Руководство пользователя ZESB

ООО "ОРПЛАБ"

2022

Гибридная интеграционная платформа "ZESB" предоставляет несколько программных продуктов для удовлетворения потребностей компании в управлении API и переходе на микросервисную архитектуру. Требуется ли создать api или микросервис, программное обеспечение "ZESB" поможет удовлетворить изменяющиеся потребности в масштабе предприятия.

1. "ZESB" Application Server представляет собой сервер приложений на основе Net.Core и среду развертывания, а также включает
встроенный механизм для логирования и мониторинга исполняемых процессов.

2."ZESB" студия – декларативный, визуальный инструмент графической разработки и тестирования для "ZESB" Application Server. Он обеспечивает визуальную среду разработки для моделирования, тестирования, конфигурирования и развертывания решений интеграции.

ООО "ОРПЛАБ"

+7(969)117-77-01

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Данное руководство предназначено для разработчиков, DevOps, системных администраторов.
Оно познакомит вас:

- Как создавать новое приложение в студии ZESB (Zingy ESB);
- Как управлять конфигурациями приложения;
- Как развертывать приложение;
- Как конфигурировать и запускать сервер приложений "ZESB" (Zingy ESB)

1.1. О ПРОГРАММЕ

Гибридная интеграционная платформа "ZESB" предоставляет несколько программных продуктов для удовлетворения потребностей компании в управлении API и переходе на микросервисную архитектуру. Требуется ли создать api или микросервис, программное обеспечение "ZESB" поможет удовлетворить изменяющиеся потребности в масштабе предприятия.

1. "ZESB" Application Server представляет собой сервер приложений на основе Net.Core и среду развертывания, а также включает встроенный механизм для логирования и мониторинга исполняемых процессов.

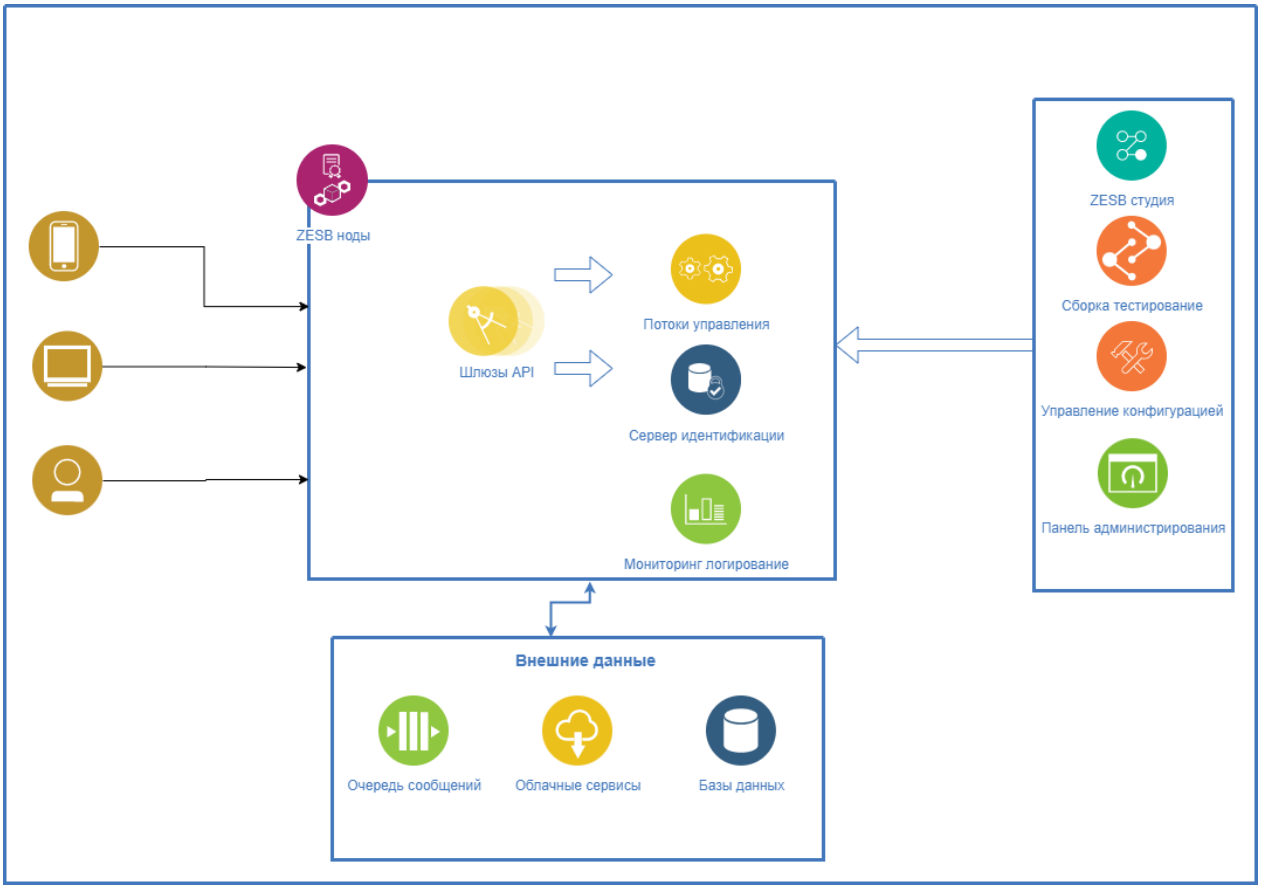
2. "ZESB" студия – декларативный, визуальный инструмент графической разработки и тестирования для "ZESB" Application Server. Он обеспечивает визуальную среду разработки для моделирования, тестирования, конфигурирования и развертывания решений интеграции.

1.2. ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Термины	Описание
ZESB	Сокращенное название от Zingy ESB
Студия ZESB	"Zingy ESB" студия – декларативный, визуальный инструмент графической разработки и тестирования для "ZESB" сервер приложений.
Сервер приложений ZESB	Прикладное приложение для исполнения программ на языке DSL ZESB
API	Описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

Микросервис	Это архитектурный подход к проектированию приложений. Программный модуль в IT инфраструктуре отвечающий за выполнение одной задачи
DSL	Предметно ориентированный язык (Domain Specific Language)
JPATH	Язык навигации по свойствам JSON объекта, аналог XPATH (см. раздел JPATH)

1.3. АРХИТЕКТУРА



2. РУКОВОДСТВО ZESB СТУДИЯ

В данном руководстве вы узнаете как пользоваться студией разработки "ZESB".
Создадите свое первое приложение с использованием предметно ориентированного языка ZESB.

2.1. ЗНАКОМСТВО

"ZESB" студия – декларативный, визуальный инструмент
графической разработки и тестирования для "ZESB" Application Server. Он обеспечивает визуальную среду разработки для моделирования, тестирования, конфигурирования и развертывания решений интеграции.

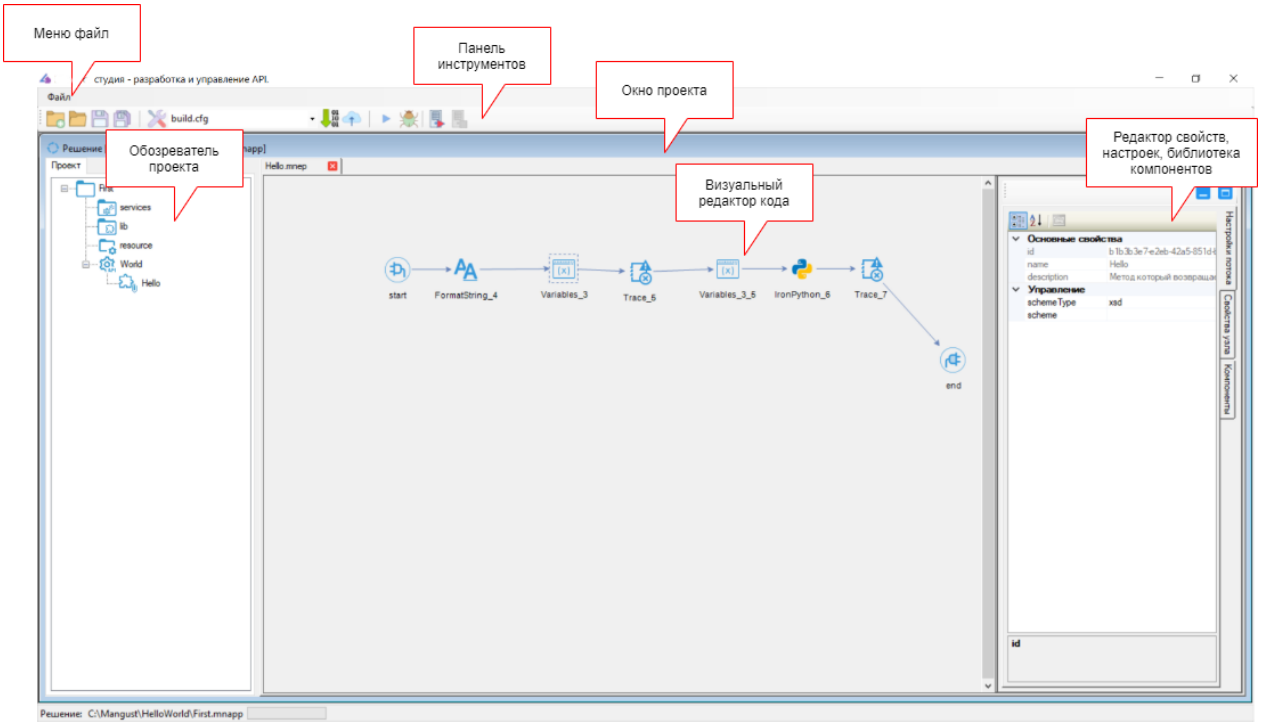
"ZESB" студия упрощает разработку решений:
- в области создания API, микросервисов;
- интеграционных задач с использованием богатых функциональных возможностей языка DSL;
- повторного использования разработанных API и микросервисов как компонентов сервисов.

"ZESB" студия позволяет создавать комплексные бизнес-решения – процессы, шлюзы, адаптеры, компоненты кода – в удобной среде разработки. Предоставляемые визуальные редакторы минимизируют необходимость написания исходного кода.
"ZESB" студия, предоставляет единый инструментарий для разработчиков приложений и решений в таких областях как управление API, Микросервисы, интеграция.

2.2. ТЕРМИНЫ

1. **Проект** - конфигурационный файл описывающий решение и устанавливающий связи между объектами решения;
2. **Шлюз** - точка входа для http запросов (определяемая Хост:Порт/<Имя шлюза>;
3. **Эндпоинт** - точка доступа и обработки запроса;
4. **Поток управления** - подпрограмма для обработки контекста данных описанная на DSL "ZESB";
5. **Служебный сервис** - это фоновое задание которое работает как потоковый обработчик либо как задание по расписанию;

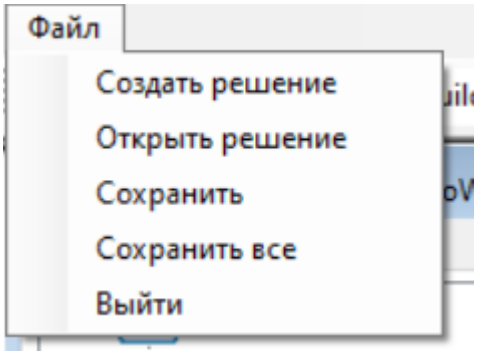
2.3. ИНТЕРФЕЙС



- *Главное меню файл*
  Меню файл - содержит основные процедуры по работе с проектом. Позволяющие создавать новое решение (Проект), открывать и сохранять решение и все изменения внутри него.
- *Панель инструментов*
  Необходимые функции для управления работой проекта.
- Окно проекта
  Окно проекта - основное окно где происходит редактирования кода программы. И доступ ко всем компонентам проекта.
- Обозреватель проекта
  Обозреватель проекта - предоставляет доступ к основным блокам проекта и составляющим проекта.
  Проект подразделяется на четыре области:
  1. Сервисы - это задачи выполняемы в фоне для обработки потоковых данных и по расписанию
  2. Библиотеки - наборы библиотек на языке DSL ZESB
  3. Ресурсы - конфигурационные данные проекта
  4. Шлюзы - точки развертывания API. Может быть много шлюзов развернуто и создано в рамках одного проекта.
- Визуальный редактор кода
  Визуальный редактор кода - позволяет создавать и редактировать потоки управления для обработки контекста данных.

2.3.1. МЕНЮ FILE

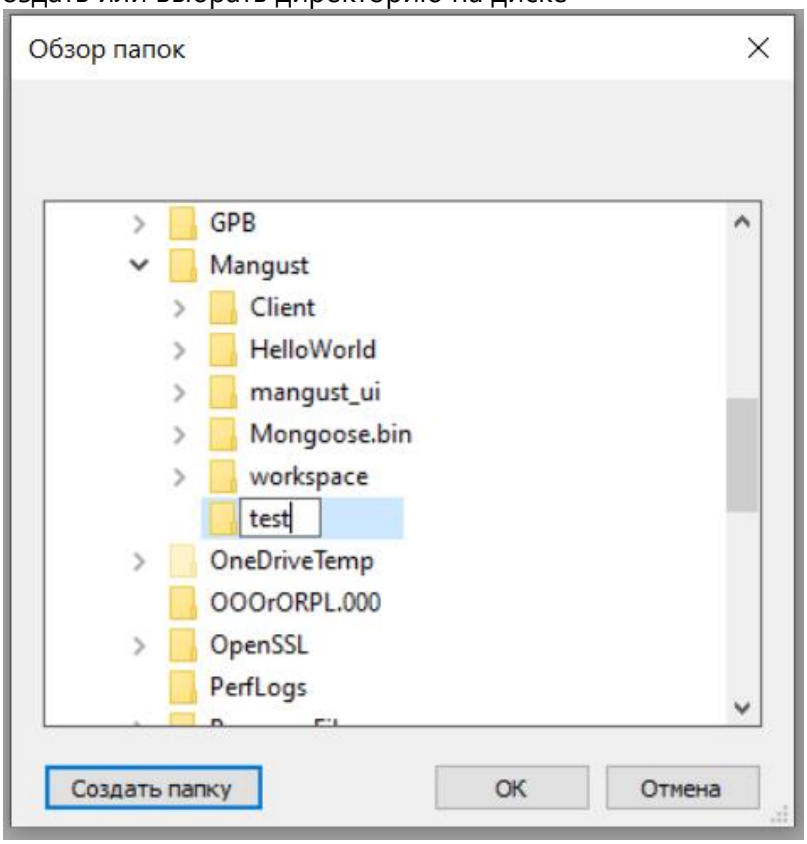
В данном разделе представлены основные



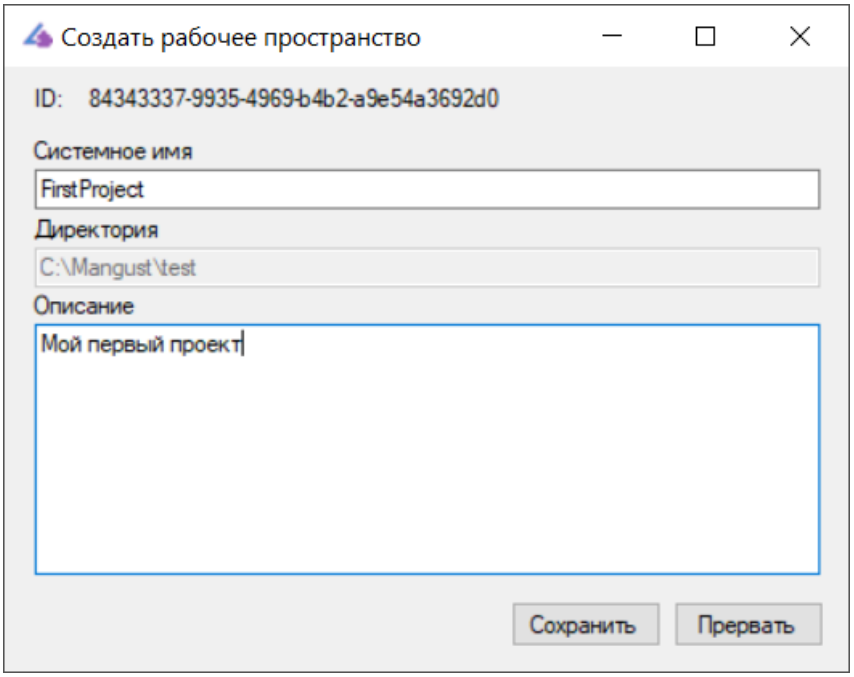
- Создать проект

  Для того чтобы создать новый проект необходимо сделать следующие операции:

  1. Выбрать пункт меню **Файл > Создать решение**
  2. Создать или выбрать директорию на диске



  3. Указать название проекта и заполнить комментарий к проекту и кликнуть на кнопку сохранить **.**

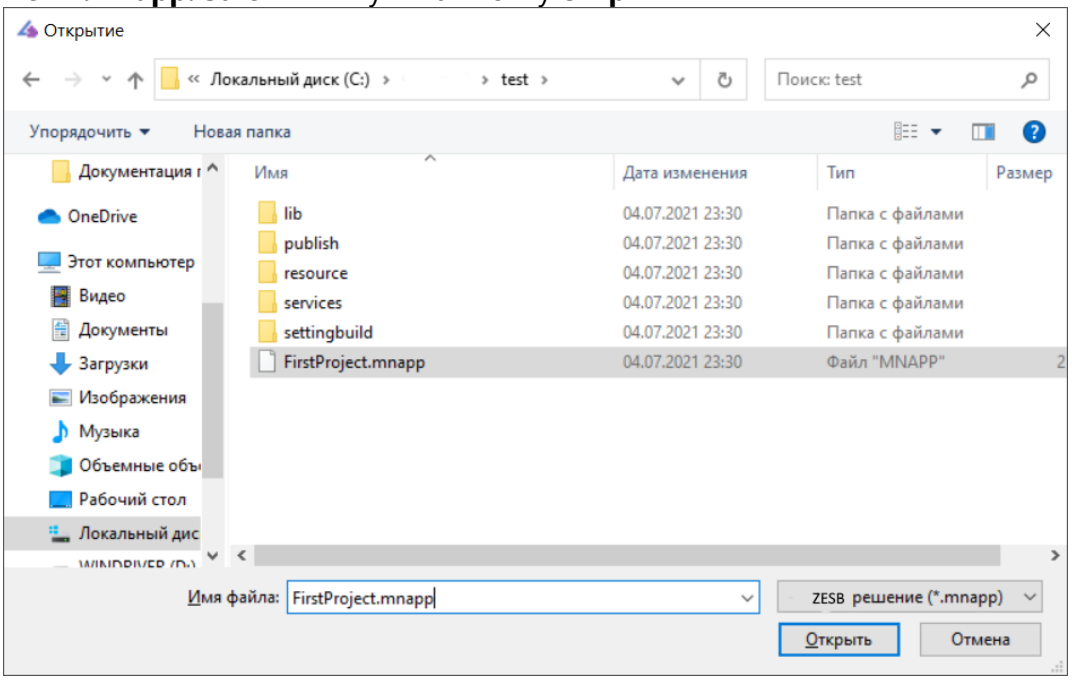


После этого проект создан и можно приступать к работе.

