БИПЛЕКС-BPLEX

Руководство программиста

Содержание

[1. Аннотация 4](#_Toc69730468)

[1.1. Назначение документа 4](#_Toc69730469)

[1.2. Краткое изложение основной части документа 4](#_Toc69730470)

[1.2.1.Требования к квалификации системного программиста 4](#_Toc69730471)

[2. Общие сведения о программе 5](#_Toc69730472)

[2.1. Назначение программы 5](#_Toc69730473)

[2.2. Функции программы 5](#_Toc69730474)

[2.3. Сведения о технических и программных средствах, обеспечивающих выполнение
 данной программы 5](#_Toc69730475)

[2.3.1. Характеристики оборудования и конфигурация программного обеспечения,
 необходимые для выполнения сборки исходного кода 5](#_Toc69730476)

[2.3.2. Характеристики оборудования и конфигурация программного обеспечения,
 необходимые для функционирования программы 7](#_Toc69730477)

[3. Структура программы 9](#_Toc69730478)

[3.1. Сведения о структуре программы 9](#_Toc69730479)

[3.2. Сведения о составных частях программы 10](#_Toc69730480)

[3.3. Сведения о связях между составными частями программы 11](#_Toc69730481)

[3.4. Сведения о связях с другими программами 11](#_Toc69730482)

[4. Настройка программы 12](#_Toc69730483)

[4.1. Установка Системы 12](#_Toc69730484)

[4.2. Процесс запуска программного обеспечения 12](#_Toc69730485)

[5. Проверка программы 14](#_Toc69730486)

[5.1. Описание способов проверки, позволяющих дать общее заключение о
 работоспособности программы 14](#_Toc69730487)

[5.2. Контрольные примеры 14](#_Toc69730488)

[5.3. Результаты проверки программы 14](#_Toc69730489)

[6. Сообщения системному программисту 15](#_Toc69730490)

[6.1. Тексты сообщений, выдаваемых в ходе настройки программы 15](#_Toc69730491)

[6.2. Тексты сообщений, выдаваемых в ходе проверки программы 15](#_Toc69730492)

[6.3. Тексты сообщений, выдаваемых в ходе выполнения программы 15](#_Toc69730493)

[Список обозначений и сокращений 16](#_Toc69730494)

[Приложение 1. Public-API: документация по интеграции BPLEX 19](#_Toc69730495)

[Registry – реестр описания процессов 19](#_Toc69730496)

[Execution – логика, касающаяся исполнения расчета 20](#_Toc69730497)

[Monitoring – средства мониторинга инфраструктуры 21](#_Toc69730498)

[History – просмотр исторической информации 22](#_Toc69730499)

[Schedule – обеспечение запуска по расписанию 23](#_Toc69730500)

[Сгенерированный API 24](#_Toc69730501)

[Methods 24](#_Toc69730502)

[Execution 24](#_Toc69730503)

[DELETE /execution/process 24](#_Toc69730504)

[GET /execution/process/output 24](#_Toc69730505)

[POST /execution/process 25](#_Toc69730506)

[GET /execution/process/status 25](#_Toc69730507)

[History 26](#_Toc69730508)

[GET /history/process/input 26](#_Toc69730509)

[GET /history/process/log 26](#_Toc69730510)

[GET /history/process/output 27](#_Toc69730511)

[GET /history/process/status 27](#_Toc69730512)

[GET /history/processes 28](#_Toc69730513)

[GET /history/stage 29](#_Toc69730514)

[GET /history/stage/input 29](#_Toc69730515)

[GET /history/stage/log 30](#_Toc69730516)

[GET /history/stage/output 30](#_Toc69730517)

[GET /history/stage/status 30](#_Toc69730518)

[Monitoring 31](#_Toc69730519)

[GET /monitoring/process 31](#_Toc69730520)

[GET /monitoring/service 32](#_Toc69730521)

[Registry 33](#_Toc69730522)

[DELETE /registry/process 33](#_Toc69730523)

[GET /registry/process 33](#_Toc69730524)

[POST /registry/process 34](#_Toc69730525)

[GET /registry/process/{uuid} 34](#_Toc69730526)

[DELETE /registry/service 35](#_Toc69730527)

[GET /registry/service 35](#_Toc69730528)

[POST /registry/service 36](#_Toc69730529)

[Schedule 36](#_Toc69730530)

[DELETE /schedule 36](#_Toc69730531)

[POST /schedule 37](#_Toc69730532)

[GET /schedule/{uuid} 37](#_Toc69730533)

[GET /schedules 38](#_Toc69730534)

[Models 38](#_Toc69730535)

[Date 38](#_Toc69730536)

[InputStream 38](#_Toc69730537)

[ListServiceInfo 38](#_Toc69730538)

[ListStageDescription 38](#_Toc69730539)

[ListString 38](#_Toc69730540)

[MultipartBody 38](#_Toc69730541)

[ProcessDTO 39](#_Toc69730542)

[ProcessInfo 39](#_Toc69730543)

[ProcessStatus 39](#_Toc69730544)

[ScheduleInfo 39](#_Toc69730545)

[ServiceInfo 39](#_Toc69730546)

[ServiceInfoList 39](#_Toc69730547)

[StageDescription 39](#_Toc69730548)

[StageType 39](#_Toc69730549)

[UUIDList 39](#_Toc69730550)

# Аннотация

## Назначение документа

В настоящем документе представлены сведения для установки, настройки на условия конкретного применения, проверки и обеспечения функционирования комплекса программ «БИПЛЕКС-BPLEX» (далее – Система, BPLEX).

## Краткое изложение основной части документа

Настоящий документ включает описание:

1. назначения и структуры Системы;
2. действий по установке Системы;
3. действий по настройке Системы;
4. требований к техническим и программным средствам, необходимым для работы Системы;
5. описание выводимых в процессе установки и работы Системы текстовых сообщений.

### Требования к квалификации системного программиста

Для решения задач, указанных в настоящем руководстве, системному программисту необходимо обладать:

* знаниями основ информационной безопасности;
* пониманием архитектуры Системы;
* знанием основ веб-технологий;
* навыками администрирования, в том числе резервирования и восстановления баз данных (БД), системы управления базами данных (СУБД) PostgreSQL, MS SQL, Mongo;
* навыками администрирования операционных систем RedHat, CentOS;
* навыками сборки и установки обновлений общесистемного программного обеспечения;
* навыками использования веб-браузеров (установка подключений, доступ к вебсайтам, навигация, формы и другие типовые интерактивные элементы).

