне отладки

```
2 {
3   "options": {},
4   "headers": {},
5   "payload": {
6     "MultiplyResult": 112
7   },
8   "_msgid": "7c10c7c011faa72c"
9 }
```

Код потока Launching the SOAP service

4.8 Ожидание исполнения узлов

Пример потока с использованием узла `luxmsbi-wait`, который ждет выполнения узлов `SQL источник` и собирает результат их выполнения в массив.

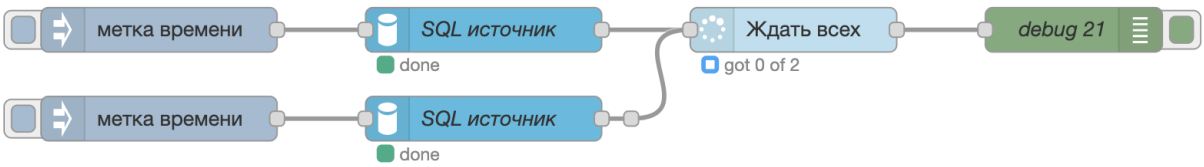


Рис. 4.27 **Поток отправки сообщение на вход узлу Ждать всех**

> Данный узел принимает уникальные сообщения, инициализированными узлами `меткой времени`.

Результат выполненного потока, отображенного в окне отладки

```
[
  {
    "id": "1d55ec772780a987",
    "payload": {
      "num": 2
    }
  },
  {
    "id": "39fb76d0f77058ff",
    "payload": {
      "num": 1
    }
  }]
```

Код потока Waiting for node execution

4.9 Использование библиотеки exceljs

Данная библиотека позволяет производить следующие операции:

1. Чтение/запись файлов формата `Excel`,
2. Работа с данными в ячейках,
3. Работа со стилем.

Ссылка на библиотеку GitHub

4.9.1 Основы: импорт, типы данных

Вызов библиотеки происходит следующим образом:

```
const Excel = global.get('EXCEL');
```

Основным объектом является т.н. `Workbook`, в котором производится дальнейшая работа.

```
const workbook = new Excel.Workbook();
```

После создания объекта и можно производить основные манипуляции с файлами в формате `Excel`. С помощью данной библиотеки можно не только считывать готовые `Excel` файлы, но и создавать свои с помощью различных методов. Рассмотрим случай, когда нам необходимо вытащить все данные из готового `Excel`.

Для загрузки экселя в `Workbook` необходимо выполнить команду:

```
await workbook.xlsx.readFile("PATH_TO_FILE");
```

`PATH_TO_FILE` - путь к файлу, хранящемуся на сервере Luxms Data Boring.

Теперь в нашем `Workbook` загружены данные и можно приступать к их считыванию.

Sheet1 Sheet2

org_id	dt	name	value	formula	some_column	colors		
9	5/1/21	ОМС	2000	222,22222222222	some text			
2	5/1/21	ДМС	8770	4385	some text			
6	5/1/21	ОМС	9587	1597,8333333333	some text			
8	5/1/21	Платные услуги	6791	848,875	some text			
7	5/1/21	ОМС	6540	934,285714285714	some text			50
5	5/1/21	ДМС	7237	1447,4	some text			
4	5/1/21	Платные услуги	2144	536	some text			
10	5/1/21	Личный шифр	3297	329,7	some text			
3	5/1/21	Платные услуги	9664	3221,3333333333	some text			
5	5/1/21	Бюджет	4370	874	some text			
7	5/1/21	Платные услуги	3526	503,714285714286	some text			
2	5/1/21	Бюджет	6986	3493	some text			
2	5/1/21	Личный шифр	9721	4860,5	**some text**			
			60					

Рис. 4.28 **Структура файла**

`Excel` файл представляет из себя:

1. 2 листа - Sheet, Sheet2
2. Различные типы колонок (числа, дата, формула, ячейки с измененными стилями написания и цвета)
3. Значения, выходящие за пределы таблицы.

4.9.1.1 Чтение всех данных из листов

Данные можно считывать разными способами. Все зависит от задачи, которая стоит перед вами.

Например, если необходимо просто считать данные с листа, то можно воспользоваться следующим кодом:

```
const e = global.get('EXCEL');

const wb = new e.Workbook();
await wb.xlsx.readFile("/tmp/exceljs_xlsx.xlsx");

let data = new Array();

wb.eachSheet(function (ws, sheetId) {

    let wsData = new Array();

    ws.eachRow({ includeEmpty: true }, function (row, rowNumber){

        let rowData = new Array();

        row.eachCell({ includeEmpty : true }, function (cell, colNumber) {

            let cellData = cell.value;
            rowData.push(cellData);
        });

        wsData.push(rowData);
    });

    data.push(wsData);
});

msg.payload = {};

msg.payload.data = data;

return msg;
```

Результат выполнения кода следующий:

отладка

все узлы all

23/12/2022, 21:22:38 node: debug 122

msg.payload : Object


```
▾object
▾data: array[2]
  ▾0: array[16]
    ▾[0 … 9]
      ▾0: array[7]
          0: "org_id"
          1: "dt"
          2: "name"
          3: "value"
          4: "formula"
          5: "some_column"
          6: "colors"
      ▾1: array[7]
          0: 9
          1: "2021-05-01T00:00:00.000Z"
          2: "OMC"
          3: 2000
        ▾4: object
            formula: "D2/A2"
            result: 222.22222222222223
          5: "some text"
          6: null
      ▸2: array[7]
      ▸3: array[7]
      ▸4: array[7]
      ▸5: array[9]
      ▸6: array[7]
      ▸7: array[7]
      ▸8: array[7]
      ▸9: array[7]
    ▸[10 … 15]
```


Рис. 4.29 **Результат выполения кода**

Таким образом, у нас получается структура, которая повторяет наполнение `Excel` файла. В дальнейшем можно производить различные манипуляции над этими данными.

Отметим некоторые моменты:

1. Параметр `{includeEmpty : true [false] }` отвечает за то, возвращать ли в результат пустые листы/строки/ячейки,

2. Парсится весь документ как вширь, так и вглубь. Поэтому если выбрать параметр `{includeEmpty : true }`, то конечный массив получится тем больше, чем дальше у вас какое-нибудь одиночное мисс-клик значение. Рекомендаций по выбору параметра нет, это зависит от чистоты входных данных.

3. Параметр `{includeEmpty : true }` вместо пустых значений вставляет значение `null`

4. Формулы понимаются как `object`, у которого есть свойства:

   1) `result` - результат выполнения формулы,
   2) `sharedFormula` - указатель ячейки, откуда бралась формула,
   3) `formula` - формула, по которой происходило вычисление,
   4) `ref` - ссылка на ячейки, на которые распространяется формула,
   5) `shareType` - тип шейринга.

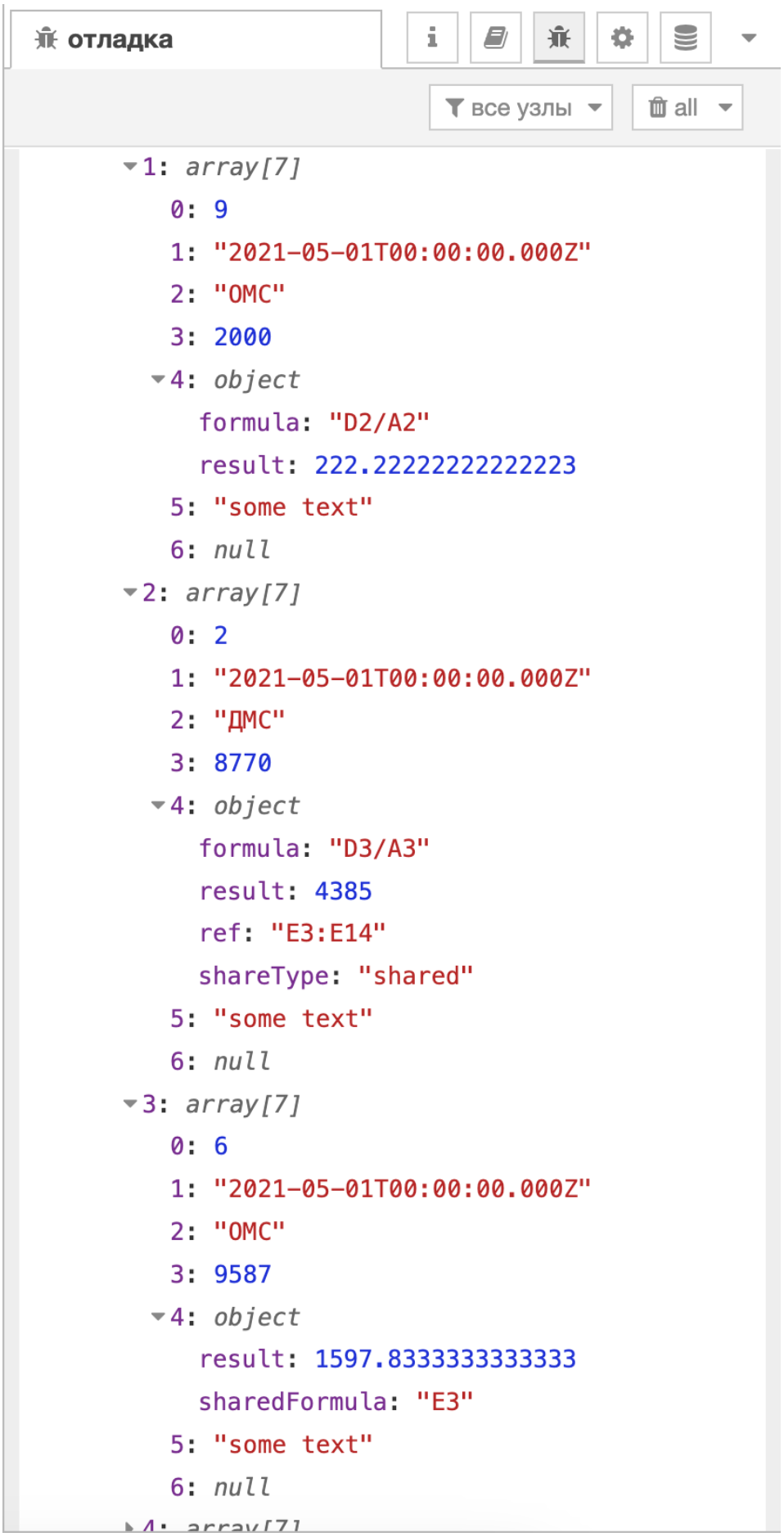


Рис. 4.30 **Свойства формул**

В данном примере мы видим только свойства объектов, где стоит формула. Добавление следующего кода позволит нам доставать именно значения формул:

```
let cellData = cell.value;
if (typeof cellData==='object' && colNumber===5) {
    cellData = cell.value.result;
};
rowData.push(cellData)
```

Но с данной конструкцией стоит быть внимательным, потому что бывают случаи, когда формула расшарена, но нет никакого результата выполнения (например, когда вызов происходит из пустой ячейки), следовательно не может произойти и вызова к свойству `result`. Данную ситуацию можно эффективно обходить с помощью конструкции `try...catch`.

Также стоит отметить, что дата (столбец `dt`) тоже имеет тип `object`, поэтому в данном случае лучше указать конкретный столбец, где находится формула, чтобы не получать `TypeError`.

Нумерация как для столбцов, так и для строк начинается с `1`.

4.9.1.2 Типы данных в ячейках

Помимо вызова к ячейке метода `value` можно обращаться методом `type`, который определяет какой тип значения стоит в конкретной ячейке.

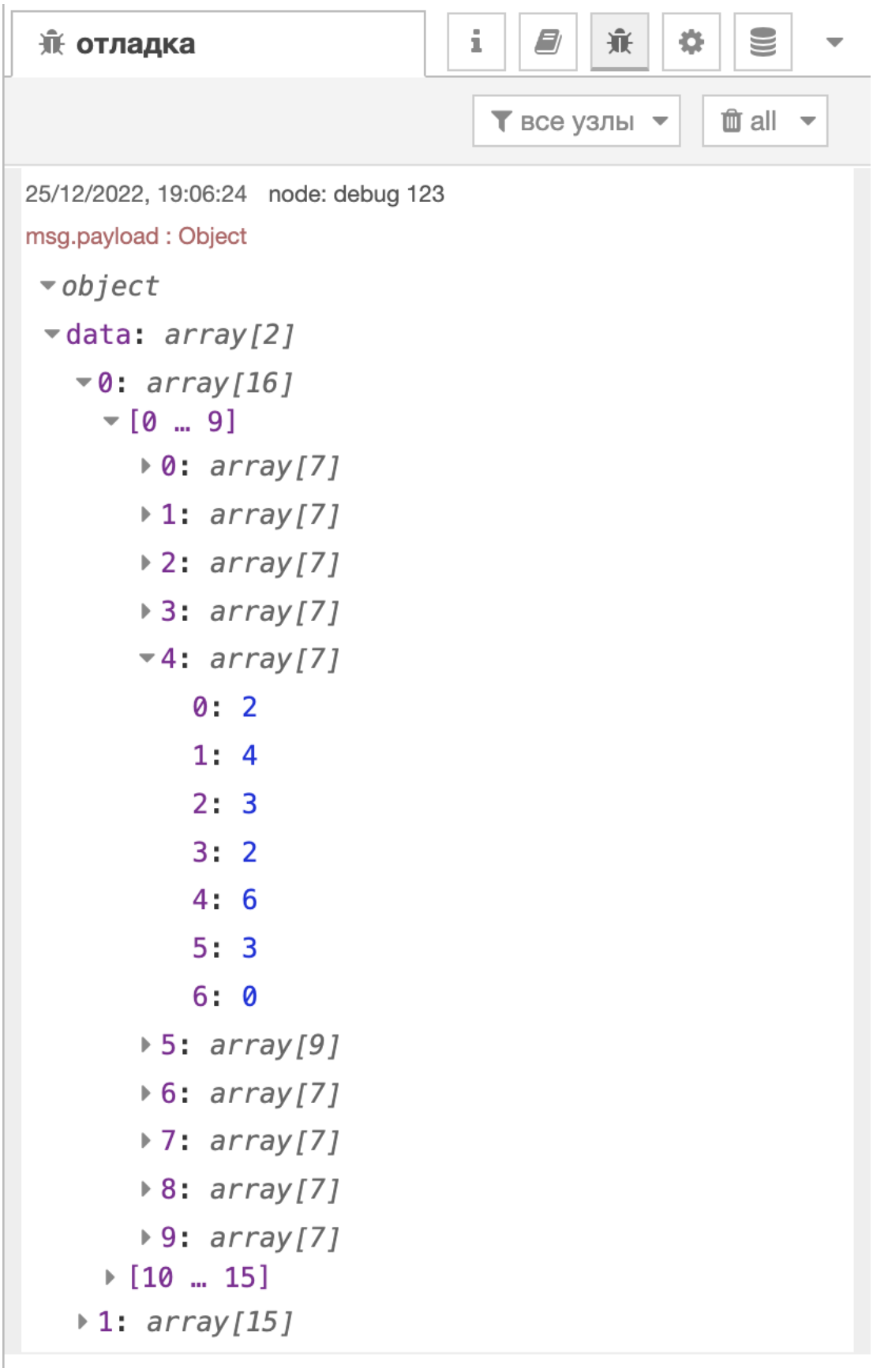


Рис. 4.31 **Типы данных**

Ниже представлен перевод численного значения

Null	0
Merge	1
Number	2
String	3
Date	5
Hyperlink	5
Formula	6
SharedString	7
RichText	8
Boolean	9
Error	10

4.9.1.3 Стиль ячеек

У ячеек, помимо значения и типа, также существует стиль, метод `style` помогает нам работать с ним.