- Открыть решение

  Для открытия проекта вам потребуется сделать следующие действия:

  1. Выбрать пункт меню **Файл > Открыть решение**
  2. Выбрать рабочую директорию где расположен проект и кликнуть на файл с расширением ***.mnapp.** Затем кликнуть на кнопку **открыть**



  После чего проект будет открыт в приложении.

- Сохранить и сохранить все

  Для сохранения изменений в проекте предусмотрены следующие функции:

1. Выбрать пункт меню **Файл>Сохранить** или **Файл > Сохранить все**

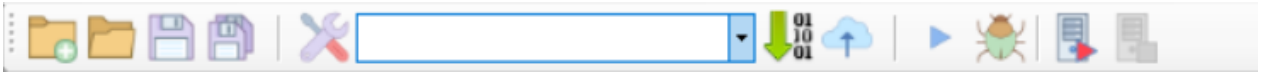
После этого изменения будут сохранены.

Отличие Сохранить и Сохранить все заключается в следующем :

- Сохранить - сохраняет содержимое текущего открытого активного окна
- Сохранить все - сохраняет изменения всех открытых окон.

2.3.2. ПАНЕЛЬ ИНСТРУМЕНТОВ

Панель инструментов предоставляет быстрый доступ к основным функциям среды разработки ZESB.



▼ Управление проектом


Создать проект


Открыть уже созданный проект

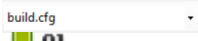

Сохранить изменения в активном окне


Сохранить все изменения в открытых окнах проекта

Смотри описание Меню файл

▼ Развертывание и сборка


При нажатии на данную кнопку будет вызвана форма конфигурирования сборки и развертывания приложения. Проект позволяет иметь несколько версий конфигураций для разных сред развертывания (продуктовой, качества, тестовой и т.д).смотри Подготовка к развертыванию



Выбор конфигурации для развертывания и исполнения приложения.


При нажатии на данную кнопку. Будет собрано приложение на основе конфигурационного файла.

И помещено в папку **<Корень проекта>/Deploy/<Версион>/**


Зарезервировано (Облачное развертывание)

▼ Отладка потока управления


При нажатии на данную кнопку будет запущен активный поток управления из текущего проекта. Смотри Отладка приложения.


Зарезервировано

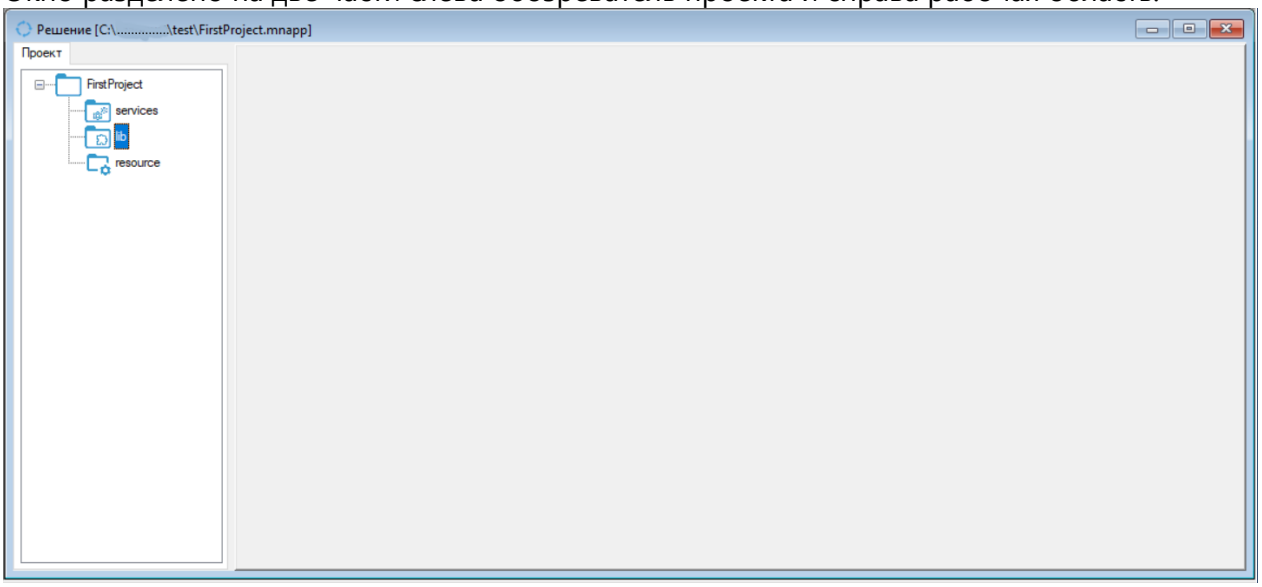
▼ Локальный запуск


При нажатии на данную кнопку будет запущен локально прикладной сервер ZESB. В качестве сборки возьмет текущий проект и текущую конфигурацию проекта.


При нажатии на данную кнопку запущенный локально прикладной сервер ZESB будет остановлен.

2.3.3. ОКНО ПРОЕКТА

После того как был создан новый проект на экране отобразиться основное окно управления для работы с проектом.
Окно разделено на две части слева обозреватель проекта и справа рабочая область.



Легенда иконок в обозревателе проекта

Данный раздел является корневым в дереве проекта. Содержит вложенные папки служебные сервисы, библиотека потоков управления, ресурсы проекта а также шлюзы API.

Раздел служебных сервисов

Раздел библиотек проекта

Раздел внутренних ресурсов проекта - соединения с базами данных, очередями а также настроечные таблицы параметров.

Шлюз. В одном проекте можно создавать больше чем 1 шлюз. Количество их не ограничено. Шлюз агрегирует в себе шаблон взаимодействия с api (аутентификацию, авторизацию, маршрутизацию на Endpoint).

Объекты

Ресурс отвечающий за соединение с БД или с очередью

Настройки приложение в виде пары ключ значение

Поток управления

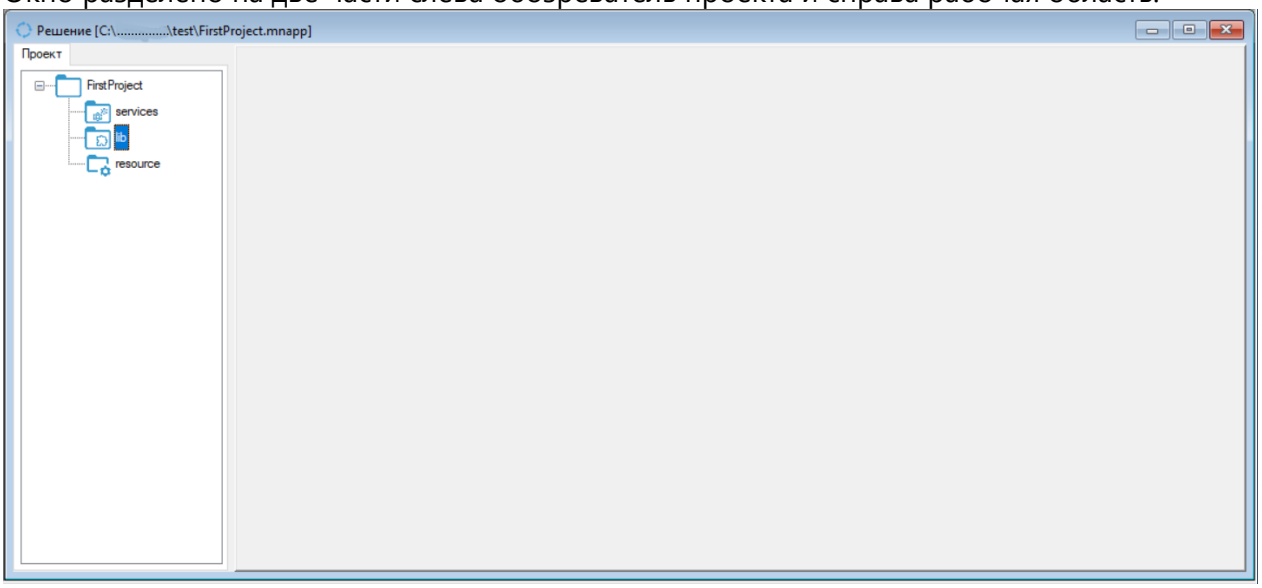
Потоковый служебный сервис. Обрабатывает бесконечно источник данных

Служебный сервис по расписанию - обрабатывает источник данных по расписанию. Расписание настраивается с помощью Cron.

Точка доступа в рамках шлюза (Endpoint)

2.4. РАБОТА С ПРОЕКТОМ

После того как был создан новый проект на экране отобразиться основное окно управления для работы с проектом.
Окно разделено на две части слева обозреватель проекта и справа рабочая область.



Легенда иконок в обозревателе проекта

Данный раздел является корневым в дереве проекта. Содержит вложенные папки служебные сервисы, библиотека потоков управления, ресурсы проекта а также шлюзы API.

Раздел служебных сервисов

Раздел библиотек проекта

Раздел внутренних ресурсов проекта - соединения с базами данных, очередями а также настроечные таблицы параметров.

Шлюз. В одном проекте можно создавать больше чем 1 шлюз. Количество их не ограничено. Шлюз агрегирует в себе шаблон взаимодействия с api (аутентификацию, авторизацию, маршрутизацию на Endpoint).

Объекты

Ресурс отвечающий за соединение с БД или с очередью


Настройки приложение в виде пары ключ значение


Поток управления


Потоковый служебный сервис. Обрабатывает бесконечно источник данных


Служебный сервис по расписанию - обрабатывает источник данных по расписанию. Расписание настраивается с помощью Cron.

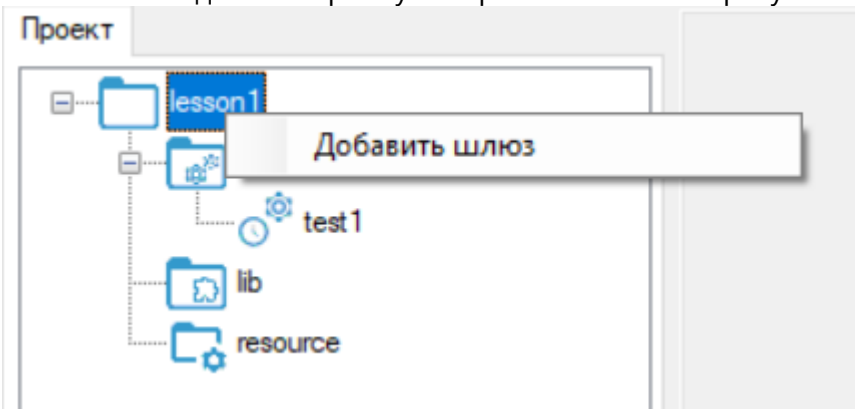

Точка доступа в рамках шлюза (Endpoint)

2.4.1. ШЛЮЗ

Шлюз - Программный элемент, который осуществляет авторизацию, валидацию и маршрутизацию запросов.

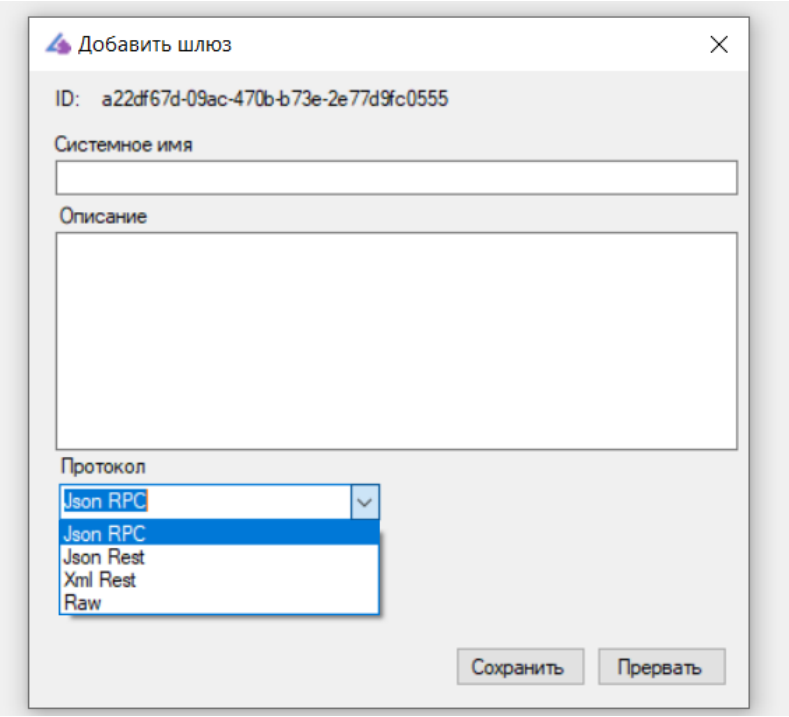
▼ Создать шлюз

Шаг 1: Необходимо выбрать узел проекта и нажать правую кнопку мыши.



Шаг 2: Заполнить поля формы

- **Системное имя** - осознанное имя относящееся к предметной области на английском языке;
- **Описание** - Краткое описание для чего нужен данный шлюз и что он делает
- **Протокол** - ожидаемый протокол обмена данными по умолчанию Json RPC (на данный момент поддерживается Json RPC)

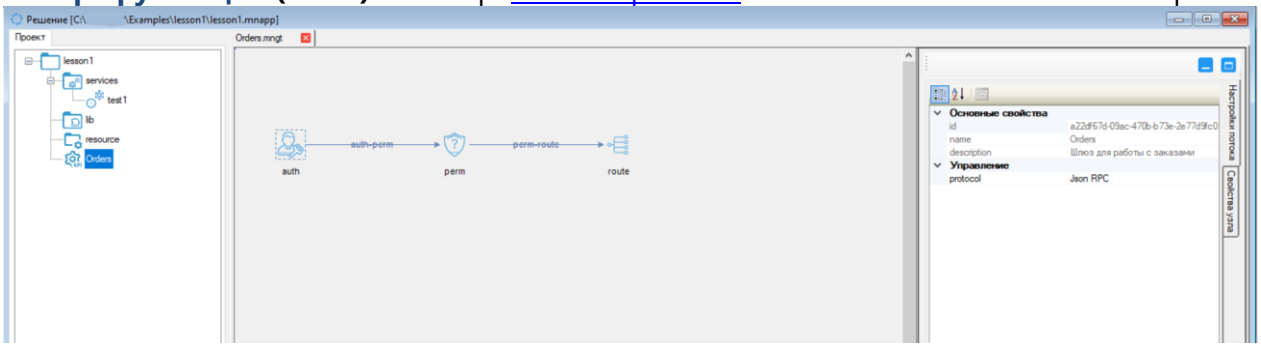


Шаг 3: После того как поля заполнены нажать кнопку сохранить.
Вот и готов шлюз для приема запросов.

▼ Настроить шлюз

После создания шлюза откроется закладка с внутренним устройством.
Шлюз состоит из трех блоков:

- **Аутентификация (auth)** - проверка доступен ли данный шлюз для клиента обращающегося к шлюзу
- **Авторизация (perm)** - проверка прав доступа к точкам обработки внутри шлюза
- **Маршрутизация (route)** - выбор точки обработки в зависимости от контекста запроса



Аутентификация свойства компонента(auth):

Свойства	Описание
UrlAuthorization	Путь к точке доступа для получения JWT токена по логину и паролю. Также поддерживается базовая аутентификация.
Method	Выбор типа проверки аутентификации • **NONE** - не проверять, анонимный доступ • **JWT** - авторизация по Json Web Token

Проверка прав доступа свойства компонента(perm):

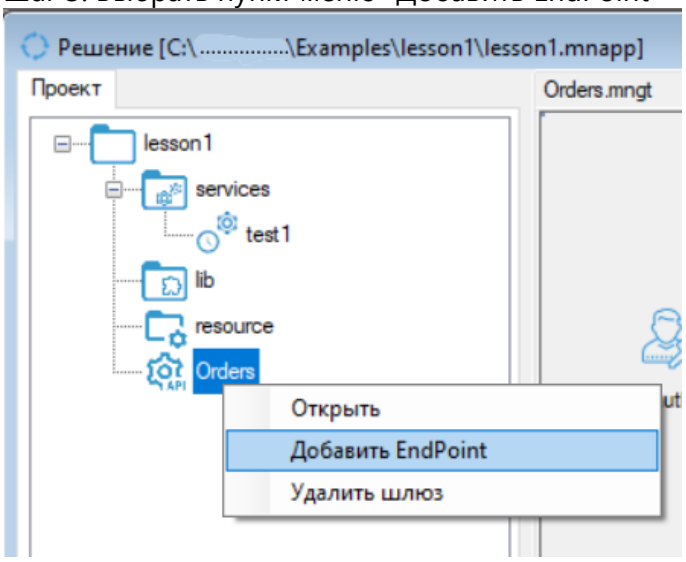
Свойства	**Описание**
Verify	Булевое значение **true** или **false.** Если выбрано значение **истина** то компонент проверяет имеет ли клиент право доступа к вызываемому методу.

Маршрутизация свойства компонента (route):

Свойства	**Описание**
Routes	Таблица маршрутов для определения точки обработки (необязательна для заполнения). На текущий момент не поддерживается запланиравано на будущее.