# Общие сведения о программе

## Назначение программы

Комплекс программного обеспечения «БИПЛЕКС-BPLEX» предназначен для использования в системах, ключевые возможности которых лежат в области поиска оптимальных вариантов и принятия к реализации максимально эффективных сценариев.

## Функции программы

BPLEX является системой управляемого исполнения математических моделей, позволяющей выполнять настраиваемые последовательности операций, в том числе математических моделей, в заданной последовательности в распределенной среде вычислений. Набор гибких методов ПО обеспечивает:

* использование как идемпотентных математических моделей (с идентичными по типу наборами данных на входе и выходе), так и трансформационных (с разными по типу наборами данных);
* помимо основных данных, математические модели могут принимать на вход наборы ограничений, параметров и формировать на выходе опциональные наборы данных.

Система оперирует вспомогательными механизмами: модулями загрузки, выгрузки, преобразования данных, условий, условных и безусловных циклов. При создании моделей возможно связывать информационные потоки между ними (с учетом типов данных), а также настраивать параметры работы конкретных модулей.

Создание и настройка последовательностей осуществляется с помощью наглядного web-интерфейса в пользовательском интерфейсе системы.

Система открыта к добавлению в нее новых моделей, разработанных в соответствии с SDK.

BPLEX реализует ролевую модель доступа к ее использованию и управлению на принципах Explicit Deny.

## Сведения о технических и программных средствах, обеспечивающих выполнение данной программы

### Характеристики оборудования и конфигурация программного обеспечения, необходимые для выполнения сборки исходного кода

Сборка исходного кода производится на персональном компьютере, с техническими характеристиками не ниже указанных в таблице (Таблица 1).

Таблица 1. Минимальные технические характеристики для персонального компьютера, необходимого для сборки исходного кода

| Наименование | Характеристики |
| --- | --- |
| Тип процессора | i686 |
| Частота процессора | 2Ghz |
| Количество ядер | 2 |
| Объем кэша L3 | 256K |
| Объем кэша L2 | 6144K |
| Чипсет | Intel B75 |
| Тип оперативной памяти | DDR3 |
| Объем оперативной памяти | 8Gb |
| Тип накопителя | HDD |
| Объем накопителя | 500Gb |
| Оптический привод | - |
| Тип видеокарты | встроенная |
| Видеопамять | 128Mb |
| Количество портов PS/2 | 1 |
| Количество портов USB | 2 |
| COM порт | - |
| Количество разъемов VGA | 1 |
| Разъем RJ-45 | 1 |
| Мощность блока питания | 450W |

Сборка исходного кода может производиться на персональном компьютере под управлением операционной системы (ОС), версия которой не ниже, чем указана в таблице (Таблица 2).

Таблица 2. Минимальные требования к ОС для сборки исходного кода

| Наименование ОС | Версия |
| --- | --- |
| RedHat Linux (RHEL) | 7.3 |
| MS Windows | Выше Windows 7 |

Таблица 3. Утилиты, необходимые для успешной сборки исходного кода

| Наименование утилиты | Версия |
| --- | --- |
| Quarkus | 13.0  |
| Gitlab CI | 8.13 |
| Java development kit | 8u40 и выше |
| Docker  | 1.17 и выше |
| Kafka  | 2.0.0 |

### Характеристики оборудования и конфигурация программного обеспечения, необходимые для функционирования программы

Комплекс технических средств, обеспечивающих работу BPLEX, включает (Таблица 4):

Таблица 4. Требования к комплексу технических средств

| Узел | Технические характеристики | Общесистемное программное обеспечение – операционная система (ОС) | Специальное программное обеспечение |
| --- | --- | --- | --- |
| ПК оператора | Процессор: Intel Pentium/Celeron 1800 МГц и выше;ОЗУ: не менее 1 Гб;HDD: 1 Гб и более | ОС: Windows XP (SP3) и выше, Linux, MAC OS, Android v6 и выше | Веб-браузер:Google Chrome |
| Управляющий сервер виртуальной среды (оркестрация контейнеров) | Kubernetes Master | Процессор: 2 х Intel/AMD x64;ОЗУ: не менее 4 Gb;HDD: не менее 32 Gb | ОС:Linux семейства CentOS 7.4 | Kubernetes  |
| Kubernetes Slave | Процессор: 16 х Intel/AMD x64;ОЗУ: не менее 64 Gb;HDD: не менее 64 Gb | ОС:Linux семейства CentOS 7.4 |
| Сервер приложений виртуальной среды | Процессор: 4х Intel/AMD x64;ОЗУ: не менее 16 Gb;HDD: не менее 64 Gb | ОС:Linux семейства CentOS 7.4 | Kubernetes |
| Сервер БД | Процессор: 16 х Intel/AMD x64;ОЗУ: не менее 32 Gb;HDD: не менее 200 Gb | ОС:Linux семейства CentOS 7.4 | СУБД:PostgreSQL, MS SQL, Mongo |
| Сервер обмена сообщениями (основная шина) | Процессор: 8 х Intel/AMD x64;ОЗУ: не менее 8 Gb;HDD: не менее 40 Gb | ОС:Linux семейства CentOS 7.4 | Kafka |
| Сервер журналирования (логов) | Процессор: 4 х Intel/AMD x64;ОЗУ: не менее 8 Gb;HDD: не менее 200 Gb | ОС:Linux семейства CentOS 7.4 | Elastic Search (сбор), Kibana (отображение) |
| Сервер мониторинга (логов) | Процессор: 4 х Intel/AMD x64;ОЗУ: не менее 8 Gb;HDD: не менее 200 Gb | ОС:Linux семейства CentOS 7.4 | Prometeus (сбор), Graphana (отображение) |

Дополнительной специализированное ПО:

* Контроль версий – Gitlab/Gitlab Registry;
* Права доступа – Basic/Kerberos/LDAP;
* Контейнеры – Docker;
* Среда проектирования и программные языки – Quarkus, Java, Python.

# Структура программы

## Сведения о структуре программы

Обобщенная архитектура системы, использующей BPLEX, представлена ниже на рисунке (Рисунок 1).



Рисунок 1. Архитектура системы, использующей BPLEX

Комплекс программ BPLEX имеет архитектуру, представленную на рисунке ниже (Рисунок 2).

****

**Рисунок 2. Диаграмма компонент ПО «БИПЛЕКС-BPLEX»**

## Сведения о составных частях программы

Программные элементы структуры Системы представлены в таблице (Таблица 5).