Заменив в коде выше метод `value` на `style` получим следующее:

```
отладка
все узлы
all
25/12/2022, 19:10:28  node: debug 124
msg.payload : Object
▾object
▾data: array[2]
  ▾0: array[16]
    ▾[0 … 9]
      ▸0: array[7]
      ▾1: array[7]
        ▾0: object
          ▾font: object
            size: 11
            ▾color: object
              theme: 1
            name: "Calibri"
            family: 2
            scheme: "minor"
          ▾border: object
            ▾right: object
              style: "thin"
              ▾color: object
                indexed: 64
            ▾top: object
              style: "thin"
              ▾color: object
                indexed: 64
            ▾bottom: object
              style: "thin"
              ▾color: object
                indexed: 64
          ▾fill: object
            type: "pattern"
            pattern: "none"
        ▸1: object
```


Рис. 4.32 **Стиль ячеек**

С помощью этого метода можно смотреть такие данные, как:

1. `font` - информация о шрифте,
2. `border` - информация о границах,
3. `fill` - информация о цвете наполнения ячейки,
4. `alignment` - информация о расположении текста в ячейке (напр. тип выравнивания).

4.9.1.4 Вывод

С помощью библиотеки `exceljs` можно работать со всеми мета-данными, которые определяют конечный вид `Excel` файла, такими как значения в ячейках, их тип данных и стиль, в котором исполнена конкретная ячейка.

В зависимости от конкретной задачи это бывает очень полезно, например, когда дата документа подсвечена жёлтым цветом и мы легко сможем определить именно эту конкретную ячейку и потом каким-либо образом использовать эту информацию в дальнейшем.

В конечном итоге, результат парсинга будет определяться лишь набором эвристических правил, по которым будет осуществлен поиск и важно понимать какую именно информацию можно получать с помощью данной библиотеки.

4.9.2 Базовые методы работы

4.9.2.1 Автоматический поиск заголовка

Для примера будем использовать следующий `Excel` файл со следующей структурой:

L30 | fx | 6

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4	fixed acidi	volatile ac	citric acid	residual su	chlorides	free sulfur	total sulfu	density	pH	sulphates	alcohol	quality
5	7	0,27	0,36	20,7	0,045	45	170	1,001	3	0,45	8,8	6
6	6,3	0,3	0,34	1,6	0,049	14	132	0,994	3,3	0,49	9,5	6
7	8,1	0,28	0,4	6,9	0,05	30	97	0,9951	3,26	0,44	10,1	6
8	7,2	0,23	0,32	8,5	0,058	47	186	0,9956	3,19	0,4	9,9	6
9	7,2	0,23	0,32	8,5	0,058	47	186	0,9956	3,19	0,4	9,9	6
10	8,1	0,28	0,4	6,9	0,05	30	97	0,9951	3,26	0,44	10,1	6
11	6,2	0,32	0,16	7	0,045	30	136	0,9949	3,18	0,47	9,6	6
12	7	0,27	0,36	20,7	0,045	45	170	1,001	3	0,45	8,8	6
13	6,3	0,3	0,34	1,6	0,049	14	132	0,994	3,3	0,49	9,5	6
14	8,1	0,22	0,43	1,5	0,044	28	129	0,9938	3,22	0,45	11	6
15	8,1	0,27	0,41	1,45	0,033	11	63	0,9908	2,99	0,56	12	5
16	8,6	0,23	0,4	4,2	0,035	17	109	0,9947	3,14	0,53	9,7	5
17	7,9	0,18	0,37	1,2	0,04	16	75	0,992	3,18	0,63	10,8	5
18	6,6	0,16	0,4	1,5	0,044	48	143	0,9912	3,54	0,52	12,4	7
19	8,3	0,42	0,62	19,25	0,04	41	172	1,0002	2,98	0,67	9,7	5
20	6,6	0,17	0,38	1,5	0,032	28	112	0,9914	3,25	0,55	11,4	7
21	6,3	0,48	0,04	1,1	0,046	30	99	0,9928	3,24	0,36	9,6	6
22	6,2	0,66	0,48	1,2	0,029	29	75	0,9892	3,33	0,39	12,8	8

White Wine

Рис. 4.33 **Структура Excel wine_data**

Рассмотрим тривиальную задачу - необходимо загрузить `Excel` файл в БД с использованием `Luxms Data Boring`. Следующий поток обработки имеет вид:

Рис. 4.34 **Простой пайплайн загрузки Excel**

Для загрузки подобным образом помимо стандартных параметров необходимо указать:

1. Номер строки, на которой находятся заголовки,
2. Номер строки, с которой начинаются данные.

Перед вставкой нужно предварительно проанализировать `Excel` файл. Для динамической подстановки данных параметров можно воспользоваться библиотекой `exceljs`. Для этого используется следующий алгоритм - строка, содержащая заголовки, содержит все значения типа `string`.

Добавив в поток обработки узел `function`, в котором прописан алгоритм, имеем следующий вид:

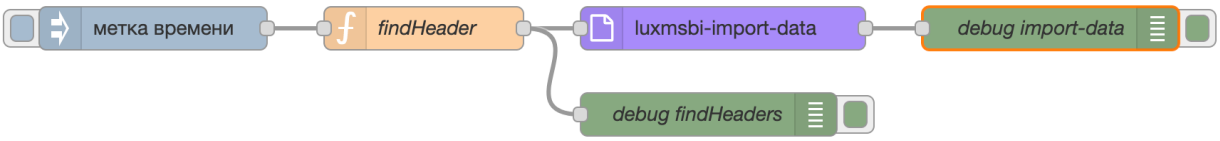


Рис. 4.35 **Автоматический поиск заголовка**

У узла `function` с наименованием `findHeader` вставлен следующий код:

```
const Excel = global.get('EXCEL');

const workbook = new Excel.Workbook();
await workbook.xlsx.readFile("/tmp/wine_data.xlsx");
const worksheet = workbook.getWorksheet("White Wine");

const isString = (currentValue) => typeof currentValue == "string";
let findHeader = true;
let headerName;
let headerRow;
let dataRow;

worksheet.eachRow({ includeEmpty: true }, function(row, rowNumber) {

    let readRow = new Array();

    row.eachCell({ includeEmpty: true }, function(cell, colNumber) {
```

```
20          readRow.push(cell.value);
21      });

23      if (findHeader) {
24          if (readRow.every(isString) && readRow.length !== 0) {
25              headerName = readRow;
26              headerRow = rowNumber;
27              dataRow = rowNumber + 1;
28              findHeader = false;
29          };
30      };
31  });

33  msg.payload = {};
34  msg.headerRow = headerRow;
35  msg.firstDataRow = dataRow;
36  msg.headerName = headerName;

38  return msg;
```

Алгоритм, реализованный в узле `function`:

1. Проходим по файлу `Excel` и записываем каждое значение в строку `readRow`.

2. Если установлена опция `findHeader`, входим в блок `if`:

    1) Если все значения в строке `readRow` типа `string` и строка не пуста:

        1) Записываем заголовки, находящиеся в строке readRow, в переменную `headerName`.

        2) Записываем номер текущей строки `rowNumber` в переменную `headerRow`.

        3) Записываем в переменную `dataRow` номер строки `rowNumber + 1`.

        4) Сбрасываем флаг `findHeader` так как заголовки уже найдены.

Результат работы алгоритма:

```
27/12/2022, 09:55:53   node: debug findHeaders
msg : Object
▾object
  _msgid: "3aa6f3196b22836d"
  ▸payload: object
  topic: ""
  headerRow: 4
  firstDataRow: 5
  ▾headerName: array[12]
    ▾[0 … 9]
        0: "fixed acidity"
        1: "volatile acidity"
        2: "citric acid"
        3: "residual sugar"
        4: "chlorides"
        5: "free sulfur dioxide"
        6: "total sulfur dioxide"
        7: "density"
        8: "pH"
        9: "sulphates"
    ▾[10 … 11]
        10: "alcohol"
        11: "quality"
```

Рис. 4.36 **Результат поиска заголовка**

По выводу видно, что алгоритм работает корректно, так как удалось найти заголовки и вывести их имена, а также загрузить `Excel` файл в БД.

4.9.2.2 Частичное чтение информации из файла

Не только чтение, но и запись в файлы с помощью библиотеки `exceljs` также возможна. Например, можно записывать части исходного `excel` файла в отдельный `csv` файл. Рассмотрим пример на основе файла `wine_data.xlsx`.

Для решения этой задачи понадобятся следующие методы:

rowCount	Общее количество строк в документе. Равняется номеру последней строки, в которой есть хотя бы одно значение.
actualRowCount	Общее количество непустых строк. То есть, если в середине документа строки будут пустыми, то они не будут учитываться.
columnCount	Общее количество столбцов в документе.
actualColumnCount	Общее количество столбцов в документе, в которых есть хотя бы одно значение.

Примечание: разница между total и actual - это количество пустых строк/столбцов, которые встречаются в документе. Рассмотрим этот момент на примере, запустив следующий код:

```
const Excel = global.get('EXCEL');

const workbook = new Excel.Workbook();

await workbook.xlsx.readFile('/tmp/wine_data.xlsx');
const worksheet = workbook.getWorksheet('White Wine');

msg.rowCount = worksheet.rowCount
msg.actualRowCount = worksheet.actualRowCount
msg.columnCount = worksheet.columnCount
msg.actualColumnCount = worksheet.actualColumnCount

return msg;
```

После запуска потока обработки объект `msg` имеет следующее содержание:

```
{
  "_msgid": "b4500b5b7ee142b7",
  "payload": 1672122420477,
  "topic": "",
  "rowCount": 4902,
  "actualRowCount": 4899,
  "columnCount": 12,
  "actualColumnCount": 12
}
```

Выяснено, что методы работают корректно, подсчитывая строки и столбцы. Первые 3 строки были исключены из подсчета `actualRowCount`.

Пусть требуется записать последние 5 столбцов первых 200 строк `excel` файла в отдельный `csv` файл. Можно воспользоваться следующим кодом:

```
const Excel = global.get('EXCEL');
msg.filename = '/tmp/wine_data_csv.csv';

const workbook = new Excel.Workbook();
```

```
6  await workbook.xlsx.readFile('/tmp/wine_data.xlsx');
7  const worksheet = workbook.getWorksheet('White Wine');

9  const newWorkbook = new Excel.Workbook();
10 const newWorksheet = newWorkbook.addWorksheet('wineSheet');

12 let startColumn = worksheet.actualColumnCount - 4;
13 let endColumn = worksheet.actualColumnCount;
14 let startRow = 5;
15 let endRow = 205;


18 for (let r=startRow, k=1; r <= endRow; r++, k++) {
19     var row = worksheet.getRow(r);
20     var newRow = await newWorksheet.getRow(k);

22     for (let col=startColumn; col<=endColumn; col++) {
23         newRow.getCell(col-startColumn+1).value = row.getCell(col).value;
24     };
25 };

27 await newWorkbook.csv.writeFile(msg.filename, { sheetName: 'wineSheet' });

29 return msg;
```

После выполнения этого кода и использования узла `read file` можно увидеть следующий вывод в окне отладки:

```
27/12/2022, 10:32:41   node: debug 100
msg : Object
▾object
  _msgid: "fedf170b83864cb8"
  ▾payload: string
    1.001,3,0.45,8.8,6
    0.994,3.3,0.49,9.5,6
    0.9951,3.26,0.44,10.1,6
    0.9956,3.19,0.4,9.9,6
    0.9956,3.19,0.4,9.9,6
    0.9951,3.26,0.44,10.1,6
    0.9949,3.18,0.47,9.6,6
    1.001,3,0.45,8.8,6
    0.994,3.3,0.49,9.5,6
    0.9938,3.22,0.45,11,6
    0.9908,2.99,0.56,12,5
    0.9947,3.14,0.53,9.7,5
    0.992,3.18,0.63,10.8,5
    0.9912,3.54,0.52,12.4,7
    1.0002,2.98,0.67,9.7,5
    0.9914,3.25,0.55,11.4,7
    0.9928,3.24,0.36,9.6,6
    0.9892,3.33,0.39,12.8,8
    0.9917,3.12,0.53,11.3,6
    0.9955,3.22,0.5,9.5,5
```

Рис. 4.37 **Сохранение файла в CSV**

Работа алгоритма заключается в проходе по `excel` файлу с помощью двух циклов `for`. Внешний цикл отвечает за проход по строкам, внутренний отвечает за проход по столбцам.

Для записи файла следует инициализировать новый `Workbook` и записать в него нужную часть исходного `excel` файла с помощью конструкции `.getCell(col).value`. Затем этот новый `excel` файл можно преобразовать в `CSV` с помощью метода `.csv.writeFile(msg.filename, { sheetName: 'wineSheet' });`, где:

1. `msg.filename` - путь, куда мы сохраняем `CSV` файл;

2. `sheetName` - название листа, куда сохраняли необходимые значения.

4.9.2.3 Сохранение таблиц со сложной шапкой в БД из Excel файла

В случае, когда необходимо парсить файл и записать его сразу в БД, не всегда стандартные средства `Luxms Data Boring` могут помочь, например, при наличии сложной шапки таблицы. Проблема заключается в том, что невозможно точно указать, с какой строки начинаются заголовки и начинаются данные.

Чтобы решить проблему с парсингом файла со сложной шапкой и записью в БД, можно использовать средства `exceljs`. В качестве примера можно взять файл со следующей структурой:

D43 | fx | 2,5

	A	B	C	D	E	F	G	H
1			float				integer	
2	fixed_acid	volatile_a	citric_acid	residual_s	chlorides	free_sulfu	total_sulfi	quality
3	7	0,27	0,36	20,7	0,045	45	170	6
4	6,3	0,3	0,34	1,6	0,049	14	132	6
5	8,1	0,28	0,4	6,9	0,05	30	97	6
6	7,2	0,23	0,32	8,5	0,058	47	186	6
7	7,2	0,23	0,32	8,5	0,058	47	186	6
8	8,1	0,28	0,4	6,9	0,05	30	97	6
9	6,2	0,32	0,16	7	0,045	30	136	6
10	7	0,27	0,36	20,7	0,045	45	170	6
11	6,3	0,3	0,34	1,6	0,049	14	132	6
12	8,1	0,22	0,43	1,5	0,044	28	129	6
13	8,1	0,27	0,41	1,45	0,033	11	63	5
14	8,6	0,23	0,4	4,2	0,035	17	109	5
15	7,9	0,18	0,37	1,2	0,04	16	75	5
16	6,6	0,16	0,4	1,5	0,044	48	143	7
17	8,3	0,42	0,62	19,25	0,04	41	172	5
18	6,6	0,17	0,38	1,5	0,032	28	112	7
19	6,3	0,48	0,04	1,1	0,046	30	99	6
20	6,2	0,66	0,48	1,2	0,029	29	75	8
21	7,4	0,34	0,42	1,1	0,033	17	171	6

White Wine

Рис. 4.38 **Структура Excel со сложной шапкой**

Появились новые заголовки, которые указывают на то, является ли столбец целочисленным или дробным числом. Требуется сохранить эту информацию.

Для выполнения данной задачи используется поток следующего вида:

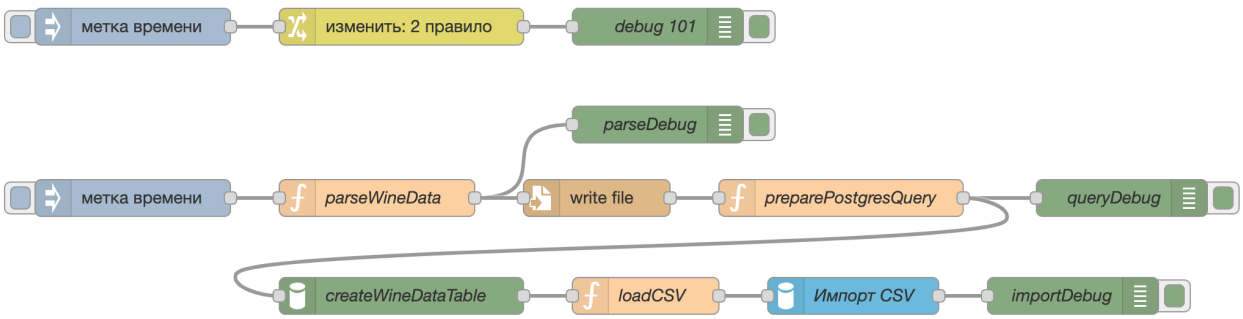


Рис. 4.39 **Поток для парсинга сложной шапки**

Для начала нужно задать имя таблицы и путь сохранения `csv` файла в узле `change`.