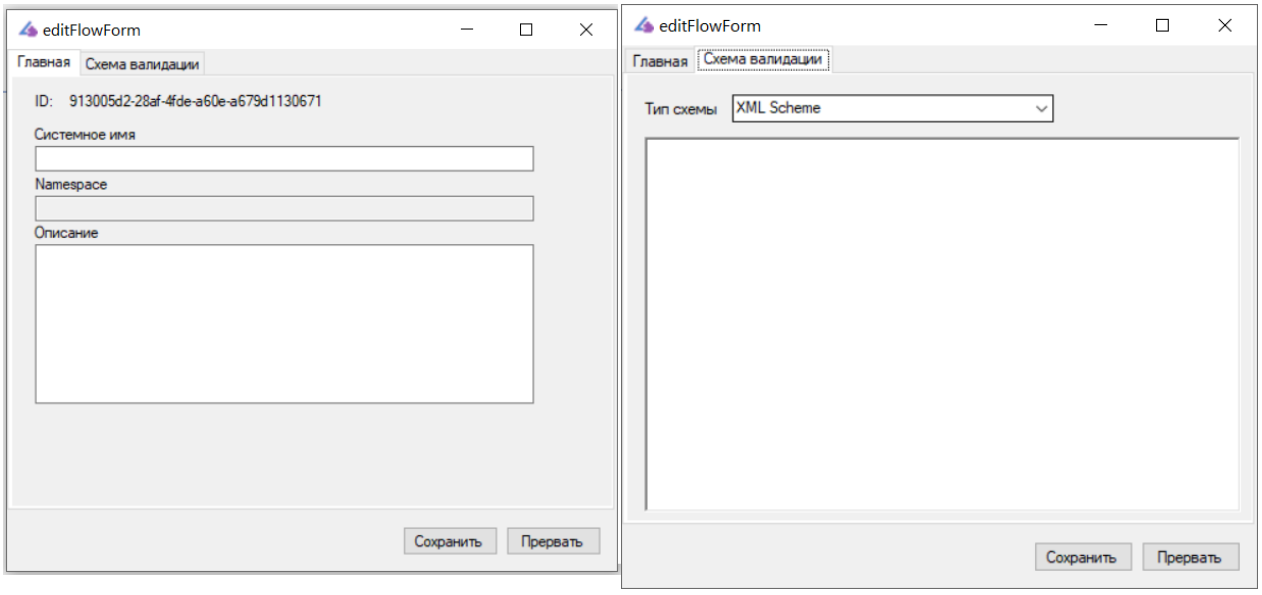
2.4.1.1. ТОЧКА ОБРАБОТКИ

Точка обработки- это точка вызова сценария обработки данных на DSL "ZESB".
Для добавления точки обработки необходимо выполнить несколько шагов:
Шаг 1: Выбрать шлюз
Шаг 2: Нажать правую кнопку мыши
Шаг 3: Выбрать пункт меню "Добавить EndPoint"



Шаг 4: Заполнить поля следующей формы.

- **Системное имя** -осознанное имя (имя метода, операции) на английском языке;
- **Описание** - краткое описание для точки обработки;
- **Namespace** - **(зарезервировано);**
- **Тип схемы** (схема валидации) - указывает тип схемы валидации это XML Scheme (xsd) или Json Scheme **(зарезервировано);**
- **Схема валидации** - **(зарезервировано);**



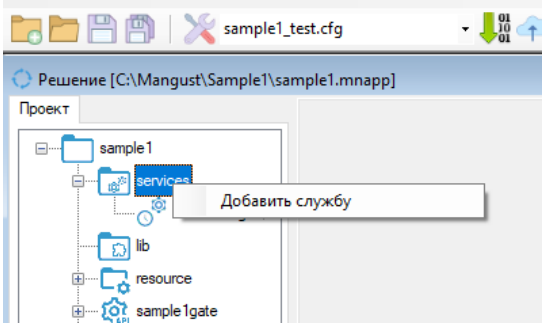
После того как заполнены необходимые поля нажимаем на кнопку "**сохранить".**

2.4.2. СЛУЖБА

Служба - это задание которое запускается по расписанию или выполняется постоянно в режиме много поточности.
Есть два варианта фоновых заданий:

1. Задание по расписанию где расписание устанавливается с точностью до минуты, формат описания времени запуска такой же как у **Cron** задания.
2. Потоковая обработка - это задание которое запускается постоянно в цикле и его можно масштабировать увеличивая количество потоков (обработчиков).

Для добавления фонового задания выполнить следующие шаги:
Шаг 1. Выбрать раздел services.
Шаг 2. Нажать правую кнопку мыши. И выбрать пункт **"Добавить службу"**.

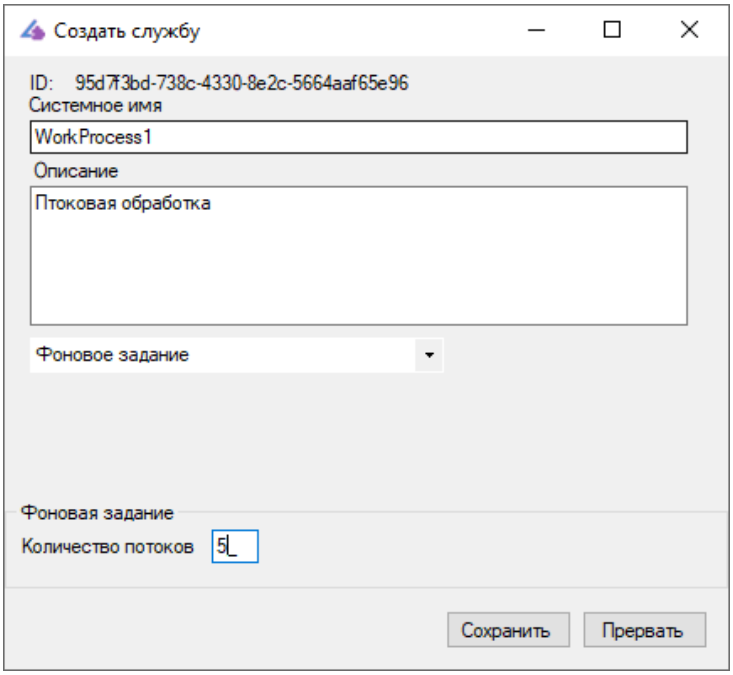


Шаг 3. Заполнить параметры формы.

- Фоновое задание

  Заполнить следующие поля:

  - **Системное имя** - уникальное имя задачи.
  - **Описание** - краткое описание.
  - **Количество потоков** - определяет количество потоков запускаемых в фоне

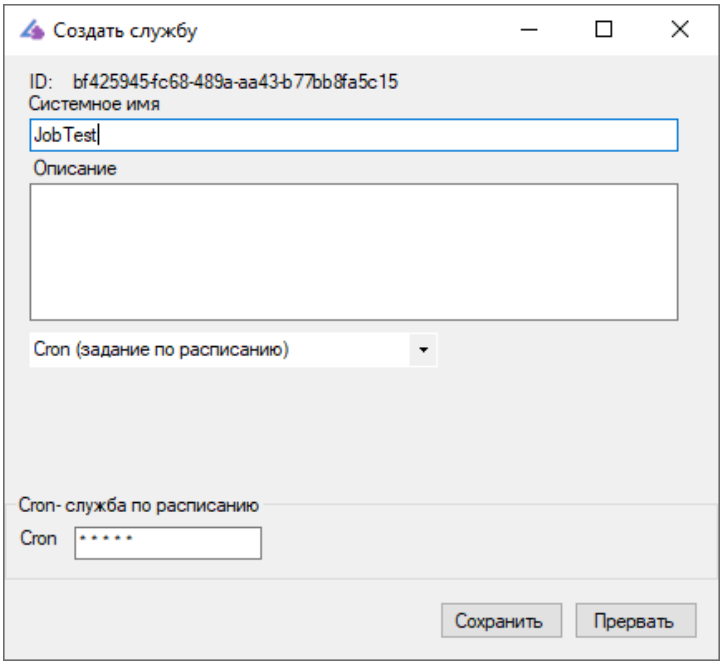


Нажать кнопку **сохранить**.

- Задание по расписанию

  Заполнить следующие поля:

  - **Системное имя** - уникальное имя задачи.
  - **Описание** - краткое описание.
  - **Cron** - строка расписания в формате cron.



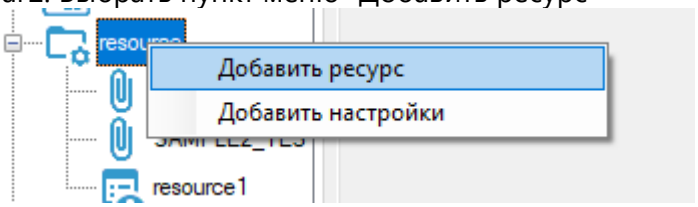
Нажать кнопку **сохранить**.

2.4.3. РЕСУРС

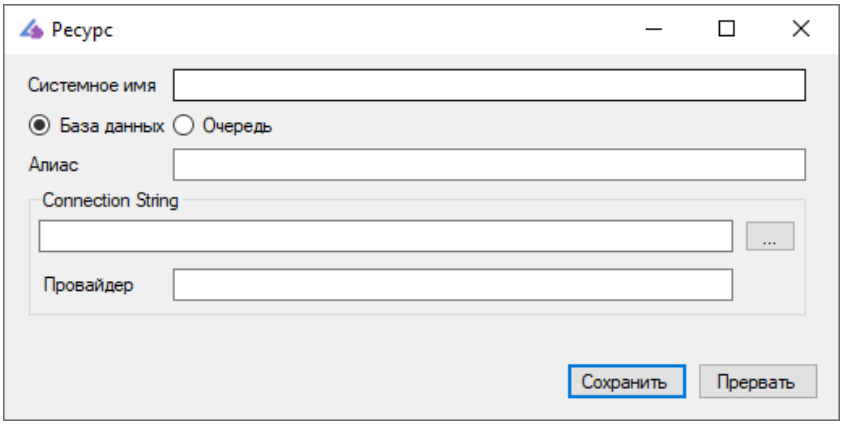
Раздел resource - это настройки среды выполнения необходимые для исполнения приложения.
Настройки бывают двух видов:
Ресурс (источник данных) - это строка соединения с источником данных (БД, очереди), которая формируется либо вручную либо с помощью помощника
Настройка - это список настроек (ключ, значение, тип данных)

- Строка соединения с источниками данных

  Шаг1: Выбрать раздел "resource" и нажать правую клавишу мыши.
  Шаг2: Выбрать пункт меню "Добавить ресурс"



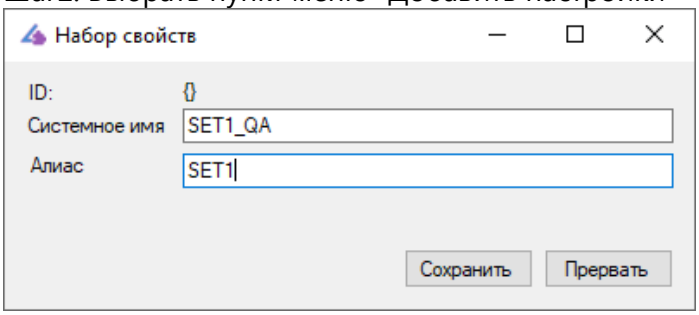
  Шаг3: Заполнить поля формы
  Все поля формы обязательны для ввода.
  Внимание: По значению в поле "алиас" осуществляется доступ к данным внутри приложения.



  Шаг4: Для сохранения изменений нажать кнопку "Сохранить"

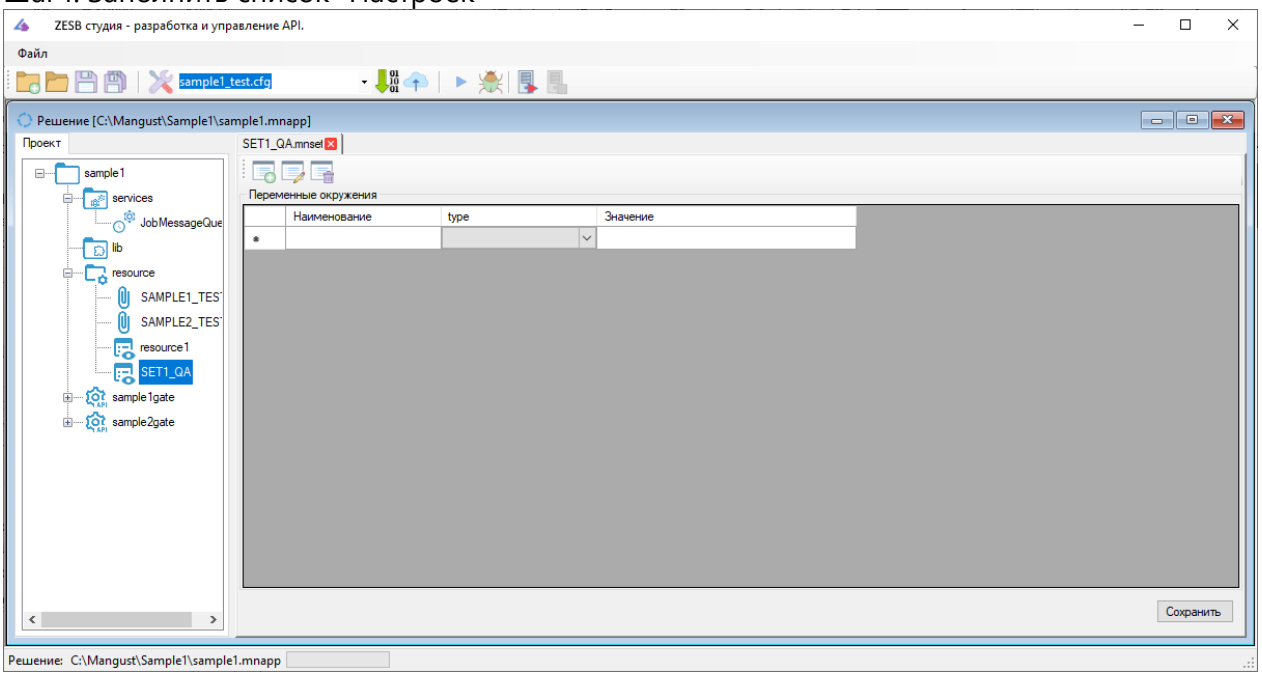
- Таблица настроек

Шаг1: Выбрать раздел "resource" и нажать правую клавишу мыши.
Шаг2: Выбрать пункт меню "Добавить настройки"



Шаг3: Заполнить поля и нажать копку сохранить

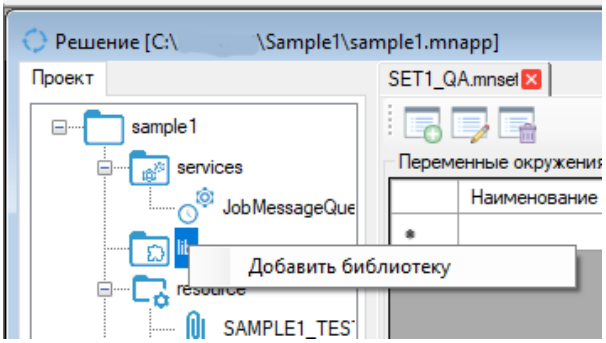
Шаг4: Заполнить список "Настроек"



Список настроек добавлен

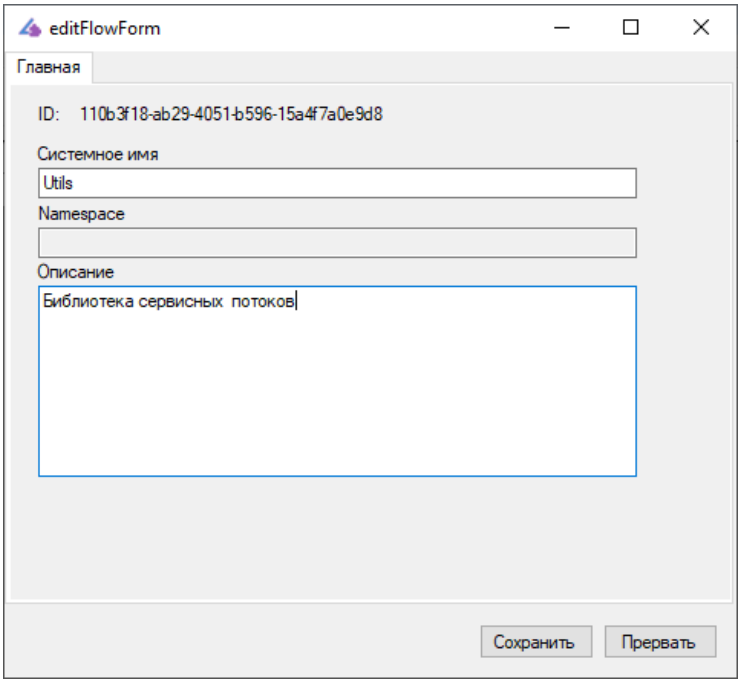
2.4.4. БИБЛИОТЕКА

Библиотеки необходимые для работы приложения реализованные на языке DSL (ZESB).
Для добавления библиотеки нужно выполнить следующие шаги:
Шаг1: Выбрать раздел "lib" и нажать правую клавишу мыши.
Шаг2: Выбрать пункт меню "Добавить библиотеку"



Шаг3: Заполнить поля следующей формы

- **Системное имя** - имя библиотеки на английском языке;
- **Описание** - краткое описание библиотеки к какой области относится;
- **Namespace** - **(зарезервировано);**

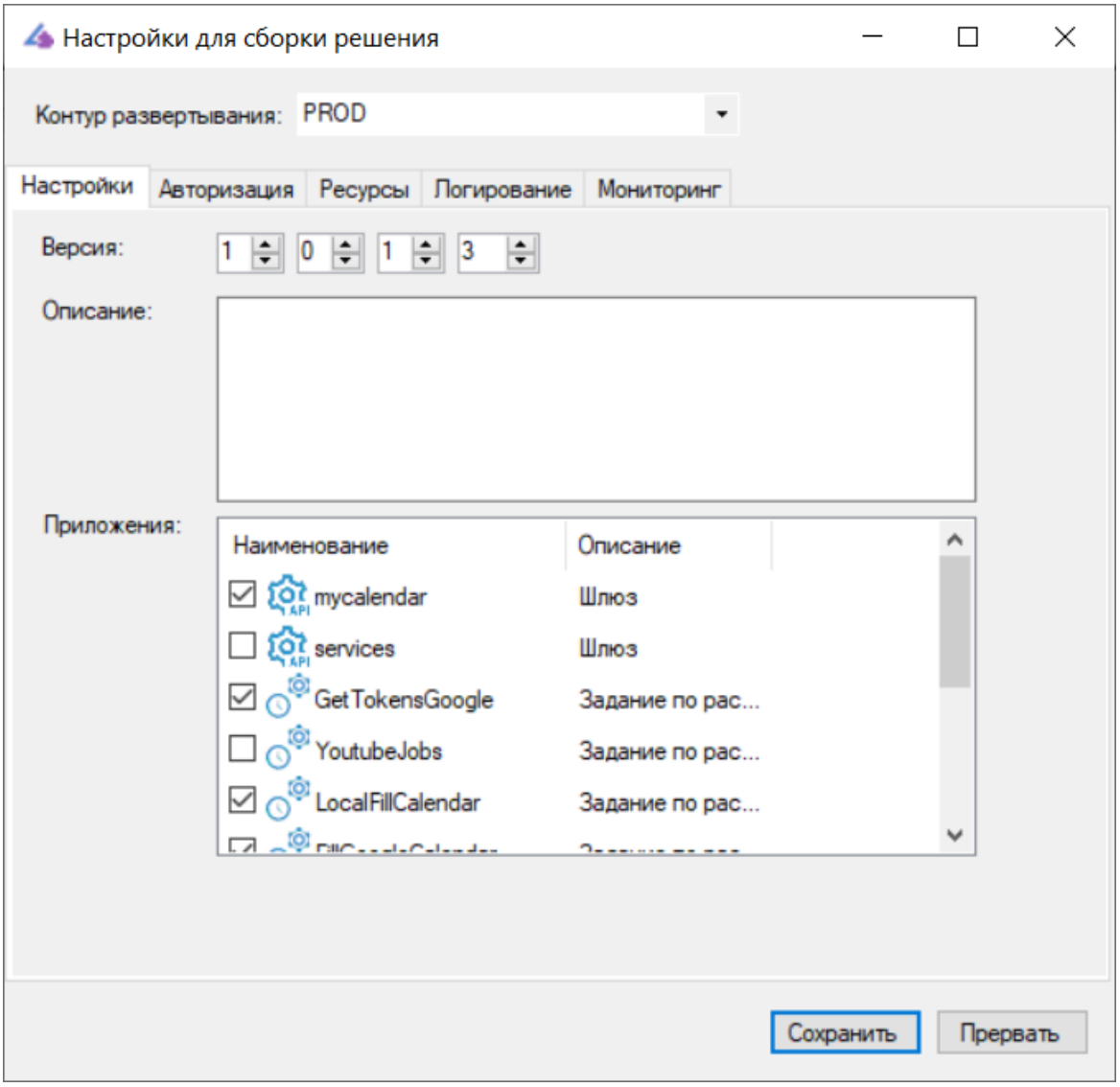


Шаг4: После завершения описания нажать кнопку сохранить.