Таблица 6. Структура Системы

| **Наименование** | **Описание** |
| --- | --- |
| UI | Публичный и административный интерфейсы системы |
| API Gateway | Интерфейс взаимодействия с BPLEX для UI и внешних систем |
| Validator | Служба проверки консистентности модели  |
| Storage | Подсистема хранения моделей, служебной информации, промежуточных результатов вычислений, состояний |
| Scheduler | Служба настройки расписания запуска |
| Controller | Механизм управления ресурсами для параллельного исполнения моделей и их элементов, в том числе за счет динамического создания новых Executor-ов и управления выделяемыми им ресурсами |
| Interpreter | Служба интерпретации модели для конкретных Executor-ов |
| Executor | Прикладной модуль линейной оптимизации |
| Reader | Служба загрузки данных из внешних источников |
| Writer | Служба передачи данных во внешние источники |
| ESB | Шина данных для межсервисного взаимодействия |
| Permissions | Служба управления правами доступа |
| Logging | Подсистема журналирования |
| Monitoring | Подсистема мониторинга |
| Version control | Подсистема версионирования моделей |

## Сведения о связях между составными частями программы

Входящие в состав Системы элементы в процессе функционирования осуществляют обмен информацией с использованием API для приема и передачи данных.

Описание методов взаимодействия представлено в Приложении 1. «Public-API: документация по интеграции».

## Сведения о связях с другими программами

В процессе функционирования Система взаимодействует с источниками метаданных, которыми являются внешние системы, данные справочников.

Информационная связь между элементами Системы и внешней средой осуществляется посредством API.

Описание методов взаимодействия представлено в Приложении 1. «Public-API: документация по интеграции».

# Настройка программы

## Установка Системы

Для развертывания BPLEX необходимо выполнить следующие действия в установленном порядке.

## Процесс запуска программного обеспечения

Развёртывание бэкенда

Чтобы подготовить к работе и запустить бэкенд системы BPLEX (проект https://gitlab.dellin.ru/OMOiM\_developers/bplex-system/), используя сервер или рабочую станцию с ОС Linux, необходимо:

Установить и настроить JDK 11

Установить и настроить Docker

Установить и настроить клиентское ПО Kubernetes

Скачать исходники проекта (командой git clone https://gitlab.dellin.ru/OMOiM\_developers/bplex-system.git или иным способом).

В файле *resources/application.properties* указать актуальные параметры доступа к БД и другие привязанные к развёртыванию.

Собрать файл jar командой *./gradlew quarkusBuild compileTestKotlin*

Собрать и опубликовать Docker-образ:

*docker build -t* *Название\_образа* *-f src/main/docker/Dockerfile* .

*docker push Название\_образа*

Подготовить и развернуть конфигурацию на кластере Kubernetes.

Типовой шаблон конфигурации находится в файле *src/main/kubernetes/deployment.template.yaml* проекта. В него необходимо подставить требуемые значения параметров для новой конфигурации и сохранить файл с названием *deployment.yaml*

Развернуть конфигурацию командой *kubectl replace --force --cascade -f deployment.yaml*

Развёрнутый бэкенд имеет UI, сгенерированный с использованием Swagger, расположенный по URL *http://web-test.bplex.sandbox.bia-tech.ru/q/swagger-ui/* (здесь и далее домен в указанном URL характерен для демо-стенда). Рабочее API соответствует содержимому этой страницы.

Развёртывание фронтенда

Чтобы подготовить к работе и запустить фронтенд системы BPLEX (проект *https://gitlab.dellin.ru/OMOiM\_developers/bplex-ui/*), используя сервер или рабочую станцию с ОС Linux, необходимо:

Установить и настроить Docker.

Установить и настроить клиентское ПО Kubernetes.

Скачать исходники проекта (командой *git clone https://gitlab.dellin.ru/OMOiM\_developers/bplex-ui.git* или иным способом)

В файле *src/api.ts* указать актуальный адрес бэкенда (в соответствии с Ingress или Service).

Собрать и опубликовать Docker-образ:

*docker build -t Название\_образа* .

*docker push Название\_образа*

Подготовить и развернуть конфигурацию на кластере Kubernetes.

Пример конфигурации находится в файле *kubernetes/deployment.json* проекта. При необходимости можно изменить некоторые параметры.

Развернуть конфигурацию командой *kubectl rollout restart -f ./kubernetes/deployment.json*

# Проверка программы

## Описание способов проверки, позволяющих дать общее заключение о работоспособности программы

Общее состояния системы отображается на */q/health*

## Контрольные примеры

Примеры вводимых и получаемых данных, а также примеры выполнения операций в BPLEX, приведены в документе «БИПЛЕКС-BPLEX. Руководство пользователя».

## Результаты проверки программы

Комплекс программного обеспечения «БИПЛЕКС-BPLEX» является работоспособным при одновременном выполнении следующих условий:

1. в процессе запуска нет сообщения (сообщений) об ошибках;
2. доступны все рабочие интерфейсы Системы.

В ходе выполнения сценариев работы программы не должно быть зафиксировано критических ошибок, останавливающих или блокирующих работу программы. Перечень сообщений и ошибок, допустимых при работе Системы, приведен в п. 6.3 настоящего документа.

# Сообщения системному программисту

## Тексты сообщений, выдаваемых в ходе настройки программы

Сообщения, выдаваемые при настройке Системы, описаны в разделе 4 «Настройка программы» настоящего документа.

## Тексты сообщений, выдаваемых в ходе проверки программы

Сообщения, выдаваемые программой в ходе выполнения проверки, аналогичны сообщениям, выдаваемым программой в ходе ее выполнения, и приведены в п.6.3.

## Тексты сообщений, выдаваемых в ходе выполнения программы

Сообщения, которые выдаются системой в ходе работы, описаны в документе «БИПЛЕКС-BPLEX. Руководство пользователя».