В узле `function` с наименованием `parseWineData` содержится следующий код:

```
const Excel = global.get('EXCEL');

const workbook = new Excel.Workbook();

await workbook.xlsx.readFile('/tmp/wine_data_head.xlsx');
const worksheet = workbook.getWorksheet('White Wine');

const isString = (currentValue) => typeof currentValue == "string";
var headerName = new Array();
var headerRow;
var wb_csv = new Array();

worksheet.eachRow({ includeEmpty: true }, function (row, rowNumber) {

    let readRow = new Array();

    row.eachCell({ includeEmpty: true }, function (cell, colNumber) {
        let valueCell = cell.value;
        readRow.push(valueCell);
    });

    if (readRow.every(isString)) {
        headerName.push(readRow);
        headerRow = rowNumber;
    };

    wb_csv.push(readRow);
});

// Создание csv
let wbRaw = new Array();
wb_csv.slice(headerRow).forEach(function (infoArray, index) {

    var line = infoArray.map(function (elem, i) {
        if (String(elem) === 'null') {
```

```
36                 return ''
37             } else {
38                 return '\"' + String(elem) + '\"'
39             };
40         }).join(",");
41         wbRaw.push(line);
42     });
43     let wbPrepare = wbRaw.join("\n");

45     // Работа с заголовками
46     const regex = /[^a-z]/gi;
47     let header = headerName[0]
48     for (let i = 1; i < headerName.length; i++) {

50         let headerPast = headerName[i - 1];
51         let headerNow = headerName[i];

53         for (let j = 0; j < headerPast.length; j++) {
54             if (headerPast[j] !== headerNow[j]) { header[j] = header[j] +  '_' + ↩
  headerNow[j]; };
55         };
56     };

58     let clearHeader = header.map(element => element.replace(regex, ' ↩
  ').trim().replace(/\s+/g, '_').toLowerCase().substr(0, 60));

60     msg.payload = {};

62     msg.a = clearHeader;
63     msg.payload = wbPrepare;
64     msg.filename = flow.get('path');

66     return msg;
```

Данный код можно, условно, разделить на три части:

1. Формирование массива строк `wb_csv`, который затем будет превращен в `csv` файл.

   Для каждой ячейки пробегается, затем складывает в строку, и строку складывает в массив. Если все значения в строке текстовые, номер строки запоминается для последующего обрезания массива.

2. Непосредственно создание `csv`.

   Созданный ранее массив конкатенируется. Удобство данного способа в том, что мы сами выбираем какое значение будеть иметь статус `NULL`.

3. Работа с заголовками.

   Алгоритм выполняет простую операцию с заголовками - если следующий заголовок равен предыдущему, то он пропускает. В противном случае конкатенирует строки между собой, преобразуя их в формат, который будет использоваться в запросе для создания БД в `PostgresSQL`.

Следует отметить, что значения в объединенной колонке читаются для каждой ячейки отдельно, что позволяет не упускать значения в конкретной ячейке.

Заголовки сохраняются в объекте `msg`, чтобы в дальнейшем построить по ним таблицу в БД.

Вывод этого узла будет выглядеть следующим образом:

Рис. 4.40 **Результат работы узла `function` с наименованием `parseWineData`**

Узел `write file` возьмёт из потока `msg.payload` и `msg.filename` и создаст необходимый `csv` файл.

В узле `preparePostgresQuery` содержится следующий код:

```
msg.payload = {};

msg.a = (msg.a.join('" text NULL, \r\n"') + '" text NULL');
msg.payload.query = 'CREATE TABLE IF NOT EXISTS ' + flow.get('table') + '\n' + ↩
  '(' + '\n' + '"' + msg.a + '\n' + ')' + '\;' + '\n' + 'TRUNCATE TABLE ' + ↩
  flow.get('table') + '\;';

return msg;
```

Данный код просто строит `SQL` запрос в БД, который построит таблицу. Вывод этого узла будет таким:

Рис. 4.41 **Результат работы узла `function` с наименованием `preparePostgresQuery`**

В узле `createWineDataTable` происходит создание таблицы из запроса, который лежит в `msg.payload.query`.

Узел `loadCSV` содержит следующий код:

```
msg.payload = {};

msg.payload.sink = 'file://' + flow.get('path');
msg.payload.table = flow.get('table');

return msg;
```

Таким образом, параметры задаются для следующего узла `Импорт CSV`. Это удобно, когда заранее известно, в какую таблицу и откуда нужно что-то складывать.

После загрузки данные успешно были загружены в БД.

4.9.2.4 Вывод

Библиотека `exceljs` может покрыть большинство задач связанных с чтением и записью файлов. Она также мощна и может быть интегрирована с хранилищем `S3`, что позволяет отправлять данные туда, где они реально нужны, минуя различные этапы. Возможности библиотеки не ограничиваются только работой с `Excel` файлами.

4.10 Чтение файлов

С помощью узла `read file` можно читать файлы, например можно прочитать файл в формате `.csv`.

В данном примере рассмотрим вариант чтения файла с несколькими строками, каждая из которых будет содержать информацию о определённом файле и его кодировку, для корректного импорта данных в базу.

Так же в поток добавим узел `csv` для разбиения строк под свой отдельный объект сообщения.

Рис. 4.42 **Поток с использованием `read file` и `csv`**

Сам узел имеет следующие параметры:

- Имя файла: абсолютный путь к файлу;
- Вывод:

1. одна utf-8 строка;

2. сообщение для каждой строчки;
3. один объект буфера;
4. поток буфера.

- Выбор кодировки;
- Имя узла.

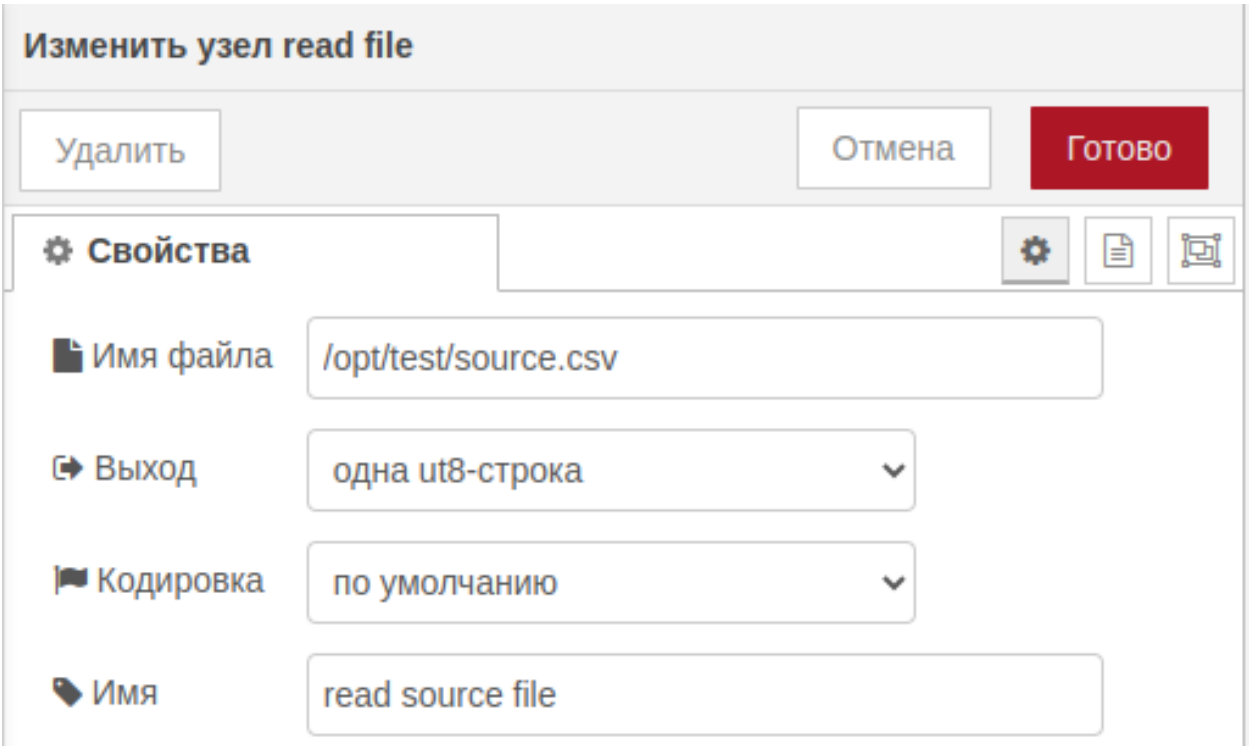


Рис. 4.43 **Узел `read file`**

После прочтения простейшего файла следующего вида:

```
buffer_table,file_name,encoding
custom.test_1,test_1.csv,utf-8
custom.test_2,test_2.csv,windows-1251
```

Получим следующий ответ:

12/14/2022, 3:09:42 PM node: 67b7ec31b6068e61
msg : Object
object
_msgid: "e77b4d4e2beca00b"
payload: string
buffer_table,file_name,encoding
custom.test_1,test_1.csv,utf-8
custom.test_2,test_2.csv,windows-1251
topic: ""
filename: "/opt/test/source.csv"

Рис. 4.44 **Результат исполнения `read file`**

Разобьем файл на отдельные объекты сообщения с помощью узла `csv`, где пропишем параметры нашего csv файла:

Изменить узел csv

Удалить | Отмена | Готово

Свойства

Столбцы: buffer_table,file_name,encoding

Разделитель: запятая

Имя: convert csv to json

Опции CSV -> Объект

Вход: Пропускать первые 0 строк(и)

- [x] первый ряд содержит имена столбцов
- [ ] разбирать числовые значения
- [ ] включать пустые строковые значения
- [ ] включать null-значения

Выход: сообщение для каждой строки

Опции Объект -> CSV

Выход: никогда не отправлять заголовки ст

Новая строка: Linux (\n)

Рис. 4.45 **Параметры узла `csv`**

Так же для удобства можно переопределить объект сообщения на другой с помощью узла `change`:

Изменить узел change

Удалить Отмена Готово

Свойства

Имя set table props

Правила

Установить msg. sql.buffer_table
to the value msg. payload.buffer_table
Deep copy value

Установить msg. sql.file_name
to the value msg. payload.file_name
Deep copy value

Установить msg. sql.encoding
to the value msg. payload.encoding
Deep copy value

Рис. 4.46 **Переименование объектов сообщения**

В результате всех произведённых действий получаем следующий ответ:

12/14/2022, 11:09:13 AM node: 8a7e1771e65f1841
msg : Object
▾object
_msgid: "74d4fbec5c4863cb"
▸payload: object
topic: ""
filename: "/opt/test/source.csv"
columns: "buffer_table,file_name,encoding"
▾parts: object
id: "74d4fbec5c4863cb"
index: 0
count: 2
▾sql: object
buffer_table: "custom.test_1"
file_name: "test_1.csv"
encoding: "utf-8"

12/14/2022, 11:09:13 AM node: 8a7e1771e65f1841
msg : Object
▾object
_msgid: "74d4fbec5c4863cb"
▸payload: object
topic: ""
filename: "/opt/test/source.csv"
columns: "buffer_table,file_name,encoding"
▾parts: object
id: "74d4fbec5c4863cb"
index: 1
count: 2
▾sql: object
buffer_table: "custom.test_2"
file_name: "test_2.csv"
encoding: "windows-1251"

Рис. 4.47 **Результат исполнения потока с `read file` и `csv`**

В результате разделения на несколько сообщений, добавляется специальный объект `parts`, который служит для понимания системой из скольких частей состояло исходное сообщение (на следующих этапах может понадобится для узлов по типу `join`)

Код потока Reading files

4.11 Экспорт запроса данных в csv

Пример потока, который сохраняет результат запроса в файл формата `.csv`

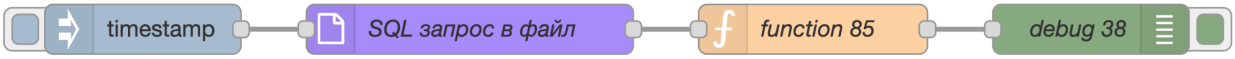


Рис. 4.48 **Поток Экспорта запроса данных в csv**

Код запроса, который прописан в поле узла `SQL запрос в файл`

```
SELECT t.day::date as dt, random() * 100 + 44 AS plan,  random() * 100 AS fakt
FROM   generate_series(timestamp '2022-05-01'
                     , timestamp '2022-10-30'
                     , interval  '1 day') AS t(day)
order by t.day::date
```

Код потока Exporting a data requrest to csv

Edit data transfer node

Delete | Cancel | Done

Properties

Name: SQL запрос в файл

Retry count: 1

Retry delay (ms): 5000

Source

Source type: SQL | XLS | CSV | DBF | QVD | Avro | Parquet

Connection ID: luxmsbi

```
1  SELECT t.day::date as dt, random() * 100 + 44 AS plan,  random() * 100
2  FROM   generate_series(timestamp '2022-05-01'
3                        , timestamp '2022-10-30'
4                        , interval  '1 day') AS t(day)
5  order by t.day::date
```

Destination

Destination type: SQL | CSV

Target Folder Path:

CSV with headers: ☑

CSV delimiter: ,

Zipped: ☐

Рис. 4.49 **Свойства узла, установленные для выгрузки в NATS-бакет**

Выбранные свойства для выгрузки результата запроса из базы `PostgreSQL` в формате `CSV` в распределённое документное хранилище работающее по протоколу `NATS`.

Работа потока приводит к следующему результату.

```
24/10/2023, 12:22:36   node: debug 38
msg.payload : Object
▾object
  sink: "dds:///DataSourceService/yvz8nzqyewh.csv"
  count: 183
```

Рис. 4.50 **Результат работы потока**

4.12 Исполнение unix-команд

С помощью узла exec можно запускать любые sh скрипты. При этом, в комбинации с узлом function можно создать свой sh скрипт с запуском любых bash запросов.

В данном примере используем function для запуска скрипта python.

> Для использования скриптов python, сам python должен быть развёрнут на сервере со своим витруальным окружением и со всеми установленными необходимыми библиотеками.

Рис. 4.51 **Поток с использованием exec**

4.12.1 function (file)

В данном узле мы прописываем запросы, для запуска нашего скрипта, как делали бы это в терминале bash. 1. Переходим в папку со скриптом 2. Активируем виртуальное окружение 3. Запускаем скрипт python

Код функции:

```
#!/bin/bash

cd /opt/test
source env/bin/activate
python some_script.py
```

4.12.2 write file (save to file)

Записываем этот запрос в sh файл. В `Имя файла` прописываем абсолютный путь для файла, который хотим создать.