2.5. ПОДГОТОВКА К РАЗВЕРТЫВАНИЮ

Особенную роль занимает процесс подготовка к развертыванию приложения. Процесс заточен под лучшие практики DevOps
используемые для развертывания приложений в docker контейнерах. Приложение поддерживает возможность одновременно
управлять несколькими конфигурациями одного и того же проекта для разных сред (Prod, Stage,Test,Dev)
Для того чтобы создать или изменить конфигурацию приложения, необходимо на рабочей панели кликнуть на кнопку 
После данного действия появиться следующая форма редактирования в которой нужно указать для какого контура делается конфигурация.
На форме представлены четыре закладки отвечающие за настройки приложения:

- *Настройки*
- *Авторизация*
- *Ресурсы*
- *Логирование*
- *Мониторинг*

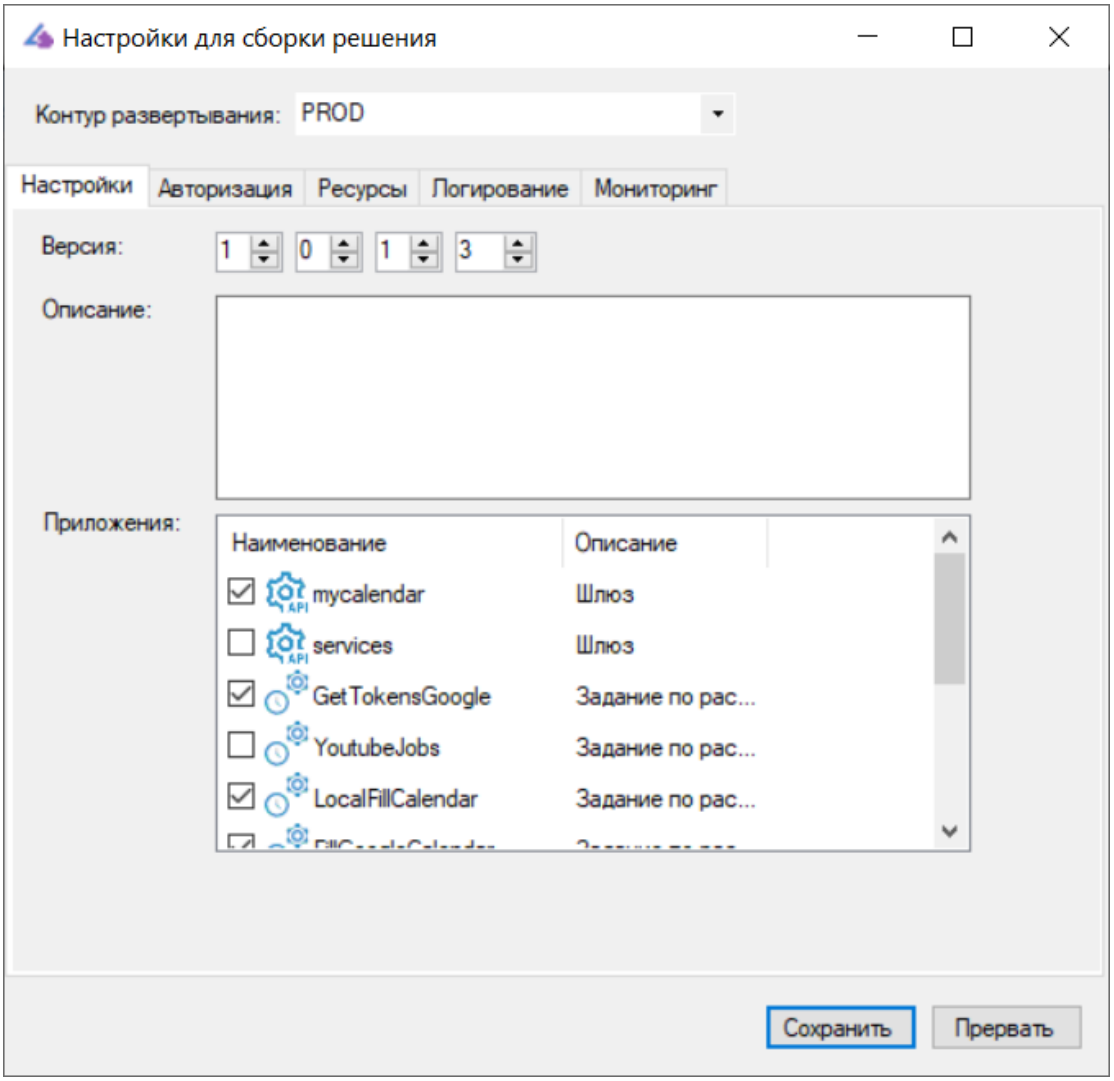


2.5.1. НАСТРОЙКИ

На главной странице формы "Настройки развертывания" необходимо заполнить следующие поля:

1. Контур развертывания - выпадающий список из следующих значений (PROD,STAGE,TEST,DEVELOP,NONE) которые определяют для какого контура будет собрано приложение.
2. Версия - является 4 -х значным числом определяет номер изменения в приложении
3. Описание - краткое описание сборки что в нее вошло нового
4. Приложения - список шлюзов, служебных заданий. Чекбокс - определяет войдут изменения в эту версию или нет.

Для того чтобы шлюз или служебный сервис попал в сборку надо отметить их в списке.

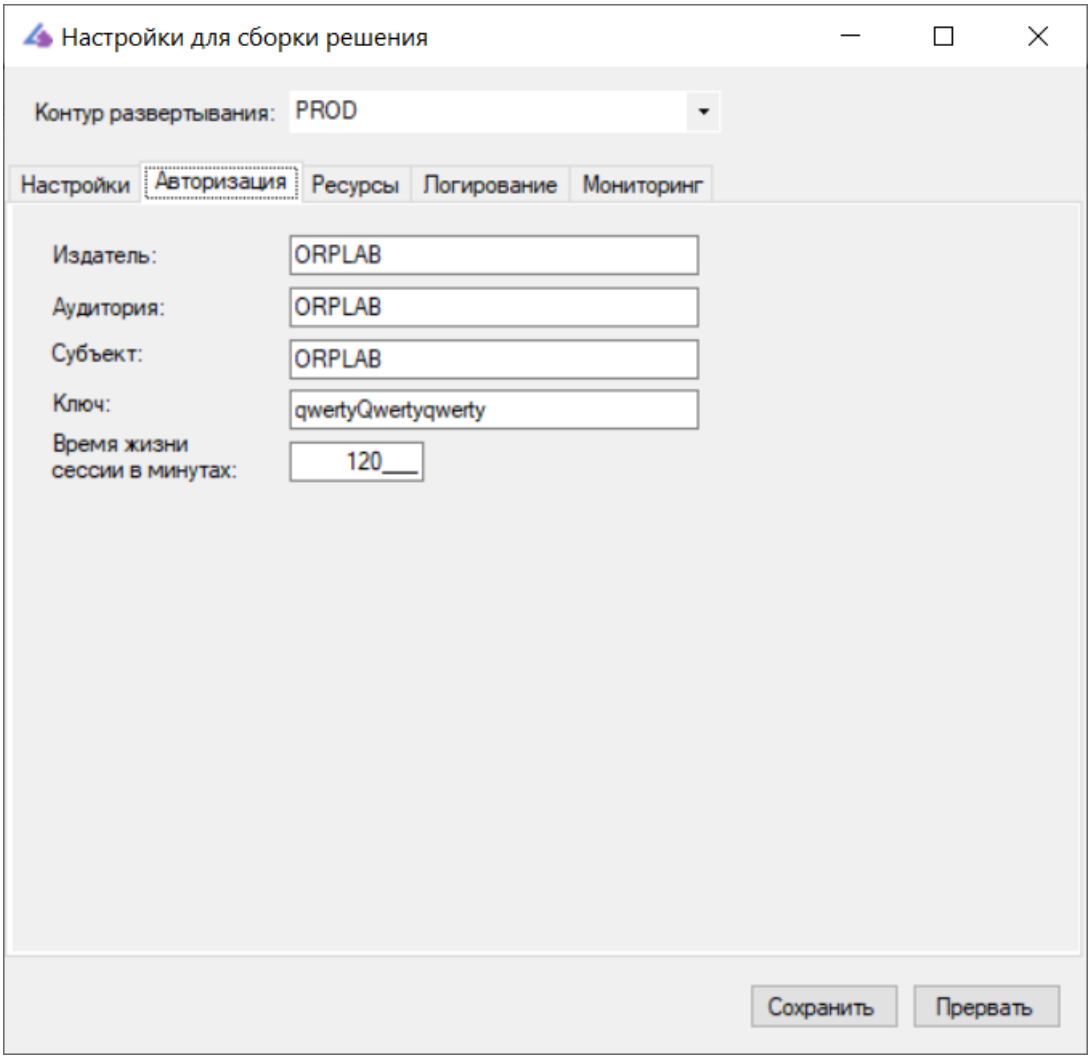


2.5.2. АВТОРИЗАЦИЯ

Закладка авторизация предназначена для работы с JWT токеном. В ней указываются необходимые атрибуты для формирования JWT токена.

Описание полей:

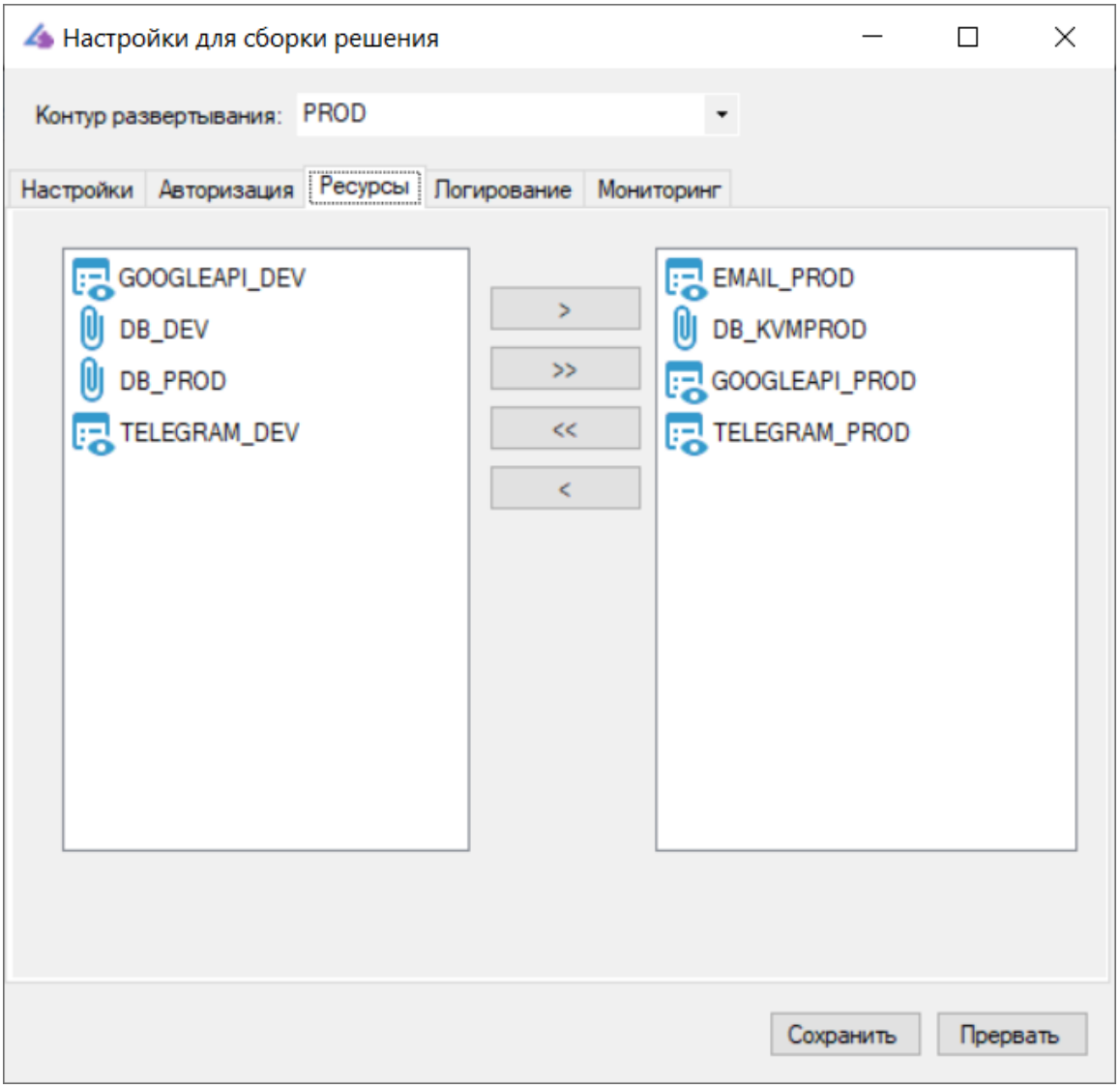
Издатель	Указывает кто выпустил JWT Token	**Не обязательное**
Аудитория	Что за аудитория работает с данным приложением	**Не обязательное**
Субъект	Тоже что и издатель	**Не обязательное**
Ключ	Необходим для формирования подписи в JWT токене. Длина > 16 символов	**Обязательное**
Версия жизни сессии в минутах	Время жизни JWT токена	**Обзательное**



2.5.3. РЕСУРСЫ

На данной закладке определяется какие ресурсы будут подключены к приложению в момент его сборки или отладки. В простом понимании какие константы будут доступны во время работы приложения (runtime). Также ресурсы должны быть привязаны к контуру развертывания. Смотри работу с ресурсами в проекте.
Форма разделена на два списка элементов. С левой стороны все доступные ресурсы для использования в приложении, справа те которые будут использованы в приложении после его сборки или в режиме отладки.
Для изменения состава ресурсов в списке слева и справа используются кнопки:

- **>** перенести отмеченный в правый список
- **>>** перенести все элементы из левого списка в правый список
- **<<** Убрать все элементы из правого списка в левый
- **<** Убрать отмеченный элемент из правого в левый

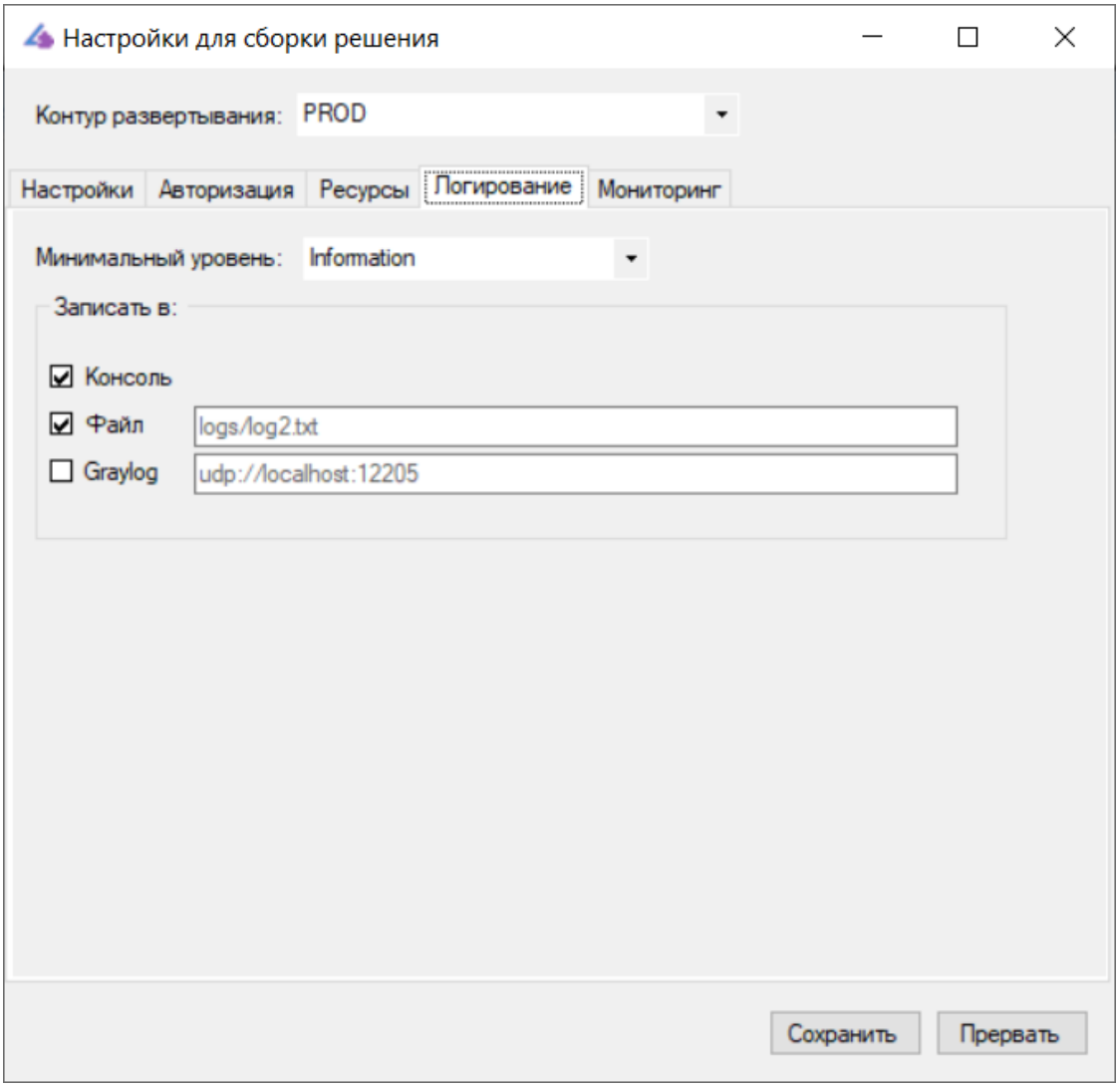


2.5.4. ЛОГИРОВАНИЕ

Параметры на панели логирования играют важную роль в настройке сбора и хранения логов приложения.