# Список обозначений и сокращений

| Термин/ Сокращение | Значение |
| --- | --- |
| Ant | (Another Neat Tool) – инструмент сборки на основе XML, который является широко используемым инструментом сборки на основе Java с полной переносимостью чистого кода Java |
| API | (Application Programming Interface) – программный интерфейс приложения, интерфейс [прикладного программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)), описание способов (набор [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), [процедур](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), [структур](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) или [констант](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29)), которыми одна компьютерная программа может взаимодействовать с другой программой |
| Basic  | Beginner's All-purpose Symbolic Instruction Code (Всецелевой язык программирования для начинающих) |
| CentOS | ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Community ENTerprise Operating System) – [дистрибутив](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) [Linux](https://ru.wikipedia.org/wiki/Linux), основанный на коммерческом [Red Hat Enterprise Linux](https://ru.wikipedia.org/wiki/Red_Hat_Enterprise_Linux) компании [Red Hat](https://ru.wikipedia.org/wiki/Red_Hat) и совместимый с ним |
| CPLEX  | Пакет программного обеспечения («решатель»), предназначенный для решения задач линейного и квадратичного программирования, в том числе целочисленного программирования |
| Docker | Программное обеспечение для автоматизации развёртывания и управления приложениями в среде [виртуализации на уровне операционной системы](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) |
| Elastic Search  | Высокомасштабируемая распределенная поисковая система полнотекстового поиска и анализа данных, работающая в режиме реального времени |
| Explicit Deny | Приоритет отказа над разрешением |
| Gitlab | Веб-инструмент жизненного цикла DevOps с открытым исходным кодом, представляющий систему управления репозиториями кода для Git с собственной вики, системой отслеживания ошибок, CI/CD пайплайном и другими функциями |
| Gitlab Registry  | GitLab Container Registry – это безопасный приватный реестр для образов (images) Docker, разработанный с помощью ПО с открытым кодом |
| Gradle | Система сборки |
| Graphana | Инструмент с открытым исходным кодом для визуализации данных из различных систем сбора статистики |
| Gurobi | Внешний модуль решения задач с параллельными алгоритмами для задач линейного программирования, задач квадратичного программирования и смешанно-целочисленных задач |
| ID (ИД) | Идентификатор |
| Java | [Строго](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8_%D1%81%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [объектно-ориентированный язык программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения |
| Java Runtime Environmen | Среда выполнения Java |
| JDK | JDK (Java SE Development Kit) – набор инструментов разработчика для создания программ на Java. Включает в себя JRE плюс инструменты для разработки, отладки и мониторинга Java приложений |
| JSON | (JavaScript Object Notation) – [текстовый формат](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82) [обмена данными](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D0%B5%D0%BD_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8), основанный на [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) |
| Kafka | Распределённый программный [брокер сообщений](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%BE%D0%BA%D0%B5%D1%80_%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B9), проект с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), разрабатываемый в рамках фонда [Apache](https://ru.wikipedia.org/wiki/Apache_Software_Foundation). Написан на языках программирования [Java](https://ru.wikipedia.org/wiki/Java) и [Scala](https://ru.wikipedia.org/wiki/Scala_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) |
| Kerberos | Kerberos – сетевой протокол аутентификации, позволяющий передавать данные через незащищённые сети для безопасной идентификации. Ориентирован , в первую очередь , на клиент-серверную модель и обеспечивает взаимную аутентификацию – оба пользователя через сервер подтверждают личности друг друга |
| Kibana | Платформа для анализа и визуализации с открытым исходным кодом, предназначенная для работы с Elasticsearch |
| Kubernetes | Программное обеспечение с открытым [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) |
| LDAP | (англ. Lightweight Directory Access Protocol – «легковесный протокол доступа к каталогам») – протокол прикладного уровня для доступа к службе каталогов X.500, разработанный IETF как облегчённый вариант разработанного ITU-T протокола DAP |
| Mongo | [Документоориентированная](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) [система управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), не требующая описания схемы таблиц. Считается одним из классических примеров [NoSQL](https://ru.wikipedia.org/wiki/NoSQL)-систем, использует [JSON](https://ru.wikipedia.org/wiki/JSON)-подобные документы и схему базы данных. Написана на языке [C++](https://ru.wikipedia.org/wiki/C%2B%2B). Применяется в веб-разработке, в частности, в рамка [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-ориентированного стека [MEAN](https://ru.wikipedia.org/wiki/MEAN) |
| MS SQL | СУБД реляционного типа. Для манипуляции данными используется специально разработанный язык Transact-SQL |
| PostgreSQL  | [Свободная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) [объектно-реляционная](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) |
| Prometeus | Система мониторинга серверов и программ с открытым исходным кодом |
| Python | Интерпретируемый язык программирования |
| Quarkus | Полная, многоплатформенная среда проектирования, легко адаптируемая к требованиям конкретного проекта |
| Red Hat Linux | Дистрибутив [Linux](https://ru.wikipedia.org/wiki/Linux) компании [Red Hat](https://ru.wikipedia.org/wiki/Red_Hat) |
| Red Hat OpenShift Container Platform | Платформа для самостоятельной подготовки, сборки и развертывания приложений и их компонентов. [OpenShift](https://u.tmtm.ru/axoft_redhat_openshift) позволяет собрать воедино множество разрозненных процессов внутри компании и вести разработку в соответствии с принципами DevOps и CI/CD |
| SDK | Набор средств разработки, позволяющий специалистам по программному обеспечению создавать приложения для определённого пакета программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, игровых консолей, операционных систем и прочих платформ |
| Sparx | Sparx Enterprise Architect (EA) - инструмент визуального моделирования и проектирования, основанный на OMG UML |
| Swagger | фреймворк для спецификации RESTful API |
| UID | (Unique identifier) – уникальный идентификатор |
| БД | База данных |
| Бэкенд | (англ. back-end) – программно-аппаратная часть сервиса. Бэкенд отвечает за осуществление функционирования внутренней части веб-сайта |
| ОС | Операционная система |
| ПК | Персональный компьютер |
| ПО | Программное обеспечение |
| СУБД | Система управления базами данных |
| Сущность | Совокупность коллекции объектов с присущей им базовой (низкоуровневой) бизнес-логикой. Сущность обладает набором характеристик, значения которых подчиняются определенным правилам обработки |
| Фронтенд | (англ. front-end) – клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса |

# Приложение 1. Public-API: документация по интеграции BPLEX

В настоящем документе приведен формат обмена сообщениями описание API BPLEX <http://web-test.bplex.sandbox.bia-tech.ru/q/swagger-ui/#/>

API сервис работает по протоколу HTTP, HTTPS и представляет собой набор методов, с помощью которых совершаются запросы и возвращаются ответы для каждой операции. Все ответы приходят в виде JSON структур.

Интеграции, в том числе, возможна на основе существующих механизмов интеграции (Excel/CSV, OLEDB, XML, JSON, Rest API).