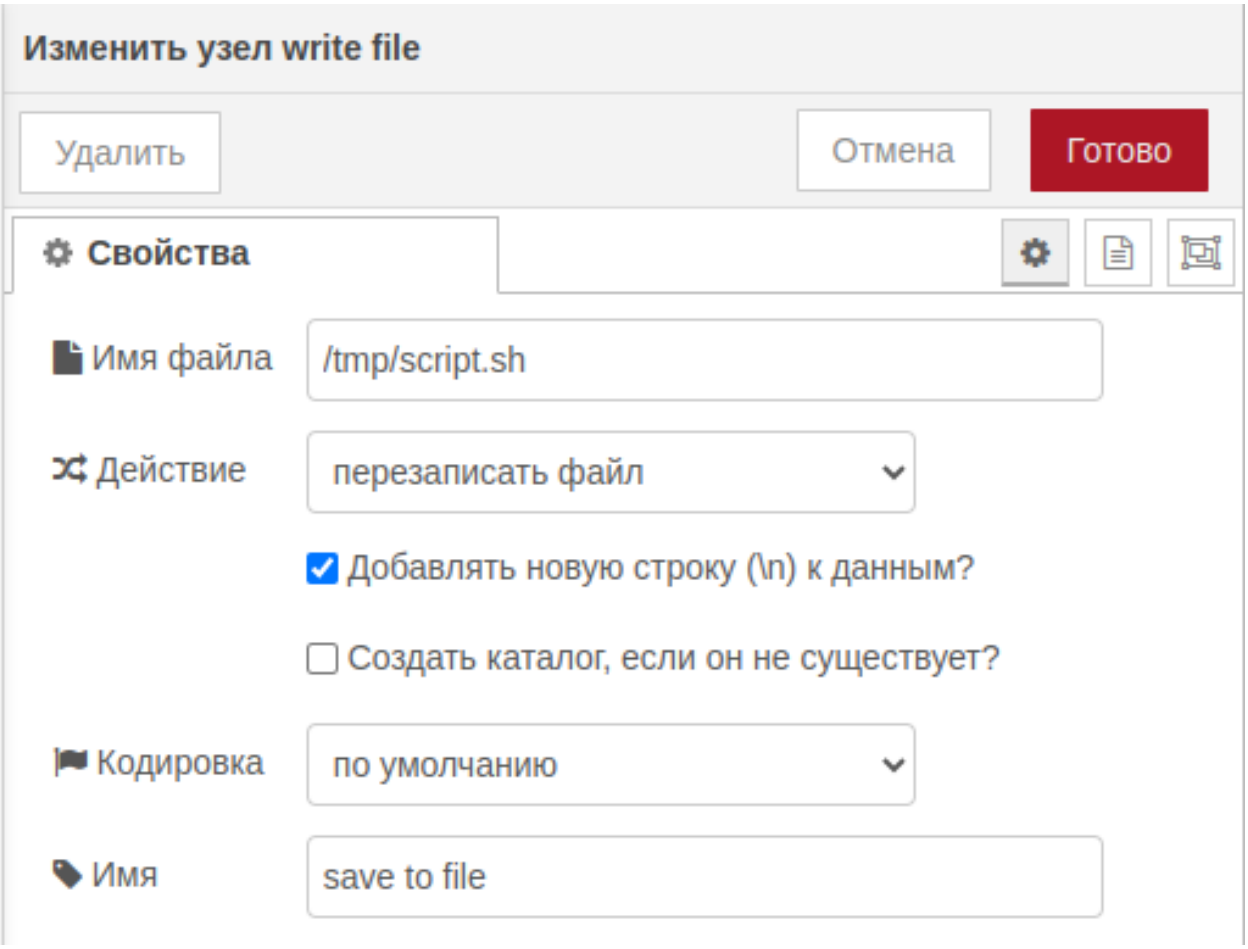


Рис. 4.52 **Запись запроса в sh**

4.12.3 exec (make script file executable)

Меняем права на созданный файл sh (добавляем возможность исполняемости).

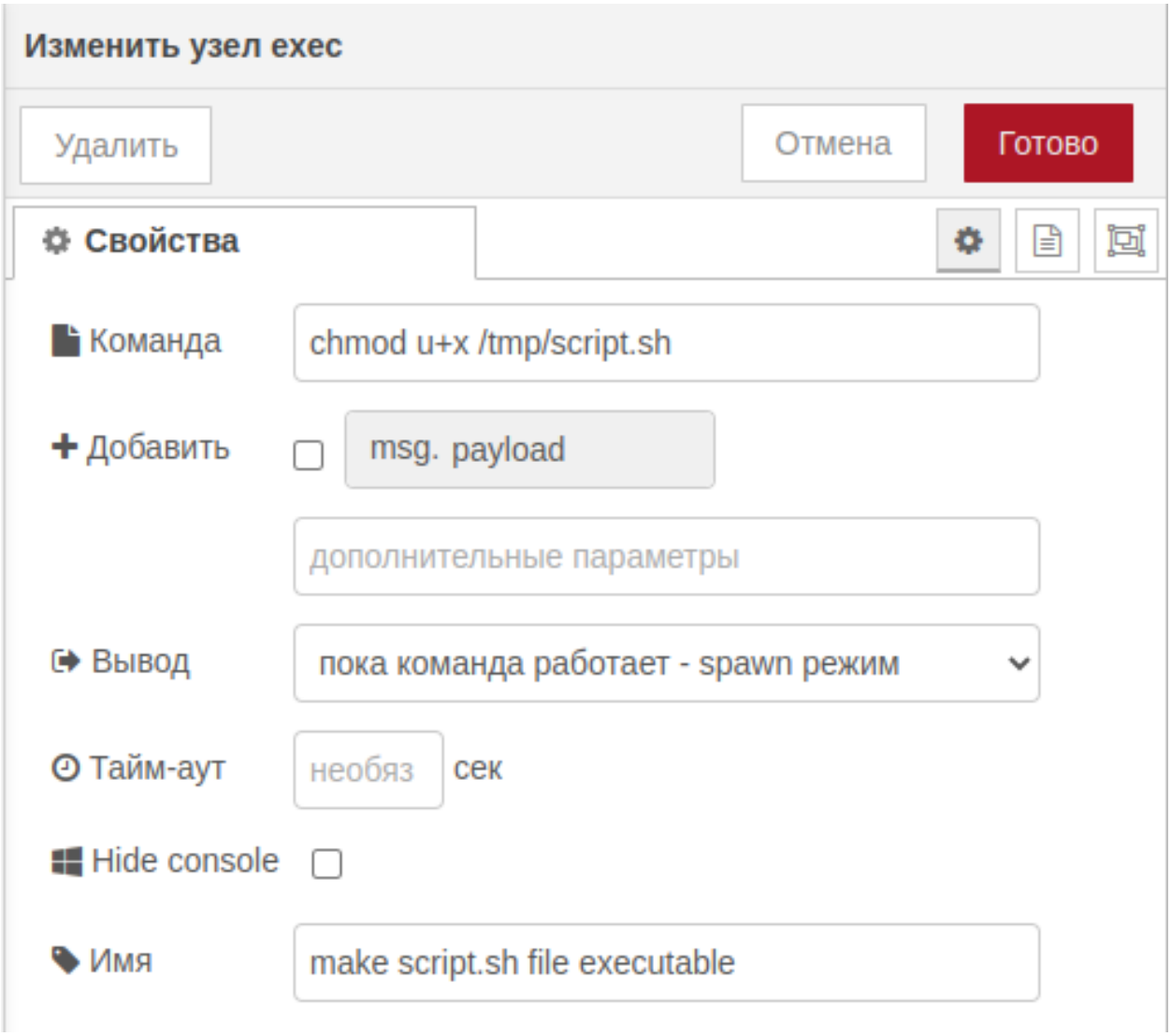


Рис. 4.53 **Делаем файл исполняемым**

4.12.4 exec (execute script)

Выполняем sh.

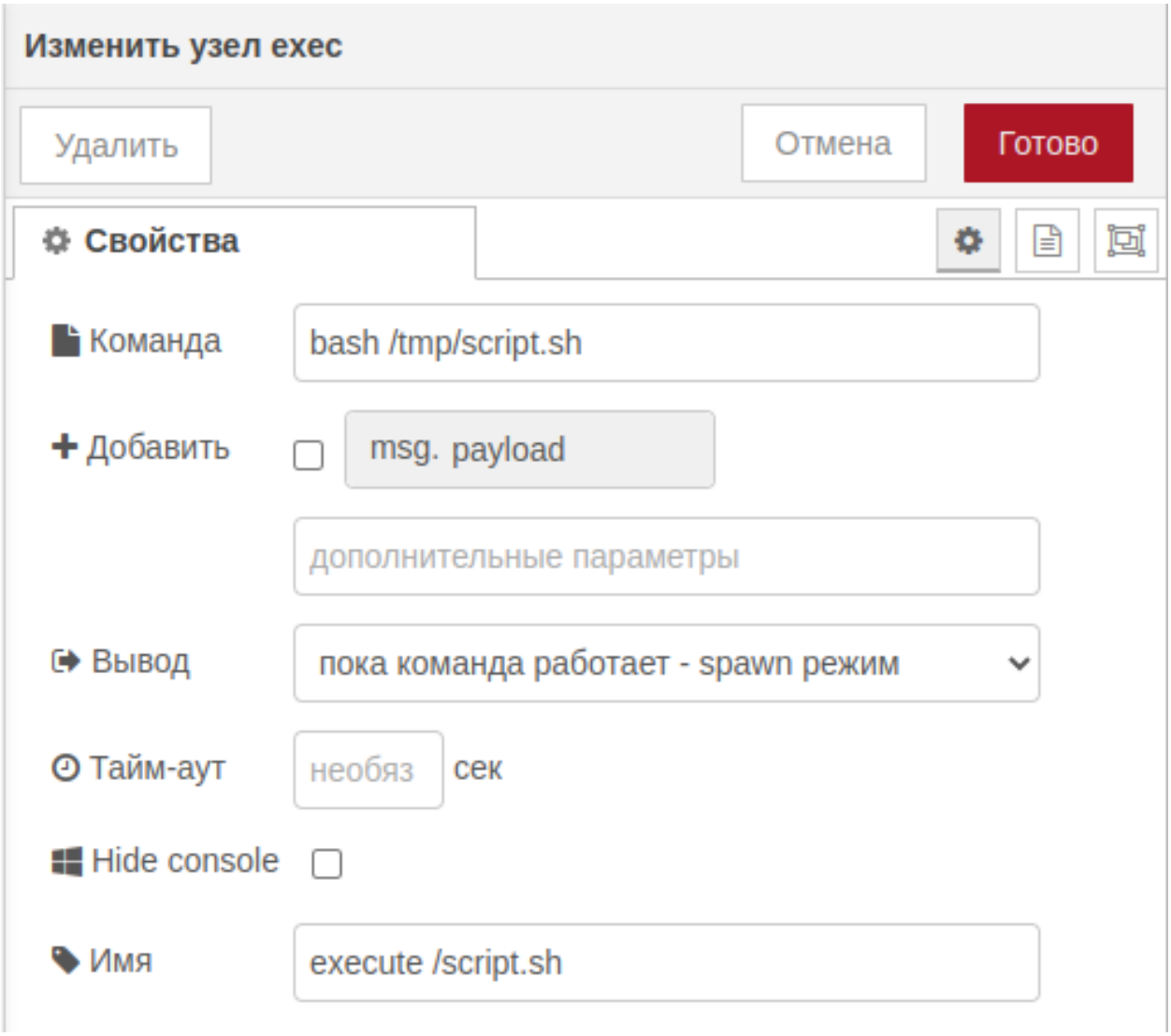


Рис. 4.54 **Исполняем файл**

4.12.5 Вывод результатов

Exec поддерживает вывод различных типов возвратов функций:

1. Стандартный вывод - stdout;
2. Стандартный вывод ошибок - stderr;
3. Код возврата - вывод 1/0.

Если скрипт python настроен под вывод таких типов, их можно будет получать через разные выводы exec. На основании этих выводов можно будет управлять продолжением исполнения потока.

Код потока Execution of unix commands

4.13 Циклические потоки

Пример потока, который позволяет создавать циклы на основе узла `loop` из палитры Luxms Data Boring группы `функция`.

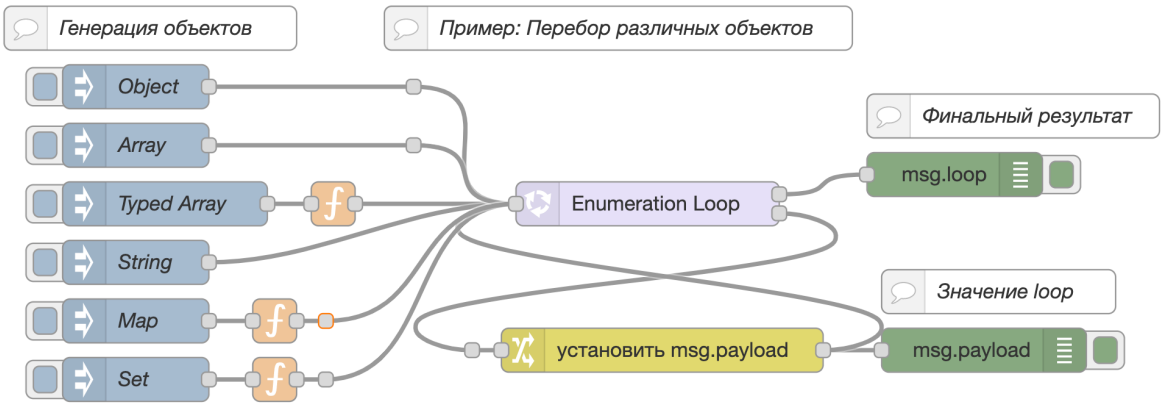


Рис. 4.55 **Поток обработки входящего объекта по циклу**

В узле `метка времени` в свойстве `msg.payload` устанавливается значение `{"first":↩ "Hello World","second":8,"third":true}`, которое подается на вход узлу `loop`.

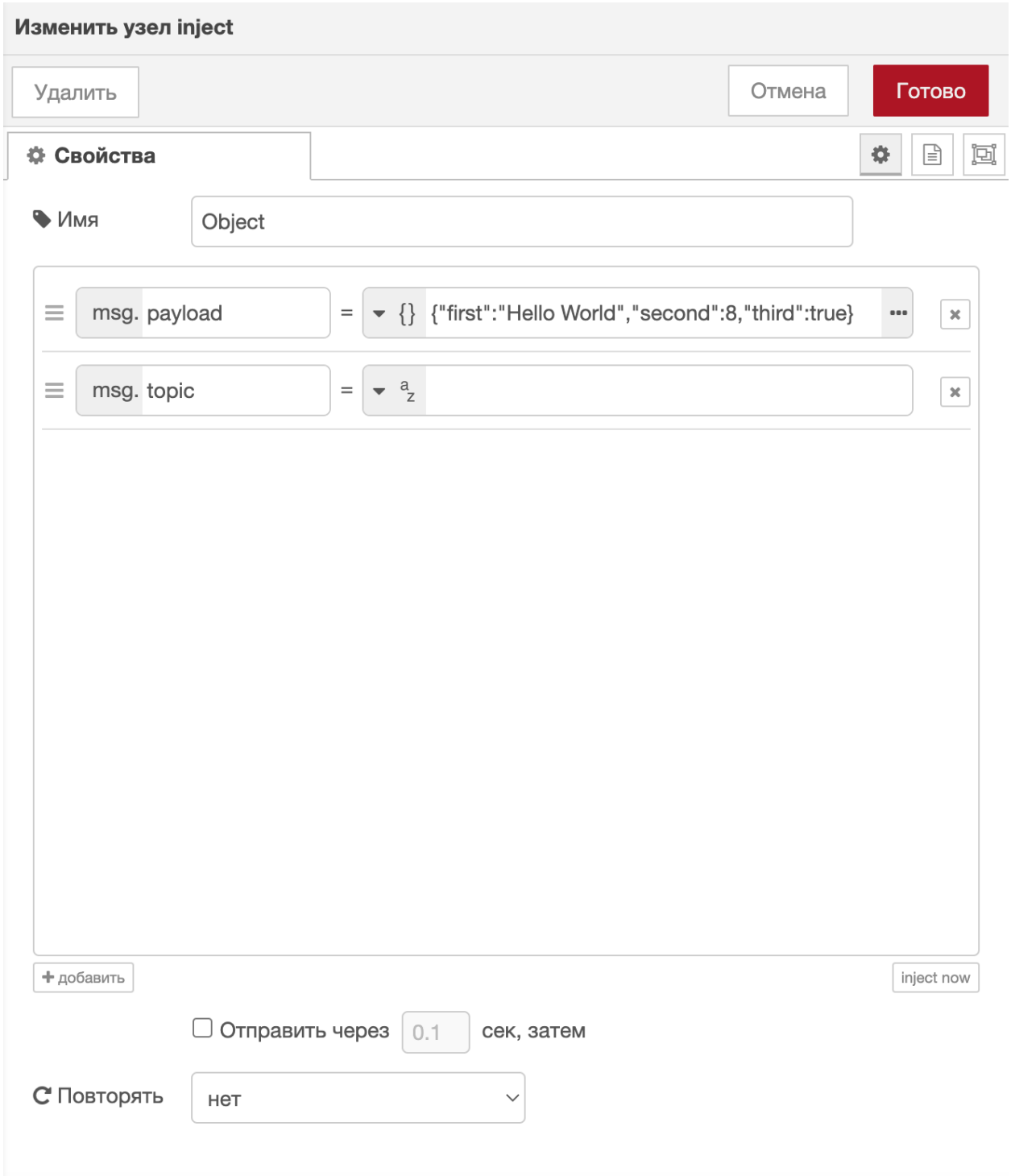


Рис. 4.56 **Свойства объекта, заданного в узле метка времени**

В свойствах узла `loop` устанавливаются параметры обработки объекта, который пришел на вход. В данном случае свойство было определено в msg.payload.