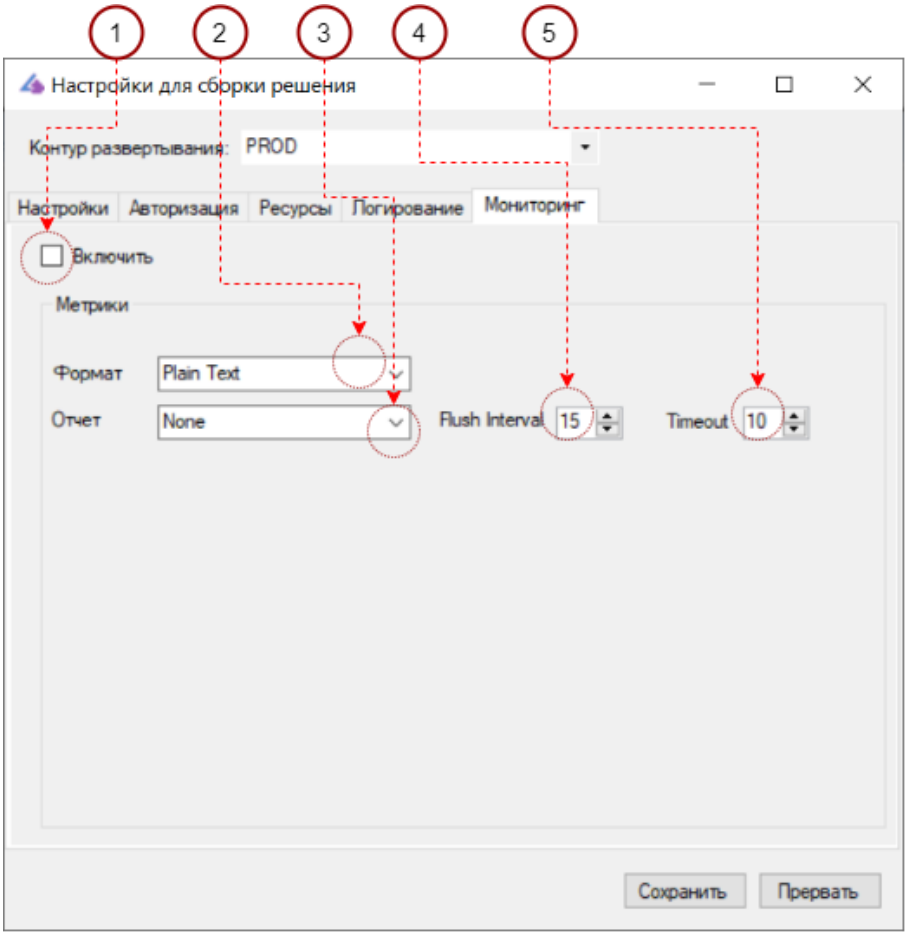
Минимальный уровень	Параметр определяет уровень сбора логов в приложении ○ Verbose - Самый широкий уровень логирования ○ Debug ○ Information - стандарт собирает логи с уровнем Information и выше ○ Warning - собирает (Warning, Error, Fatal) ○ Error- собирает (Error, Fatal) ○ Fatal- собирает (Fatal) ○ None - не собирать логи

Записать в	Параметр определяет куда записывать логируемую информацию в стандартную консоль,файл, внешний сервис Graylog. В случае Graylog необходимо развертывать в ландшафте приложение Graylog (www.graylog.com). Отметьте чекбоксы напротив имен куда нужно сохранять лог в приложении.
Консоль	Вывод в стандартную консоль input/output
Файл	Вывод в файл
Graylog	Отправка лога вовнешний сервис по протоколу udp в реальном времени



2.5.5. МОНИТОРИНГ

Параметры задаваемые в панели мониторинга позволяют снимать метрики как с самого приложения, так и с процессов запущенных внутри приложения.
Пример: Загрузка процессора, использование памяти, время исполнения того или иного потока, количество выполненных потоков.



- *1. Включить мониторинг*
  Активация данного чекбокса включает сбор метрик во всем приложении. После чего в приложении становятся доступны следующие страницы /metrics, /metrics-text.
  На которые выводится собранная информация из приложения.
- 2. Формат
  1. Plain Text - метрики выводятся в текстовом формате
  2. Json - метрики выводятся в формате Json
  3. Prometeus - метрики выводятся в формате для приложения Prometeus (система сбора метрик)
- 3. Отчет

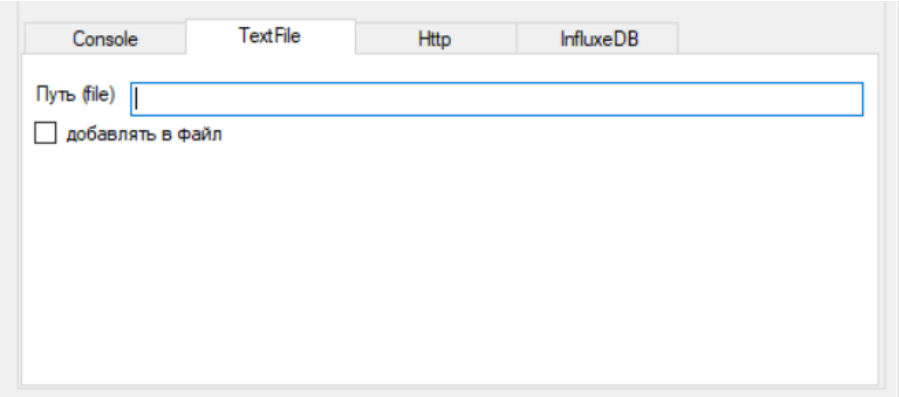
Основная цель данного пункта определить систему для сбора и агрегации метрик.

None Означает что собранные данные никуда не отправляются

Console Собранные данные отправляются в устройство Console с периодичностью установленной в параметре 4 (Flush Interval)

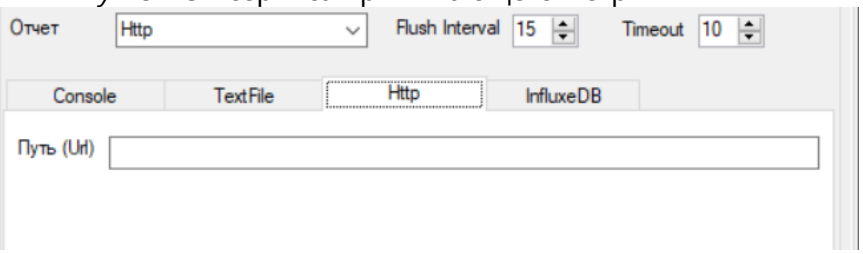
File Собранные метрики сохраняются в файл.
Параметры окна:

1. *Путь* - месторасположение файла
2. *Добавлять в файл* - не пересоздавать файл, а добавлять в конец созданного файла данные.



Http Отправка метрик в систему которая умеет собирать их по http.
Параметры окна:

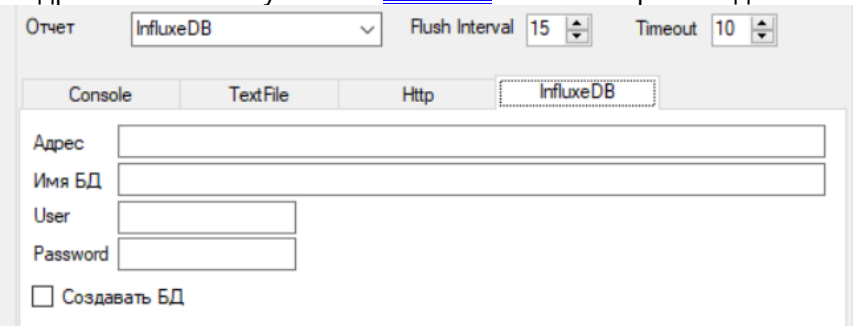
- *Путь* - Url сервиса принимающего метрики



In-fluxDB Отправка метрик в специализированную БД.

1. *Адрес* - место расположение БД ip адрес
2. *Имя БД* - название базы данных куда агрегировать
3. *User* - пользователь БД
4. *Password* - Пароль доступа пользователя
5. *Создавать БД* - нужно ли создавать БД , если БД нет то создать.

Подробнее можно узнать о InfluxDB на сайте производителя.



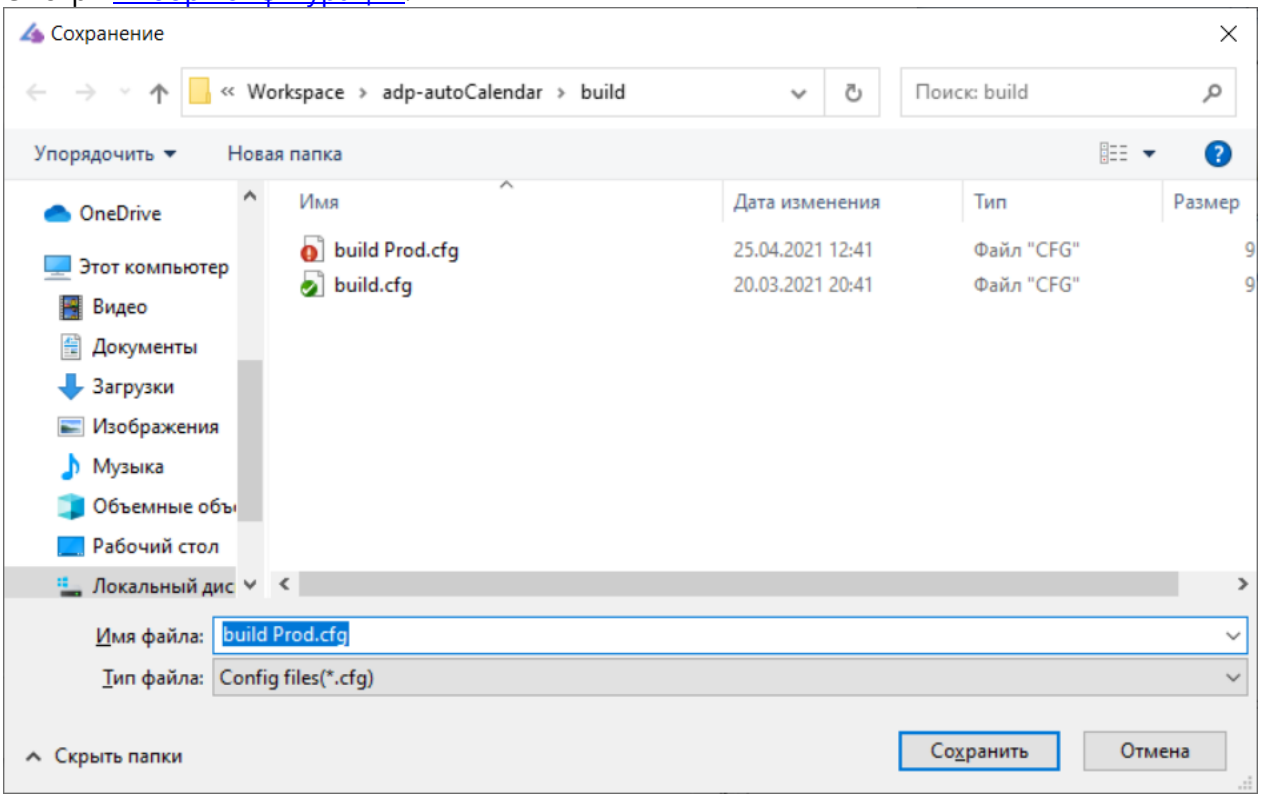
▼ 4,5 Временные интервалы

1. *FlushInterval* - интервал записи отчета в систему приемник
2. *Timeout* - время ожидания ответа от системы приемника

2.5.6. СОХРАНИТЬ

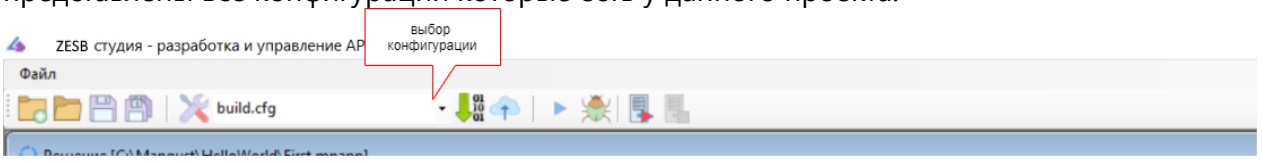
После того как было завершено заполнение всех разделов в форме редактирования и была нажата кнопка "Сохранить".

Появляется следующее окно сохранения в файл текущей конфигурации. Имя файла может быть любое, желательно
чтобы было понятно из имени файла что это за конфигурация и для чего. После сохранения на диск данная конфигурация
в проекте становиться активной и отображается в в панели инструментов главного окна.
Смотри выбор конфигурации.



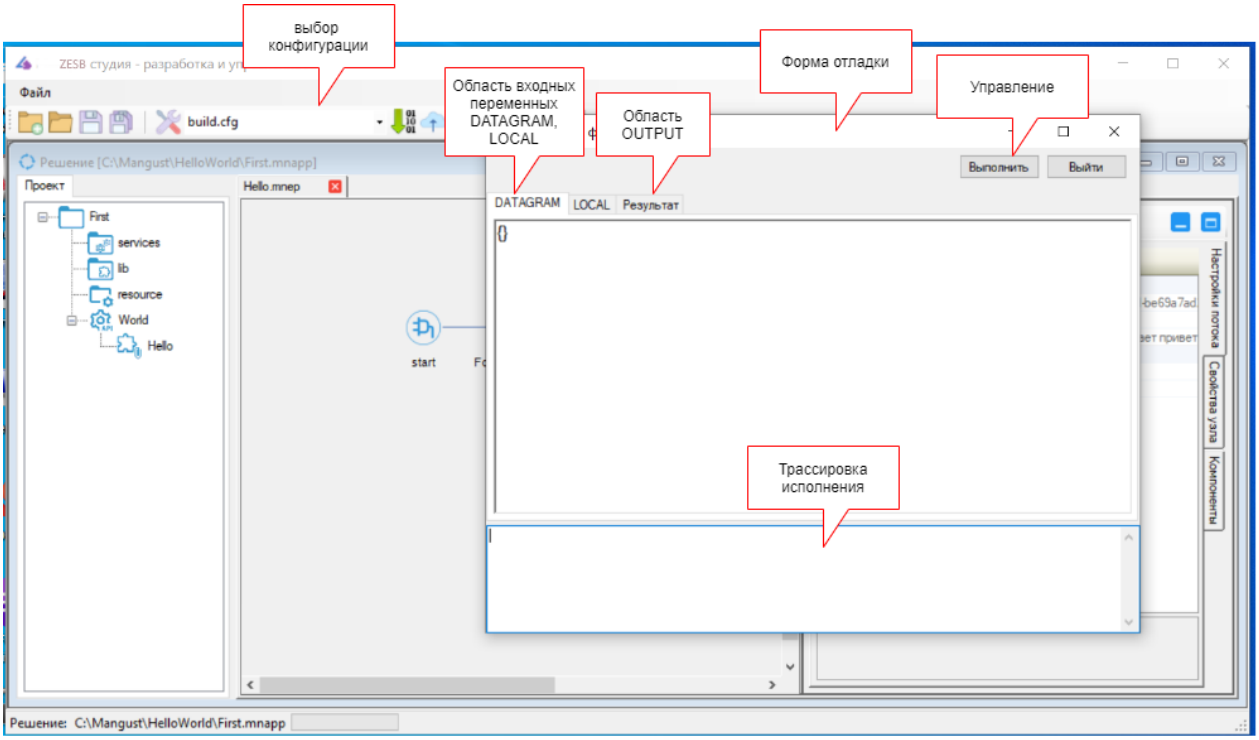
2.5.7. ВЫБРАТЬ КОНФИГУРАЦИЮ

Для выбора новой активной конфигурации, в панели инструментов присутствует выпадающий список выбора конфигурации. В элементе выпадающего списка
представлены все конфигурации которые есть у данного проекта.