**Терминология**

**Процесс –** сущность, представляющая собой решение значимой для бизнеса проблемы, сводящаяся к преобразованию входных данных в выходные через выполнение набора стадий по цепочке вызовов.

**Стадия** – сущность, которая для входных данных через определенный промежуток времени возвращает выходные данные. Используется для решения как общей бизнес задачи целиком, важной именно для пользователя, так и только для неделимой части какого-то более сложного вычисления.

**Сервис –** конкретный сервис с унифицированным API, развернутый на конкретном URL, описывающий логику, за которую отвечает стадия.

**API**

## Registry – реестр описания процессов

**POST /registry/process** – зарегистрировать новый процесс или обновить информацию о процессе с теми же тегами и версией.

UUID: String – UUID описания процесса, именно по UUID будет потом этот процесс идентифицироваться.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят идентифицировать конкретную версию процесса. String – версия процесса.

body: String – собственно информация о расчете, DSL по стадиям.

DSL описывает расчет, как однонаправленный ациклический граф стадий, с двумя особыми вершинами, называемая начальная и конечная. На начальную вершину подаются входные данные для всего расчета, с конечной вершины получаются результаты целиком. В DSL описаны набор стадий, последовательность и их связи. Стадия описывается именем (тегами), версией (если не указана, то используется версия расчета), при интерпретации будут найдены те URL сервисов стадий, которые будут содержать все нужные теги. При выполнении процесса будет использован любой из свободных сервисов, который найден.

**POST /registry/service** – зарегистрировать URL сервиса, отвечающий за стадию расчета.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретную стадию.

version: String – версия сервиса.

url: String – уникальный URL сервиса, на котором развернута стадия.

**GET /registry/processes –** получить информацию о процессах.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный процесс, optional.

version: String –версия процесса, optional.

dateFrom: optional Date – дата регистрация расчета, от которой вести поиск, optional.

dateTo: optional Date – дата регистрация расчета, до которой вести поиск, optional.

limit: optional Int – максимальное количество записей результата, optional.

offset: optional Int – возвращать со смещением, начиная с определенной записи (по умолчанию – с начала).

Responses – возвращается список UUID, соответствующий критериям, а также в результате возвращается общее количество записей. В случае если, например, не указана версия, то будет передана информация обо всех версиях процесса. Если вообще не указаны никакие параметры, то возвращается максимально допустимое количество записей о процессах и общее количество записей.

**GET /registry/process/UUID –** получить информацию о конкретном процессе.

UUID: String – UUID процесса.

Responses – то, что передавалось ранее на POST.

**GET /registry/service –** получить список сервисов, удовлетворяющим критериям поиска.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретную стадию, optional.

version: String – optional версия стадии.

Responses – то, что передавалось ранее на POST. Передается список, соответствующий критериям. В случае, если не указана, например, версия, то будут передана информация обо всех версиях стадии.

**delete /registry/process –** удалить информацию о процессе.

UUID: String – UUID описания процесса.

**delete /registry/service/url –** сделать URL сервиса, отвечающим за стадии расчета, недоступным для использования.

URL: String – URL сервера, на котором развернут сервис.

## Execution – логика, касающаяся исполнения расчета

**POST /execution/process –** запуск процесса на исполнение.

parameters:

UUID: String – UUID конкретного процесса, именно по UUID потом будет идентифицироваться этот процесс.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный процесс.

version: String –версия расчета, optional. Если версия не указана, то используется последняя.

body: object – входные данные.

**GET /execution/process/status**

parameters:

UUID – UUID процесса.

Responses – текущая стадия, время запуска, ожидаемое время завершения, на каком сервере выполняется и подобная информация.

**DELETE /execution/process –** отмена процесса. При отмене процесса освобождаются все занятые ресурсы.

parameters:

UUID – UUID процесса.

**GET /execution/process/result –** результаты процесса.

Responses – данные результата или подробная информация об ошибках.

## Monitoring – средства мониторинга инфраструктуры

**GET /monitoring/process –** получить информацию о процессах.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный процесс, optional.

version: String – версия процесса, optional.

dateFrom: optional Date – дата регистрация расчета, от которой вести поиск, optional.

dateTo: optional Date – дата регистрация расчета, до которой вести поиск, optional.

limit: optional Int – максимальное количество записей результата, optional.

offset: optional Int – возвращать со смещением, начиная с определенной записи (по умолчанию – с начала).

Responses – возвращается список данных о процессах, соответствующий критериям, а также в результате возвращается общее количество записей. В случае, если, например, не указана версия, то будет передана информация обо всех версиях процесса. Если не указаны вообще никакие параметры, то возвращается максимально допустимое количество записей о процессах и общее количество записей. Каждый элемент списка содержит полный набор тегов процесса, версию, текущий статус сервиса, время последнего изменения статуса, список сервисов, занятых конкретным процессом.

**GET /monitoring/service –** получить информацию о сервисах.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный сервис, optional.

version: String – версия сервиса, optional.

dateFrom: optional Date – дата регистрация расчета, от которой вести поиск, optional.

dateTo: optional Date – дата регистрация расчета, до которой вести поиск, optional.

limit: optional Int – максимальное количество записей результата, optional.

offset: optional Int – возвращать со смещением, начиная с определенной записи (по умолчанию – с начала).

Responses – возвращается список данных о сервисах, соответствующий критериям, а также в результате возвращается общее количество записей. В случае, если, например, не указана версия, то будет передана информация обо всех версиях сервиса. Если не указаны вообще никакие параметры, то возвращается максимально допустимое количество записей о сервисах и общее количество записей. Каждый элемент списка содержит полный набор тегов стадии, с которой связан сервис, версию, URL, на котором развернут сервис, текущий статус сервиса, время последнего изменения статуса.

## History – просмотр исторической информации

**GET /history/processes –** получить список процессов по критериям поиска.

tags: optional String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный процесс.

version : optional String – версия процесса.

dateFrom: optional Date – дата регистрация процесса, от которой вести поиск, optional.

dateTo: optional Date – дата регистрация процесса, до которой вести поиск, optional.

limit: optional Int – максимальное количество записей результата, optional.

offset: optional Int – возвращать со смещением, начиная с определенной записи (по умолчанию – с начала).

Responses – набор UUID процессов, удовлетворяющих критериям поиска.

**GET /history/stage –** получить список стадий, участвующих в процессе, по критериям поиска.

calcUUID: UUID процесса.

Responses – набор UUID стадий, связанных с выполнением расчета с данным UUID.