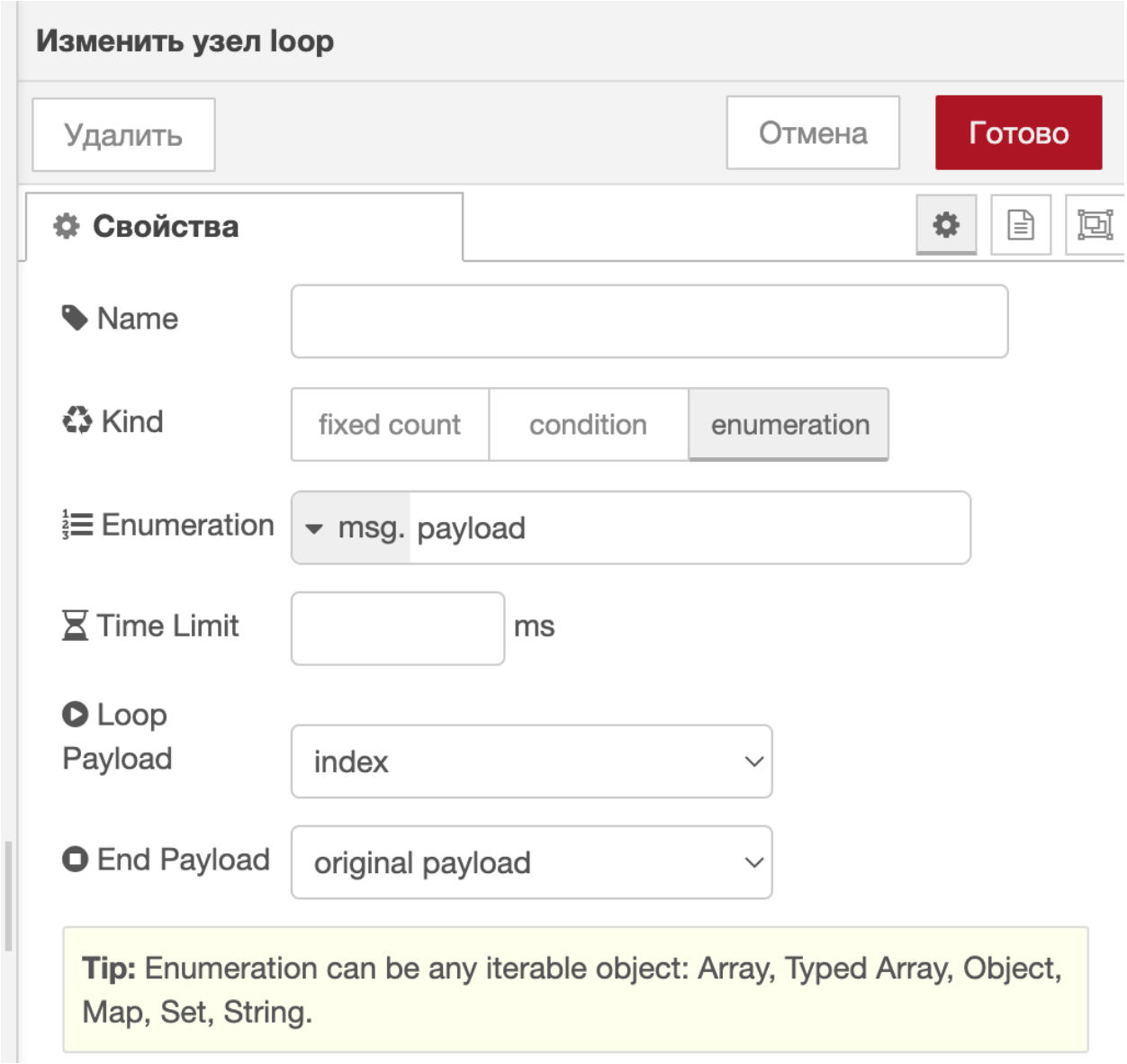


Рис. 4.57 **Свойства узла loop**

Результат выполненного потока, отображённого в окне отладки

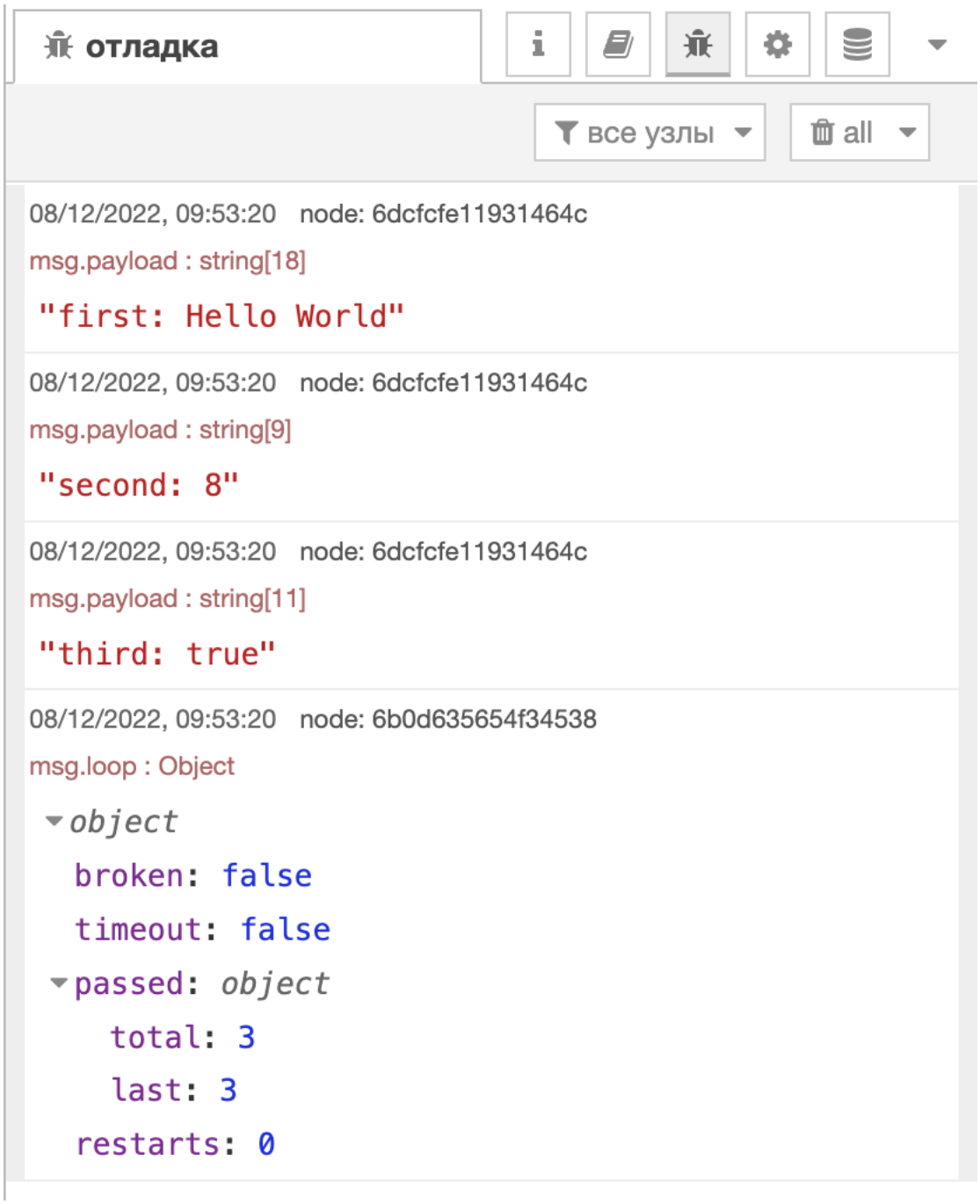


Рис. 4.58 **Результат работы узла `loop`**

Код потока Cyclic flows

4.14 Динамическая загрузка Excel файлов

Пример потока, который динамически инициализирует загрузку `.xlsx` файлов, появившихся в папке `/tmp`, используя узел `watch`. Библиотека `exceljs` позволяет опеределить два параметра: `Номер строки заголовка` и `Номер первой строки с данными`, которые подаются на вход узлу `Перенос данных`.

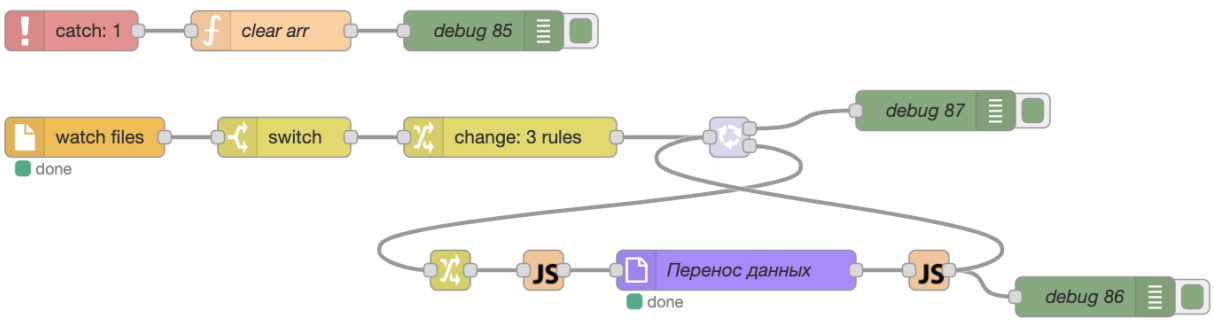


Рис. 4.59 **Поток с динамической загрузкой**

Узел `Перенос данных` принимает на вход параметры, которые были определены ранее в узле `change`. Поля в данном примере не заполняются.

Edit data transfer node

Delete | Cancel | Done

Properties

Name: Перенос данных

Retry count: 1

Retry delay (ms): 5000

Source

Source type: SQL | XLS | CSV | DBF | QVD | Avro | Parquet

Data source file path: Data source file path

Sheets to import: Sheets to import

Header rows: Header rows

First data rows: First data rows

Stream processing: ☐

Drop file on complete: ☐

Destination

Destination type: SQL | CSV

Connection ID: Connection ID

Schema name: Schema name

Table name: Table name

Mode: Truncate

Convert types: ☐

Chunk size: 1000

Рис. 4.60 **Свойства узла ‘Импорт файлов’**

Входные параметры, установленные в узле `change`

Edit change node

Delete Cancel Done

Properties

Name

Name

Rules

Set	msg. payload	
to the value	$type(msg.loop.key) != "undefined" ? msg.loop.value != '' ? msg.loop.key & ": " & msg.loop.value :	
Set	msg. destinationSqlConnIdent	
to the value	luxmsbi	
Set	msg. destinationSqlSchemaName	
to the value	custom	
Set	msg. destinationSqlTableName	
to the value	$replace($replace(msg.loop.value, '/tmp/', ''), '.xlsx', '')	
Set	msg. sourceXlsSheetNumbers	
to the value	1	
Set	msg. sourceXlsFilePath	
to the value	msg.loop.value	

Рис. 4.61 **Входные параметры**

Код потока Dynamic loading of Excel files

4.15 Циклическая проверка данных с помощью использования http

Входящие данные для данного примера: в начало данного потока приходит наименование датасета для поиска запроса соответствующей функции по этому наименованию.

> Для работы данного потока необходима подготовленная функция в базе.

Алгоритм работы функции: на вход функции приходит код ошибки (где 0 это старт), при получении определённого кода будет производиться определённая проверка.

Проверка, которая завершилась с ошибкой, возвращает код ошибки и описание ошибки -> их мы покажем на веб интерфейсе.

На веб интерфейсе будет 2 варианта, при возникновении ошибки: ознакомиться с ошибкой и игнорировать её (продолжить) и остановить загрузку.

При варианте `Продолжить` мы возвращаемся в начало потока и запускаем функцию, но уже не сначала, а с кода ошибки, которую проигнорировали.

Каждый узел будет описан отдельно.

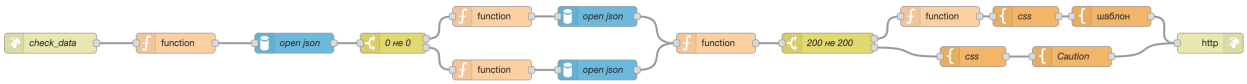


Рис. 4.62 **Циклическая проверка данных с помощью использования http**

4.15.1 Первичная проверка запуска фунцкии

- **http in**

Наименование пути по http, на данном этапе нам уже необходимо получить номер датасета в данный объект (например с помощью глобальных или потоковых переменных). Или же, если нет необходимости в динамическом изменении скрипта для проверки, можно сразу указать его.

- **function №1**

Функция принимает наименование датасета и внедряет его в запрос к базе.

В данном примере у нас уже есть созданная таблица, в которой наименование датасета соответствует запросу процедуры. Соответственно мы просто забираем из базы запрос к функции проверки.

Код функции:

```
let ds = msg.payload.ds;
msg.payload.query = "select json_extract_path_text(config::json, ↩
  'function_check') FROM custom.etl_matrix WHERE schema_name = '" + ds + "'";
return msg;
```

- **open json**

Получаем запрос для функции проверки. Сам узел оставляем пустым, так как запрос мы прописали на предыдущем узле в `msg.payload.query`.

- **switch 0 не 0**

Создаем простейшую проверку: если функция уже запускалась, у нас будет прописан определенный ключ возврата в потоковых переменных, если нет - запускаем процедуру с ключом старта.

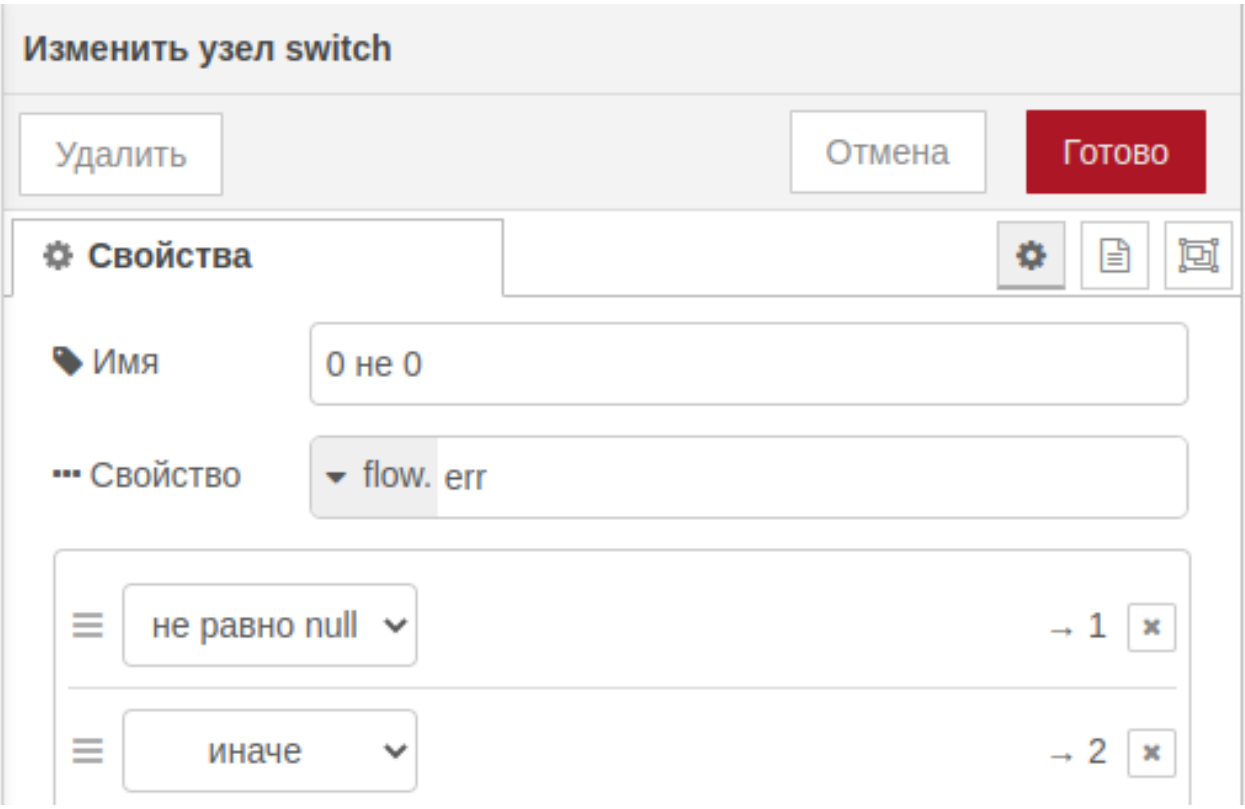


Рис. 4.63 **Первичная проверка ключа**

- **function если код возврата существует**

Запускаем процедуру проверки с определённым ключом (с соответствующего этапа).

Код функции:

```
let err = flow.get("err");
let func = msg.payload.json_extract_path_text;
msg.payload.query = "SELECT " + func + "(" + err + ")";
return msg;
```

- **function если код возврата не существует**

Запускаем процедуру с начальным ключом (0).

Код функции:

```
let func = msg.payload.json_extract_path_text;
msg.payload.query = "SELECT " + func + "(0)";
return msg;
```

4.15.2 Проверка кода возврата из функции

- **function №4**

Заносим ключ возврата в потоковые переменные.

Код функции:

```
let err_tmp = msg.payload.check_new_data.f1;
flow.set("err", err_tmp);
return msg;
```

- **swith 200 не 200**

Если ключ 200 (OK) - возвращаем страницу с окончанием проверок. Иначе возвращаем страницу с ошибкой и описанием.

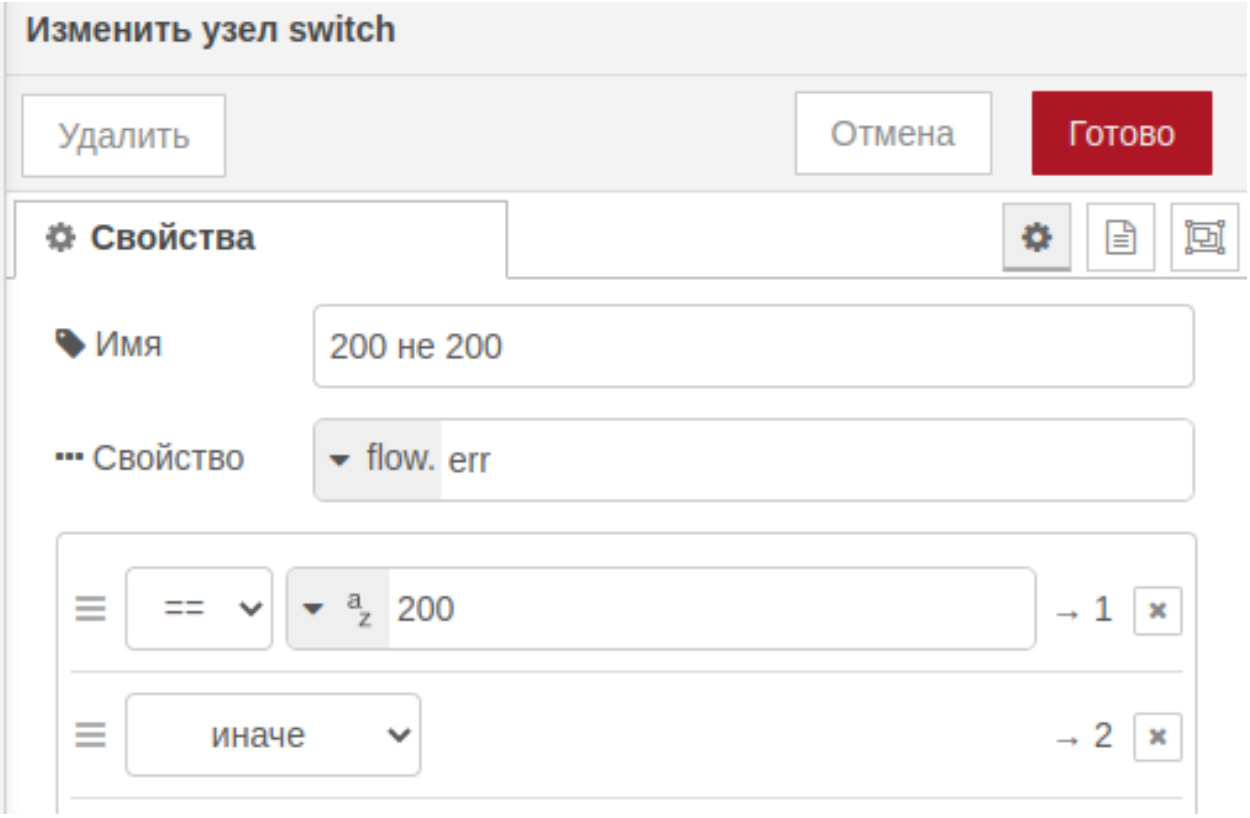


Рис. 4.64 **Проверка кода ошибки**

- **function если код 200**

Если пришло 200 - очищаем потоковую переменную, чтобы при следующем запуске начать сначала.

Код функции:

```
flow.set("err", undefined);
return msg;
```

- **template для кода 200**

Страница с успешным окончанием проверок. Выводит сообщение об успешном окончании проверок.

- **template для кода не 200**

Страница с вариантами игнорирования проверки и продолжения проверок (возврат в начало к http in) или остановки прогрузки (две разных кнопки).

Код потока Cyclic data validation using http

4.16 Мониторинг выполнения потоков при помощи KeyDB как брокера сообщений

4.16.1 Концепт:

В данном решении существует три части: 1) глобальный поток, который можно поместить в один любой поток - эта часть, отвечающая за рассылку сообщений в случае зависания или долгого выполнения потока. Сюда же входит часть с глобальными параметрами, которую необходимо единоразово запустить; 2) локальный поток, к которому необходимо подключить в начало и в конец отслеживаемого потока, эта часть, сопровождающая начало и конец отслеживаемого потока. Он должен располагаться вместе с отслеживаемым потоком; 3) сам отслеживаемый поток

Идея заключается в том, что потоку в момент его запуска присваивается счётчик=время жизни (TTL) в redis/keyDB с информацией об этом потоке. Далее канал отслеживающий все действия, в том числе время жизни потока, даёт сигнал об истечении, а дальше этот сигнал используется для объявлении об этом в сообщении переменной msg.msg.

При этом в данном случае, время жизни берётся на основании истории успешного выполнения потоков, на основании этой таблицы строится представление, которое откидывает 25 и 75 персентили, работая с адекватными значниями без выбросов. А далее считается СКО - дисперсия. Существует функция, которая на выход получает имя потока и на сколько сигм=дисперсий надо отступить, чтобы захватить время исполнения.

> Для работы данного потока необходимо настроенное подключение к myself_lamel (в данном примере, это бд postgres). Развернуть таблицу с историей выполнения потоков, если нет замены - public.upload_log_sdc, представление - public.mat_stat_upload_log_sdc и функцию - public.get_all_sigma_from_flow(TEXT, double precision). Это развёртывается при помощи нод в импортируемых json-ах. Так же подключение к keyDb. Не забыть ввести HOST стенда, где производится опробация решения для ноды с http запросом.

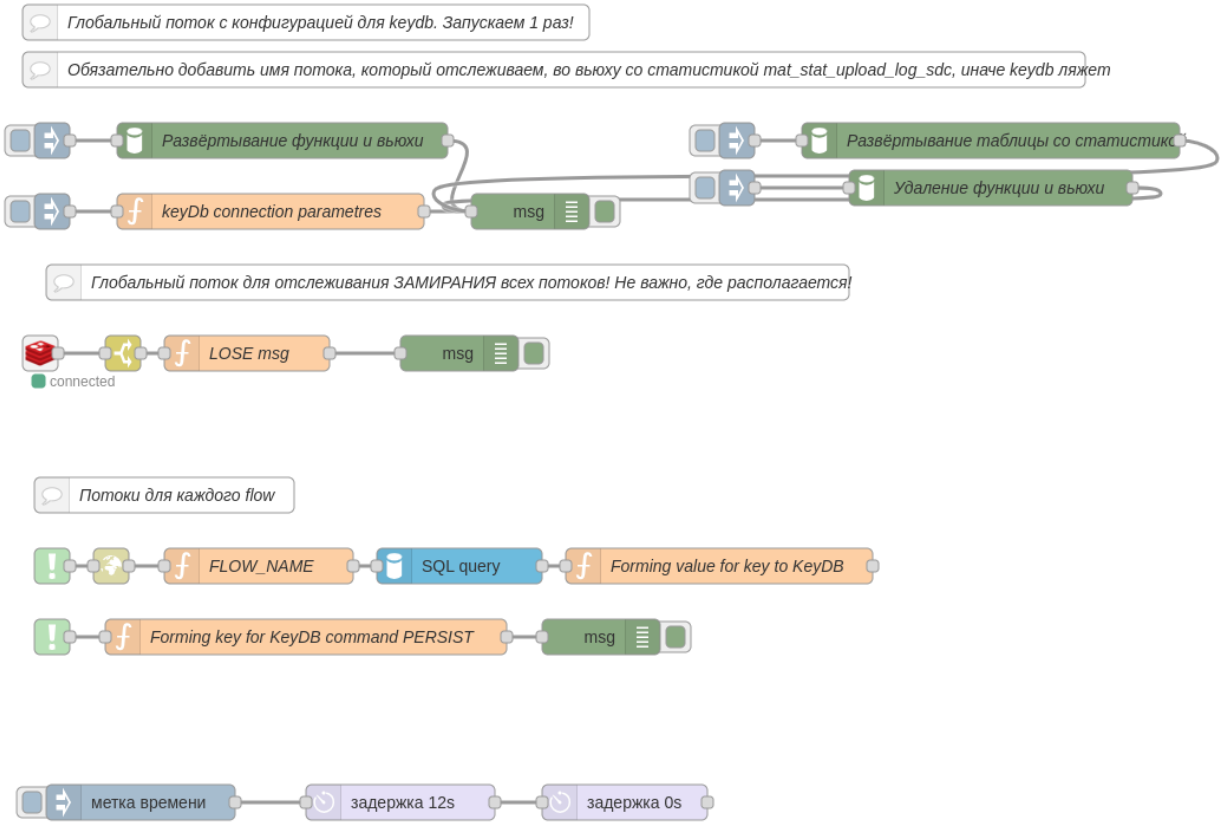


Рис. 4.65 **Скриншот потока**

4.16.2 Глобальная часть

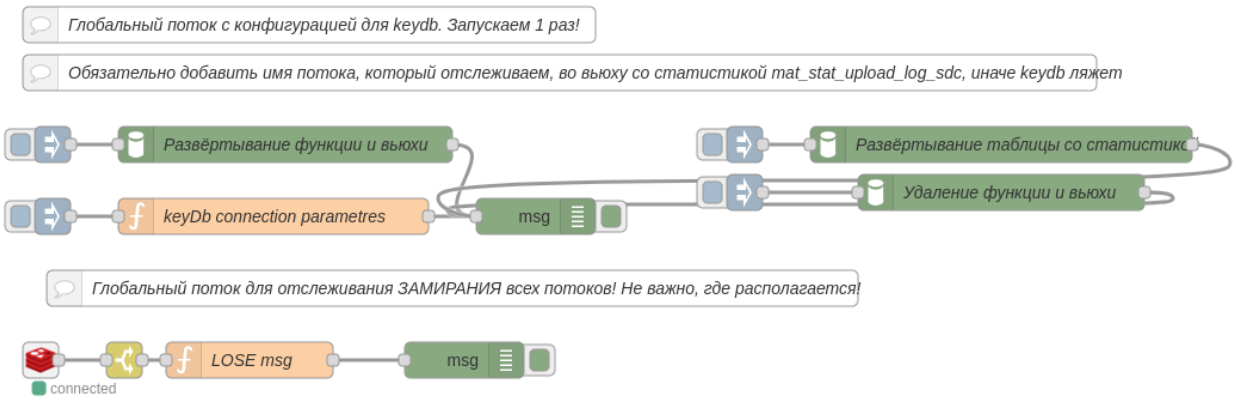


Рис. 4.66 **Скриншот Глобальной части**

- **Развёртывание таблицы со статистикой**

Код функции:

```
DROP table if exists public.upload_log_sdc;

CREATE TABLE public.upload_log_sdc (
```

```
 start_time timestamp NULL,
 raw_log text NULL,
 job_result text NULL,
 pipeline_success text NULL,
 end_time timestamp NULL,
 pipe_name text NULL
);


INSERT INTO public.upload_log_sdc
(start_time, raw_log, job_result, pipeline_success, end_time, pipe_name)
VALUES('2021-07-21 18:45:00.001', NULL, NULL, 'true', '2021-07-21 23:45:05.212',
   'cron_008_MUST_RUN_BUFFER');
INSERT INTO public.upload_log_sdc
(start_time, raw_log, job_result, pipeline_success, end_time, pipe_name)
VALUES('2021-07-21 20:15:00.001', NULL, NULL, 'true', '2021-07-22 01:15:05.254',
   'cron_008_MUST_RUN_BUFFER');
INSERT INTO public.upload_log_sdc
(start_time, raw_log, job_result, pipeline_success, end_time, pipe_name)
VALUES('2021-07-22 04:45:00.002', NULL, NULL, 'true', '2021-07-22 09:45:05.266',
   'cron_008_MUST_RUN_BUFFER');
```

- **Удаление функции и вьюхи**

Код функции:

```
--
вью для вычисления статистики времени выполнения потоков
DROP VIEW IF exists public.mat_stat_upload_log_sdc;--

Функция, которая будет возвращать границы одной-двух-трёх сигм при получении наи
  менования потока
DROP FUNCTION if EXISTS public.get_all_sigma_from_flow;
```

- **Развёртывание функции и вьюхи**

Код функции:

```
--
вью для вычисления статистики времени выполнения потоков
CREATE OR REPLACE VIEW public.mat_stat_upload_log_sdc AS
WITH diff AS (
 SELECT pipe_name,
 EXTRACT(EPOCH FROM end_time) - EXTRACT(EPOCH FROM start_time) diff
 FROM public.upload_log_sdc uls
 WHERE pipeline_success = 'true'
),

percentile_value AS (
SELECT pipe_name,
percentile_disc(0.25) WITHIN GROUP (ORDER BY diff) perc_25,
percentile_disc(0.75) WITHIN GROUP (ORDER BY diff) perc_75
```

```
FROM diff
GROUP BY pipe_name
)
SELECT DISTINCT pipe_name, floor(avg(diff) OVER (PARTITION BY diff.pipe_name))::int avg_v,
ceil(sqrt(var_pop(diff) OVER (PARTITION BY diff.pipe_name)))::int sigma
FROM diff
LEFT JOIN percentile_value USING (pipe_name)
WHERE diff BETWEEN perc_25 AND perc_75
UNION
SELECT 'Мониторинг потоков', 8, 1
UNION
SELECT 'Lamel', 8, 1;


COMMENT ON VIEW public.mat_stat_upload_log_sdc IS 'Представление с расчётами для каждого потока среднего значения выполнения потока с учётом выбросов, а также областей: одна-две-три сигма от этого среднего';--

Функция, которая будет возвращать границы одной-двух-трёх сигм при получении наименования потока

CREATE OR REPLACE FUNCTION public.get_all_sigma_from_flow(flow_name TEXT, multiplier double precision)
RETURNS TABLE (sigma int)
AS $$
BEGIN
 IF flow_name IN (
     SELECT pipe_name
     FROM public.mat_stat_upload_log_sdc
     )
     THEN
      RETURN QUERY EXECUTE 'SELECT abs(avg_v+sigma*floor($1))::int
        FROM public.mat_stat_upload_log_sdc
        where pipe_name = $2' USING multiplier, flow_name;
     ELSE
      RAISE EXCEPTION 'Flow name Error : %', flow_name;
 END IF;
END;
$$
LANGUAGE plpgsql;

COMMENT ON FUNCTION public.get_all_sigma_from_flow(text, double precision) IS 'Функция, которая будет возвращать значение в зависимости от множителя для сигмы и получении наименования потока';
```

- **keyDb connection parametres**

Код функции:

```
// Работа с keyDB
global.set('con_config_h', '127.0.0.1'); //конфиг подключения к KeyDB хост
global.set('con_config_p', 6379); //конфиг подключения к KeyDB порт
```

```
4 return msg;
```

- **redis in**

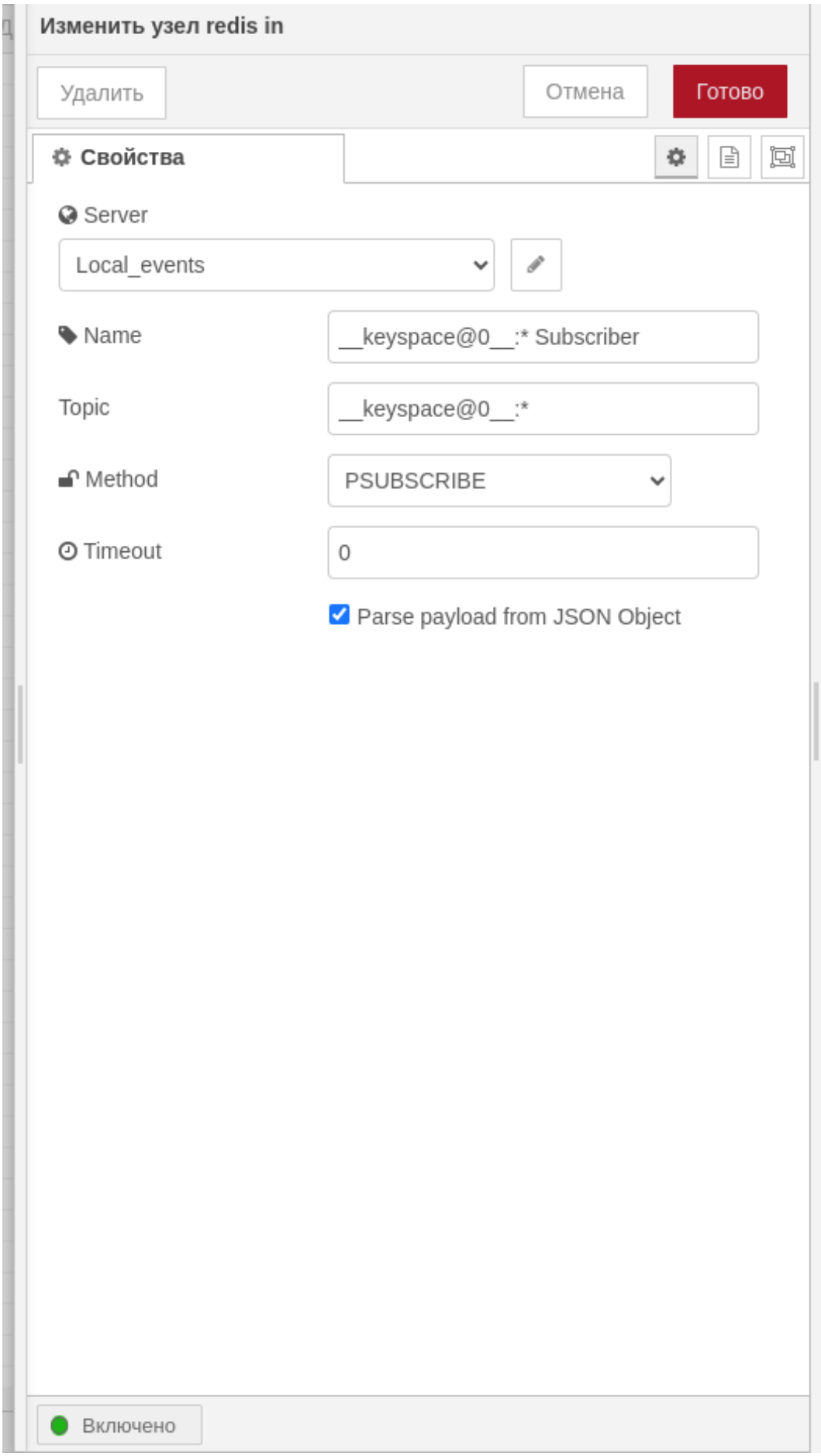


Рис. 4.68 **Скриншот redis in**

- **switch**

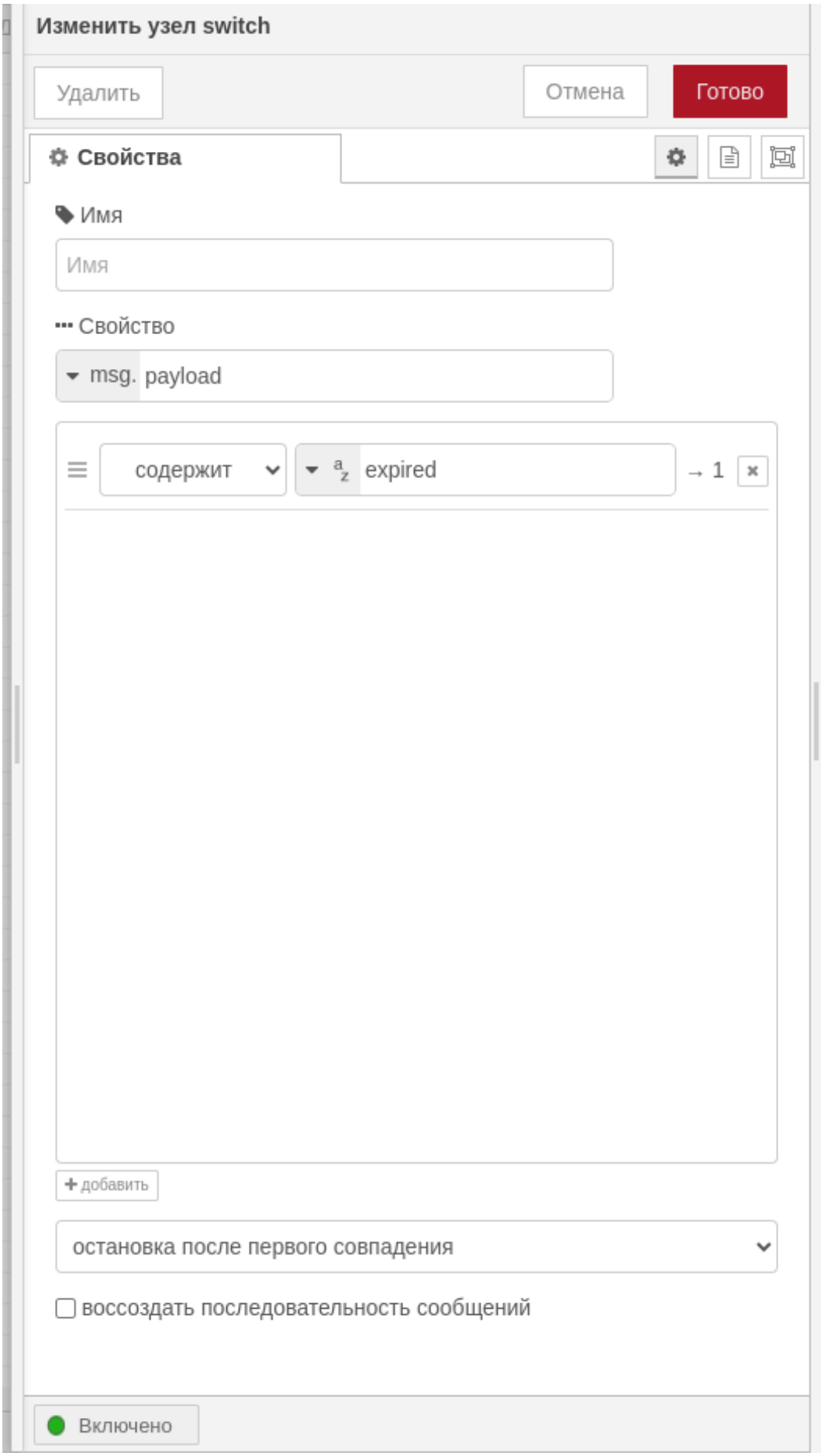


Рис. 4.69 **Скриншот switch**

- **LOSE msg**

Код функции

```
// Работа с keyDB
let con_config_h = global.get('con_config_h'); //конфиг подключения к KeyDB хост
let con_config_p = global.get('con_config_p'); //конфиг подключения к KeyDB порт
const red = new redis(con_config_p, con_config_h); //подключение к нужному хосту
    и порту

let key = msg.topic.replace('__keyspace@0__:', '');
let store_key = 'store:' + key; //ключ для хранилища всех ключей и значений

let info = await red.get(store_key);

msg.msg = 'Поток долго выполняется. ИНФО: ' + info;
return msg;
```

4.16.3 Локальный поток

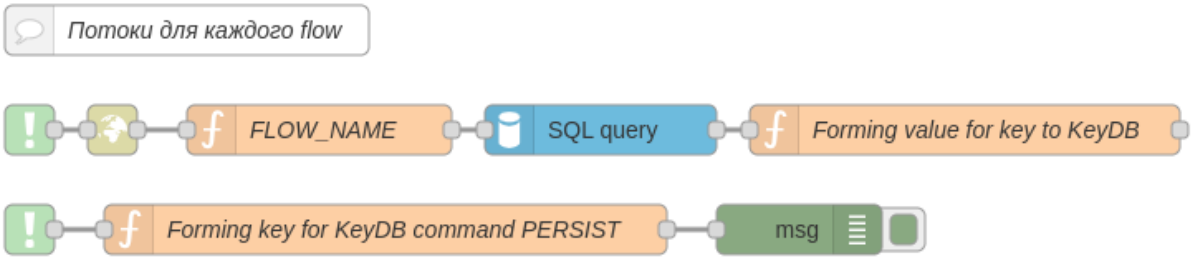


Рис. 4.70 **Локальный поток**

- **complete**

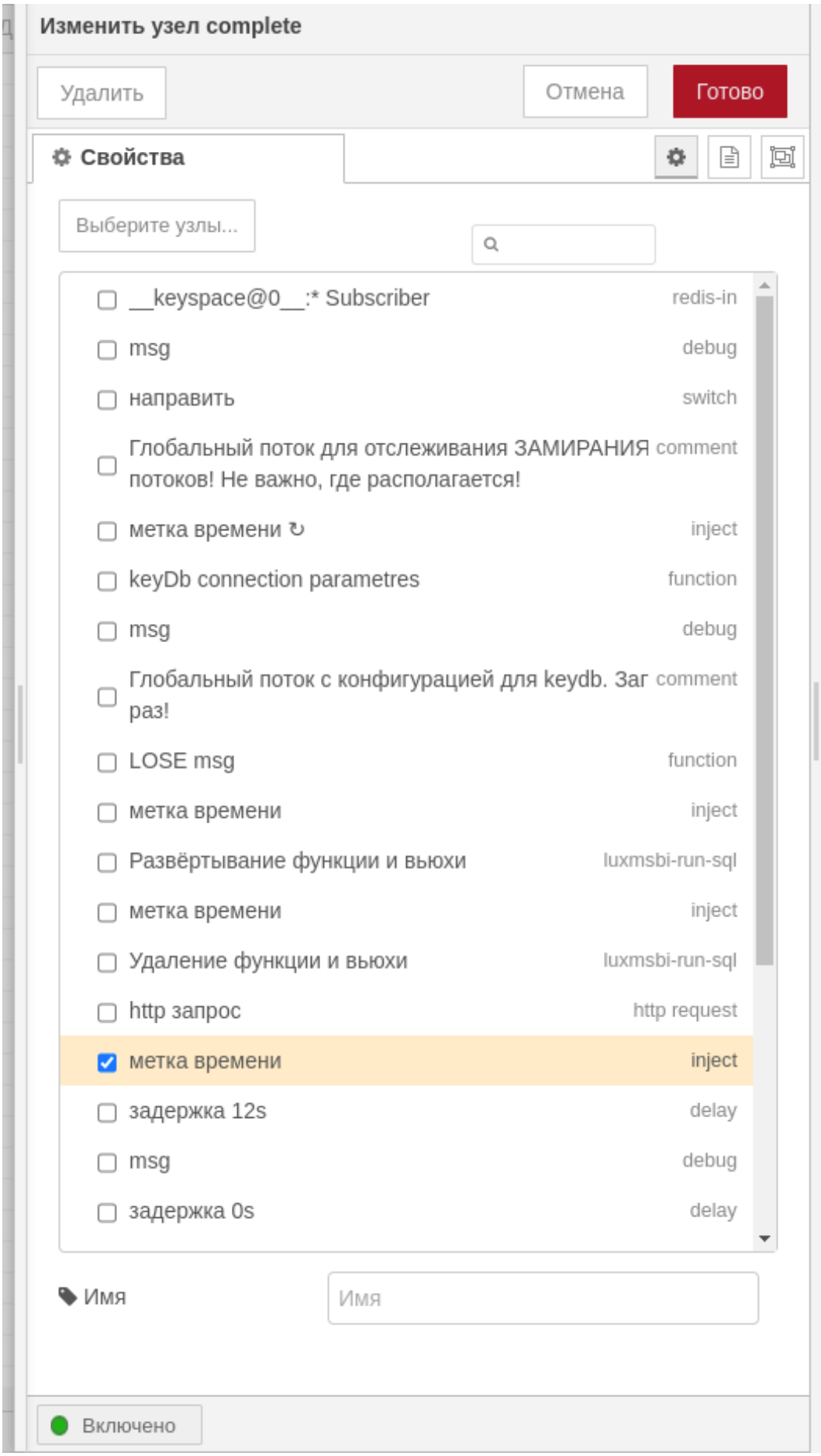


Рис. 4.71 **Скриншот complete 1**

- **http request**

Изменить узел http request

Удалить | Отмена | Готово

Свойства

Метод

GET

URL https://bi-et.luxmsbi.com/databoring/proces

Данные

Игнорировать

- [ ] Включить безопасное (SSL/TLS) соединение
- [ ] Использовать аутентификацию
- [ ] Включить keep-alive соединение
- [ ] Использовать прокси
- [ ] Only send non-2xx responses to Catch node
- [ ] Disable strict HTTP parsing

Возврат

строка UTF-8

Заголовки

+ добавить

Имя Имя

Включено

Рис. 4.72 **http request**

- **FLOW_NAME**

Код функции

```
msg.name = env.get('NR_FLOW_NAME'); // наименование потока

msg.user = JSON.parse(msg.payload)[0]['username']; // пользователь из databoring
  a

return msg;
```

- **SQL query**

Код функции

```
SELECT * FROM public.get_all_sigma_from_flow('${msg.name}', 6)
```

- **Forming value for key to KeyDB**

Код функции

```
let time = msg.payload.sigma; // среднее время выполнения потока
msg.dt_start = new Date().toISOString();
let title = env.get('NR_FLOW_ID'); //идентификатор flow
let name = env.get('NR_FLOW_NAME'); // наименование потока
let sys_user = env.get('USER'); // пользователь системный

// формирование JSON с информацией о потоке
let info = '{' + '"flow_id":' + '"' + title + '"' + ',' + '"flow_name":' + '"' +
   name + '"' + ',' + '"sys_user_name":' + '"' + sys_user + '"' + ',' +
  '"user_name":' + '"' + msg.user + '"' + ',' + '"dt_start":' + '"' +
  msg.dt_start + '"' + '}';

//На вход в keyDB выполнение скрипта - команда
let run_fvalue = JSON.parse(info);

// ключ - значение для keyDB
let run_key = msg.user + title;
let run_value = '\'' + info + '\'';

flow.set('flow_key', run_key); // потоковая переменная ключ потока


// Работа с keyDB
let con_config_h = global.get('con_config_h'); //конфиг подключения к KeyDB хост
let con_config_p = global.get('con_config_p'); //конфиг подключения к KeyDB порт

let store_key = 'store:' + run_key; //ключ для хранилища всех ключей и значений
const red = new redis(con_config_p, con_config_h); //подключение к нужному хосту
   и порту
```

```
red.set(run_key, run_value); // устанавливаем ключ значение для текущего потока
red.set(store_key, run_value); // устанавливаем ключ значение для текущего поток↩
  а
red.expire(run_key, time); // ставим установленному ключу для потока время жизни

msg.payload = await red.get(run_key);
// msg.store_key = await red.get(store_key);

return msg;
```

- **complete**

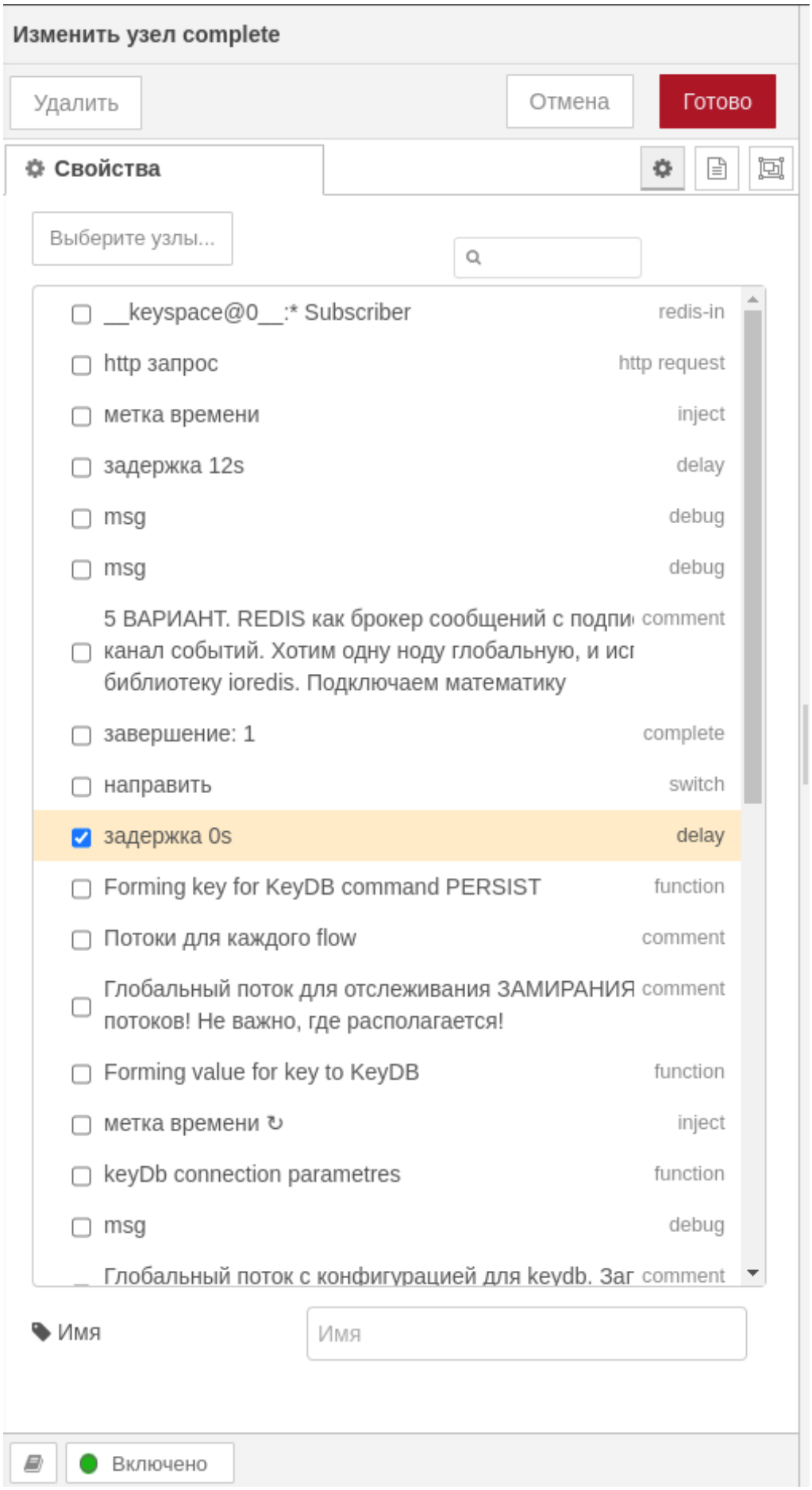


Рис. 4.73 **Скриншот complete 2**

- **Forming key for KeyDB command PERSIST**

Код функции

```
let flow_key = flow.get('flow_key');// использование потоковой переменной ключ

// Работа с keyDB
let con_config_h = global.get('con_config_h'); //конфиг подключения к KeyDB хост
let con_config_p = global.get('con_config_p'); //конфиг подключения к KeyDB порт
const red = new redis(con_config_p, con_config_h); //подключение к нужному хосту
    и порту

msg.persist = await red.persist(flow_key); // устанавливаем ключ значение для те
  кущего потока

let store_key = 'store:' + flow_key; //ключ для хранилища всех ключей и значений
let get_value = await red.get(store_key);

//message для информации
msg.msg = 'Поток выполнился. ИНФО: ' + get_value;

return msg;
```

Рис. 4.74 **Скриншот отслеживаемого потока**

Код потока Monitoring the execution of threads using KeyDB 1

4.16.4 Таблица с историей выполнения потоков

Код таблицы с историей выполнения потоков Table with thread execution history

Поток можно разворачивать и тестировать у себя на стенде

4.17 Отправка уведомлений на почту

Поток помогает отправлять уведомления на почту.

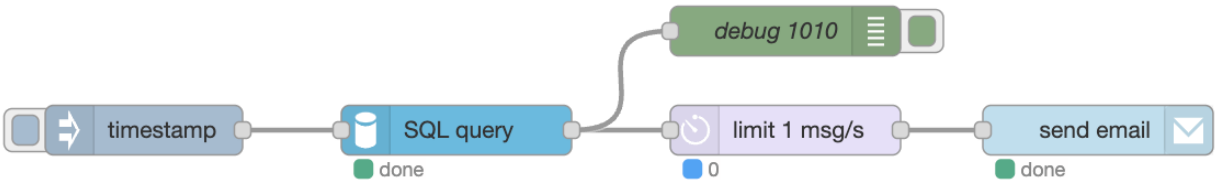


Рис. 4.75 **Поток SQL Query**

Параметры приходят на вход узлу динамически, поэтому их не нужно устанавливать.

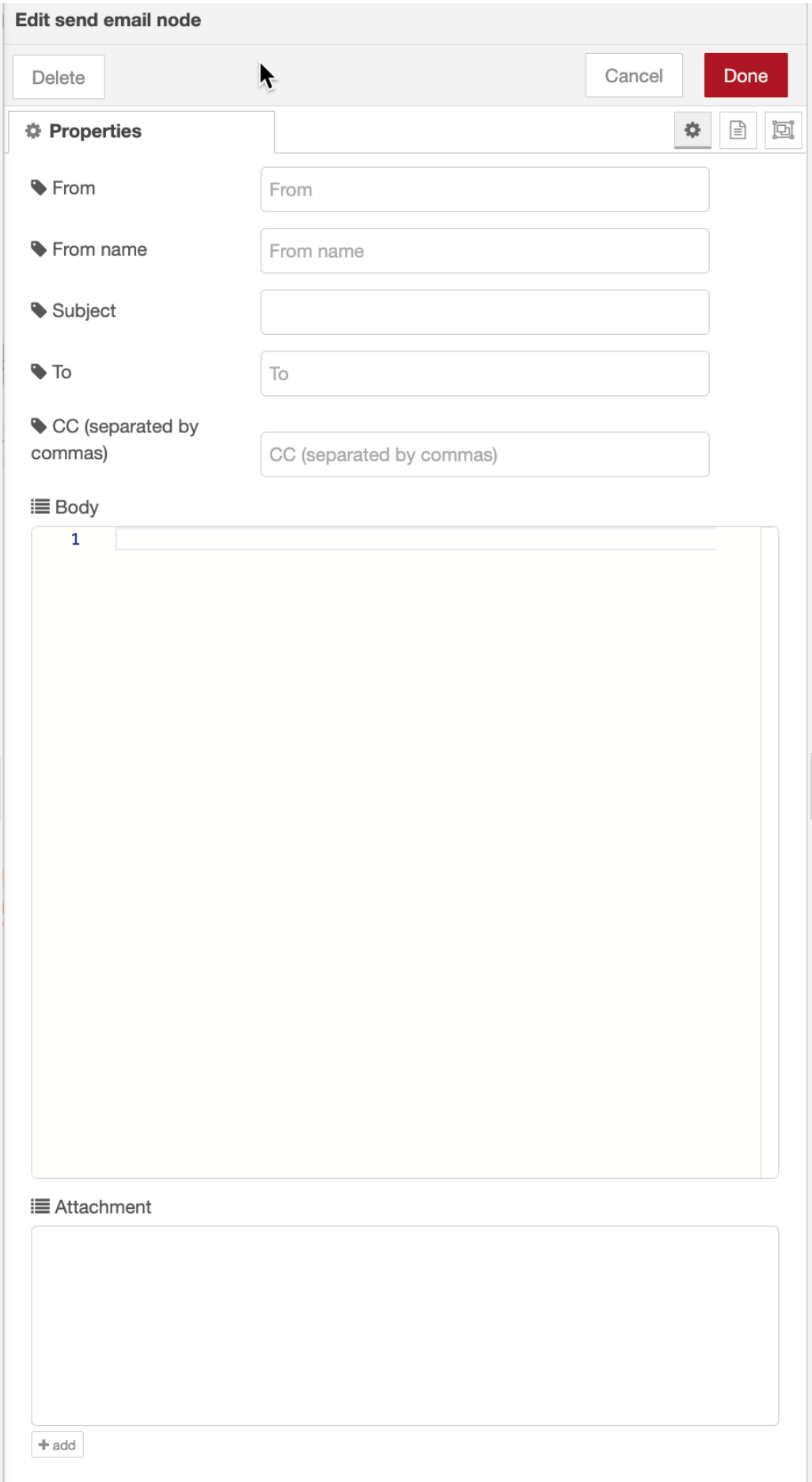


Рис. 4.76 **Редактирование узла**

Формирование параметров в узле `SQL query` базы `PostgeSQL`, которая передаёт на вход узлу `send email`, приведен ниже. Данный запрос формирует два сообщения в окне `debug`, который позволяет отправить сигнал на вход узлу дважды.

```
SELECT 'Luxms Data Boring 1' AS subject
    ,'user1@mail.com' AS from
    ,'user2@mail.com' AS to
    ,'«I sent you data, check it out. 📩⬆️' AS body

UNION ALL

SELECT 'Luxms Data Boring 2' AS subject
    ,'user1@mail.com' AS from
    ,'user3@mail.com' AS to
    ,'«I sent you information about docs check it out. 📩⬆️' AS body
```

Результат работы узлы `SQL query` видим в окне `debug`.

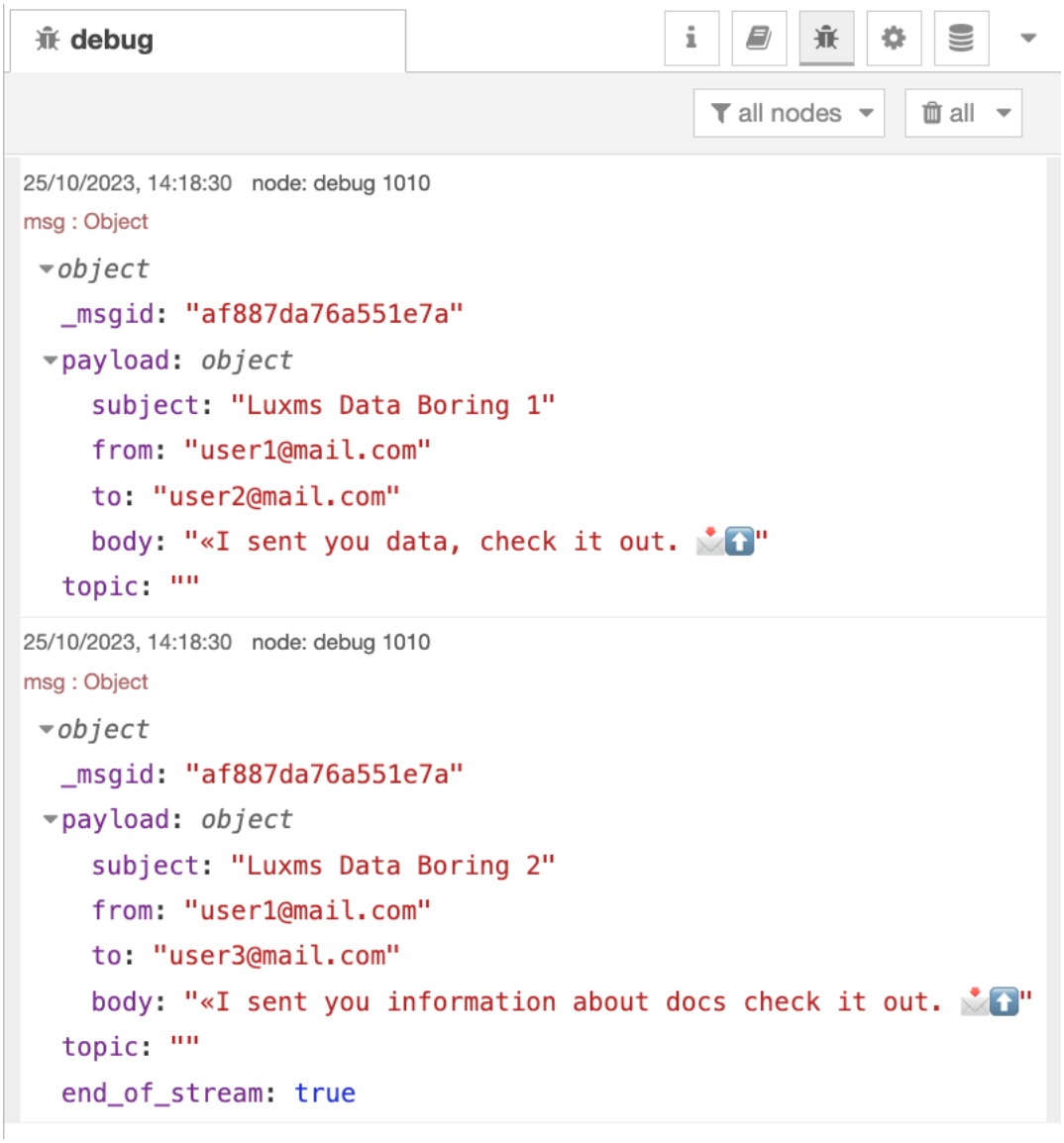


Рис. 4.77 **Окно debug**

Код потока Sending notifications to the mail