2.6. ОТЛАДКА ПРИЛОЖЕНИЯ

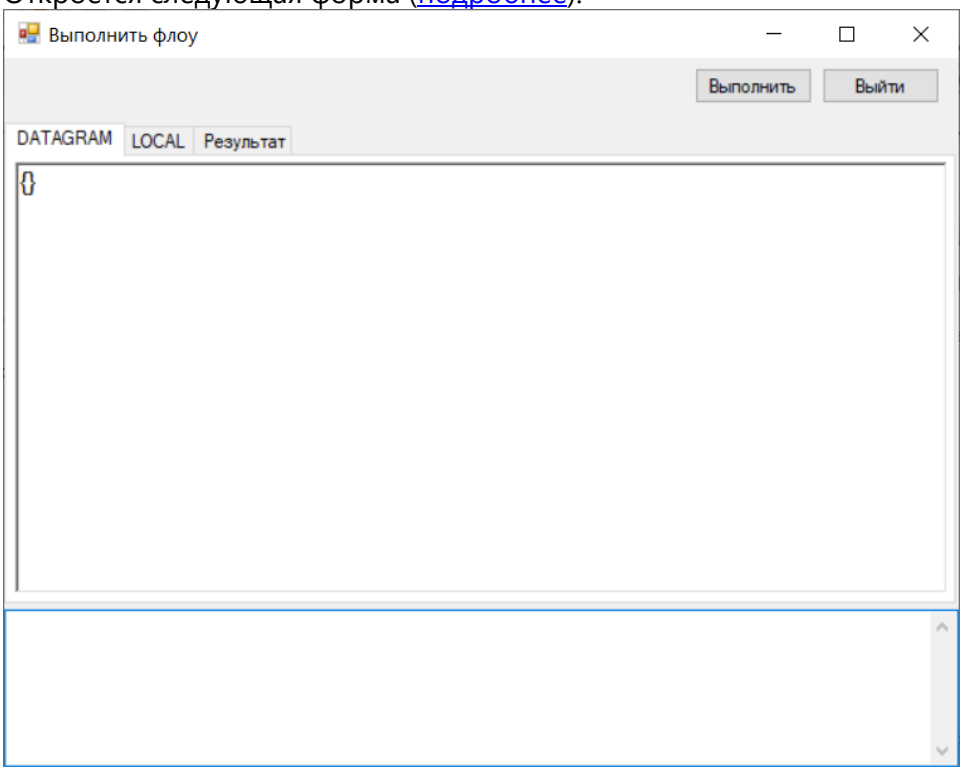
Как и любая low-code система - студия ZESB имеет механизм для отладки кода. Любой **поток управления** можно выполнить и отладить.



Для того чтобы выполнить отладку потока управления необходимо сделать следующее:

1. Поток управления сделать активным
2. Панели инструментов выбрать конфигурация
3. Панели инструментов кликнуть на кнопку

Откроется следующая форма (подробнее):

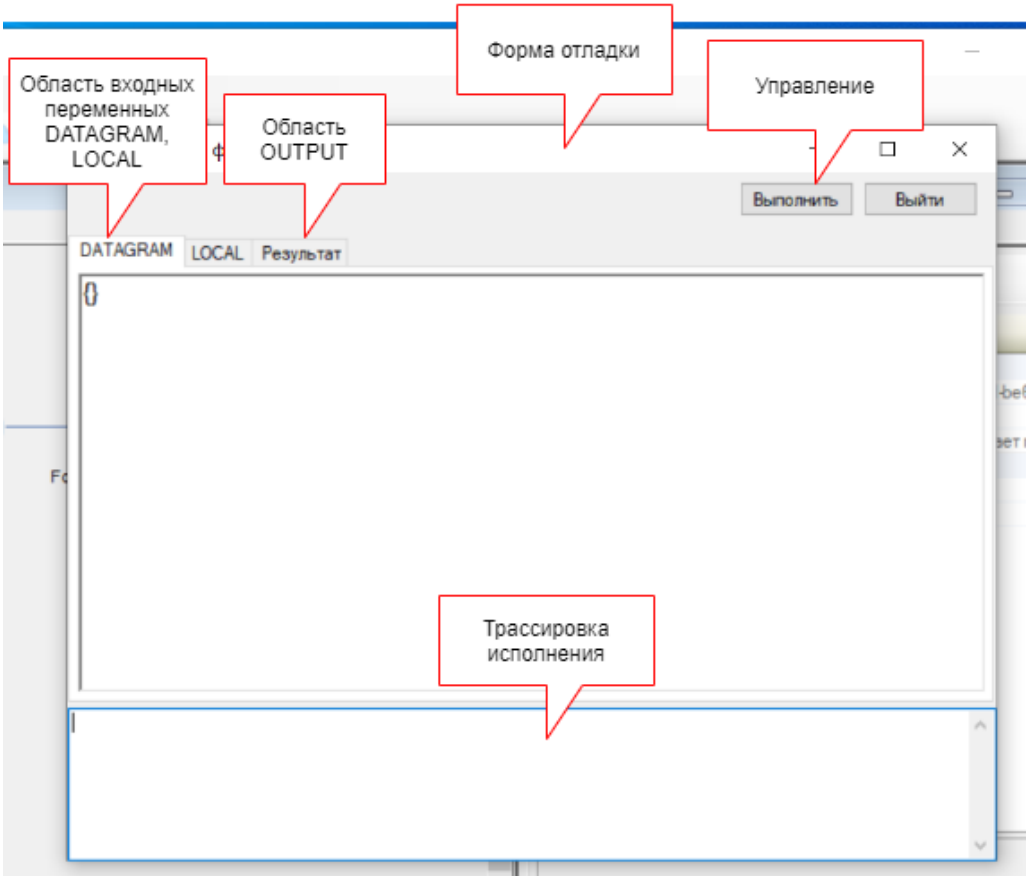


2.6.1. КАК ВЫПОЛНИТЬ ОТЛАДКУ

Что бы подготовить поток управления к тестированию необходимо задать параметры контекста.
В поля формы DATAGRAM, LOCAL параметры задаются в формате JSON и при исполнении потока управления
инициализирую области памяти в контексте данных потока.
Контекст данных потока управления делиться на несколько областей памяти. У каждой области памяти
есть своя задача.

- Области памяти
  - DATAGRAM - всегда содержит входящее сообщение для потока управления. Предназначен для тестирование rest сервисов.
  - LOCAL - локальная область памяти потока управления. Все компоненты потока управления видят содержимое данной области памяти
  - SETTINGS - константы из конфигурации приложения. Подключения к базам данных, очередям, настройки в виде списков ключ значение
  - OUTPUT - Область памяти для предназначенная для возврата результата.
- Область трассировки
  В поле трассировки выводиться все лог сообщения с уровнем логирования trace и выше. По содержимому лог сообщений можно понять успешно завершился поток управления иди произошла ошибка

После того как выше описанные поля были заполнены достаточно нажать на кнопку **"выполнить"**. И смотреть поля формы трассировка и результат.
В поле трассировка можно увидеть пошаговое исполнение потока управления, а в поле результат можно увидеть содержимое контекста данных
после завершения работы потока управления.



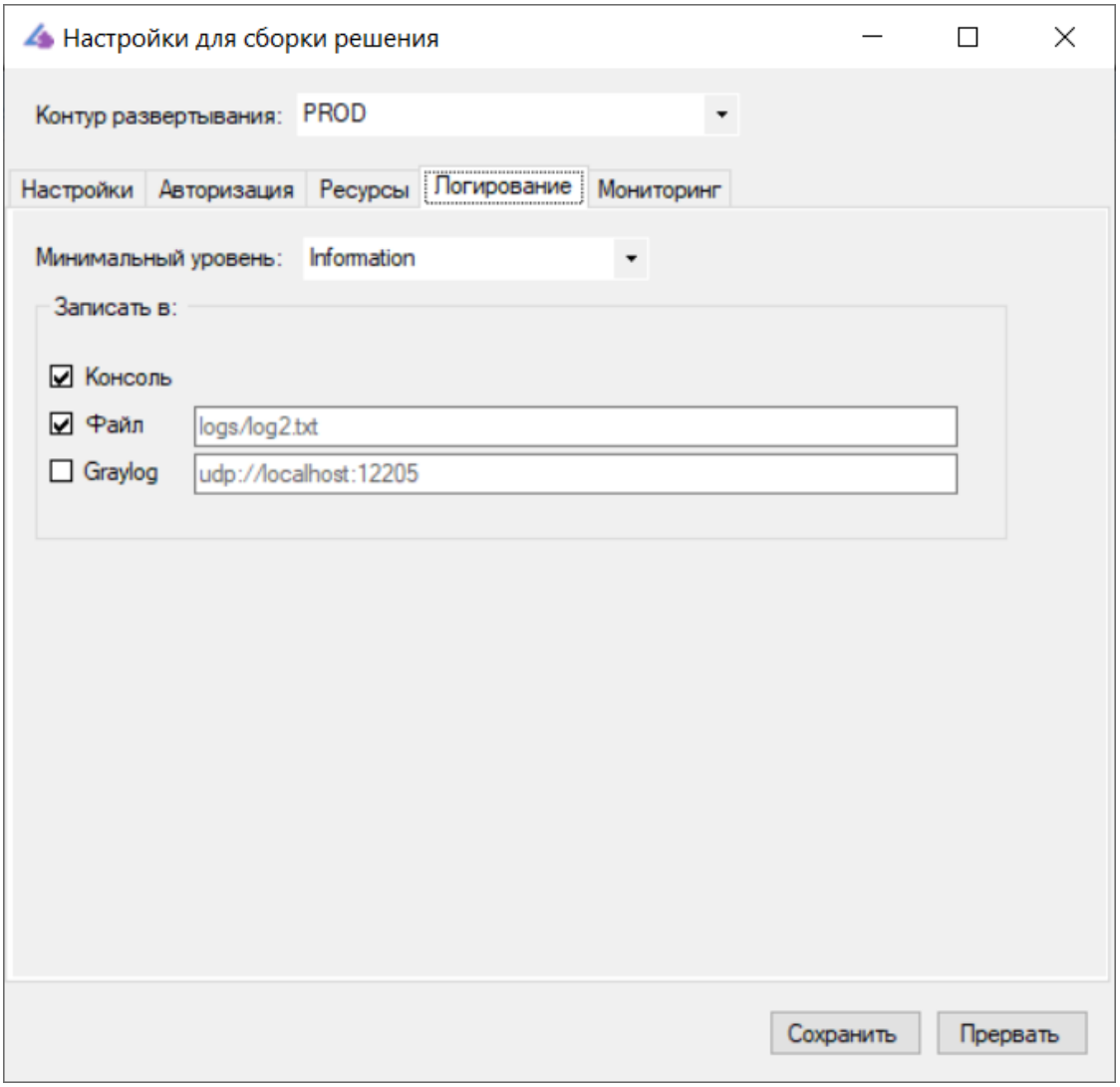
2.7. МОНИТОРИНГ, ЛОГИРОВАНИЕ

▼ Логирование

Параметры на панели логирования играют важную роль в настройке сбора и хранения логов приложения.

Минимальный уровень	Параметр определяет уровень сбора логов в приложении ○ Verbose - Самый широкий уровень логирования ○ Debug ○ Information - стандарт собирает логи с уровнем Information и выше ○ Warning - собирает (Warning, Error, Fatal) ○ Error- собирает (Error, Fatal) ○ Fatal- собирает (Fatal) ○ None - не собирать логи
Записать в	Параметр определяет куда записывать логируемую информацию в стандартную консоль,файл, внешний сервис Graylog. В случае Graylog необходимо развертывать в ландшафте приложение Graylog (www.graylog.com). Отметьте чекбоксы напротив имен куда нужно сохранять лог в приложении.
Консоль	Вывод в стандартную консоль input/output

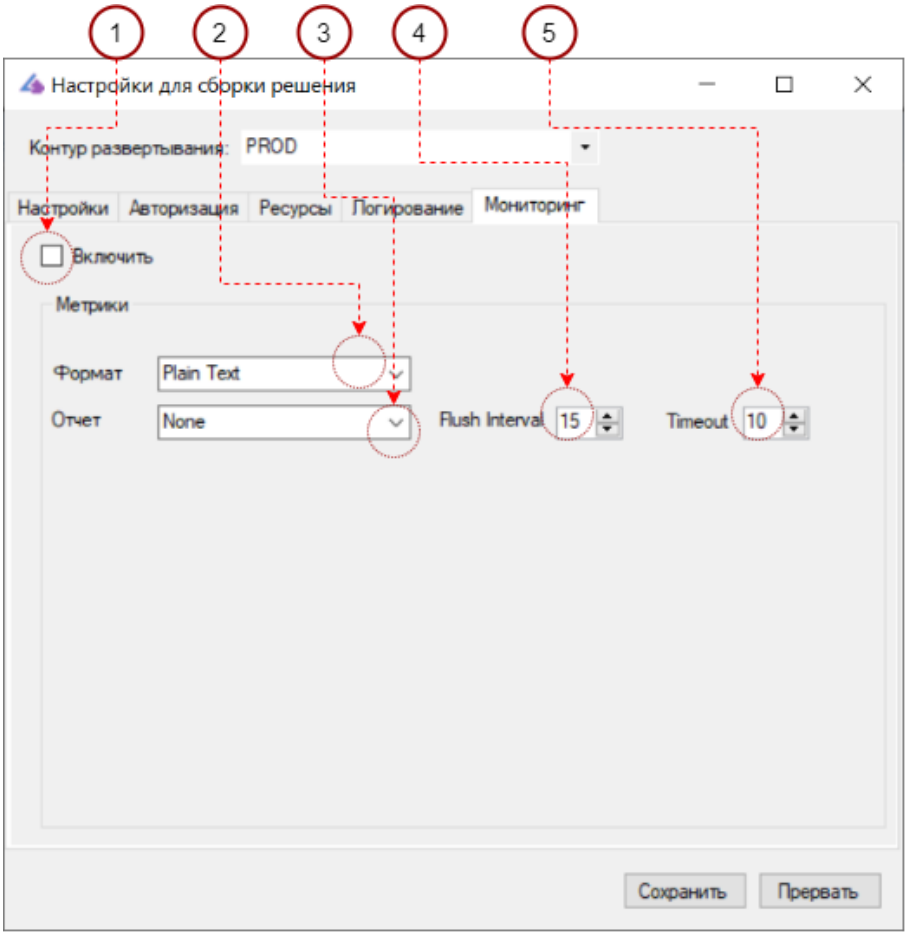
Файл	Вывод в файл
Graylog	Отправка лога вовнешний сервис по протоколу udp в реальном времени



▼ Мониторинг

Параметры задаваемые в панели мониторинга позволяют снимать метрики как с самого приложения, так и с процессов запущенных внутри приложения.

Пример: Загрузка процессора, использование памяти, время исполнения того или иного потока, количество выполненных потоков.



- *1. Включить мониторинг*
  Активация данного чекбокса включает сбор метрик во всем приложении. После чего в приложении становятся доступны следующие страницы /metrics, /metrics-text.
  На которые выводится собранная информация из приложения.
- 2. Формат
  1. Plain Text - метрики выводятся в текстовом формате
  2. Json - метрики выводятся в формате Json
  3. Prometeus - метрики выводятся в формате для приложения Prometeus (система сбора метрик)
- 3. Отчет

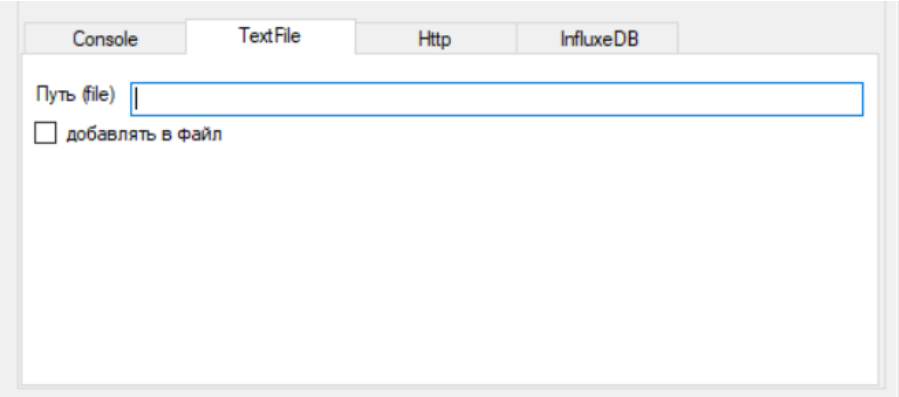
Основная цель данного пункта определить систему для сбора и агрегации метрик.

None Означает что собранные данные никуда не отправляются

Console Собранные данные отправляются в устройство Console с периодичностью установленной в параметре 4 (Flush Interval)

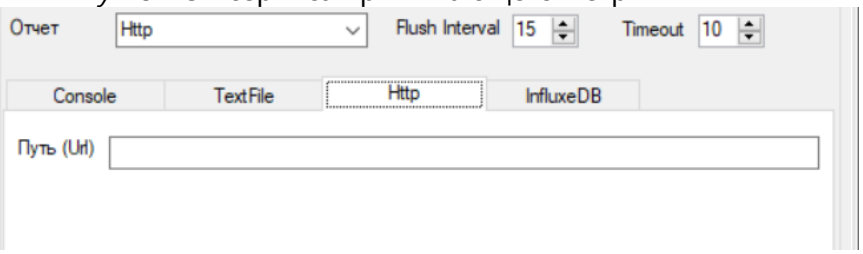
File Собранные метрики сохраняются в файл.
Параметры окна:

1. *Путь* - месторасположение файла
2. *Добавлять в файл* - не пересоздавать файл, а добавлять в конец созданного файла данные.



Http Отправка метрик в систему которая умеет собирать их по http.
Параметры окна:

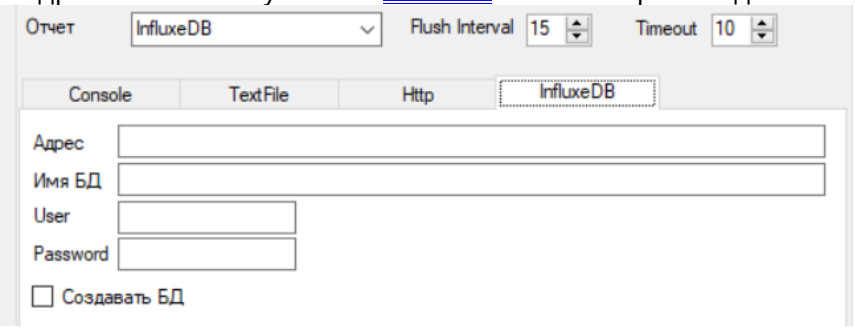
- *Путь* - Url сервиса принимающего метрики



In-fluxDB Отправка метрик в специализированную БД.

1. *Адрес* - место расположение БД ip адрес
2. *Имя БД* - название базы данных куда агрегировать
3. *User* - пользователь БД
4. *Password* - Пароль доступа пользователя
5. *Создавать БД* - нужно ли создавать БД , если БД нет то создать.

Подробнее можно узнать о InfluxDB на сайте производителя.



▼ 4,5 Временные интервалы

1. *FlushInterval* - интервал записи отчета в систему приемник
2. *Timeout* - время ожидания ответа от системы приемника

3. БЫСТРЫЙ СТАРТ

Enter topic text here.

4. DSL ZESB

Специализированный язык для предметной области связанной с разработкой API, микро-сервисов. Предназначен для быстрого ввода в эксплуатацию новых API предприятия.

4.1. ВВЕДЕНИЕ

Что такое предметно-ориентированные язык программирования "ZESB"?

Предметно-ориентированные язык (DSL) "ZESB" — это язык программирования с более высоким уровнем абстракции, который отражает
специфику решаемых с его помощью задач. Такой язык оперируют понятиями и правилами из определенной предметной области,
а именно интеграция ПО.

Чем предметно-ориентированный язык "ZESB" отличается от «настоящих» языков программирования?

Предметно-ориентированные языки обычно не такие сложные, как языки общего назначения, например Java, C или Ruby.
Как правило, предметно-ориентированные языки разрабатываются в сотрудничестве со специалистами в той сфере деятельности,
для которой проектируется язык. Зачастую для использования таких языков не требуется квалификация разработчика
и на них программируют люди, хорошо разбирающиеся в предметной области.

4.2. ТЕРМИНЫ

1	**DSL**	Предметно ориентированный язык (Domain Specific Language), направлен на реализацию задач данной конкретной предметной области.
2	**Gate**	Шлюз накоторый поступают запросы и маршрутизируются на нужные точки обработки
3	**Области данных**	Разделение области памяти на логические сегменты отвечающие за доступ и этапность обработки. (см. область данных)
4	**Нода, узел**	Атомарный элемент потока обработки данных .
5		

4.3. ОБЛАСТИ ДАННЫХ

DATAGRAM	это область данных для входящих запросов
LOCAL	локальная область переменных с которыми работает поток
OUTPUT	область переменных исходящих данных (получаемый результат)
ENV	область переменных окружения среды (настройки для корректной работы приложения)

4.4. JPATH

JPATH - синтаксис

A JSONPath expression specifies a path to an element (or a set of elements) in a JSON structure. Paths can use the dot notation:

```
$.store.book[0].title
```

or the bracket notation:

```
$['store']['book'][0]['title']
```

The leading `$` represents the root object or array and can be omitted. For example, `$.foo.bar` and `foo.bar` are the same, and so are `$[0].status` and `[0].status`.

Other syntax elements are described below.

Expression	Description
`$`	The root object or array.
`.property`	Selects the specified property in a parent object.
`['property']`	Selects the specified property in a parent object. Be sure to put single quotes around the property name. **Tip:** Use this notation if the property name contains special characters such as spaces, or begins with a character other than `A..Za..z_`.
`[n]`	Selects the *n*-th element from an array. Indexes are 0-based.
`[index1,index2,…]`	Selects array elements with the specified indexes. Returns a list.
`..property`	Recursive descent: Searches for the specified property name recursively and returns an array of all values with this property name. Always returns a list, even if just one property is found.

`*`	Wildcard selects all elements in an object or an array, regardless of their names or indexes. For example, `address.*` means all properties of the `address` object, and `book[*]` means all items of the `book` array.
`[start:end]` `[start:]`	Selects array elements from the *start* index and up to, but not including, *end* index. If *end* is omitted, selects all elements from *start* until the end of the array. Returns a list.
`[:n]`	Selects the first *n* elements of the array. Returns a list.
`[-n:]`	Selects the last *n* elements of the array. Returns a list.
`[?(expression)]`	Filter expression. Selects all elements in an object or array that match the specified filter. Returns a list.
`[(expression)]`	Script expressions can be used instead of explicit property names or indexes. An example is `[(@.length-1)]` which selects the last item in an array. Here, `length` refers to the length of the current array rather than a JSON field named `length`.
`@`	Used in filter expressions to refer to the current node being processed.

Notes:

- JSONPath expressions, including property names and values, are **case-sensitive**.
- Unlike XPath, JSONPath does not have operations for accessing parent or sibling nodes from the given node.

Filters are logical expressions used to filter arrays. An example of a JSONPath expression with a filter is
`$.store.book[?(@.price < 10)]`

where `@` represents the current array item or object being processed. Filters can also use `$` to refer to the properties outside of the current object:
`$.store.book[?(@.price < $.expensive)]`

An expression that specifies just a property name, such as `[?(@.isbn)]`, matches all items that have this property, regardless of the value.

Additionally, filters support the following operators:

Operator	Description
`==`	Equals to. `1` and `'1'` are considered equal. String values must be enclosed in single quotes (not double quotes): `[?(@.color=='red')]`.
`!=`	Not equal to. String values must be enclosed in single quotes.
`>`	Greater than.
`>=`	Greater than or equal to.
`<`	Less than.
`<=`	Less than or equal to.

`=~`	Match a JavaScript regular expression. For example, `[?(@.description =~ /cat.*/i)]` matches items whose description starts with *cat* (case-insensitive). **Note:** Not supported at locations that use ReadyAPI 1.1.
`!`	Use to negate a filter: `[?(!@.isbn)]` matches items that do not have the `isbn` property. **Note:** Not supported at locations that use ReadyAPI 1.1.
`&&`	Logical AND, used to combine multiple filter expressions: `[?(@.category=='fiction' && @.price < 10)]`
`||`	Logical OR, used to combine multiple filter expressions: `[?(@.category=='fiction' || @.price < 10)]` **Note:** Not supported at locations that use ReadyAPI 1.1.

4.5. ПРОТОКОЛ JRPC

JSON RPC - специальный протокол по взаимодействию с сервером. Подробнее можно ознакомиться на сайте с спецификацией данного протокола JSON RPC .

Примеры использования:
7 Examples
Syntax:

--> data sent to Server
<-- data sent to Client
rpc call with positional parameters:

--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}

--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}
<-- {"jsonrpc": "2.0", "result": -19, "id": 2}
rpc call with named parameters:

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend": 42}, "id": 3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend": 23}, "id": 4}
<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
a Notification:

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
--> {"jsonrpc": "2.0", "method": "foobar"}
rpc call of non-existent method:

--> {"jsonrpc": "2.0", "method": "foobar", "id": "1"}
<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": "1"}
rpc call with invalid JSON:

--> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id": null}
rpc call with invalid Request object:

--> {"jsonrpc": "2.0", "method": 1, "params": "bar"}
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
rpc call Batch, invalid JSON:

--> [
  {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
  {"jsonrpc": "2.0", "method"
]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id": null}
rpc call with an empty Array:
--> []
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
rpc call with an invalid Batch (but not empty):
--> [1]
<-- [
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
]
rpc call with invalid Batch:

--> [1,2,3]
<-- [
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
]
rpc call Batch:

--> [
        {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]},
        {"jsonrpc": "2.0", "method": "subtract", "params": [42,23], "id": "2"},
        {"foo": "boo"},
```

```
      {"jsonrpc": "2.0", "method": "foo.get", "params": {"name": "myself"}, "id": "5"},
      {"jsonrpc": "2.0", "method": "get_data", "id": "9"}
   ]
<-- [
      {"jsonrpc": "2.0", "result": 7, "id": "1"},
      {"jsonrpc": "2.0", "result": 19, "id": "2"},
      {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
      {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": "5"},
      {"jsonrpc": "2.0", "result": ["hello", 5], "id": "9"}
   ]
```

rpc call Batch (all notifications):

```
--> [
      {"jsonrpc": "2.0", "method": "notify_sum", "params": [1,2,4]},
      {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]}
   ]
<-- //Nothing is returned for all notification batches
```

4.6. КОМПОНЕНТЫ ЯЗЫКА

Основным строительным блоком потоков обработки в DSL "ZESB" , является - нода (это компонент который выполняет атомарную операцию). Каждая нода
представляет из себя черный ящик. У ноды есть как входные параметры, так и выходной параметр который сохраняется в локальны контекст данных.

Для добавления ноды на экран редактирования потока обработки. Необходимо выделить ноду на экране левым кликом мыши,
перевести курсор мыши в правую часть экрана, выбрать раздел компоненты. После чего на нужном компоненте сделать double-click
За выделенной нодой появится новый вставленный компонент.

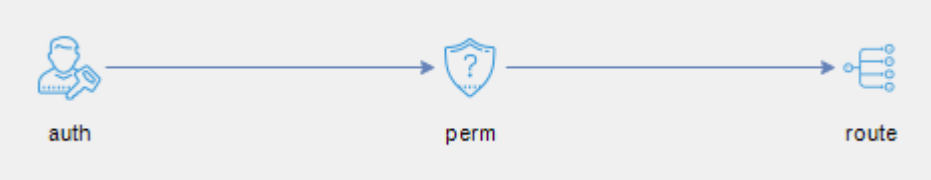
4.6.1. DSL ZESB COMPONENTS OVERVIEW

Основным строительным блоком потоков обработки в DSL "ZESB" , является - нода (это компонент который выполняет атомарную операцию). Каждая нода
представляет из себя черный ящик. У ноды есть как входные параметры, так и выходной параметр который сохраняется в локальны контекст данных.

Для добавления ноды на экран редактирования потока обработки. Необходимо выделить ноду на экране левым кликом мыши,
перевести курсор мыши в правую часть экрана, выбрать раздел компоненты. После чего на нужном компоненте сделать double-click
За выделенной нодой появится новый вставленный компонент.

4.6.2. ШЛЮЗ (GATE)

Сложно составной компнент, который отвечает за авторизация, проверку прав доступа и исполнение метода переданноо по протоколу JSON RPC.
Данный элемент состоит из трех взаимосвязанных компонентов:



Поддерживает следующие типы аутентификации(авторизации) Basic, JWT.

Алгоритм работы следующий:
1. Basic проверка пользователя и возврат JWT токена в заголовке http ответа.
2. Когда только JWT отправка запроса вида {"user":"test","password":"test123456"} на специализированный url указанный в компоненте и получение в ответ тела JWT токена.

4.6.2.1. АУТЕНТИФИКАЦИЯ

Проверка авторизации по JWT токену. Если в свойстве Method выбрать **NONE** проверка токена не производится и обработка запроса
будет передана следующему компоненту Permission(Авторизация).

Свойства:

Наименование	Значения
UrlAuthorization	**Тип значения**: **String** Задается точка входа (Url) для получения JWT токена. (Зарезервировано)
Method	**Тип значения**: **Enum** Варианты выбора значений свойства: 1. NONE - не проверять токен, выдает токен с ролью anonimus; 2. JWT - проверять токен.

4.6.2.2. АВТОРИЗАЦИЯ

Данный компонент отвечает за управление доступом к вызываемым методам системы. Проверку прав доступа.

Свойства:

Наименование	Значения

Verify	**Тип данных: Boolean**; Данное свойство отключает или включает проверку прав доступа к вызываемым методам по ролям. Значение **true** проверка доступа осуществляется. Сочетание свойств **Method** = **NONE** и **Verify** = **false** дает полный доступ ко всем потокам выполнения внутри шлюза.

4.6.2.3. МАРШРУТИЗАЦИЯ

Компонент "Управления точками доступа" - выполняет функцию вызова по имени необходимого потока выполнения (EndPointFlow). Обладает свойствами routes , которое позволяет переопределять маршруты.

Свойства:

Наименование	Значения
Routes	**Тип данных: List**; Свойства объекта: **id** — **Тип данных: String**; Уникальный id метода **name** — **Тип данных: String**; Задается имя метода маршрутизации **verify** — **Тип данных: Enum**; принимаемые значения (BY_ID,BY_NAME) по умолчанию BY_NAME

- Пример:

Запрос на endpoint шлюза:

1. Url адреса - https://localhost:5001/CrmData
2. Пример запроса {"jsonrpc": "2.0", "method": "ClientSave", "params": [42, 23], "id": 1}

Под номером один красным цветом отмечено имя шлюза, под номером два красным цветом отмечено имя потока исполнения.

4.6.3. РАЗДЕЛ SYSTEM

Базовые компоненты:

Наименование	Краткое описание
Variables	Работа с областями памяти
Trace	Трассировка данных

TryCatch	Перехват исключительной ситуации
Throw	Сгенерировать ошибку и остановить обработку потока

4.6.3.1. VARIABLES

Компонент управления переменными в памяти. Копирование переменной из одной области памяти в другую.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Mapping	Устанавливает соответствие из каких полей в какие поля копировать данные. Если поле не существует оно будет создано. **Тип данных: List**; Свойства объекта:

	Тип	Описание
	Enum	Перечисление областей переменных откуда брать данные.
	String	Строковое выражение JPATH начинается с $. Пример: $.JSON.result.code
	Enum	Перечисление из двух значений (**LOCAL, OUTPUT**) из области переменных.

	String	Строковое выражение пути куда сохранить перемен-ную перечисление идет через символ "."

4.6.3.2. TRACE

Компонент trace позволяет записать в лог любую переменную из памяти или набор таких переменных.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Level	**Тип данных: Enum**; Параметр определяет уровень логируемых данных ○ Verbose(Trace) - максимальный уровень логирования ○ Debug ○ Information ○ Warning ○ Error ○ Fatal - минимальный уровень логирования ○ None - лог не сохраняется
Variables	Список переменных памяти которые необходимо сохранить в процессе исполнения скрипта в лог файл. **Тип данных: List**; Свойства объекта:

	Тип	Описание
	String	Строковое выражение JPATH начинается с $. Пример:

			$.JSON.result.code

4.6.3.3. TRYCATCH

Компонент TryCatch позволяет перехватить исключительную ситуацию (ошибку) если она произошла ниже по потоку исполнения и произвести
альтернативную обработку исключительной ситуации.

4.6.3.4. THROW

Компонент Throw отвечает за генерацию в коде исключительной ситуации. Необходим для тех случаев когда необходимо
прервать обработку потока и выдать прикладную ошибку для данной ситуации.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
ErrorCode	Свой код ошибки
ErrorMessage	Свое сообщение об ошибке

4.6.4. РАЗДЕЛ TRANSFORMATION

Компоненты отвечающие за трансформацию данных:

Наименование	Краткое описание
TransformUtil	Базовые функции преобразования текстовых данных
TransformXSLT	Компонент xslt для преобразования xml по шаблонам
FormatString	Преобразование строки по шаблону
IronPython	Вызов Python Script

4.6.4.1. TRANSFORMUTIL

Компонент позволяет преобразовать данные из переменной From применяя следующие функции преобразования данных
JSON to XML, XML to JSON, URL ENCODE, URL DECODE, ENCODE BASE64, DECODE BASE64.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
From	Строковое выражение JPATH начинается с $. Пример: $.JSON.result.code

Direct	**Тип данных: Enum**; 1. JSON2XML 2. XML2JSON 3. ENCODEBASE64 4. DECODEBASE64 5. URLENCODE 6. URLDECODE 7. MD5HASH 8. NEWID - сгенерировать guid
Output	**Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти

4.6.4.2. TRANSFORMXSLT

Выполняет xslt преобразование над объектом xml.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
From	Переменная памяти откуда берется значение для преобразования Строковое выражение JPATH начинается с $. Пример: $.JSON.result.code
Direct	Выбор откуда загружать шаблон xslt из файла или из поля body. **Тип данных: Enum** Принимает следующие значения: 1. DATA - в данном случае шаблон берется из поля **Body**. 2. FILE - загружается из файла по пути указанному в поле **Path**
Body	Шаблон xslt сохраненный в переменную
Path	Путь до шаблона xslt на диске
Output	Переменная куда помещается преобразованное значение. **Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти

4.6.4.3. FORMATSTRING

Преобразование строки по шаблону. Для вставки значения переменной используется следующий шаблон фигурные кавычки и значение переменной из списка по имени .

Свойства:

<table>
<tr><th>Наименование</th><th>Значения</th></tr>
<tr><td>id</td><td>Уникальный идентификатор компонента, только чтение.</td></tr>
<tr><td>name</td><td>Название компонента</td></tr>
<tr><td>pattern</td><td>Шаблон форматирования строки
Пример:
Шаблон строки "Первый {test}, Второй {test2}"</td></tr>
<tr><td>Params</td><td>Устанавливает соответствие из каких полей в какие поля копировать данные. Если поле не существует оно будет создано.
Тип данных: List;
Свойства объекта:
<table>
<tr><th>Тип</th><th>Описание</th></tr>
<tr><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr>
<tr><td>String</td><td>Значение переменной взятое из локальной памяти с по мощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr>
</table></td></tr>
<tr><td>Output</td><td>Переменная куда помещается преобразованное значение.
Тип данных: String
Пример значения:
test.out_data //Переменная будет сохранена в локальную область памяти</td></tr>
</table>

4.6.4.4. IRONPYTHON

Компонент позволяющий исполнить скрипт на языке Python. Расширение DSL ZESB.

Свойства:

<table>
<tr><th>Наименование</th><th>Значения</th></tr>
<tr><td>id</td><td>Уникальный идентификатор компонента, только чтение.</td></tr>
<tr><td>name</td><td>Название компонента</td></tr>
<tr><td>Script</td><td>Тело программы на языке Python</td></tr>
<tr><td>Params</td><td>Устанавливает соответствие из каких полей в какие поля копировать данные. Если поле не существует оно будет создано.
Тип данных: List;
Свойства объекта:
<table>
<tr><th>Тип</th><th>Описание</th></tr>
<tr><td>String</td><td>Имя переменной для передачи в скрипт данных и получения из скрипта. Обязательное условие переменная должна быть объявлена в скрипте.</td></tr>
<tr><td>Enum</td><td>Boolean, Int, Decimal, DateTime, String, Blob</td></tr>
<tr><td>Enum</td><td>Направление передачи значения в метод на питон скрипте.
• Input
• Output
• InputOutput</td></tr>
</table>
</td></tr>
</table>

	String	Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную. Строковое выражение JPATH начинается с $. Пример: $.JSON.result.code или Просто текст
Output	Переменная куда помещается параметры с paramtype = (Output, InputOutput). **Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти	

4.6.5. РАЗДЕЛ MAIL

Компоненты отвечающие за отправку email сообщений:

Наименование	Краткое описание
SMTP	Отправка email сообщений по протоколу smtp

4.6.5.1. SMTP

Компонент отвечает за отправку почты по протоколу SMTP.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Server	Имя хоста или ip адрес SMTP сервера, согласно стандартам uri может указываться через : с портом. Пример: smtp.mail.ru:25
User	Имя пользователя Можно указать прямо в поле или через выражение JPATH.
Password	Пароль пользователя Можно указать прямо в поле или через выражение JPATH.
From	Адрес(а) от кого отправляется письмо
To	Адрес(а) кому отправляется письмо
Cc	Адрес(а) кому отправляется копия писем
EnableSsl	**Тип данных: Boolean** Включение поддержки защищенного протокола SSL
AttachmentSource	**Тип данных: Enum** 1. **DATAFROMCONTEXT -** содержимое и имя файла берется из контекста исполнения

	2. **FILENAME -** в переменной указывается полный путь до файла на диске
Attachments	Выражение JPATH которое возвращает либо список имен файлов, либо список структур представления файла в памяти.
IsBodyHtml	**Тип данных: Boolean** Включение представления тела письма в виде HTML
Subject	Заголовок письма. Можно указать прямо в поле или через выражение JPATH
Body	Тело письма.Можно указать прямо в поле или через выражение JPATH
IsCollection	Зарезервировано
Output	Переменная куда возвращается результат отправки письма **Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти

4.6.6. РАЗДЕЛ WEB

Компоненты отвечающие за отправку данных по протоколу http:

Наименование	Краткое описание
HTTP	Отправка запроса по протоколу http
REST	Отправка запроса по протоколу rest
JRPC	Отправка запроса по протоколу jrpc

4.6.6.1. HTTP

Компонент для отправки запроса по http протоколу.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Url	Ссылка на адрес сервиса .
Method	Устанавливает необходимый метод для получения ответа по адресу Url. Описание доступных методов (Methods) **Тип данных: Enum**;
ResponseParse	Разобрать ответ от сервера. **Тип данных: Enum**; • NONE - не разбирать • JSON - ожидается ответ в формате JSON • XML - ожидается ответ в формате XML
Headers	Устанавливает http заголовки. **Тип данных: List**;

<table>
<tr><td></td><td>Свойства объекта:<table><tr><th></th><th>Тип</th><th>Описание</th></tr><tr><td></td><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr><tr><td></td><td>String</td><td>Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr></table></td></tr>
<tr><td>Params</td><td>Устанавливает параметры запроса.
Тип данных: List;
Свойства объекта:<table><tr><th></th><th>Тип</th><th>Описание</th></tr><tr><td></td><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr><tr><td></td><td>String</td><td>Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr></table></td></tr>
</table>

Body	Тело запроса. Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.
Output	Результат выполнения запроса. **Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти

4.6.6.2. REST

Компонент для отправки запроса по http протоколу с использованием паттерна Rest.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Url	Ссылка на адрес сервиса .
Method	Устанавливает необходимый метод для получения ответа по адресу Url. Описание доступных методов (Methods) **Тип данных: Enum**;
Headers	Устанавливает http заголовки. **Тип данных: List**; Свойства объекта:

	Тип	Описание
	String	Имя переменной для подстановки в шаблон
	String	Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную. Строковое выражение JPATH начинается с $. Пример: $.JSON.result.code или Просто текст

<table>
<tr><td>Params</td><td>Устанавливает параметры запроса.
Тип данных: List;
Свойства объекта:<table><tr><th></th><th>Тип</th><th>Описание</th></tr><tr><td></td><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr><tr><td></td><td>String</td><td>Значение переменной взятое из локальной памяти с по
мощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr></table></td></tr>
<tr><td>Body</td><td>Тело запроса. Значение переменной взятое из локальной памяти с по-
мощью JPATH или введенное вручную.</td></tr>
<tr><td>Output</td><td>Результат выполнения запроса.
Тип данных: String
Пример значения:
test.out_data //Переменная будет сохранена в локальную область памяти</td></tr>
</table>

4.6.6.3. JRPC

Компонент для отправки запроса по JRPC протоколу.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Url	Ссылка на адрес сервиса .
RpcMethod	Имя метода на стороне сервера. **Тип данных: String**;
AuthType	Тип аутентификации, авторизации

<table>
<tr><td></td><td>Тип данных: Enum;
• NONE - запрос идет без авторизации
• Basic - передается логин пароль в момент вызова метода
• JWT - формируется специальный токен и проверяется на стороне сервера</td></tr>
<tr><td>Login</td><td>Имя пользователя</td></tr>
<tr><td>Password</td><td>Пароль пользователя</td></tr>
<tr><td>Token</td><td>Передается JWT токен, полученный при авторизации.</td></tr>
<tr><td>Headers</td><td>Устанавливает http заголовки.
Тип данных: List;
Свойства объекта:
<table>
<tr><td></td><td>Тип</td><td>Описание</td></tr>
<tr><td></td><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr>
<tr><td></td><td>String</td><td>Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr>
</table></td></tr>
<tr><td>Params</td><td>Параметры запроса.
Тип данных: List;
Свойства объекта:
<table>
<tr><td></td><td>Тип</td><td>Описание</td></tr>
</table></td></tr>
</table>

		String	Имя переменной для подстановки в шаблон
		String	Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную. Строковое выражение JPATH начинается с $. Пример: $.JSON.result.code или Просто текст
ParamsBody	Тело запроса. Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.		
Output	Результат выполнения запроса. **Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти		

4.6.7. РАЗДЕЛ DATABASE

Компоненты отвечающие за работу с источниками данных:

Наименование	Краткое описание
SQL	Исполнение запроса на стороне сервера.
CRUD	Компонент формирования и исполнения операций (insert, delete,update, select) на стороне сервера.
Memtable	Таблица в памяти для меж поточного обмена данными

4.6.7.1. SQL

Компонент работы с базами данных поддерживаются следующие БД: (oracle, postgresSQL, sqlite, mssql)

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
DataSource	Строка соединения с базой данных или ссылка на ресурс объявленный в проекте.
CommandText	Текст SQL запроса **Тип данных: String**;
CommandType	Тип выполнения команды **Тип данных: Enum**;

<table>
<tr><td></td><td>

QUERY - возвращает результат выполнения запроса в виде RowSet
EXECUTENONQUERY - возвращает только статус исполнения запроса
SCALAR - возвращает только одно скалярное значение

</td></tr>
<tr><td>Params</td><td>
Параметры запроса.
Тип данных: List;
Свойства объекта:
<table>
<tr><th></th><th>Тип</th><th>Описание</th></tr>
<tr><td></td><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr>
<tr><td></td><td>Enum</td><td>

Boolean
Int
Decimal
DateTime
String
Blob

</td></tr>
<tr><td></td><td>Enum</td><td>Задает направление передачи значения в запросе

INPUT - только вход
OUTPUT - только выход
INPUTOUTPUT - вход и выход

</td></tr>
<tr><td></td><td>String</td><td>Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr>
</table>
</td></tr>
</table>

StopProcess	Остановка исполнения процесса целиком если в режиме QUERY установлено значение true и Rowset пустой. **Тип данных: Boolean;**
Output	Результат выполнения запроса. **Тип данных: String** **Пример значения:** test.out_data //Переменная будет сохранена в локальную область памяти

4.6.7.2. CRUD

Зарезервировано

4.6.7.3. MEMTABLE

Зарезервиравано

4.6.8. РАЗДЕЛ MESSAGE QUEUE

Компоненты отвечающие за обращение источники данных типа очереди:

Наименование	Краткое описание
MQueue	Работа с очередями по протоколу amqp
RabbitMQ	Работа с брокером сообщений Rabbit MQ

4.6.8.1. MQUEUE

Зарезервировано для поддержки протокола AMQP

4.6.8.2. RABBITMQ

Компонент для работы с очередями Rabbit MQ.

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
DataSource	Строка соединения с брокером сообщений или ссылка на ресурс объявленный в проекте.
CommandType	Тип операции **Тип данных: Enum**; • GET - Получить сообщение из очереди • PUT - Поместить сообщение в очередь
Queue	Имя очереди
Exchange	Точка обмена
	Параметры запроса.

<table>
<tr><td>Headers</td><td>Тип данных: List;
Свойства объекта:<table><tr><td></td><td>Тип</td><td>Описание</td></tr><tr><td></td><td>String</td><td>Имя переменной для подстановки в шаблон</td></tr><tr><td></td><td>String</td><td>Значение переменной взятое из локальной памяти с помощью JPATH или введенное вручную.
Строковое выражение JPATH начинается с $.
Пример:
$.JSON.result.code или Просто текст</td></tr></table></td></tr>
<tr><td>Body</td><td>Тело сообщения
параметр обязателен только при выборе CommandType = GET
Можно значение передавать с помощью JPATH</td></tr>
<tr><td>MessageType</td><td>Тип сообщения. Устанавливает специальный заголовок. Необязательно
Можно значение передавать с помощью JPATH</td></tr>
<tr><td>MessageId</td><td>Уникальный идентификатор сообщения. (Необязательно)
Можно значение передавать с помощью JPATH</td></tr>
<tr><td>CorrelationId</td><td>Уникальный идентификатор соответствия сообщения (Необязательно)
Можно значение передавать с помощью JPATH</td></tr>
<tr><td>WhenDataNull Cancel</td><td>Если при получении сообщения пусто не выполнять поток дальше
Тип данных: Boolean;</td></tr>
<tr><td>Output</td><td>Результат выполнения запроса.
Тип данных: String
Пример значения:
test.out_data //Переменная будет сохранена в локальную область памяти</td></tr>
</table>

4.6.9. РАЗДЕЛ ROUTERS

Компоненты отвечающие за маршрутизацию обработки данных:

Наименование	Краткое описание

Router	Компонент маршрутизации аналог switch в языке программирования c#
Parallel	Компонент за параллельную обработку потока
Neuro Network	Нейро сеть для обработки данных (зарезервировано)

4.6.9.1. ROUTER

Компонент разветвитель принцип действия аналогичен такой конструкции как if -else if

Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Routes	Указывает на какой узел нужно переключить поток исполнения если условие истина. Если все условия оказались ложны то процесс исполнения завершается. **Тип данных: List**; Свойства объекта:

	Тип	Описание
	String	Имя следующего узла
	String	Значение переменной взятое из локальной памяти с по мощью JPATH или введенное вручную.

4.6.9.1.1. EXPRESSION

Функции и выражения используемые в условиях.
Переменная из локального контекста обязательно берется в фигурные скобки **{$.id}**

Имена переменных

Если имя переменной содержит какой-либо из этих специальных символов ~ ( ) # \ / = > < + - * % & | ^ ' " [ ], необходимо заключить имя столбца в квадратные скобки.
Если имя переменной содержит правую скобку или обратную косую черту, пропустите ее с помощью обратной косой черты (\] или \\).

Литералы

Строковые значения заключены в одинарные кавычки. Если строка содержит одинарную кавычку, то кавычка должна быть удвоена.
Пример:

```
{$.name} = 'John'      // string value
{$.name} = John ''A'''  // string with single quotes "John 'A'"
```

Числовые значения не заключены в символы.

```
{$.Year} = 2008        // integer value
{$.Price} = 1199.9     // float value
```

Значения дата заключены в символы # #.

```
{$.Date} = #12/31/2008#"       // date value (time is 00:00:00)
{$.Date} = #2008-12-31#"       // also this format is supported
{$.Date} = #12/31/2008 16:44:58#" // date and time value
```

Операторы сравнения

Равно, не равно, меньше, больше операторы используются для включения только значений, которые подходят для выражения сравнения.
Вы можете использовать эти операторы = < > < = > >= .

```
{$.Num} = 10          // number is equal to 10
{$.Date} < #1/1/2008#    // date is less than 1/1/2008
{$.Name} <> 'John'       // string is not equal to 'John'
{$.Name} >= 'Jo'         // string comparison
```

Оператор IN используется для включения только значений из списка. Оператор можно использовать для всех типов данных,
таких как числа или строки.

```
{$.Id} IN (1, 2, 3)                    // integer values
{$.Price} IN (1.0, 9.9, 11.5)          // float values
{$.Name} IN ('John', 'Jim', 'Tom')     // string values
{$.Date} IN (#12/31/2008#, #1/1/2009#) // date time values

{$.Id} NOT IN (1, 2, 3)  // values not from the list
```

Оператор LIKE используется для включения только тех значений, которые соответствуют шаблону с подстановочными знаками. Подстановочный знак - это *или %, он может быть в начале шаблона '*value', в конце 'value*'или в обоих случаях '*value*'. Подстановочный знак в середине патерна 'va*lue'не допускается .

```
{$.Name} LIKE 'j*'        // values that start with 'j'
{$.Name} LIKE '%jo%'       // values that contain 'jo'
{$.Name} NOT LIKE 'j*'    // values that don't start with 'j'
```

Если шаблон в предложении LIKE содержит какие-либо из этих специальных символов * % [ ], эти символы должны быть заключены в скобки , [ ]подобные этим [*], или .[%][[][]]

```
{$.Name} LIKE '[*]*'        // значения, начинающиеся с '*'
{$.Name} LIKE '[[]*'        // значения, начинающиеся с '['
```

Логические операторы

Булевы операторы AND используются для объединения выражений OR. Оператор NOT имеет приоритет над оператором AND
и имеет приоритет над оператором OR.

```
// оператор И имеет приоритет над оператором ИЛИ, нужны скобки
{$.City} = 'Tokyo' AND ({$.Age} < 20 OR {$.Age} > 60)
// следующие примеры делают то же самое
{$.City} <> 'Tokyo' AND {$.City} <> 'Paris'
NOT {$.City} = 'Tokyo' AND NOT {$.City} = 'Paris'
NOT ({$.City} = 'Tokyo' OR {$.City} = 'Paris')
{$.City} NOT IN ('Tokyo', 'Paris')
```

Функции

Также поддерживаются следующие функции. Подробное описание можно найти здесь .

CONVERT – преобразует конкретное выражение в указанный тип .NET Framework
LEN - получает длину строки
ISNULL – проверяет выражение и либо возвращает проверенное выражение, либо значение замены
IIF – получает одно из двух значений в зависимости от результата логического выражения
TRIM - удаляет все начальные и конечные пробелы, такие как \r, \n, \t, , '
SUBSTRING – получает подстроку заданной длины, начиная с указанной точки строки

4.6.9.1.1.1. ФУНКЦИИ

CONVERT

Описание	Преобразует конкретное выражение в указанный тип .NET Framework.
Синтаксис	**CONVERT(`expression`, `type`)**

Аргументы	`expression`-- Выражение для преобразования. `type`-- Тип .NET, в который будет преобразовано значение.

```
Пример: Convert({$.total}, 'System.Int32')
```

LEN

Описание	Получает длину строки
Синтаксис	**LEN(`expression`)**
Аргументы	`expression`-- выражение.

```
Пример: Len({$.ItemName})
```

ISNULL

Описание	Проверяет выражение и либо возвращает проверенное выражение, либо замещающее значение.
Синтаксис	**`ISNULL`(`expression`, `replacementvalue`)**
Аргументы	`expression`-- Выражение для проверки. `replacementvalue`-- Если выражение `null`,возвращается `replacementvalue`.

```
Пример: IsNull({$.price}, -1)
```

IIF

Описание	Получает одно из двух значений в зависимости от результата логического выражения.
Синтаксис	**IIF(`expr`, `truepart`, `falsepart`)**
Аргументы	`expr`-- Выражение для оценки. `truepart`-- Возвращаемое значение, если выражение истинно. `falsepart`-- Возвращаемое значение, если выражение ложно.

```
Пример: IIF({$.total}>1000, 'дорого', 'дорого')
```

TRIM

Описание	Удаляет все начальные и конечные пробелы, такие как \r, \n, \t, ' '
Синтаксис	**TRIM (`expression`)**
Аргументы	`expression`-- Выражение для обрезки.

SUBSTRING

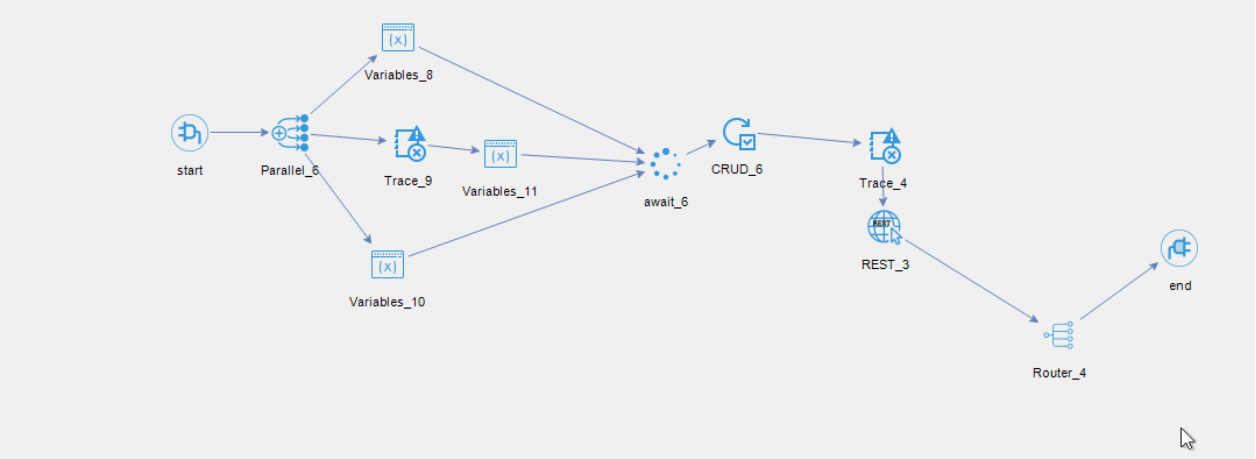
Описание	Получает подстроку указанной длины, начиная с указанной точки строки.
Синтаксис	**SUBSTRING(`expression`, `start`, `length`)**
Аргументы	`expression`-- Исходная строка для под-строки. `start`-- Целое число, указывающее, где начинается подстрока. `length`-- Целое число, указывающее длину подстроки.

```
Пример: SUBSTRING({$.phone}, 7, 8)
```

4.6.9.2. PARALLEL

Компонент создан для реализации параллельных вычислений. Выглядит точно также как Router за одним исключением все
ветви выполняются параллельно. И состоит он из двух узлов - узел разветвитель и узел синхронизации.
При этом каждая ветвь выполняется в своем защищенном контексте.
Свойства:

Наименование	Значения
id	Уникальный идентификатор компонента, только чтение.
name	Название компонента
Output	Переменная куда помещаются параллельные вычисления после синхро-низации. **Тип данных: String** **Пример значения:** var.out_data //Переменная будет сохранена в локальную область памяти



4.6.9.3. NEURO NETWORK

Зарезервировано.

KEYWORD INDEX

Элементы указателя не найдены.