**GET /history/process/log –** получить логи конкретного расчета.

UUID: String – UUID конкретного расчета.

**GET /history/stage/log –** получить логи конкретной стадии.

UUID: String – UUID конкретного расчета.

**GET /history/process/input** – получить входные данные для конкретного расчета.

UUID: String – UUID расчета.

**GET /history/stage/input –** получить входные данные для конкретной стадии.

UUID: String – UUID стадии.

**GET /history/process/output –** получить данные результата для конкретного расчета.

UUID: String – UUID расчета.

**GET /history/stage/output –** получить данные результата для конкретной стадии.

UUID: String – UUID стадии.

**GET /history/process/status –** получить статистическую информацию о выполнении конкретного расчета (начало выполнения, конец выполнения, на каком сервере считалось, успех или неудача).

UUID: String – UUID расчета.

**GET /history/stage/status –** получить статистическую информацию о выполнении конкретной стадии (начало выполнения, конец выполнения, на каком сервере считалось, успех или неудача).

UUID: String – UUID.

## Schedule – обеспечение запуска по расписанию

**POST /shedule – зарегистрировать события автоматического старта процесса.**

UUID: String – UUID конкретного расписания, именно по UUID потом будет идентифицироваться это расписание.

tags: optional String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный процесс.

version : optional String – версия процесса.

initialDateTime: DateTime – дата и время первого запуска.

delayMinites: Int –промежуток времени в минутах между запусками.

untilStart: Boolean – если True, то отсчитывать промежуток рестарта от начала процесса, если False, то от завершения. В случае, если требуется стартовать процесс, но предыдущий еще не завершился, событие старта в этот момент пропускается.

body: входные данные процесса.

**GET /shedules –** получить информацию о зарегистрированных событиях автоматических стартов.

tags: String – разделенные запятой теги (например, test, preprod, full, small, reduced, moscow), которые позволят отличить конкретный сервис, optional.

version: String – optional версия сервиса.

dateFrom: optional Date – дата регистрация расчета, от которой вести поиск, optional.

dateTo: optional Date – дата регистрация расчета, до которой вести поиск, optional.

limit: optional Int – максимальное количество записей результата, optional.

offset: optional Int – возвращать со смещением, начиная с определенной записи (по умолчанию – с начала).

Responses – список UUID расписаний, удовлетворяющих условиям поиска, которые изначально отправлялись на POST, а также общее количество зарегистрированных записей.

**GET /shedule/UUID –**  получить информацию о конкретном расписании.

UUID: String – UUID расписания.

Responses – те данные, которые передавались на POST.

**DELETE** **/shedule – пометить событие автоматического старта на удаление.**

UUID: String – UUID конкретного расписания.



## Сгенерированный API

(сгенерировано Swagger Codegen https://github.com/swagger-api/swaggercodegen).

Version: 1.0.

All rights reserved http://apache.org/licenses/LICENSE-2.0.html

**Methods**

Execution

 **DELETE /execution/process**

Отмена процесса. Процесс отменяется, при этом освобождаются все занятые ресурсы (executionProcessDelete).

*Query parameters*

**UUID (optional)**

*Query Parameter* – uuid процесса

*Responses*

**204**

No Content

 **GET /execution/process/output**

Результаты процесса (executionProcessOutputGet).

*Query parameters*

UUID (optional)

*Query Parameter* – uuid процесса

*Return type*

MultipartBody

*Example data*

Content-Type: application/json

{

 "data" : { },

 "name" : "name"

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 multipart/form–data

*Responses*

200 данные результата или подробная информация об ошибках MultipartBody

 **POST /execution/process**

Запуск процесса на исполнение (executionProcessPost).

*Consumes*

This API call consumes the following media types via the Content-Type request header:

 multipart/form-data

*Query parameters*

UUID (optional)

*Query Parameter* – uuid конкретного процесса, именно по UUID будет потом этот процесс идентифицироваться tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса. Если не указана, используется последняя.

Form parameters

data (optional) *Form Parameter* — name (optional) *Form Parameter* —

*Responses*

201

Created

 **GET /execution/process/status**

Плучение статуса процесса (executionProcessStatusGet).

*Query parameters*

UUID (optional)

*Query Parameter* – uuid процесса

Return type

ProcessStatus

*Example data*

Content-Type: application/json

|  |
| --- |
| {  "planningEndTime" : "2000-01-23",  "stage" : "stage",  "serviceURL" : "serviceURL" } |

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Текущая стадия, время запуска, ожидаемое время завершения, на каком сервере выполняется и подобная информация ProcessStatus.

History

 **GET /history/process/input**

Получить входные данные для конкретного процесса (historyProcessInputGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid процесса

Return type

MultipartBody

*Example data*

Content-Type: application/json

{

 "data" : { },

 "name" : "name"

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 multipart/form-data

*Responses*

200 входные данные MultipartBody

 **GET /history/process/log**

Получить логи конкретного процесса (historyProcessLogGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid процесса

Return type

String

*Example data*

Content-Type: application/json

""

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Логи String

 **GET /history/process/output**

Получить выходные данные для конкретного процесса (historyProcessOutputGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid процесса

Return type

MultipartBody

*Example data*

Content-Type: application/json

{

 "data" : { },

 "name" : "name"

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 multipart/form-data

*Responses*

200 выходные данные MultipartBody

 **GET /history/process/status**

Получить статистическую информацию о выполнении конкретного процесса (historyProcessStatusGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid процесса

Return type

ProcessStatus

*Example data*

Content-Type: application/json

|  |
| --- |
| {  "planningEndTime" : "2000-01-23",  "stage" : "stage",  "serviceURL" : "serviceURL" } |

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Начало выполнения, конец выполнения, на каком сервере считалось, успех или неудача ProcessStatus.

 **GET /history/processes**

Получить список процессов по критериям поиска (historyProcessesGet).

*Query parameters*

dateFrom (optional)

*Query Parameter* – дата регистрация расчета, от которой искать dateTo (optional).

*Query Parameter* – дата регистрация расчета, до которой искать limit (optional).

*Query Parameter* – максимальное количество записей результата format: int32 offset (optional).

*Query Parameter* – возвращать со смещением, начиная с определенной записи, по умолчанию с начала, format: int32 tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* — версия процесса

*Return type*

[UUIDList](#_UUIDList_Up)

*Example data*

Content-Type: application/json

{

 "totalRows" : 0,

 "uuids" : [ "uuids", "uuids" ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Возвращается список UUID, соответствующий критериям, а также в результате возвращается общее количество записей. В случае, если, например, не указана версия, то будет передана информация обо всех версиях процесса. Если не указаны вообще никакие параметры, то возвращается максимально допустимое количество записей о процессах и общее количество записей

[UUIDList](#_UUIDList_Up)

 **GET /history/stage**

Плучить список стадий, участвующих в процессе (historyStageGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid процесса

Return type

[UUIDList](#_UUIDList_Up)

*Example data*

Content-Type: application/json

{

 "totalRows" : 0,

 "uuids" : [ "uuids", "uuids" ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200 набор uuid стадий, связанных с выполнением расчета с данным UUID [UUIDList](#_UUIDList_Up)

 **GET /history/stage/input**

Получить входные данные для конкретной стадии (historyStageInputGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid стадии

Return type

MultipartBody

*Example data*

Content-Type: application/json

{

 "data" : { },

 "name" : "name"

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 multipart/form-data

*Responses*

200 входные данные MultipartBody

 **GET /history/stage/log**

Получить логи конкретной стадии (historyStageLogGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid стадии

Return type

String

*Example data*

Content-Type: application/json

""

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Логи String

 **GET /history/stage/output**

Получить выходные данные для конкретной стадии (historyStageOutputGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* – uuid стадии

Return type

MultipartBody

*Example data*

Content-Type: application/json

{

 "data" : { },

 "name" : "name"

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 multipart/form-data

*Responses*

200 выходные данные MultipartBody

 **GET /history/stage/status**

Получить статистическую информацию о выполнении конкретной стадии (historyStageStatusGet).

*Query parameters*

calcUUID (optional)

*Query Parameter* — uuid стадии

Return type

ProcessStatus

*Example data*

Content-Type: application/json

|  |
| --- |
| {  "planningEndTime" : "2000-01-23",  "stage" : "stage",  "serviceURL" : "serviceURL" } |

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200 выходные данные ProcessStatus

Monitoring

 **GET /monitoring/process**

Плучить информацию о процессах (monitoringProcessGet).

*Query parameters*

dateFrom (optional)

*Query Parameter* – дата регистрация расчета, от которой искать dateTo (optional).

*Query Parameter* – дата регистрация расчета, до которой искать limit (optional).

*Query Parameter* – максимальное количество записей результата format: int32 offset (optional).

*Query Parameter* – возвращать со смещением, начиная с определенной записи, по умолчанию с начала, format: int32 tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса

Return type

[UUIDList](#_UUIDList_Up)

*Example data*

Content-Type: application/json

{

 "totalRows" : 0,

 "uuids" : [ "uuids", "uuids" ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Возвращается список UUID, соответствующий критериям, плюс в результате возвращается общее количество записей. В случае, если не указана, например, версия, то будут передана информация обо всех версиях процесса. Если не указаны вообще никакие параметры возвращается максимально допустимое количество записей о процессах и общее количество [UUIDList](#_UUIDList_Up)

 **GET /monitoring/service**

Получить информацию о сервисах (monitoringServiceGet).

*Query parameters*

dateFrom (optional)

*Query Parameter* – дата регистрация расчета, от которой искать dateTo (optional).

*Query Parameter* – дата регистрация расчета, до которой искать limit (optional).

*Query Parameter* – максимальное количество записей результата format: int32 offset (optional).

*Query Parameter* – возвращать со смещением начиная с определенной записи, по умолчанию с начала, format: int32 tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса

Return type

ServiceInfoList

*Example data*

Content-Type: application/json

{

 "totalRows" : 0,

 "infos" : [ {

 "version" : "version",

 "url" : "url",

 "tags" : [ "tags", "tags" ]

 }, {

 "version" : "version",

 "url" : "url",

 "tags" : [ "tags", "tags" ]

 } ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Возвращается список данных о сервисах, соответствующий критериям, а также в результате возвращается общее количество записей. В случае, если, например, не указана версия, то будет передана информация обо всех версиях сервиса. Если не указаны вообще никакие параметры, то возвращается максимально допустимое количество записей о сервисах и общее количество записей. Каждый элемент списка содержит полный набор тегов стадии, с которой связан сервис, версию, url, на котором развернут сервис, текущий статус сервиса, время последнего изменения статуса ServiceInfoList

Registry

**DELETE /registry/process**

Удалить информацию о процессе (registryProcessDelete).

*Query parameters*

UUID (optional)

*Query Parameter* – uuid описания процесса

*Responses*

204

No Content

**GET /registry/process**

Получить информацию о процессах (registryProcessGet).

*Query parameters*

dateFrom (optional)

*Query Parameter* – дата регистрация расчета, от которой искать dateTo (optional).

*Query Parameter* – дата регистрация расчета, до которой искать limit (optional).

*Query Parameter* – максимальное количество записей результата format: int32 offset (optional).

*Query Parameter* – возвращать со смещением начиная с определенной записи, по умолчанию с начала, format: int32 tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow) которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса

Return type

[UUIDList](#_UUIDList_Up)

*Example data*

Content-Type: application/json

{

 "totalRows" : 0,

 "uuids" : [ "uuids", "uuids" ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

Возвращается список UUID, соответствующий критериям, а также в результате возвращается общее количество записей. В случае, если, например, не указана версия, то будет передана информация обо всех версиях процесса. Если не указаны вообще никакие параметры, то возвращается максимально допустимое количество записей о процессах и общее количество [UUIDList](#_UUIDList_Up)

 **POST /registry/process**

Зарегистрировать новый процесс или обновить информацию о процессе с теми же тегами и версией (registryProcessPost).

<pre><code> body – собственно информация о расчете, DSL по стадиям. DSL описывает расчет как однонаправленный ациклический граф стадий, с двумя особыми вершинами, называемая начальная и конечная. На начальную вершину подаются входные данные для всего расчета, с конечной вершины получаем результаты целиком. В DSL описаны набор стадий, последовательность и их связи, стадия описывается именем, тегами, версией (если не указана, то используется версия расчета), при интерпретации будут найдены те URL сервисов стадий, которые будут содержать все нужные теги. При выполнении процесса будет использован любой из свободных сервисов, который был найден </code></pre>

*Consumes*

This API call consumes the following media types via the Content-Type request header:

 application/json

Request body

body ProcessDTO (optional) *Body Parameter* —

*Query parameters*

UUID (optional)

*Query Parameter* – uuid описания процесса, именно по UUID будет потом этот процесс идентифицироваться tags (optional).

*Query Parameter* – разделенные запятой теги (например test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса.

*Responses*

201

Created

 **GET /registry/process/{uuid}**

Получить информацию о конкретном процессе (registryProcessUuidGet).

*Path parameters*

uuid (required)

*Path Parameter* – UUID процесса

Return type ProcessInfo

*Example data*

Content-Type: application/json

|  |
| --- |
| {  "dsl" : {  "dsl" : [ {  "type" : "MAP",  "version" : "version",  "additionalIn" : [ "additionalIn", "additionalIn" ],  "tags" : "tags" }, {  "type" : "MAP",  "version" : "version",  "additionalIn" : [ "additionalIn", "additionalIn" ],  "tags" : "tags"  } ]  },  "uuid" : "uuid",  "version" : "version" } |

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

То, что передавалось ранее на post. ProcessInfo

 **DELETE /registry/service**

Сделать недоступным для использования url сервиса, отвечающего за стадии расчета (registryServiceDelete).

*Query parameters*

url (optional)

*Query Parameter* – URL сервера, на котором развернут сервис.

*Responses*

204

No Content

 **GET /registry/service**

Плучить список сервисов, удовлетворяющим критериям поиска (registryServiceGet).

*Query parameters*

tags (optional)

*Query Parameter* – разделенные запятой теги (например test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса.

Return type ServiceInfo

*Example data*

Content-Type: application/json

{

 "version" : "version",

 "url" : "url",

 "tags" : [ "tags", "tags" ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200

То, что передавалось ранее на post. Передается список, соответствующий критериям. В случае, если не указана, например, версия, то будут передана информация обо всех версиях стадии ServiceInfo

 **POST /registry/service**

Зарегистрировать URL сервиса, отвечающего за стадию расчета (registryServicePost).

*Query parameters*

tags (optional)

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс url (optional).

*Query Parameter* – URL сервиса, на котором развернута стадия, уникален version (optional).

*Query Parameter* – версия процесса

*Responses*

201

Created

Schedule

 **DELETE /schedule**

Пометить событие автоматического старта на удаление (scheduleDelete).

*Responses*

204

No Content

 **POST /schedule**

Зарегистрировать события автоматического старта процесса (schedulePost).

*Consumes*

This API call consumes the following media types via the Content-Type request header:

 application/json;charset=UTF-8

Request body

body MultipartBody (optional) *Body Parameter*

*Query parameters*

delayMinutes (optional)

*Query Parameter* – промежуток времени в минутах между запусками format: int32 initialDateTime (optional).

*Query Parameter* – дата и время первого запуска tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс untilStart (optional).

*Query Parameter* – если true, то отсчитывать промежуток рестарта от начала процесса, если false, то от завершения. В случае, если требуется стартовать процесс, но предыдущий еще не завершился, событие старта в этот момент пропускается uuid (optional).

*Query Parameter* – uuid конкретного расписания, именно по UUID будет потом это расписание идентифицироваться version (optional).

*Query Parameter* – версия процесса

*Responses*

201

Created

 **GET /schedule/{uuid}**

Получить информацию о конкретном расписании (scheduleUuidGet).

*Path parameters uuid (required) Path Parameter – UUID расписания*

Return type ScheduleInfo

*Example data*

Content-Type: application/json

{

 "initialDateTime" : "2000-01-23",

 "delayMinutes" : 0,

 "untilStart" : true,

 "uuid" : "uuid",

 "version" : "version",

 "tags" : "tags"

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200 те данные, которые передавались на POST ScheduleInfo

 **GET /schedules**

Получить информацию о зарегистрированных событиях автоматических стартов (schedulesGet).

*Query parameters*

dateFrom (optional)

*Query Parameter* – дата регистрация расчета, от которой искать format: date dateTo (optional).

*Query Parameter* – дата регистрация расчета, до которой искать format: date limit (optional).

*Query Parameter* – максимальное количество записей результата format: int32 offset (optional).

*Query Parameter* – возвращать со смещением начиная с определенной записи, по умолчанию с начала, format: int32 tags (optional).

*Query Parameter* – разделенные запятой теги (например, test, preprod, full, small, reduced, moskow), которые позволят идентифицировать конкретный процесс version (optional).

*Query Parameter* – версия процесса

Return type

 [UUIDList](#_UUIDList_Up)

*Example data*

Content-Type: application/json

{

 "totalRows" : 0,

 "uuids" : [ "uuids", "uuids" ]

}

*Produces*

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

 application/json

*Responses*

200 список UUID расписаний, удовлетворяющих условиям поиска, которые изначально отправлялись на POST плюс общее количество зарегистрированных. [UUIDList](#_UUIDList_Up)

**[Models](#_Сгенерированный_API)**

**Date**

**InputStream**

**ListServiceInfo**

**ListStageDescription**

**ListString**

**MultipartBody**

**data (optional)** *InputStream* **name (optional)** *String*

**ProcessDTO**

**dsl (optional)** *ListStageDescription*

**ProcessInfo**

**dsl (optional)** *ProcessDTO* **tags (optional)** *ListString* **uuid (optional)** *String*

**version (optional)** *String*

**ProcessStatus**

**planningEndTime (optional)** *Date*

**serviceURL (optional)** *String*

**stage (optional)** *String*

**startingTime (optional)** *Date*

**ScheduleInfo**

**delayMinutes (optional)** *Integer* format: int32 **initialDateTime (optional)** *Date*

**tags (optional)** *String*

**untilStart (optional)** *Boolean* **uuid (optional)** *String*

**version (optional)** *String*

**ServiceInfo**

**tags (optional)** *ListString* **url (optional)** *String*

**version (optional)** *String*

**ServiceInfoList**

**infos (optional)** *ListServiceInfo* **totalRows (optional)** *Integer* format: int32

**StageDescription**

**additionalIn (optional)** *ListString*

**nextStage (optional)** *StageDescription* **previousStages (optional)** *ListStageDescription* **tags (optional)** *String*

**tasks (optional)** *ListString* **type (optional)** *StageType* **version (optional)** *String*

**StageType**

**UUIDList**

**totalRows (optional)** *Integer* format: int32 **uuids (optional)**

*ListString*

****Лист регистрации изменений****

| Номера листов (страниц) | Всего листов (страниц) в докум. | № доку­мента | Входящий № сопроводи­тельного докум. и дата | Подп. | Дата |
| --- | --- | --- | --- | --- | --- |
| Изм | Изменен­ных | Заменен­ных | Новых | Аннули­рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |