
Документация

Выпуск 6

XSQUARE - REPORTS

февр. 23, 2026

1	Общие сведения	1
1.1	Знакомство с сервером отчетов	1
1.2	Модули сервера отчетов	1
1.3	Установка сервера отчетов	2
1.4	Примеры использования	2
1.5	Получение первого отчета	2
2	Архитектура и системные требования	3
2.1	Архитектура	3
2.2	Базовая архитектура	3
2.3	Архитектура для высоконагруженных систем	4
2.4	Среда исполнения	5
2.5	Системные требования	5
3	Установка	6
3.1	Установка сервера отчетов	6
3.1.1	Установка шрифтов для RPM-based ОС	6
3.1.2	Установка Libre Office	7
3.1.3	Файл конфигурации config.json	7
4	Эксплуатация	9
4.1	Запуск сервера отчетов	9
4.2	Проверка состояния сервера отчетов	9
4.3	Сбор информации об ошибках	10
5	Модули	11
5.1	Сервис формирования отчетов из документа-шаблона	11
5.1.1	Общее описание сервиса	11
5.1.2	Работа с документами-шаблонами	11
5.1.3	Формирование отчета в формате DOCX	12
5.1.4	Формирование отчета формате XLSX	35
5.1.5	Конвертация отчета в PDF	57
5.1.6	Работа со штрихкодом	58
5.1.7	Работа с графическими кодами (QR code)	59
5.2	Генератор отчетов PDF	60
5.2.1	Общее описание сервиса	60
5.2.2	Формирование документов с помощью генератора PDF	60

5.2.3	Пример вызова сервиса	64
5.2.4	Команды генератора	66
5.2.5	Координаты	83
5.2.6	Теги	84
5.2.7	Работа со штрихкодом	85
5.3	Сервис объединения PDF-документов	86
5.4	Сервис формирования печатной формы	89
5.5	Сервис заполнения PDF-форм	95
5.6	Сервис конвертации XLSX документов в JSON, XML, CSV	98
5.7	Сервис конвертации XLS документов в JSON, XML, CSV	101
5.8	Сервис генерации кодов (QR code)	102
5.9	Сервис конвертации документов DOCX/XLSX в формат PDF	107
6	Свойства готовых отчетов (метаданные)	109
6.1	Поддерживаемые форматы	109
6.2	Доступные метаданные	109
6.3	Структура запроса	109
7	Генератор запросов	111
7.1	Описание сервиса	111
7.2	Путь к сервису	111
7.3	Интерфейс	111
7.4	Описание работы	113
7.5	Именованние тегов в шаблонах	113
7.6	API генератора	114
8	Примеры	116
8.1	Архив примеров	116
8.2	Формирование отчета из архива примеров	116
9	Мой первый отчет	118
9.1	Общее описание	118
9.2	Первый шаблон	118
9.3	Первый запрос	118
9.3.1	Описание	119
9.3.2	Пример запроса	120
9.3.3	Пример запроса для сохранения результата в файл без кодирования в base64	120
9.4	Получение первого ответа	121
9.4.1	Формат возвращаемого ответа	121
9.4.2	Пример ответа	121
9.5	Завершение	121
10	Приложение	122
10.1	Быстрая установка	122

1.1 Знакомство с сервером отчетов

XSQUARE-REPORTS (**XREPORTS**) - это сервер отчетов, который обеспечивает создание документов на основе документов-шаблонов с помощью HTTP-запросов. В качестве документов-шаблонов используются файлы в форматах DOCX, XLSX, PDF. Клиентские запросы используют форматы JSON и XML.

Основными функциями сервера отчетов являются:

- генерация отчетов в форматах DOCX, XLSX, PDF;
- объединение PDF-документов;
- формирование на основе ранее сформированного PDF-документа печатной формы в формате PDF со штампом.
- заполнение PDF-форм.
- экспорт (конвертация) данных из формата XLSX и XLS (Microsoft Office Excel Binary 2003) в JSON, XML, CSV

1.2 Модули сервера отчетов

Сервер отчетов включает следующие модули:

- *Сервис формирования отчетов из документа-шаблона*
- *Генератор отчетов PDF*
- *Сервис объединения PDF-документов*
- *Сервис формирования печатной формы*
- *Сервис заполнения PDF-форм*
- *Сервис экспорта (конвертации) XLSX и XLS файлов в JSON, XML, CSV*
- *Сервис генерации кодов (QR code)*

- *Сервис конвертации документов DOCX/XLSX в формат PDF*
- *Сервис генератора запросов*

1.3 Установка сервера отчетов

Установка сервера отчетов описана в разделе *Установка*

1.4 Примеры использования

Примеры запросов и шаблонов приведены в разделе *Примеры*.

1.5 Получение первого отчета

Процесс создания простого отчета с помощью сервера отчетов описан в разделе *Мой первый отчет*.

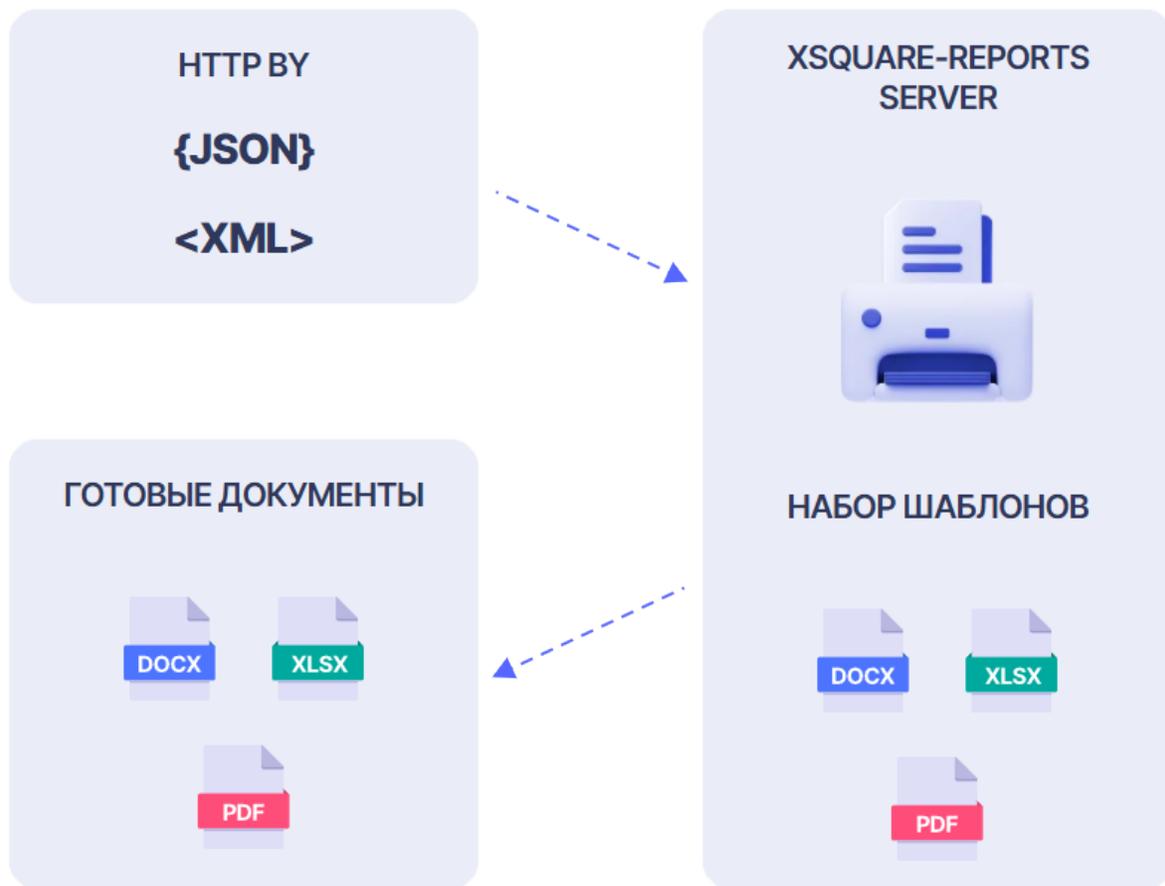
Архитектура и системные требования

2.1 Архитектура

Архитектура приложения представляет собой веб-сервер, который обрабатывает клиентские HTTP-запросы посредством формирования отчетов на основе документов-шаблонов и возвращает результат клиенту.

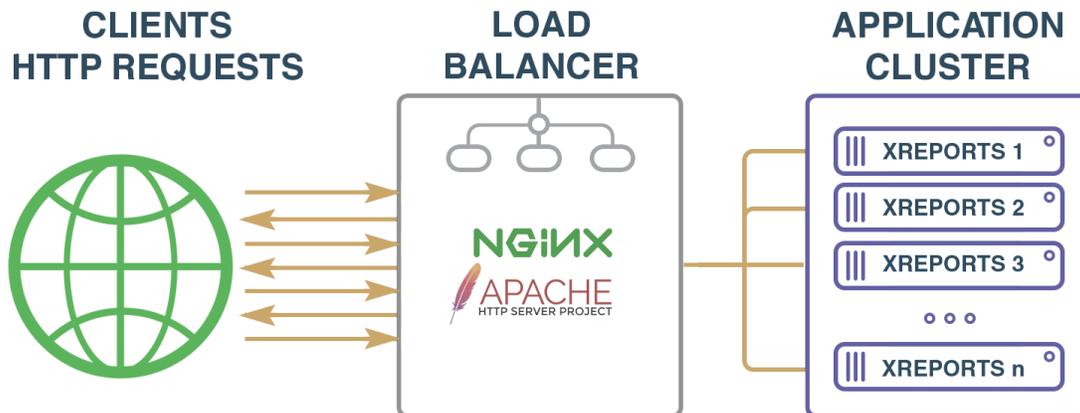
2.2 Базовая архитектура

Принципиальная или базовая архитектура приложения выглядит следующим образом для промышленного окружения:



2.3 Архитектура для высоконагруженных систем

Принципиальная или базовая архитектура приложения для высоконагруженных систем выглядит следующим образом:



2.4 Среда исполнения

Поддерживаемые архитектуры:

- x86-64
- ARM
- Loongson

Поддерживаемые ОС:

- DEB-based – любые
- RPM-based – любые
- Debian 12 - рекомендуемая

2.5 Системные требования

Минимальные системные требования:

- CPU - 1 Ядро
- RAM - 100 Мб
- HDD - 100 Мб + Логи

Модули генерации DOCX/XLSX/PDF независимы и работают полностью автономно.

Для конвертации документов DOCX/XLSX в PDF используются компоненты проекта с открытым исходным кодом Libre Office.

Установка системы виртуализации/контейнеризации, операционной системы осуществляется на усмотрение Администратора исходя из потребностей Организации.

3.1 Установка сервера отчетов

Рассмотрим установку сервера отчетов на примере ОС Debian.

Все команды следует выполнять с правами суперпользователя (root).

1. Создаем каталог для дистрибутива PGHS

```
mkdir /root/xsquare
```

переходим в каталог

```
cd /root/xsquare
```

2. Скачиваем/получаем дистрибутив в созданный каталог

```
wget https://lcdp.xsquare.ru/files/xreports/rpm_dep/6.0.0.3/deb/xsquare.  
↪xreports_6.0.0.3.deb
```

3. Устанавливаем xreports

```
dpkg -i xsquare.xreports_6.0.0.3.deb
```

4. Проверяем состояние сервера отчетов

```
curl http://localhost:8886/status
```

3.1.1 Установка шрифтов для RPM-based ОС

```
dnf install msttcore-fonts-installer  
fc-cache -fv
```

3.1.2 Установка Libre Office

Для конвертации документов DOCX/XLSX в PDF необходимо установить компоненты Libre Office.

i Примечание

Если конвертация DOCX/XLSX в PDF не требуется, то установку Libre Office можно пропустить.

1. Скачиваем дистрибутив Libre Office

```
wget https://lmdp.xsquare.ru/files/libreoffice/LibreOffice_24.8.4_Linux_
↳x86-64_deb.tar.gz
```

2. Устанавливаем Libre Office

```
apt -y install libxinerama1 libcairo2 libcups2 default-jre
tar -xvzf LibreOffice_24.8.4_Linux_x86-64_deb.tar.gz
dpkg -i ./LibreOffice_24.8.4.2_Linux_x86-64_deb/DEBS/*.deb
```

3.1.3 Файл конфигурации config.json

Для работы сервера отчетов необходимо, чтобы в директории с сервисом присутствовал файл конфигурации config.json. Файл конфигурации содержит 3 раздела:

1. Описатель App, где можно определить базовые настройки сервиса:

```
{
  "app":
  {
    "port": "8886",
    "debug": false,
    "data-directory": "",
    "save-file-directory": "/tmp/xreports",
    "save-file-path-mask": "%DOMAIN%/%YYYY%/%MM%/%DD%/%HH%/%GUID%/
↳%FILENAME%"
  }
}
```

- **port** - строка. Определяет номер сетевого порта, на котором будет запущен сервис (по умолчанию - 8886)
- **debug** - логическое значение. Включает режим отладки, при котором доступен подробный лог обработки запросов, а также происходит сохранение всех запросов и готовых документов в локальной директории **reports_debug** сервиса (аналогично действию флага *enable-debug-report-save* в свойствах запроса). Значение по умолчанию - **false**.
- **data-directory** - строка. Определяет абсолютный путь к каталогу с данными сервера отчета (assets, templates, reports_debug)
- **save-file-directory** - строка. Определяет абсолютный путь к каталогу для сохранения отчетов когда, сервер отчетов работает в режиме файлового сервера и возвращает ссылки на готовые отчеты
- **save-file-path-mask** строка. Определяет файловую маску для сохранения готовых отчетов. Маска поддерживает шаблонизированные параметры, которые можно использовать при составлении пути
 - %DOMAIN% - домен, задается в опциях запроса
 - %YYYY% - текущий год

- %MM% - текущий месяц
- %DD% - текущий день
- %HH% - текущий час
- %GUID% - случайный идентификатор
- %FILENAME% - имя файла

2. Описатель formatConversion, где определяются настройки для приложения soffice из пакета Libre Office :

```
"formatConversion":
{
  "format-conversion-dir": "",
  "soffice-max-process-count": 0,
  "soffice-path": ""
}
```

- **format-conversion-dir** - строка. Определяет директорию для хранения временных файлов (по умолчанию - «/tmp»)
- **soffice-max-process-count** - число. Определяет количество ядер процессора, которое может использовать soffice для конвертации (по умолчанию - «0», использовать все доступные ядра)
- **soffice-path** - строка. Определяет путь до исполняемого файла soffice (по умолчанию - путь, записанный в переменные окружения при установке Libre Office)

Если после установки Libre Office запуск soffice не доступен по имени - необходимо прописать полный путь в параметр «soffice-path».

Например:

```
"formatConversion":
{
  "format-conversion-dir": "",
  "soffice-max-process-count": 0,
  "soffice-path": "/opt/libreoffice24.8/program/soffice"
}
```

Для применения новых параметров конфигурации необходимо перезапустить сервер отчетов:

```
systemctl restart xsquare.xreports.service
```

Данный раздел описывает порядок поддержания работоспособности приложения и порядок загрузки компонентов.

4.1 Запуск сервера отчетов

Для загрузки `xsquare.xreports` пользователю необходимо убедиться в наличие правильно настроенного конфигурационного файла.

```
cat /usr/local/xsquare.xreports/config.json
```

Команда должна отобразить правильный конфигурационный файл, описанный в разделе “Установка”.

Далее необходимо запустить сервер отчетов выполнив команду:

```
systemctl start xsquare.xreports
```

После проверить состояние сервера приложений:

```
systemctl status xsquare.xreports
```

В случае возникновения ошибок они будут записаны в журнал. Проверить сообщения об ошибках можно выполнив команду:

```
journalctl -u xsquare.xreports
```

4.2 Проверка состояния сервера отчетов

Для проверки состояния сервера отчетов предназначен сервисный эндпоинт `/status`, при обращении к которому можно получить следующие параметры:

- статус

- версия сборки
- дата сборки
- время работы
- занятая оперативная память
- pid процесса
- параметры лицензии сервера отчетов

```
curl http://localhost:8886/status
```

4.3 Сбор информации об ошибках

Для удобства сбора информации об ошибках предназначен сервисный эндпоинт **/bugreport**, при обращении к которому, сервер отчетов отправит архив со всеми необходимыми файлами для предоставления в поддержку. В кеше хранится только последний запрос, поэтому вызвать сбор информации необходимо сразу после возникновения ошибки.

```
curl -O -J http://localhost:8886/bugreport
```

5.1 Сервис формирования отчетов из документа-шаблона

5.1.1 Общее описание сервиса

Сервис формирует отчет в формате OOXML (документы .docx и .xlsx) из документа-шаблона по HTTP-запросу в формате XML или JSON. Опционально возможна конвертация отчета в PDF.

Алгоритм работы сервиса: теги документа-шаблона заменяются конкретными данными, указанными в запросе.

Основные возможности сервиса

1. Генерация отчетов в формате DOCX на основе документа-шаблона. Могут быть сгенерированы:
 - простые отчеты в формате DOCX;
 - мульти-отчеты в формате DOCX (отчеты из одного шаблона на основе множества входных данных).
2. Генерация отчетов в формате XLSX на основе документа-шаблона.
3. Конвертация отчетов в формат PDF.

5.1.2 Работа с документами-шаблонами

Для формирования отчета необходимо создать шаблон в формате DOCX или XLSX.

Расположение файлов шаблонов

Документы шаблонов в форматах DOCX и XLSX хранятся в локальной директории *templates*, расположенной в директории приложения сервиса или в директории заданной опцией *data-directory*.

Формат файлов шаблонов

Для создания документов-шаблонов необходимо использовать текстовый процессор (MS Word/Excel, LibreOffice Writer/Calc, Google Docs и т.п.).

Важно:

- Шаблоны для формирования документов в формате DOCX должны быть сохранены в формате DOCX.
- Шаблоны для формирования документов в формате XLSX должны быть сохранены в формате XLSX или XLSM.
- Шаблоны для последующей конвертации в PDF должны быть сохранены в формате DOCX (желательно с использованием Libre Office, так как результирующий PDF будет полностью соответствовать визуальному представлению документа в Libre Office).

Использование тегов в шаблоне

Шаблоны документов могут содержать теги, которые будут заменены входными данными из запроса. Тег задается в квадратных скобках. Пример: [задолженность]

В теле запроса входные данные для тегов содержатся в **input-data**. В запросе скобки не указываются (см. примеры из пункта “Структура запроса” подраздела “Формирование отчета в формате DOCX” и из пункта “Структура запроса” подраздела “Формирование отчета в формате XLSX”).

Работа с изображениями

Принцип работы с изображениями:

- в шаблон добавляется изображение, предназначенное для последующей замены;
- при формировании документа оно заменяется файлом изображения, который передается в запросе. В запросе передается новое содержимое файла изображения, закодированное в BASE64, либо данные для динамического создания изображения (например QR code)

Теги изображений в шаблоне задаются в свойстве изображения «Замещающий текст» («Alt text») без квадратных скобок.

Поддерживаемые форматы: JPEG, PNG.

Ограничения для изображений

1. Формат изображения (JPEG, PNG) в запросе должен совпадать с форматом замещаемого изображения в шаблоне.
2. Если необходима подмена нескольких изображений шаблона разными изображениями из входных данных, в шаблоне изображения также должны быть разными. Связано с тем, что изображение в теле документа ссылается на его контент (файл), откуда следует, что при замене контента в документе изменятся все изображения, которые на этот контент ссылаются.

Примеры шаблонов

С примерами шаблонов можно ознакомиться в архиве примеров [архиве примеров](#)

5.1.3 Формирование отчета в формате DOCX

Описание сервиса

Наименование сервиса	Сервис формирования отчетов из документа-шаблона в формате DOCX
Путь к сервису	Простые отчеты в формате DOCX: <ul style="list-style-type: none"> • [host]:[port]/word_report_json • [host]:[port]/word_report_xml Мульти-отчеты в формате DOCX: <ul style="list-style-type: none"> • [host]:[port]/word_multi_report_1_N_json • [host]:[port]/word_multi_report_1_N_xml
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON или XML. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт файл документа в формате DOCX, закодированный в base64. При конвертации отчета в формат PDF сервис возвращает файл документа в формате PDF, закодированный в base64.
Назначение	Сервис предназначен для формирования отчетов из документа-шаблона в формате DOCX . Возможна конвертация в PDF.

Структура тела запроса

Тело запроса содержит объект в формате JSON или XML, который включает:

1. Описатель шаблона.
2. Входные данные для подстановки в шаблон вместо тегов.
3. Опции запроса.
4. Формат ответа.

Описатель шаблона

Элемент (объект) описателя шаблона `template` содержит:

- **uri** - строка. Задаёт расположение файла документа-шаблона в зависимости от того, как трактуется параметр **id**.

Поддерживаемые значения:

- **local** - расположение шаблона в каталоге `templates`.
 - **embedded** - шаблон включен в запрос в виде закодированной base64 строки. Значение строки должно быть записано в параметре `value`.
 - **remote** - шаблон расположен на удаленном файловом сервере. Ссылка на шаблон должна быть указана в параметре `value`.
- **id** - строка. Идентификатор шаблона. Путь к файлу документа-шаблона относительно директории `templates`. Также используется для записи файла отчета в отладочном режиме и для идентификации запроса в логге.
 - **value** - строка. Значение используется в режимах `embedded` и `remote`.

Примеры для формата XML:

```
<template uri="local" id="Информационное письмо"/>
```

```
<template uri="embedded" id="Письмо" value="UESDBBQACAA ... nJlbH0tk1KA0EMh"/>
```

```
<template uri="remote" id="Письмо" value="http://directlink/templates/letter.docx"/>
```

Пример для формата JSON:

```
"template":
{
  "uri": "local",
  "id": "template_example_1"
}
```

```
"template":
{
  "uri": "embedded",
  "id": "Письмо",
  "value": "UESDBBQACAgIAPyFilcAAAAAAAAAAAAAAAA...LAAAX3JlbHMvLnJlbH0tk1KA0EMh"
}
```

```
"template":
{
  "uri": "remote",
  "id": "Письмо",
  "value": "http://directlink/templates/letter.docx"
}
```

Входные данные

Элемент (объект) входных данных **input-data** содержит:

- Простые строковые данные для подстановки в шаблон вместо тегов.
- Значения для условных выражений.
- Описатели таблиц.
- Описатели списков.
- Данные для подстановки в теги изображений.
- Описатели блоков.

i Примечание

Отличие структуры данных для формата JSON от XML в том, что вместо XML-элементов данные представлены как объекты и списки объектов. Имена тегов задаются именами полей объектов. Списки, таблицы, изображения являются дочерними объектами по отношению к объекту **input-data**.

i Примечание

Для мульти-отчетов используется список (массив) **input-data-array**, содержащий элементы (объекты), по структуре совпадающие с **input-data**. Подробнее с мульти-отчетами можно ознакомиться в подразделе “Работа с документом мульти-отчета”.

Простые строковые данные для подстановки в шаблон вместо тегов

Для формата XML

Элемент **tags** содержит список дочерних элементов **tag** с атрибутом **name** и нужным конкретным значением, на который тег в шаблоне будет заменен при генерации отчета. Атрибут **name** совпадает с тегом в шаблоне.

Пример:

```
<input-data><tags>
  <tag name="ORGANIZATION">АО «xxxxxyyyy»</tag>
</tags></input-data>
```

Для формата JSON

Имена тегов задаются именами объектов внутри **input-data**.

Пример:

```
"input-data":
{
  "ORGANIZATION": "АО «xxxxxyyyy»"
}
```

Описание таблиц

Для формата XML

Элемент описателей таблиц **tables** содержит список дочерних элементов **table** с атрибутом **name**. Атрибут **name** совпадает с тегом в шаблоне. Тег в шаблоне будет заменен на таблицу, созданную согласно описанию таблицы в запросе.

Пример:

```
<input-data>
  <tables>
    <table name="TABLE-FORMATTED">
      <rows>
        <row>
          <cell tag="номер_пп">1</cell>
          <cell tag="номер_договора">Пример номера 1</cell>
          <cell tag="район">Пример района 1</cell>
          <cell tag="предприятие">Пример предприятия 1</cell>
          <cell tag="дата_отключения">Пример даты 1</cell>
          <cell tag="адрес">Пример адреса 1</cell>
          <cell tag="комплексное_поле">Пример суммы 1</cell>
        </row>
      </rows>
    </table>
    <table name="TABLE-NO-FORMAT">
      <header>
        <cell>№ п/п</cell>
        <cell>№ дела в Арбитражном суде</cell>,
        <cell>Период задолженности</cell>,
        <cell>Основной долг, руб. с НДС</cell>,
        <cell>Проценты, руб.</cell>,>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

        <cell>Госпошлина, руб.</cell>,
        <cell>Неустойка по решению суда</cell>,
        <cell>Общая сумма погашения, руб.</cell>,
        <cell>Срок погашения, не позднее</cell>
    </header>
    <rows>
        <row>
            <cell>1</cell>,
            <cell>A12-1234/2030</cell>,
            <cell>Октябрь 2017</cell>,
            <cell>12 345,58</cell>,
            <cell/>
            <cell>23 456,00</cell>,
            <cell>78 912,41</cell>,
            <cell/>
            <cell>31.08.2019</cell>
        </row>
    </rows>
</table>
</tables>
</input-data>

```

Для формата JSON

Описатели таблиц являются дочерними объектами по отношению к объекту **input-data**. Именами полей объектов таблиц совпадают с тегами в шаблоне.

Пример:

```

"input-data":
{
  "TABLE-NO-FORMAT":
  {
    "header":
    [
      "№ п/п",
      "№ дела в Арбитражном суде",
      "Период задолженности",
      "Основной долг, руб. с НДС",
      "Проценты, руб.",
      "Госпошлина, руб.",
      "Неустойка по решению суда",
      "Общая сумма погашения, руб.",
      "Срок погашения, не позднее"
    ],
    "rows":
    [
      [
        "1",
        "A12-1234/2030",
        "Октябрь 2017",
        "12 345,58",
        null,

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "23 456,00",
    "78 912,41",
    null,
    "31.08.2019"
  ]
],
"TABLE-FORMATTED":
{
  "rows":
  [
    {
      "номер_пп": "1",
      "номер_договора": "Пример номера 1",
      "район": "Пример района 1",
      "предприятие": "Пример предприятия 1",
      "дата_отключения": "Пример даты 1",
      "адрес": "Пример адреса 1",
      "комплексное_поле": "Пример суммы 1"
    }
  ]
}
}

```

Описатели списков

Для формата XML

Элемент описателей списков **lists** содержит список дочерних элементов **list** с атрибутом name. Атрибут name совпадает с тегом в шаблоне. Тег в шаблоне будет заменен параграфом, отформатированным как список с данными элементов из описателя списка в запросе.

Пример:

```

<input-data>
  <lists>
    <list name="BULLET_LIST">
      <items>
        <item>bullet item 1</item>
        <item>bullet item 2</item>
        <item>bullet item 3</item>
      </items>
    </list>
    <list name="NUMBERED_LIST">
      <items>
        <item>numbered item 1</item>
        <item>numbered item 2</item>
        <item>numbered item 3</item>
      </items>
    </list>
  </lists>
</input-data>

```

Для формата JSON

Описатели списков являются массивами внутри объекта **input-data**. Именами полей объектов списков совпадают с тегами в шаблоне.

Пример:

```
"input-data":
{
  "BULLET_LIST":
  [
    "bullet item 1",
    "bullet item 2",
    "bullet item 3"
  ],
  "NUMBERED_LIST":
  [
    "numbered item 1",
    "numbered item 2",
    "numbered item 3"
  ]
}
```

Данные для подстановки в теги изображений

Для формата XML

Элемент **images** содержит список дочерних элементов **image** с атрибутом **name**. Атрибут **name** совпадает с тегом в шаблоне. Изображения в шаблоне будут заменены на данные изображений из запроса.

Пример:

```
<input-data>
  <images>
    <image name="IMAGE-1-JPEG">/9j/4AAQSkZJ ... CigAooAKKACigD//Z</image>
    <image name="IMAGE-2-JPEG">/9j/4AAQSkZJ ... AooAKKACigAooA//9k=</image>
  </images>
</input-data>
```

Для формата JSON

Данные для подстановки в теги изображений находятся внутри объекта **images**, который является дочерним для объекта **input-data**.

Пример:

```
"input-data":
{
  "images":
  {
    "IMAGE-1-JPEG":
    {
      "data": "/9j/4AAQSkZJ ... CigAooAKKACigD//Z"
    },
    "IMAGE-2-JPEG":
    {
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "data": "/9j/4AAQSkZJ ... AooAKKACigAooA//9k="
  }
}

```

Также описатель **data** может содержать данные для генерации кодов (например QR code). Подробнее [Работа с графическими кодами](#).

Описатели блоков

Для формата XML

Элемент описателей блоков **blocks** содержит список дочерних элементов **block** с атрибутом **block-template** (совпадает с именем шаблона блока в документе-шаблоне). Блок имеет те же элементы, что и **input-data**, кроме элементов изображений и блоков. Шаблоны блоков в документе-шаблоне будут использованы для вставки в документ произвольного количества экземпляров блоков с разными наборами данных. Места вставки определяются тегами экземпляра шаблона в документе-шаблоне. Детали в подразделе “Работа с документом → Блоки”.

Для формата JSON

Список (массив) описателей блоков **blocks** находится внутри объекта **input-data**.

Пример:

```

"input-data":
{
  "blocks":
  [
    {
      "block-template": "block1",
      "data":
      {
        "таблица1":
        {
          "rows":
          [
            {
              "дней": "670",
              "доля": "130",
              "пени": "12564,46",
              "дата_с": "11.02.2021",
              "период": "Реализация 0000165 от 31.01.2021 23:59:59",
              "ставка": "7,5",
              "дата_по": "12.12.2022",
              "формула": "32505,07 * 670 * 0.08 * 1/130",
              "задолженность": "32505,07"
            }
          ]
        }
      }
    },
    "сумма_пени": "12345,67",
    "сумма_долга": "78910,23",
    "номер_договора": "1"
  }
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

},
{
  "block-template": "block2",
  "data":
  {
    "таблица1":
    {
      "rows":
      [
        {
          "дней": "670",
          "пени": "12564,46",
          "дата_с": "11.02.2021",
          "период": "Реализация 0000165 от 31.01.2021 23:59:59",
          "ставка": "7,5",
          "дата_по": "12.12.2022",
          "задолженность": "32505,07"
        }
      ]
    },
    "сумма_пени": "76543,21",
    "сумма_долга": "987654,32",
    "номер_договора": "2"
  }
}
]
}

```

Опции запроса

Элемент (объект) опций запроса **options** содержит:

- **output-mode** - строка. Определяет форму представления результата.

Поддерживаемые значения:

- **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
- **binary** - результат будет представлен в виде бинарного потока.
- **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

Примечание: флаг `enable-binary-output` считается устаревшим, но для поддержки старых отчетов, продолжает использоваться.

- **report-format** - строка. Указывает формат файла отчета, если он отличен от формата, упомянутого в url запроса. Единственное непустое поддерживаемое значение: **pdf**. Значение по умолчанию: пустая строка. См. пункт “Конвертация отчета в PDF” ниже.
- **formatting** - объект опций форматирования. Поддерживается только для запросов **word**.
 - Поле объекта: **tables** - объект опций форматирования таблиц.
 - Поле объекта: **enable-cells-auto-merge** - логическое значение. Указывает, объединять ли подряд идущие ячейки с одинаковым значением в одном столбце по вертикали. Значение по умолчанию

- **true**. Данный параметр может использовать как глобально, так и для каждой таблицы в отдельности.

- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **enable-debug-report-save** - логическое значение. Указывает создавать ли копию отчета документа и копию запроса (имя файла - идентификатор шаблона) в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.

Пример для формата XML:

```
<options>
  <enable-debug-report-save>true</enable-debug-report-save>
  <output-mode>binary</output-mode>
  <formatting>
    <tables>
      <enable-cells-auto-merge>true</enable-cells-auto-merge>
    </tables>
  </formatting>
  <report-format>pdf</report-format>
</options>
```

Пример для формата JSON:

```
"options":
{
  "enable-debug-report-save": true,
  "output-mode": "base64",
  "formatting":
  {
    "tables":
    {
      "enable-cells-auto-merge": true
    }
  },
  "clean-data-by-null": true,
  "report-format": "pdf"
}
```

Формат ответа

Элемент (объект) формата ответа **response-format** может принимать значения **json** и **xml**.

Подробнее про то, как задается формат ответа можно прочитать в подразделе “Структура ответа”.

Пример для формата XML:

```
<response-format value="xml"/>
```

Пример для формата JSON:

```
{"response-format": "json"}
```

Примеры запросов

Примеры для формата XML

Пример №1

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <template uri="..." id="..." />
  <input-data>
    <tags>...</tags>
    <tables>...</tables>
    <lists>...</lists>
    <images>...</images>
    <blocks>...</blocks>
  </input-data>
  <options>...</options>
  <response-format value="..." />
</request>>
```

Пример №2

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <template uri="local" id="template_example_1" />
  <input-data>
    <tags>
      <tag name="ORGANIZATION">АО «xxxxxyyyyy»</tag>
      <tag name="TAG_IN_HEADER_TEST">Пример верхнего колонтитула</tag>
      <tag name="TAG_IN_FOOTER_TEST">Пример нижнего колонтитула</tag>
      <tag name="CONDITIONAL_TAG_TRUE">true</tag>
      <tag name="CONDITIONAL_TAG_FALSE">false</tag>
    </tags>
    <tables>
      <table name="TABLE-FORMATTED">
        <rows>
          <row>
            <cell tag="номер_пп">1</cell>
            <cell tag="номер_договора">Пример номера 1</cell>
            <cell tag="район">Пример района 1</cell>
            <cell tag="предприятие">Пример предприятия 1</cell>
            <cell tag="дата_отключения">Пример даты 1</cell>
            <cell tag="адрес">Пример адреса 1</cell>
            <cell tag="комплексное_поле">Пример суммы 1</cell>
          </row>
        </rows>
      </table>
      <table name="TABLE-NO-FORMAT">
        <header>
          <cell>№ п/п</cell>
          <cell>№ дела в Арбитражном суде</cell>,
          <cell>Период задолженности</cell>,
          <cell>Основной долг, руб. с НДС</cell>,
          <cell>Проценты, руб.</cell>,</table>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

        <cell>Госпошлина, руб.</cell>,
        <cell>Неустойка по решению суда</cell>,
        <cell>Общая сумма погашения, руб.</cell>,
        <cell>Срок погашения, не позднее</cell>
    </header>
    <rows>
        <row>
            <cell>1</cell>,
            <cell>A12-1234/2030</cell>,
            <cell>Октябрь 2017</cell>,
            <cell>12 345,58</cell>,
            <cell/>
            <cell>23 456,00</cell>,
            <cell>78 912,41</cell>,
            <cell/>
            <cell>31.08.2019</cell>
        </row>
    </rows>
</table>
</tables>
<images>
    <image name="IMAGE-1-JPEG">/9j/4AAQSkZJR ... oAKKACigAooAKKACigD//Z</image>
    <image name="IMAGE-2-JPEG">/9j/4AAQSkZJR ... CigAooAKKACigAooA//9k=</image>
</images>
</input-data>
<options>
    <enable-debug-report-save>true</enable-debug-report-save>
    <formatting>
        <tables>
            <enable-cells-auto-merge>true</enable-cells-auto-merge>
        </tables>
    </formatting>
</options>
</request>

```

Пример для формата JSON

```

{
  "template":
  {
    "uri": "local",
    "id": "template_example_1"
  },
  "input-data":
  {
    "ORGANIZATION": "АО «xxxxxyyy»",
    "TAG_IN_HEADER_TEST": "Пример верхнего колонтитула",
    "TAG_IN_FOOTER_TEST": "Пример нижнего колонтитула",
    "CONDITIONAL_TAG_TRUE": "true",
    "CONDITIONAL_TAG_FALSE": "false",
    "TABLE-NO-FORMAT":
    {

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

"header":
[
  "№ п/п",
  "№ дела в Арбитражном суде",
  "Период задолженности",
  "Основной долг, руб. с НДС",
  "Проценты, руб.",
  "Госпошлина, руб.",
  "Неустойка по решению суда",
  "Общая сумма погашения, руб.",
  "Срок погашения, не позднее"
],
"rows":
[
  [
    "1",
    "A12-1234/2030",
    "Октябрь 2017",
    "12 345,58",
    null,
    "23 456,00",
    "78 912,41",
    null,
    "31.08.2019"
  ]
]
},
"TABLE-FORMATTED":
{
  "rows":
  [
    {
      "номер_пп": "1",
      "номер_договора": "Пример номера 1",
      "район": "Пример района 1",
      "предприятие": "Пример предприятия 1",
      "дата_отключения": "Пример даты 1",
      "адрес": "Пример адреса 1",
      "комплексное_поле": "Пример суммы 1"
    }
  ]
},
"images":
{
  "IMAGE-1-JPEG":
  {
    "data": "/9j/4AAQSkZJR ... oAKKACigAooAKKACigD//Z"
  },
  "IMAGE-2-JPEG":
  {
    "data": "/9j/4AAQSkZJR ... CigAooAKKACigAooA//9k="
  }
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    }
  },
  "options":
  {
    "enable-debug-report-save": true,
    "formatting":
    {
      "tables":
      {
        "enable-cells-auto-merge": true
      }
    }
  }
}

```

Примеры запросов в форматах XML и JSON также можно посмотреть в [архиве примеров](#).

Пример вызова сервиса

Для формата XML

```

curl --request POST --data-binary "@templates/examples/docx/example_from_xml.xml" \
  ->http://localhost:8886/word_report_xml

```

Для формата JSON

```

curl --request POST --data-binary "@templates/examples/docx/example_from_json.json" \
  ->http://localhost:8886/word_report_json

```

Структура ответа

В зависимости от **output-mode** ответ сервиса может быть либо потоком бинарных данных, либо ответом в формате JSON/XML.

Формат ответа (JSON или XML) может быть задан двумя способами:

1. в HTTP заголовке Ассерта. Поддерживаемые значения: **application/json** и **application/xml**.
2. в теле запроса в элементе (объекте) **response-format**. Поддерживаемые значения: **json** и **xml**.

Приоритет имеет формат, указанный в теле запроса.

Ответ сервиса содержит объект в формате JSON или XML, который включает:

1. Описание ошибки (код ошибки, сообщение об ошибке). В случае успешного ответа сервиса возвращается значение **null**.
2. Результат (закодированный в base64 файл документа отчета в формате docx/pdf либо ссылка на файл документа отчета).

Формат ответа

```

{
  "error":
  {
    "code": "",

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "message": ""
  },
  "result": "[base64 encoded docx/pdf file] or [url]"
}

```

Пример ответа

```

{
  "error": null,
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="
}

```

Пример для отчета предоставляемого по ссылке**Часть запроса:**

```

{
  "options":
  {
    "output-mode": "url",
    "domain-name": "test_domain",
    "file-name": "name_by_options"
  },
  "template":
  {
    "id": "report_by_url",
    "uri": "embedded",
    "value": "UESDBBQABgAIAAA..."
  }
}

```

Ответ:

```

{
  "error": null,
  "result": "test_domain/2025/02/17/13/521b84da-a837-40d3-a145-51defae86235/name_by_
↪options.docx"
}

```

Работа с документом**Общая информация**

Для документов в формате DOCX доступны:

- теги, которые будут заменены входными данными из запроса;
- таблицы;
- списки;
- изображения;
- условные выражения;

- создание единого отчета из одного шаблона на основе множества входных данных (мульти-отчеты);
- блоки.

Работа с таблицами

Поддерживаются следующие типы таблиц в шаблоне:

1. Таблица без форматирования. Задается только тегом. Таблица генерируется на ширину страницы с равной шириной столбцов.
2. Таблица с форматированием. Задается таблицей со следующими параметрами:
 - опциональный произвольный заголовок.
 - первая ячейка строки, следующей за заголовком содержит тег таблицы. Эта строка удаляется при генерации.
 - строка следующая за строкой тега содержит теги столбцов, их форматирование будет применено к сгенерированным строкам. Эта строка удаляется при генерации.
 - опциональные фиксированные строки с любыми тегами (источник данных замены - основные теги).

Примечание

таблицы в колонтитулах документа поддерживаются.

Управление слиянием ячеек

Базовый алгоритм слияния - автоматическое слияние ячеек с одинаковым содержимым по вертикали.

Изменить алгоритм слияния можно при помощи опции запроса **formatting.tables.enable-cells-auto-merge**.

Для запросов в формате JSON поддерживаются расширенные опции вертикального и горизонтального слияния.

Объект, описывающий строку таблицы, имеет опциональный объект **formatting**. Пример:

```
"formatting": {
  "column_tag_value": {
    "vertical_merge": "restart",
    "column_span": 2
  }
}
```

Имена полей этого объекта являются названиями столбцов таблицы, для которых нужно применить опцию слияния. Значения полей: объект опции слияния. Данный объект имеет поля:

- **vertical_merge** - строка. Управляет вертикальным слиянием ячейки. Поддерживаемые значения: "restart" - предотвращает слияние соседней ячейкой, расположенной в предыдущей строке того же столбца; "continue" - форсирование слияния с ячейкой выше по столбцу; "unset" - опция слияния будет унаследована от ячейки выше по столбцу;
- **column_span** - целое число. Управляет горизонтальным слиянием ячейки. Значение задает количество ячеек для объединения справа от текущей ячейки (включительно).

Примечание

это прямое управление настройками слияния в документе. Чтобы быстро понять, какие значения нужно выставить тем или иным ячейкам, нужно сперва настроить желаемое поведение в текстовом редакторе, сохранить docx-документ, разархивировать его как zip архив и посмотреть, какие значения применяются в файле document.xml.

Списки

Поддерживаются одноуровневые списки типа «маркеры» и с нумерацией.

Пример для формата XML:

```
<lists>
<list name="BULLET_LIST">
  <items>
    <item>bullet item 1</item>
    <item>bullet item 2</item>
    <item>bullet item 3</item>
  </items>
</list>
</lists>
```

Подробный пример запроса с использованием списков (lists.json) и шаблон к нему (lists.docx) можно найти в *архиве примеров*.

Добавление элементов возможно в произвольное место списка в шаблоне. В случае пустого списка во входных данных, список удаляется из отчета.

Условные выражения

Части текста шаблона могут быть исключены из отчета в зависимости от истинности условия в условном теге.

Формат условного тега:

```
[#условие]условный текст шаблона[/условие]
```

Открывающий тег начинается с символа #, за которым следует условие. Закрывающий тег начинается с символа /, за которым следует условие открывающего тега.

Примечание

Условные выражения в колонтитулах не поддерживаются.

В запросе данные для подстановки в теги условных выражений находятся внутри элемента (объекта) **input-data**. В качестве значения условных блоков в запросе поддерживаются булевы константы **true/false** и их текстовые аналоги «**true**»/«**false**».

Пример для формата XML:

```
<input-data>
<tags>
  <tag name="CONDITIONAL_TAG_TRUE">true</tag>
  <tag name="CONDITIONAL_TAG_FALSE">>false</tag>
</tags>
</input-data>
```

Пример для формата JSON:

```
"input-data":
{
  "ORGANIZATION": "АО «ххххуууу»",
  "TAG_IN_HEADER_TEST": "Пример верхнего колонтитула",
  "TAG_IN_FOOTER_TEST": "Пример нижнего колонтитула",
  "CONDITIONAL_TAG_TRUE": "true",
  "CONDITIONAL_TAG_FALSE": "false"
}
```

Подробный пример запроса с использованием условных выражений (`example_from_json.json`) и шаблон к нему (`example_from_json.docx`) можно найти в [архиве примеров](#).

Блоки

Блок - это фрагмент отчета, содержащий текст/таблицы/списки (все, кроме изображений, в текущей версии). Создается из шаблона-блока (определенного фрагмента документа-шаблона) произвольное количество раз в произвольном месте шаблона с разными наборами входных данных.

Шаблон блока в документе-шаблоне определяется тегами

[block-template:произвольное имя шаблона блока]

Шаблоны блока полностью удаляются из документа-шаблона перед заменой тегов на входные данные.

Места вставки блоков определяются тегами экземпляров шаблонов блока

[block-instances:список имен шаблонов блоков через запятую]

документа-шаблона.

Контент шаблона блока вставляется в документ (с заменой тегов в нем) по месту нахождения тега экземпляра шаблона.

Каждый тег экземпляра шаблона блока указывает, какие именно шаблоны нужно взять за основу для создания блока в отчете (другими словами, инстанцировать шаблон блока). При этом рассматриваются все входные данные блоков по порядку. При замене тегов в шаблоне блока все значения тегов берутся только из объекта **data** блока (актуально для текущей версии сервиса отчетов).

В запросе данные блоков определяются массивом **blocks**, который содержит объекты **block** с полями **block-template** (совпадает с именем шаблона блока в документе-шаблоне) и полем объекта **data**. Объект **data** имеет те же дочерние объекты, что и объект **input-data**, кроме изображений и блоков.

Для удаления блоков необходимо передать **null** в качестве значения поля **data**.

Пример:

```
"blocks":
[
  {
    "block-template": "block1",
    "data":
    {
      "таблица1":
      {
        "rows":
        [
          {}
        ]
      }
    }
  }
]
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
    ],
    "сумма_пени": "12345,67",
    "сумма_долга": "78910,23",
    "номер_договора": "1"
  }
},
{
  "block-template": "block2",
  "data":
  {
    "таблица1":
    {
      "rows":
      [
        {}
      ]
    },
    "сумма_пени": "76543,21",
    "сумма_долга": "987654,32",
    "номер_договора": "2"
  }
},
{
  "block-template": "block3",
  "data":
  {
    "номер_договора": "1",
    "название_договора": "Название договора 1",
    "дата_с": "01.01.2000",
    "дата_по": "01.01.2100"
  }
},
{
  "block-template": "block3",
  "data":
  {
    "номер_договора": "2",
    "название_договора": "Название договора 2",
    "дата_с": "02.02.2020",
    "дата_по": "02.02.2080"
  }
},
{
  "block-template": "block3",
  "data": null
}
]
```

Подробный пример запроса с использованием блоков (`example_multiblocks.json`) и шаблон к нему (`example_multiblocks.docx`) можно найти в [архиве примеров](#).

Работа с мульти-отчетами

Общая информация

Доступно создание единого отчета из одного шаблона на основе множества входных данных.

Примечание

- В текущей версии не поддерживаются нумерованные списки.
- В текущей версии не поддерживается создание единого отчета из множества шаблонов.
- Теги в колонтитулах нужно использовать разумно, так как нет возможности иметь собственные колонтитулы для каждого отчета.
- Замена изображений поддерживается только глобально. Во всех подотчетах будет набор изображений, примененный последним. Это ограничение текущей версии сервиса.

Примеры запросов

Пример для формата XML

```
<request>
  <template uri="local" id="multi_report_1-N"/>
  <input-data-array>
    <input-data>
      <tags>
        <tag name="абонент">абонент1</tag>
        <tag name="адрес_абонента">адрес_абонента1</tag>
        <tag name="период">период1</tag>
        <tag name="договоры">договоры1</tag>
        <tag name="общая_сумма_долга">общая_сумма_долга1</tag>
        <tag name="сумма_задолженности1">сумма_задолженности1_1</tag>
        <tag name="сумма_задолженности2">сумма_задолженности2_1</tag>
        <tag name="дата_пени">дата_пени1</tag>
      </tags>
      <tables>
        <table name="TABLE-FORMATTED">
          <rows>
            <row>
              <cell tag="номер_пп">1</cell>
              <cell tag="номер_договора">Пример номера 1</cell>
              <cell tag="район">Пример района 1</cell>
              <cell tag="предприятие">Пример предприятия 1</cell>
              <cell tag="дата_отключения">Пример даты 1</cell>
              <cell tag="адрес">Пример адреса 1</cell>
              <cell tag="комплексное_поле">Пример суммы 1</cell>
            </row>
          </rows>
        </table>
      </tables>
    </input-data>
    <input-data>
      <tags>
        <tag name="абонент">абонент2</tag>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

<tag name="адрес_абонента">адрес_абонента2</tag>
<tag name="период">период2</tag>
<tag name="договоры">договоры2</tag>
<tag name="общая_сумма_долга">общая_сумма_долга2</tag>
<tag name="сумма_задолженности1">сумма_задолженности1_2</tag>
<tag name="сумма_задолженности2">сумма_задолженности2_2</tag>
</tags>
<tables>
  <table name="TABLE-FORMATTED">
    <rows>
      <row>
        <cell tag="номер_пп">1</cell>
        <cell tag="номер_договора">Пример номера 3</cell>
        <cell tag="район">Пример района 3</cell>
        <cell tag="предприятие">Пример предприятия 3</cell>
        <cell tag="дата_отключения">Пример даты 3</cell>
        <cell tag="адрес">Пример адреса 3</cell>
        <cell tag="комплексное_поле">Пример суммы 3</cell>
      </row>
    </rows>
  </table>
</tables>
</input-data>
</input-data-array>
<options>
  <enable-debug-report-save>true</enable-debug-report-save>
  <formatting>
    <tables>
      <enable-cells-auto-merge>true</enable-cells-auto-merge>
    </tables>
  </formatting>
</options>
</request>

```

Пример для формата JSON

```

{
  "template":
  {
    "uri": "local",
    "id": "template_example_1"
  },
  "input-data-array":
  [
    {
      "ORGANIZATION": "АО «xxxxxyyy1»",
      "CLINAME": "МУНИЦИПАЛЬНОЕ ПРЕДПРИЯТИЕ ГОРОДА \"xxxxxxxxx\"",
      "STRNUMBER": "1234",
      "TABLE-NO-FORMAT":
      {
        "header":
        [

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "№ п/п1",
    "№ дела в Арбитражном суде1",
    "Период задолженности1",
    "Основной долг, руб. с НДС1",
    "Проценты, руб.1",
    "Госпошлина, руб.1",
    "Неустойка по решению суда1",
    "Общая сумма погашения, руб.1",
    "Срок погашения, не позднее1"
  ],
  "rows":
  [
    [
      "1_1",
      "A12-1234/2030_1",
      "Октябрь 2017_1",
      "12 345,58_1",
      null,
      "23 456,00_1",
      "78 912,41_1",
      null,
      "31.08.2019_1"
    ]
  ]
},
"TABLE-FORMATTED":
{
  "rows":
  [
    {
      "номер_пп": "1_1",
      "номер_договора": "Пример номера 1_1",
      "район": "Пример района 1_1",
      "предприятие": "Пример предприятия 1_1",
      "дата_отключения": "Пример даты 1_1",
      "адрес": "Пример адреса 1_1",
      "комплексное_поле": "Пример суммы 1_1"
    }
  ]
},
"images":
{
  "IMAGE-1-JPEG":
  {
    "data": "/9j/4AAQSkZJRgABAQEAYABgAAD...AKKACigD//Z"
  },
  "IMAGE-2-JPEG":
  {
    "data": "/9j/4AAQSkZJRgABAQEAYABg...oAKKACigAooA//9k="
  }
}
},

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

{
  "ORGANIZATION": "АО «хххххуууу_2»",
  "CLINAME": "ПРЕДПРИЯТИЕ ГОРОДА "ПАССАЖИРСКИЙ АВТОМОБИЛЬНЫЙ ТРАНСПОРТ_2"",
  "STRNUMBER": "1234_2",
  "RESTPENY": null,
  "TAG_IN_HEADER_TEST": "Пример верхнего колонтитула_2",
  "TAG_IN_FOOTER_TEST": "Пример нижнего колонтитула_2",
  "CONDITIONAL_TAG_TRUE": "true",
  "CONDITIONAL_TAG_FALSE": "false",
  "TABLE-NO-FORMAT":
  {
    "header":
    [
      "№ п/п_2",
      "№ дела в Арбитражном суде_2",
      "Период задолженности_2",
      "Основной долг, руб. с НДС_2",
      "Проценты, руб._2",
      "Госпошлина, руб._2",
      "Неустойка по решению суда_2",
      "Общая сумма погашения, руб._2",
      "Срок погашения, не позднее_2"
    ],
    "rows":
    [
      [
        "1_2",
        "A12-1234/2030_2",
        "Октябрь 2017_2",
        "12 345,58_2",
        null,
        "23 456,00_2",
        "78 912,41_2",
        null,
        "31.08.2019_2"
      ]
    ]
  },
  "TABLE-FORMATTED":
  {
    "rows":
    [
      {
        "номер_пп": "1_2",
        "номер_договора": "Пример номера 1_2",
        "район": "Пример района 1_2",
        "предприятие": "Пример предприятия 1_2",
        "дата_отключения": "Пример даты 1_2",
        "адрес": "Пример адреса 1_2",
        "комплексное_поле": "Пример суммы 1_2"
      }
    ]
  }
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    }
  }
],
"options":
{
  "enable-debug-report-save": true,
  "formatting":
  {
    "tables":
    {
      "enable-cells-auto-merge": true
    }
  }
}
}
}

```

Подробный пример запроса для формирования мульти-отчета (`multi_report_1-N_json.json`) и шаблон к нему (`multi_report_1-N_json.docx`) можно найти в *архиве примеров*.

Примеры вызова сервиса

Для формата XML

```
curl --request POST --data-binary "@templates/examples/docx/multi_report_1-N_xml.xml" \
→ http://localhost:8886/word_multi_report_1_N_xml
```

Для формата JSON

```
curl --request POST --data-binary "@templates/examples/docx/multi_report_1-N_json.json" \
→ " http://localhost:8886/word_multi_report_1_N_json
```

5.1.4 Формирование отчета формате XLSX

Описание сервиса

Наименование сервиса	Сервис формирования мульти-отчетов из документа-шаблона в формате XLSX
Путь к сервису	<ul style="list-style-type: none"> • <code>[host]:[port]/excel_report_json</code> • <code>[host]:[port]/excel_report_xml</code> • <code>[host]:[port]/excel_report_json/v2</code>
Метод	POST
Параметры	Тело запроса должно содержать JSON или XML объект. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт файл документа в формате XLSX, закодированный в BASE64. При конвертации отчета в формат PDF сервис возвращает файл документа в формате PDF, закодированный в BASE64.
Назначение	Сервис предназначен для формирования мульти-отчетов из документа-шаблона в формате XLSX . Возможна конвертация в PDF.

Структура тела запроса

Тело запроса содержит объект в формате JSON или XML, который включает:

1. Описатель шаблона.
2. Входные данные для подстановки в шаблон вместо тегов.
3. Опции запроса.
4. Формат ответа.

Описатель шаблона

Элемент (объект) описателя шаблона **template** содержит:

- **uri** - строка. Задаёт расположение файла документа-шаблона в зависимости от того, как трактуется параметр **id**.

Поддерживаемые значения:

- **local** - расположение шаблона в каталоге `templates`
 - **embedded** - шаблон включен в запрос в виде закодированной base64 строки. Значение строки должно быть записано в параметр `value`.
 - **remote** - шаблон расположен на удаленном файловом сервере. Ссылка на шаблон должна быть указана в параметре `value`.
 - **inner** - внутренний шаблон, используется для генерации таблицы с заголовком.
- **id** - строка. Идентификатор шаблона. Путь к файлу документа-шаблона относительно директории **templates**. Также используется для записи файла отчета в отладочном режиме и для идентификации запроса в логге.
 - **value** - строка. Значение используется в режимах `embedded` и `remote`.
 - **type** - строка. Данный параметр используется для указания нестандартного типа шаблона. Поддерживаемое значение: `xlsm`.

Пример для формата XML:

```
<template uri="local" id="Информационное письмо"/>
```

Пример для формата JSON:

```
"template":
{
  "uri": "local",
  "id": "template_example_1"
}
```

Входные данные

Элемент (объект) входных данных **input-data** содержит:

- Данные для подстановки в шаблон вместо тегов: строки, числа, формулы, ссылки.
- Описатели таблиц.
- Описатели блоков.
- Данные для подстановки в теги изображений.

i Примечание

Отличие структуры данных для формата JSON от XML в том, что вместо XML-элементов данные представлены как объекты и списки объектов. Имена тегов задаются именами полей объектов. Таблицы и изображения являются дочерними объектами по отношению к объекту **input-data**.

Данные для подстановки в шаблон вместо тегов**Строки**

Для формата XML

Элемент **tags** содержит список дочерних элементов **tag** с атрибутом **name** и нужным конкретным значением, на который тег в шаблоне будет заменен при генерации отчета. Атрибут **name** совпадает с тегом в шаблоне.

Пример:

```
<input-data>
  <tags>
    <tag name="ORGANIZATION">АО «xxxxxyyyy»</tag>
  </tags>
</input-data>
```

Для формата JSON

Имена тегов задаются именами объектов внутри **input-data**.

Пример:

```
"input-data":
{
  "ORGANIZATION": "АО «xxxxxyyyy»"
}
```

Числа

Для формата XML:

Элемент **tag** также поддерживает атрибут **type**. Если атрибут **type** равен **num**, значение тега будет интерпретироваться как число с плавающей точкой. В данном случае в шаблон будет подставлено числовое значение, это позволяет, например, применять формулы.

Пример:

```
<tag name="doc_cnt" type="num">2</tag>
```

Для формата JSON:

Для интерпретации значения тега в качестве числа необходимо указывать значение без кавычек, с точкой в качестве разделителя целой и дробной части.

Пример:

```
"input-data":
{
  "total": 3889.98
}
```

Формулы

Вместо тега в шаблон может быть вставлена формула, для расчета и отображения значений.

Формула в запросе задается объектом с полями:

- **type** - строка, поддерживаемое значение: "formula".
- **value** - строка, значения берутся из используемого редактора таблиц, без знака равенства.

Для формата XML type является атрибутом элемента tag, а value - его значением:

```
<tag name="sum" type="formula">SUM (AX2:AX3) </tag>
```

Для формата JSON:

```
"sum":
{
  "type": "formula",
  "value": "SUM(AX2:AX3) "
}
```

Преобразований номеров строк не выполняется. Предполагается, что номера строк актуальны для финального отчета, то есть после вставки строк таблицы, идущих перед строкой, содержащей формулу. Номер строк можно рассчитать на основе номеров строк таблицы в документе-шаблоне и количества строк таблицы на момент генерации строки с формулой в запросе.

Пример запроса: **formula.json** Его также можно найти в *архиве примеров*.

Примечание

для поддержки формул с относительной адресацией ячеек необходимо в опции запроса добавить параметр **formulas-relative-addressing**. Данный флаг позволяет использовать формулы в обоих форматах, но предполагает дополнительную обработку, которая будет производиться впустую если относительной адресации - нет.

```
"options":
{
  "formulas-relative-addressing": true
},
...
"формула":
{
  "type": "formula",
  "value": "SUM(R[-9]C+R[-1]C) "
}
```

Ссылки

Вместо тега в шаблон может быть вставлена гиперссылка.

Ссылка в запросе задается объектом с полями:

- **type** - строка, поддерживаемое значение: "link".
- **text** - строка, отображаемое в документе название ссылки.

- **value** - строка, значение ссылки для перехода.

Для формата XML `type` и `text` являются атрибутами элемента `tag`, а `value` - его значением:

```
<tag name="http://link1" type="link" text="компания «Хи-квадрат»">
  "http://xsquare.ru"
</tag>
```

Для формата JSON:

```
"http://link1":
{
  "type": "link",
  "text": "компания «Хи-квадрат»",
  "value": "http://xsquare.ru"
}
```

i Примечание

в шаблоне документа теги для ссылок должны иметь формат гиперссылок. Для этого можно имя тега начать с `http://` (или `https://`, `ftp://`)

Пример запроса: **hyperlinks.json** Его также можно найти в *архиве примеров*.

Описатели таблиц

Для отчетов XLSX также доступно формирование таблиц на основе данных из запроса.

Отличие от формата DOCX:

- теги задаются в элементе (объекте) **cell-tags** один раз для всей таблицы;
- тег ячейки отображает тег столбца на индекс ячейки в массиве, задающем строку таблицы.

Для формата XML

Элемент описателей таблиц **tables** содержит список дочерних элементов **table** с атрибутом **name**. Атрибут **name** совпадает с тегом в шаблоне. Тег в шаблоне будет заменен на таблицу, созданную согласно описанию таблицы в запросе.

Пример:

```
<input-data>
  <tables>
    <table name="таблица_застрахованные_лица">
      <cell-tags>
        <cell-tag name="номер_пп" index="0"/>
        <cell-tag name="снилс" index="1"/>
        <cell-tag name="пол" index="2"/>
        <cell-tag name="фио" index="3"/>
        <cell-tag name="дата_рождения" index="4"/>
        <cell-tag name="дата_договора" index="5"/>
        <cell-tag name="номер_договора" index="6"/>
      </cell-tags>
    </table>
  </tables>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

        <row>
          <cell>1</cell>
          <cell>001-002-003 00</cell>
          <cell>Ж</cell>
          <cell>Горбунова Екатерина Васильевна</cell>
          <cell>01.01.1970</cell>
          <cell>01.01.1988</cell>
          <cell>ABC-001-002-003-00</cell>
        </row>
      </rows>
    </table>
  </tables>
</input-data>

```

Для формата JSON

Описатели таблиц являются дочерними объектами по отношению к объекту `input-data`.

Пример:

```

"input-data":
{
  "таблица_застрахованные_лица":
  {
    "cell-tags":
    {
      "номер_пп": 0,
      "снилс": 1,
      "пол": 2,
      "фио": 3,
      "дата_рождения": 4,
      "дата_договора": 5,
      "номер_договора": 6
    },
    "rows":
    [
      [
        "1",
        "001-002-003 00",
        "Ж",
        "Горбунова Екатерина Васильевна",
        "01.01.1970",
        "01.01.1988",
        "ABC-001-002-003-00"
      ]
    ]
  }
}

```

Значениями ячеек в таблице также могут быть формулы и ссылки:

Пример JSON:

```


"input-data":
{
  "таблица_1":
  {
    "rows":
    [
      [
        "1",
        "Теплосбыт",
        "Учреждение 1",
        161975.87,
        null,
        161975.87,
        12985.51,
        "15.08.2023",
        {
          "type": "link",
          "text": "пример ссылки в таблице 1",
          "value": "http://xsquare.ru"
        }
      ],
      [
        null,
        null,
        "ИТОГО ФЕДЕРАЛЬНЫЙ БЮДЖЕТ",
        {
          "type": "formula",
          "format": "raw",
          "value": "SUM(D11:D11)"
        },
        null,
        161975.87,
        12985.51,
        null,
        null
      ]
    ],
    "cell-tags":
    {
      "наименование": 1,
      "номер_пп": 0,
      "ссылка": 8,
      "срок_исполнения": 7,
      "задолженность_общая": 3,
      "текущая_задолженность": 4,
      "наименование_организации": 2,
      "просроченная_задолженность": 5,
      "остаток_задолженности_от_просроченной": 6
    }
  }
}

```

Пример XML:

```

<tables>
  <table name="таблица_застрахованные_лица">
    <cell-tags>
      <cell-tag name="номер_пп" index="0" />
      <cell-tag name="снилс" index="1" />
      <cell-tag name="http://link" index="2" />
    </cell-tags>
    <rows>
      <row>
        <cell>1</cell>
        <cell>001-002-003 00</cell>
        <cell type="link" text="link text1">http://table1.ru</cell>
      </row>
      <row>
        <cell>2</cell>
        <cell>001-002-004 00</cell>
        <cell type="link" text="link text2">http://table2.ru</cell>
      </row>
    </rows>
  </table>
</tables>

```

Пример запроса: **example_with_links.xml** Его также можно найти в *архиве примеров*.

Описатели блоков

Для отчетов XLSX доступна генерация блоков данных в документе на основе блока-шаблона и данных из запроса.

Под блоком-шаблоном подразумевается последовательность строк в документе между ячейками с тегами **[имя_блока] - [/имя_блока]**.

Блок-шаблон может быть многократно последовательно повторен в документе. Также после каждого блока добавляется разрыв страницы, что удобно при печати документов.

Элемент описателей блоков **blocks** содержит список дочерних элементов **block** с атрибутом **template** (совпадает с именем шаблона блока в документе-шаблоне).

Блок имеет те же элементы, что и **input-data**, кроме элементов изображений и блоков. Данные для блока располагаются в описателе **data**, также для каждого блока можно задать разрыв страницы в конце блока.

Разрыв страницы можно отменить посредством указания значения **true** в описателе **no-page-break**.

Шаблоны блоков в документе-шаблоне будут использованы для вставки произвольного количества экземпляров блоков с разными наборами данных. Блоки вставляются последовательно друг за другом.

Пример XML:

```

<blocks>
  <block template="block_1">
    <data>
      <tags>
        <tag name="date">01.02.2024</tag>
      </tags>
    </data>
  </block>
  <block template="block_1">

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

<no-page-break>true</no-page-break>
<data>
  <tags>
    <tag name="date">01.02.2025</tag>
  </tags>
</data>
</block>
<block template="block_1">
  <data>
    <tags>
      <tag name="date">01.02.2026</tag>
    </tags>
  </data>
</block>
</blocks>

```

Для запросов в формате JSON имя шаблона блока указывается в описателе **block-template**

Пример JSON:

```

"blocks":
[
{
  "block-template": "block_1",
  "no-page-break": true,
  "data":
  {
    "main2": "main_for_1",
    "main3": "m1_val",
    "main4": "m1_val2",
    "table":
    {
      "cell-tags":
      {
        "f2": 0,
        "f3": 1,
        "dt_pay": 2
      },
      "rows":
      [
        [
          "f2_val1",
          "f3_val1",
          "22.12.2023"
        ],
        [
          "f2_val2",
          "f3_val2",
          "23.12.2023"
        ]
      ]
    }
  }
}
]

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

},
{
  "block-template": "block_1",
  "data":
  {
    "main2": "main_for_2",
    "main3": "m2_val",
    "main4": "m2_val2",
    "table":
    {
      "cell-tags":
      {
        "f2": 0,
        "f3": 1,
        "dt_pay": 2
      },
      "rows":
      [
        [
          "f2_val3",
          "f3_val3",
          "22.11.2022"
        ]
      ]
    }
  }
}
]

```

Пример запроса: **blocks.json**. Его также можно найти в *архиве примеров*.

i Примечание

в блоке-шаблоне не допускаются строки с пустыми ячейками. Для того, чтобы ячейки считались не пустыми - достаточно сделать заливку ячеек блока любым цветом, а затем отменить заливку и сохранить файл шаблона.

Данные для подстановки в теги изображений

Для формата XML

Элемент **images** содержит список дочерних элементов **image** с атрибутом **name**. Атрибут **name** совпадает с тегом в шаблоне. Изображения в шаблоне будут заменены на данные изображений из запроса. В запросе передается новое содержимое файла изображения, закодированное в BASE64.

Пример:

```

<input-data>
  <images>
    <image name="IMAGE-1-JPEG"/>/9j/4AAQSkZJRgABAQEAYABgAAD ... </image>
    <image name="IMAGE-2-JPEG"/>/9j/4AAQSkZJRgABAQEAYABg ... </image>
  </images>
</input-data>

```

Для формата JSON

Данные для подстановки в теги изображений находятся внутри дочерних объектов объекта **input-data**.

Пример:

```
"input-data":
{
  "images":
  {
    "IMAGE-1-JPEG":
    {
      "data": "/9j/4AAQSkZJRgABAQEAYABgAAD ... "
    },
    "IMAGE-2-JPEG":
    {
      "data": "/9j/4AAQSkZJRgABAQEAYABg ... "
    }
  }
}
```

Также описатель **data** может содержать данные для генерации кодов (например QR code). Подробнее [Работа с графическими кодами](#).

Удаление изображений

Для того, чтобы удалить изображение из шаблона необходимо вместо нового содержимого передать **null**.

Пример:

```
"input-data":
{
  "images":
  {
    "IMAGE-1-JPEG":
    {
      "data": null
    },
    "IMAGE-2-JPEG":
    {
      "data": "/9j/4AAQSkZJRgABAQEAYAB...AooAKKACigAooAKKACigAooA//9k="
    }
  }
}
```

Опции запроса

Элемент (объект) опций запроса **options** содержит:

- **output-mode** - строка. Определяет форму предоставления результата.

Поддерживаемые значения:

- **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
- **binary** - результат будет представлен в виде бинарного потока.

- **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

Примечание: флаг `enable-binary-output` считается устаревшим, но для поддержки старых отчетов, продолжает использоваться.

- **report-format** - строка. Указывает формат файла отчета, если он отличен от формата, упомянутого в url запроса. Единственное непустое поддерживаемое значение: **pdf**. Значение по умолчанию: пустая строка. См. пункт “Конвертация отчета в PDF” ниже.
- **clean-data-by-null** - логическое значение. Определяет способ обработки тегов со значениями null для xlsx, xlsxm шаблонов. Если значение опции - true, то ячейка соответствующая тегу будет полностью очищена от любого содержимого. В случае, если значение опции - false, то значение строковых тегов будет заполнено пустыми строками, а для остальных типов - очищено. Значение по умолчанию - false.
- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **enable-debug-report-save** - логическое значение. Указывает создавать ли копию отчета документа и копию запроса (имя файла - идентификатор шаблона) в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.

Примечание

в шаблоне документа должны быть заданы области печати («Set Print Area»), в соответствии с ними будет произведена конвертация в PDF

Пример для формата XML:

```
<options>
  <enable-debug-report-save>true</enable-debug-report-save>
  <output-mode>binary</output-mode>
  <report-format>pdf</report-format>
</options>
```

Пример для формата JSON:

«options»:

```
{
  "enable-debug-report-save": true,
  "output-mode": "base64",
  "report-format": "pdf"
}
```

Формат ответа

Элемент (объект) формата ответа **response-format** может принимать значения **json** и **xml**.

Подробнее про то, как задается формат ответа можно прочитать в подразделе “Структура ответа”.

Пример для формата XML

```
<response-format value="xml"/>
```

Пример для формата JSON:

```
{"response-format": "json"}
```

Примеры запросов

Пример для формата XML

```
<request>
  <response-format value="xml"/>
  <template uri="local" id="template_example"/>
  <input-data>
    <tags>
      <tag name="организация">000 «Успех»</tag>
      <tag name="число">25</tag>
      <tag name="месяц">декабря</tag>
      <tag name="год">2018</tag>
      <tag name="номер_документа">01</tag>
      <tag name="число_договоров">3</tag>
      <tag name="сотрудник">Сидорова Мария Петровна</tag>
      <tag name="тел_сотрудника">+7 (123) 456-78-90</tag>
      <tag name="тер1">тер1_значение</tag>
      <tag name="тер2">тер2_значение</tag>
    </tags>
    <tables>
      <table name="таблица_застрахованные_лица">
        <cell-tags>
          <cell-tag name="номер_пп" index="0"/>
          <cell-tag name="снилс" index="1"/>
          <cell-tag name="пол" index="2"/>
          <cell-tag name="фио" index="3"/>
          <cell-tag name="дата_рождения" index="4"/>
          <cell-tag name="дата_договора" index="5"/>
          <cell-tag name="номер_договора" index="6"/>
        </cell-tags>
        <rows>
          <row>
            <cell>1</cell>
            <cell>001-002-003 00</cell>
            <cell>Ж</cell>
            <cell>Горбунова Екатерина Васильевна</cell>
            <cell>01.01.1970</cell>
            <cell>01.01.1988</cell>
            <cell>ABC-001-002-003-00</cell>
          </row>
        </rows>
      </table>
    </tables>
    <images>
      <image name="IMAGE-1-JPEG">/9j/4AAQSkZJRgABAQEAYABgAAD/4QFQR0 ...</image>
      <image name="IMAGE-2-JPEG">/9j/4AAQSkZJRgABAQEAYABgAAD/4QFQR ...</image>
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

</images>
</input-data>
<options>
  <enable-debug-report-save>true</enable-debug-report-save>
</options>
</request>

```

Пример для формата JSON

```

{
  "response-format": "json",
  "template": {
    "uri": "local",
    "id": "template_example"
  },
  "input-data": {
    "организация": "ООО «Успех»",
    "число": "25",
    "месяц": "декабря",
    "год": "2018",
    "номер_документа": "01",
    "число_договоров": "3",
    "сотрудник": "Сидорова Мария Петровна",
    "тел_сотрудника": "+7 (123) 456-78-90",
    "тег1": "тег1_значение",
    "тег2": "тег2_значение",
    "таблица_застрахованные_лица": {
      "cell-tags": {
        "номер_пп": 0,
        "снилс": 1,
        "пол": 2,
        "фио": 3,
        "дата_рождения": 4,
        "дата_договора": 5,
        "номер_договора": 6
      },
      "rows": [
        [
          "1",
          "001-002-003 00",
          "Ж",
          "Горбунова Екатерина Васильевна",
          "01.01.1970",
          "01.01.1988",
          "ABC-001-002-003-00"
        ]
      ]
    },
    "images": {
      "IMAGE-1-JPEG": {
        "data": "/9j/4AAQSkZJRgABAQEAYABgAAD/4QFQR0 ..."
      }
    }
  }
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "IMAGE-2-JPEG": {
      "data": "/9j/4AAQSkZJRgABAQEAYABgAAD/4QFQR ..."
    }
  },
  "options": {
    "enable-debug-report-save": true
  }
}

```

Пример вызова сервиса

Для формата XML

```

curl --request POST --data-binary "@templates/examples/xlsx/base_example.xml" \
↳http://localhost:8886/excel_report_xml

```

Для формата JSON

```

curl --request POST --data-binary "@templates/examples/xlsx/base_example.json" http://
↳localhost:8886/excel_report_json

```

Структура ответа

В зависимости от **output-mode** ответ сервиса может быть либо потоком бинарных данных, либо ответом в формате JSON/XML.

Формат ответа (JSON или XML) может быть задан двумя способами:

1. В HTTP заголовке Ассерта. Поддерживаемые значения: **application/json** и **application/xml**.
2. В теле запроса в элементе (объекте) **response-format**. Поддерживаемые значения: **json** и **xml**.

Приоритет имеет формат, указанный в теле запроса.

Ответ сервиса содержит объект в формате JSON или XML, который включает:

1. Описание ошибки (код ошибки, сообщение об ошибке). В случае успешного ответа сервиса возвращается значение **null**.
2. Результат (закодированный в BASE64 файл документа отчета в формате xlsx).

Формат ответа

```

{
  "error":
  {
    "code": "",
    "message": ""
  },
  "result": "[base64 encoded docx/pdf file] or [url]"
}

```

Пример ответа

```
{
  "error": null,
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="
}
```

Пример для отчета предоставляемого по ссылке:

```
{
  "options": {
    "output-mode": "url",
    "domain-name": "test_domain",
    "file-name": "name_by_options"
  },
  "template": {
    "id": "report_by_url",
    "uri": "embedded",
    "value": "UESDBDSBQABgAIAAA..."
  }
}
```

Ответ:

```
{
  "error": null,
  "result": "test_domain/2025/02/17/13/521b84da-a837-40d3-a145-51defae86235/name_by_
↪options.docx"
}
```

Коды ошибок

Поле **code** может принимать следующие значения:

“1” - ошибка в запросе;

“2” - ошибка формирования отчета.

Работа с документом

Общая информация

Для документов в формате XLSX доступны:

- теги, которые будут заменены входными данными из запроса;
- таблицы;
- изображения.

Примечание

- Формат входных данных ячеек - текст и числа (целые или вещественные)..
- Ссылки на ячейки в формулах при генерации отчета не корректируются. Формулы могут стать неверными в случае добавления/удаления строк в отчет.
- Условные выражения в текущей версии не поддерживаются.

Работа с таблицами

Существует два варианта работы с таблицами.

v1 - API: /excel_report_json

Таблица с форматированием задается следующим образом:

- первая ячейка строки содержит тег с названием таблицы. Эта строка удаляется при генерации.
- «строка тегов», строка (строки, см ниже), следующая за строкой тега таблицы, содержит теги столбцов, их форматирование будет применено к сгенерированным строкам (если не задана строка форматирования, см ниже). Эта строка (строки) удаляется при генерации.
- опционально: строка, указывающая, что в следующей строке содержится строка форматирования. Содержит одну ячейку с текстом “[формат ячеек]”. Применяется для таблиц, где числовые данные должны быть в ячейках с форматом число (или genetal), а не строками (иначе проще задать форматирование в строке тегов).
- опционально: “строка форматирования”, строка, задающая формат ячеек вместо строки тегов. Значения ячеек игнорируются. Формат применяется к сгенерированным строкам. Должна быть полной копией строки тегов с точки зрения количества ячеек и их порядка.

Поддерживается возможность отображения одной строки входных данных на множество строк в документе. Это может понадобиться для таблиц, где в одной логической строке есть ячейки с объединением по вертикали. Количество строк шаблона на одну строку входных данных определяется как максимальное число строк, объединенных по вертикали в любой ячейке из первой строки после строки с тегом самой таблицы (см. функцию `getNumberOfSheetRowsPerInputDataRow`).

Пример шаблона можно найти в [архиве примеров](#).

Примечание

на данный момент применение форматирования к сгенерированным строкам не включает в себя настройки шрифта. Поддерживается цвет фона и настройки границ ячеек (например, горизонтальное слияние ячеек).

v2 - API: /excel_report_json/v2

Таблица с форматированием задается следующим образом:

- первая ячейка строки содержит тег с названием таблицы. Эта строка удаляется при генерации.
- «строка тегов», строка, следующая за строкой тега, содержит теги столбцов, их форматирование будет применено к сгенерированным строкам(если не задана строка форматирования, см ниже). Эта строка удаляется при генерации.
- опционально: строка, указывающая, что в следующей строке содержится строка форматирования. Содержит одну ячейку с текстом “[формат ячеек]”. Применяется для таблиц, где числовые данные должны быть в ячейках с форматом число(или genetal), а не строками(иначе проще задать форматирование в строке тегов).
- опционально: “строка форматирования”, строка, задающая формат ячеек вместо строки тегов. Значения ячеек игнорируются. Формат применяется к сгенерированным строкам. Должна быть полной копией строки тегов с точки зрения количества ячеек и их порядка.

Примеры шаблонов можно найти в [архиве примеров](#).

- `cell_format_options.xlsx`
- `cell_format_options_typed_cells.xlsx`

i Примечание

на данный момент применение форматирования к сгенерированным строкам не включает в себя настройки шрифта. Поддерживается цвет фона и настройки границ ячеек (например, горизонтальное слияние ячеек).

Управление слиянием ячеек

Базовый алгоритм слияния - применение горизонтального слияния генерируемых ячеек на основе слияния ячеек таблицы шаблона.

Поддерживаются расширенные опции вертикального слияния. Задается объектом с единственным полем:

- **vertical_span** - целое число. Управляет вертикальным слиянием ячейки. Значение задает количество ячеек для объединения вниз от текущей ячейки (включительно).

Опции могут быть привязаны к столбцам таблицы в шаблоне, задаваемым по индексу

```
"formatting": {
  "template-cells": [
    {
      "vertical_span": 2
    },
    ...
  ]
}
```

или к тегам ячеек (данный способ имеет приоритет, что удобно для избирательной коррекции опций слияния для части столбцов):

```
"formatting": {
  "tags": {
    "cell_tag_value": {
      "vertical_span": 2
    },
  },
}
```

Добавление строк в конец документа шаблона

Строки, заданные списком **rows-to-add** будут добавлены в конец указанных страниц документа шаблона или в конец первой страницы по умолчанию. Добавляемые строки могут содержать разное количество ячеек и разные данные по столбцам, то есть каждая добавляемая ячейка независима от других. См. пример запроса **add_rows_to_end.json** в *архиве примеров*.

При использовании режима **rows-to-add** значениями ячеек могут быть:

- **null**
- **строки**
- **числа**
- **формулы**
- **гиперссылки**

Пример:

```



```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "type": "formula",
    "value": "SUM(H2:H4) "
  }
]
}

```

См. пример запроса `add_rows_to_end_complex.json` в *архиве примеров*.

При необходимости задания стиля, форматирования и типа ячеек добавляемых строк можно использовать последнюю строку в документе в качестве шаблона. Для этого в опциях запроса необходимо указать флаг **use-last-row-style**. В данном режиме последняя строка будет служить источником стиля, форматирования и типа для каждой ячейки при добавлении новых строк методом **rows-add**. Последняя строка-шаблон будет удалена из результирующего документа. В запросе тип подставляемых данных должен соответствовать типу ячейки в последней строке.

Примечание

Даты и время передаются как числа, а не как строки.

Пример:

```

{
  "template":
  {
    "uri": "local",
    "id": "examples/xlsx/add_rows_to_end_with_style"
  },
  "options":
  {
    "use-last-row-style": true
  },
  "input-data":
  {
    "rows-to-add":
    [
      [
        12,
        "001-002-003 00",
        "Ж",
        "Горбунова Екатерина Васильевна",
        34021,
        null,
        "нет ссылки"
      ],
      [
        2.56,
        "004-005-006 00",
        "М",
        "Иванов Дмитрий Борисович",
        34022,
        0.6458,
        null
      ]
    ]
  }
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    {
      "type": "link",
      "text": "пример ссылки 1",
      "value": "http://xsquare.ru"
    }
  ],
  [
    3.4555555,
    "007-008-009 00",
    null,
    "Сидоров Максим Васильевич",
    34025,
    0.3,
    {
      "type": "link",
      "text": "пример ссылки 2",
      "value": "http://xsquare.ru"
    }
  ],
  [],
  [
    4,
    null,
    "М",
    "Кольев Антон Викторович",
    null,
    null,
    null
  ]
]
}

```

Для добавления строк данных на разные страницы документа необходимо указать имя страниц в запросе:

```

"rows-to-add":
{
  "Данные по регионам":
  [
    [
      100,
      "Центр",
      "Москва"
    ]
  ],
  "Лист1":
  [
    [
      12,
      "001-002-003 00",
      "Ж",
      "Горбунова Екатерина Васильевна",

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

34021,
null,
"нет ссылки"
],

```

см. пример `add_rows_by_pages.json`

Добавление строк в конец документа шаблона является самым высокопроизводительным средством выгрузки большого массива данных в документы XLSX. Так как каждая страница XLSX имеет ограничение в 1 048 576 строк, то для выгрузки большого массива данных можно использовать опцию **split-by-count**.

split-by-count - число, определяющее количество строк, на которое разбивается массив строк «rows-to-add» и распределяется по страницам. Новые страницы создаются автоматически.умолчанию

см. пример `add_rows_split_by_count.json`

Генерация таблицы без шаблона

Сервер отчетов позволяет сформировать документ в виде таблицы без шаблона. Для этого в качестве значения **uri** описателя шаблона используется **inner**. Запрос должен содержать глобальный описатель **table** для описания одной таблицы или **tables** для описания массива таблиц на разных страницах документа (описатель **table** продолжает использоваться для поддержки старых версий запросов). Описатель каждой таблицы должен содержать:

- **name** - строка, устанавливает имя страницы в документе, на которой будет размещена данная таблица
- **columns** - объект, определяющий параметры колонки таблицы.
 - **header** - строка, определяет заголовок колонки.
 - **align** - строка, определяет выравнивание данных в колонке. Поддерживаемые значения:
 - * **l** - выравнивание по левому краю ячейки(по умолчанию)
 - * **c** - выравнивание по центру ячейки
 - * **r** - выравнивание по правому краю ячейки
 - **type** - тип данных в колонке. Отсутствие значения (null) соответствует типу по умолчанию - строка. Поддерживаемые значения:
 - * **num** - число
 - * **date** - число, переданное в параметрах будет отображено как дата
 - * **time** - число, переданное в параметрах будет отображено как время
 - * **datetime** - число переданное в параметрах будет отображено как дата и время
 - **width** - ширина колонки. По умолчанию ширина колонок будет подобрана автоматически с учетом содержимого.
- **rows** - массив строк данных таблицы.

Например:

```

"input-data": {
  "tables": [
    {
      "name": "Таблица_1",
      "columns": [

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    { "header": "Строка", "align": "c", "type": null, "width": null },
    { "header": "Дата", "align": "r", "type": "date", "width": 15 }
  ],
  "rows": [
    ["A", 45920.1],
    ["B", 45921.0],
    [null, 45922.5],
    ["C", null]
  ]
},
{
  "name": "Таблица_2",
  "columns": [
    { "header": "Строка", "align": "c", "type": null, "width": null },
    { "header": "Число", "align": "l", "type": "num", "width": null },
    { "header": "Дата", "align": "r", "type": "date", "width": 15 },
    { "header": "Время", "align": "c", "type": "time", "width": 18 },
    { "header": "Дата и время", "align": "l", "type": "datetime", "width": 25 }
  ],
  "rows": [
    ["X", 100, 45925.123, 45925.456, 45925.789],
    [null, 200.5, null, 45926.111, 45926.222],
    ["Y", null, 45927.333, null, null],
    ["Z", 300, 45928.444, 45928.555, 45928.666]
  ]
},
},

```

См. пример запроса `multitables.json` и `table_column_format_new.json` в *архиве примеров*.

Использование формата XLSM в качестве шаблона

В качестве шаблонов может быть использован формат XLSm - файлы электронных таблиц с поддержкой макросов. Для использования данного типа шаблона необходимо в описателе `template` указать параметр `<type>`: `<xlsm>`

```

"template": {
  "uri": "local",
  "id": "examples/xlsx/example_macros",
  "type": "xlsm"
}

```

См. пример запроса `example_macros.json` в *архиве примеров*.

5.1.5 Конвертация отчета в PDF

Активация

В опциях запроса указать формат отчета `<pdf>`.

Пример:

```

<options>
  <report-format>pdf</report-format>
</options>

```

Зависимости

Необходим установленный пакет Libre Office. Программа **soffice** должна быть доступна для вызова. Возможно задать путь к ней в параметрах командной строки сервера отчетов или через файл конфигурации.

Принцип работы

Сервер отчетов сохраняет отчет в файл и вызывает утилиту **soffice** со следующими параметрами, для примера

```
soffice --convert-to pdf report.docx --outdir temp_dir
soffice --convert-to pdf report.xlsx --outdir temp_dir
```

затем высылает вместо оригинала отчета полученный файл PDF.

Умолчания

Путь к утилите **soffice** ищется в директориях, заданных переменной среды **PATH**. Можно указать прямой путь в аргументах командной строки сервера отчетов или через файл конфигурации.

Файлообмен с **soffice** идет через временную папку системы. Можно указать рабочую директорию в аргументах командной строки сервера отчетов или через файл конфигурации.

См. вывод команды

```
xreports --help
```

Производительность

По умолчанию время исполнения команды

```
soffice --convert-to pdf
```

для пустого документа - 2.3 секунды (для сру i7-3615M, ssd). Основное время тратится на инициализацию Libre Office.

Чтобы ускорить конверсию, можно воспользоваться режимом **quick start**, то есть предварительным исполнением команды

```
soffice --quickstart
```

В этом режиме Libre Office все время находится в оперативной памяти, что экономит примерно 2 секунды.

Примечание

для Libre Office 6.3.4.2 в Ubuntu 19.10 **soffice --quickstart** приводит к показу главного окна. Параметры **--minimized** и **--headless** не решают эту проблему, так как не позволяют пользоваться ускорением после однократной конверсии. В Windows 10 при таком запуске LO виден только в трее.

5.1.6 Работа со штрихкодом

Возможен вывод строки текста в виде штрихкода. Для корректной работы в отчетах Word и Excel в операционной системе должны быть установлены шрифты семейства Libre Barcode. Доступны в репозитории [librebarcode](#).

Каждый шрифт представляет собой определенный формат штрихкода. Поддерживаются следующие форматы:

- Code-128

- Code-39
- EAN13

Шрифт может включать оригинальный текст непосредственно под штрихкодом (шрифты с суффиксом Text). Рассмотрим применение на примере формата Code-128.

Code-128

Шрифты для установки:

- **LibreBarcode128Text-Regular.ttf**, будет доступен под именем «Libre Barcode 128 Text»
- **LibreBarcode128-Regular.ttf**, будет доступен под именем «Libre Barcode 128»

Как использовать

В шаблоне для обычного тега выбрать шрифт. Например, «Libre Barcode 128 Text».

Тестовые запросы

Документ в формате DOCX

```
curl --request POST --data-binary "@templates/examples/docx/barcode_code-128.json" \
  ->http://localhost:8886/word_report_json
```

Документ в формате XLSX

```
curl --request POST --data-binary "@templates/examples/xlsx/barcode_code-128.json" \
  ->http://localhost:8886/excel_report_json
```

5.1.7 Работа с графическими кодами (QR code)

Для подстановки на место изображения динамически создаваемых графических кодов, необходимо в параметр **data** передать описатель кода.

Пример:

```
{
  "template":
  {
    "uri": "local",
    "id": "examples/docx/qr-code"
  },
  "input-data":
  {
    "images":
    {
      "qr_link":
      {
        "data":
        {
          "type": "qr-code",
          "content": "https://xsquare.ru",
          "qr-color": "#ffa747",
          "bg-color": "#2b0054",

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

        "dot-size": 40,
        "style": "dot",
        "format": "png"
    }
}
},
"options":
{
    "enable-debug-report-save": false,
    "output-mode": "binary"
}
}

```

Параметры для генерации графических кодов аналогичны параметрам в модуле генерации кодов, с подробным описанием параметров можно ознакомиться в разделе *Сервис генерации кодов (QR code)*

5.2 Генератор отчетов PDF

5.2.1 Общее описание сервиса

Сервис формирует отчет в формате PDF по HTTP-запросу в формате JSON.

PDF-документ генерируется при помощи специальных команд генератора.

5.2.2 Формирование документов с помощью генератора PDF

Описание сервиса

Наименование сервиса	Генератор отчетов PDF
Путь к сервису	[host]:[port]/pdf_report_json
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт файл документа в формате PDF, закодированный в base64.
Назначение	Сервис предназначен для генерации документов в формате PDF.

Структура тела запроса

Тело запроса содержит объект в формате JSON, который включает:

1. Описатель генератора PDF.
2. Входные данные для подстановки в шаблон вместо тегов.
3. Опционально: генератор PDF.
4. Опции запроса.

Пример:

```
{
  "report": "report-generator",
  "uri": {
    "uri": "local",
    "id": "example_1"
  },
  "input-data": {
    "ORGANIZATION": "ООО «Успех»",
    "CLINAME": "Петров Петр Петрович"
  },
  "options": {
    "enable-debug-report-save": true
  }
}
```

Описатель генератора PDF

Объект описателя генератора PDF **report-generator** содержит:

- **uri** - строка. Задаёт положение источника команд генератора в зависимости от того, как трактуется параметр **id**. Поддерживаемые значения: **local** и **embedded**.
- **id** - строка. ID генератора.

Возможны два варианта использования генератора PDF:

- генератор как часть запроса (все команды генератора размещаются внутри объекта **embedded-report-generator** тела запроса);
- генератор как часть сервиса (все команды генератора размещаются внутри файла JSON в специальной папке **pdf_report_gen** на сервере).

Соответственно:

- При значении **uri**, равному **embedded**, **id** используется только для диагностических сообщений и идентификации запроса. Генератор должен находиться в корневом объекте **embedded-report-generator** запроса. См. пункт «Генераторы как часть запроса» ниже.
- При значении **uri**, равному **local**, **id** - это имя файла в директории сервиса **pdf_report_gen**. См. пункт «Генераторы как часть сервиса» ниже.

Генератор как часть запроса

Опционально генератор может быть встроен в запрос.

В этом случае все команды генератора указываются в объекте **embedded-report-generator**.

Пример:

```
{
  "report-generator": {
    "id": "example",
    "uri": "embedded"
  },
  "embedded-report-generator": {
    "commands": [
      {
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "name": "NewPage"
  },
  {
    "name": "setCoordinateMode",
    "params": {
      "mode": "millimeters"
    }
  },
  {
    "name": "setCellMargin",
    "params": {
      "margin": 1
    }
  }
]
},
"options": {
  "enable-debug-report-save": false
}
}

```

Генератор как часть сервиса

Цель: не передавать данные команд внутри запросов, если они не меняются от запроса к запросу.

В этом случае все команды генератора указываются в файле формата JSON, который должен находиться в директории **pdf_report_gen**, расположенной в директории приложения сервиса.

Формат генератора внутри файла в директории **pdf_report_gen**:

```

{
  "report-generator": {
    "commands": [...]
  }
}

```

Пример генератора в теле запроса:

```

"report-generator": {
  "uri": "local",
  "id": "example"
}

```

, где **id** - это имя файла в директории сервиса **pdf_report_gen**.

Пример простого запроса для генератора внутри файла **example.json** в директории **pdf_report_gen**:

```

{
  "report": "report-generator",
  "uri": {
    "uri": "local",
    "id": "example"
  },
  "options": {

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

"enable-debug-report-save": true
}
}

```

Входные данные

Объект входных данных **input-data** содержит входные, которые подставляются в теги.

Примечание

Команды генератора PDF, которые работают с текстом, могут содержать теги, которые будут заменены входными данными из запроса. Подробнее об использовании тегов можно прочитать в разделе “Теги” ниже.

Пример:

```

"input-data":
{
  "ORGANIZATION": "ООО «Успех»",
  "CLINAME": "Петров Петр Петрович"
}

```

Опционально: генератор PDF

Объект генератора PDF **embedded-report-generator** содержит команды генератора. Используется опционально при значении **uri**, равному **local**.

Подробнее описан в пункте “Генераторы как часть запроса”.

Опции запроса

Объект опций запроса **options** имеет поля:

- **output-mode** - строка. Определяет форму предоставления результата. Поддерживаемые значения:
 - **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
 - **binary** - результат будет представлен в виде бинарного потока.
 - **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

Примечание: флаг **enable-binary-output** считается устаревшим, но для поддержки старых отчетов, продолжает использоваться.

- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **enable-debug-pdf-log** - логическое значение. Включает расширенный диагностический лог создания PDF. Значение по умолчанию - **false**.

- **enable-debug-report-save** - логическое значение. Указывает создавать ли копию отчета документа и копию запроса (имя файла - идентификатор шаблона) в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.

Пример:

```
"options":
{
  "enable-debug-merged-doc-save": true,
  "output-mode": "base64",
  "enable-debug-pdf-log": true
}
```

5.2.3 Пример вызова сервиса

```
curl --request POST --data-binary "@templates/examples/pdf/embedded-report-generator/
↳request_example.json" http://localhost:8886/pdf_report_json
```

Структура ответа

В зависимости от **output-mode** ответ сервиса может быть либо потоком бинарных данных, либо ответом в формате JSON.

Ответ сервиса содержит объект в формате JSON, который включает:

1. Описание ошибки (код ошибки, сообщение об ошибке). В случае успешного ответа сервиса возвращается значение **null**.
2. Результат (закодированный в base64 сформированный PDF-файл документа или ссылка).

Формат ответа

```
{
  "error":
  {
    "code": "",
    "message": ""
  },
  "result": "[base64 encoded docx/pdf file] or [url]"
}
```

Пример ответа

```
{
  "error": null,
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="
}
```

Использование шаблонов страниц

Для формирования документа с помощью генератора PDF можно использовать готовые шаблоны страниц в формате PDF.

Для задания шаблона конкретной страницы существует специальная команда **setPageTemplate**. Подробнее про команду можно прочитать в подразделе “Команды генератора”.

Примечание

Поддерживаются многостраничные шаблоны.

Для расположения текста и других элементов на странице шаблона необходимо задавать координаты. Подробнее про задание координат элементов документа можно прочитать в подразделе “Координаты”.

Шаблоны страниц в формате PDF хранятся локально в директории **templates/pdf**, расположенной в директории приложения сервиса (*local*). Либо могут быть встроены прямо в запрос (*embedded*) в разделе **templates** в закодированном в base64 виде (*value*).

```
"templates":
[
  {
    "id": "template_1",
    "uri": "embedded",
    "value": "JVBERi0xLjIKJcfsj6IKNSA...4Kc3Rhc nR4cmVmCjM5MDE2CiU1RU9GCg=="
  }
],
"embedded-report-generator":
{
  "commands":
  [
    {
      "name": "NewPage"
    },
    {
      "name": "setPageTemplate",
      "params":
      {
        "page": 1,
        "template":
        {
          "id": "template_1",
          "uri": "embedded"
        }
      }
    }
  ]
},
```

Подробнее об использовании шаблонов в описании команды **setPageTemplate**.

Служебные директории

Скрипты генерации PDF хранятся в директории **pdf_report_gen**, расположенной в директории приложения сервиса.

Шаблоны страниц в формате PDF хранятся локально в директории **templates/pdf**, расположенной в директории приложения сервиса.

Примеры запросов находятся в директории **request_examples/pdf**.

Шрифты для использования в генераторе, находятся в директории **assets/fonts**, расположенной в директории приложения сервиса. Загружаются при старте сервиса.

i **Примечание**

При добавлении новых файлов шрифтов необходимо перезапустить сервис.

Примеры запросов и шаблонов

Примеры запросов находятся в директории **examples/pdf/embedded-report-generator** и **examples/pdf/report-generator** в *архиве примеров*.

Примеры шаблонов находятся в директории **examples/pdf** *архиве примеров*.

5.2.4 Команды генератора

PDF-документ генерируется при помощи специальных команд генератора.

Команды располагаются внутри объекта **embedded-report-generator**, если генератор встраивается в запрос (см. пункт “Генератор как часть запроса” выше).

Команды также могут располагаться в файле формата JSON, который находится в директории **pdf_report_gen**, расположенной в директории приложения сервиса (см. пункт “Генератор как часть сервиса” выше).

Команды описываются внутри массива **commands**.

Пример:

```
"commands": [
  {
    "name": "NewPage"
  },
  {
    "name": "setCoordinateMode",
    "params": {
      "mode": "millimeters"
    }
  },
  {
    "name": "setCellMargin",
    "params": {
      "margin": 1
    }
  }
]
```

Массив **commands** может включать команды, описанные ниже.

NewPage

Создает новую страницу в документе.

Параметры

- **orientation** - строка. Поддерживаемые значения:
 - «L» - альбомная ориентация
 - «P» - портретная ориентация

Значение по умолчанию - "P".

Пример:

```
{
  "name": "NewPage",
  "params": {
    "orientation": "L"
  }
}
```

setCoordinateMode

Задаёт трактовку координат и размеров, указываемых в последующих командах. Режим по умолчанию «в миллиметрах».

Параметры

- **mode** Поддерживаемые значения: «millimeters», «percents», «normalized».

setPageMargins

Устанавливает отступы от границ страницы.

Параметры

left, **right**, **top**, **bottom** - числа с плавающей точкой. Отступы от левого, правого, верхнего, нижнего краев страницы. При достижении нижнего отступа в командах типа **Row_Print** автоматически создается новая страница.

Пример:

```
{
  "name": "setPageMargins",
  "params": {
    "left": 20,
    "right": 20,
    "top": 30,
    "bottom": 15
  }
}
```

setCurrentX

Устанавливает X координату курсора.

Параметры

- **value** - число с плавающей точкой. Координата X.

setCurrentY

Устанавливает Y координату курсора.

Параметры

- **value** - число с плавающей точкой. Координата Y.

setCurrentXY

Устанавливает координаты курсора.

Параметры

- **x** - число с плавающей точкой. Координата X.
- **y** - число с плавающей точкой. Координата Y.

saveX

Сохраняет координату X для последующего использования в параметрах координат команд через тег **[PDF:savedX]**.

Параметры

- **value** - опциональный параметр, по умолчанию равен текущей координате X. Может быть числом с плавающей точкой или строкой-выражением со ссылками на встроенные теги. Пример выражения: «**[PDF:currentX] - 50**»

saveY

Сохраняет координату Y для последующего использования в параметрах координат команд через тег **[PDF:savedY]**. Внимание: при последующем создании новой страницы значение может оказаться нерелевантным.

Параметры

- **value** - опциональный параметр, по умолчанию равен текущей координате Y. Может быть числом с плавающей точкой или строкой-выражением со ссылками на встроенные теги. Пример выражения: «**[PDF:currentY] - 50**»

saveCoordinate

Сохраняет произвольное значение координат для последующего использования в параметрах координат команд через тег **[SAVED:ключ]**. Дополняет команды **saveX/saveY** в случаях, когда нужен доступ к нескольким значениям.

Параметры

- **key** - строка. Произвольный ключ для использования в теге **[SAVED:ключ]**
- **value** - может быть числом с плавающей точкой или строкой-выражением со ссылками на встроенные теги. Пример выражения: «**[PDF:currentY] - 50**»

Пример:

```
{
  "name": "saveCoordinate",
  "params": {
    "key": "x1",
    "value": 10
  }
}
```

setRotate

Задаёт поворот всех последующих объектов.

Параметры

- **angle** - число с плавающей точкой. Угол вращения в градусах.
- **x** - число с плавающей точкой. X координата точки вращения. Если null, то X координата курсора
- **y** - число с плавающей точкой. Y координата точки вращения. Если null, то Y координата курсора

Пример:

```
{
  "name": "setRotate",
  "params": {
    "angle": 45,
    "x": 105,
    "y": 130,
  }
}
```

Отмена команды:

```
{
  "name": "setRotate",
  "params": {
    "angle": 0
  }
}
```

startOpacity

Указывает степень прозрачности для всех последующих объектов. Действие отменяется командой **endOpacity**.

Параметры

- **val** - число с плавающей точкой. Допустимый диапазон от 0.0 до 1.0, где ноль - полная прозрачность.

endOpacity

Отменяет действие команды **startOpacity**.

Без параметров.

SetPrintFont

Задаёт шрифт и размер шрифта для последующих текстовых команд.

Параметры

- **font_id** Поддерживаемые значения: имена файлов шрифтов из директории **assets/fonts** без расширения. Шрифты загружаются при старте сервиса.
- **size** Размер шрифта в точках в пользовательском пространстве документа pdf.

Примечание

Единственная поддерживаемая кодовая страница - cp1251.

setColor4Text

Задаёт цвет текста для последующих текстовых команд.

Варианты входных параметров

- **r, g, b** целочисленные десятичные компоненты RGB цвета в диапазоне от 0 до 255. Пример красного цвета: “r”:255, “g”:0, “b”:0.
- **color** строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «color»:“#ff0000”.

setColor4Drawing

Задаёт цвет границ для объектов, выводимых после исполнения данной команды. Примеры объектов: ячейки таблиц, окружности, линии, прямоугольники и т.д.

Параметры: аналогично команде **setColor4Text**.

setColor4Filling

Задаёт цвет заливки внутренних областей для объектов, выводимых после исполнения данной команды. Примеры объектов: ячейки таблиц, окружности, прямоугольники и т.д.

Параметры: аналогично команде **setColor4Text**.

setLineWidth

Устанавливает толщину линий, выводимых после исполнения данной команды. Примеры объектов: границы ячеек таблиц, линии, прямоугольники и т.д.

Параметры

- **width** - число с плавающей точкой. Толщина линии.

setCellMargin

Устанавливает отступы контента табличной ячейки от левой, верхней, правой и нижней границ.

Параметры

- **margin** - число с плавающей точкой.

setCellBottomMargin

Устанавливает отступ контента табличной ячейки от нижней границы.

Параметры

- **margin** - число с плавающей точкой.

setCellLeftMargin

Устанавливает отступ контента табличной ячейки от левой границы.

Параметры

- **margin** - число с плавающей точкой.

setCellTopMargin

Устанавливает отступ контента табличной ячейки от верхней границы.

Параметры

- **margin** - число с плавающей точкой.

setCellRightMargin

Устанавливает отступ контента табличной ячейки от правой границы.

Параметры

- **margin** - число с плавающей точкой.

PrintCell

Выводит прямоугольную ячейку с текстом внутри. Можно указать цвет и наличие границ.

Параметры

- **w** - число с плавающей точкой. Ширина прямоугольника ячейки.
- **h** - число с плавающей точкой. Высота одной строки текста в ячейке. Если в ячейке однострочный текст, то высота прямоугольника ячейки равна высоте одной строки текста, а иначе высота прямоугольника ячейки будет складываться из высот ее строк текста.
- **txt** - текст
- **wrapping** - логическое значение. Указывает переносить текст на новые строки, в случае если он не умещается по ширине в ячейке. Значение по умолчанию - true. Отменить перенос текста может быть необходимо, если недопустимо изменение высоты прямоугольника ячейки.
- **line_h** - число с плавающей точкой. Параметр определяет интерлиньяж между строк, то есть межстрочный интервал. Если line_h не указан, то межстрочный интервал по умолчанию будет равен размеру шрифта.
- **border** - строка. Задаёт видимые границы ячейки. Допустимые значения: 0 - нет границ, 1 - внешняя граница прямоугольника, L - левая, T - верхняя, R - правая, B - нижняя границы. Допускается комбинация значений L,T,R и B.
- **align** - строка. Горизонтальное выравнивание текста. Значения: L - по левому краю. R - по правому краю, C - по центру, J - равномерно по ширине ячейки.
- **vert_align** - строка. Вертикальное выравнивание текста. Значения: T - по верхнему краю. B - по нижнему краю, C - по центру. Значение по умолчанию: C.
- **fill** - логическое значение. Указывает заполнять ли ячейку текущим цветом для заполнения. См. **setColor4filling**. Значение по умолчанию - false.
- **link** - строка. Ссылка, например, URL или внутренняя ссылка. В текущей версии не поддерживается.
- **clipping** - логическое значение. Указывает обрезать ли текст по границам ячейки. Значение по умолчанию - false.
- **ln** - целое число. Позиция курсора после отрисовки ячейки. Допустимые значения: 0 - рядом с ячейкой, 1 - новая строка, 2 - под ячейкой. Значение по умолчанию - 0.

Пример:

```
{
  "name": "PrintCell",
  "params": {
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "w":120,
    "h":7,
    "txt":"Персональные данные",
    "border":"0",
    "align":"L",
    "vert_align":"C",
    "fill":true,
    "clipping":false,
    "ln":2
  }
}

```

PrintMultiLineCell

Выводит многострочную ячейку

Параметры

- **w** - число с плавающей точкой. Ширина прямоугольника.
- **h** - число с плавающей точкой. Высота одной строки текста в ячейке. Если в ячейке однострочный текст, то высота прямоугольника ячейки равна высоте одной строки текста, а иначе высота прямоугольника ячейки будет складываться из высот ее строк текста.
- **txt** - текст
- **border** - строка. Задаёт видимые границы ячейки. Допустимые значения: 0 - нет границ, 1 - внешняя граница прямоугольника, L - левая, T - верхняя, R - правая, B - нижняя границы. Допускается комбинация значений L,T,R и B.
- **align** - строка. Горизонтальное выравнивание текста. Значения L - по левому краю. R - по правому краю, C - по центру, J - равномерно по ширине ячейки.
- **vert_align** - строка. Вертикальное выравнивание текста. Значения: T - по верхнему краю. B - по нижнему краю, C - по центру. Значение по умолчанию: C.
- **fill** - логическое значение. Указывает заполнять ли ячейку текущим цветом для заполнения. См. **setColor4Filling**. Значение по умолчанию - false.
- **maxline** - массив числе с плавающей точкой. Элемент массива задаёт максимальное количество строк текста в многострочной ячейке.
- **link** - строка. Ссылка, например, URL или внутренняя ссылка. В текущей версии не поддерживается.
- **clipping** - логическое значение. Указывает обрезать ли текст по границам ячейки. Значение по умолчанию - false.
- **indent** - число с плавающей точкой. Отступ для первой строки текста. Значение по умолчанию - 0.
- **ln** - целое число. Позиция курсора после отрисовки ячейки. Допустимые значения: 0 - рядом с ячейкой, 1 - новая строка, 2 - под ячейкой. Значение по умолчанию - 0.

Пример:

```

{
  "name":"PrintMultiLineCell",
  "params":{
    "h":5,
    "w":95,

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "txt":"Сумма поступивших средств за весь период",
    "border":"LTR",
    "align":"L",
    "vert_align":"T",
    "fill":true,
    "indent":0,
    "clipping":false,
    "ln":0
  }
}

```

Row_Print

Выводит строку в PDF документе. Строка состоит из многострочных ячеек(см. **PrintMultiLineCell**). Высота строки определяется ячейкой с наибольшей высотой. После выполнения команды курсор переходит на новую строку, если не задан флаг `stay_current_row`.

Параметры

- **data** - массив строк. Элемент массива задает текст ячейки в строке.
- **input_data_tag** - альтернатива data, имя массива строк-значений ячеек во входных данных запроса.
- **width** - массив чисел с плавающей точкой. Элемент массива задает ширину ячейки.
- **border** - массив строк. Элемент массива описывает видимые границы ячейки. См. описание команды **PrintMultiLineCell**. По умолчанию все границы видимы.
- **maxline** - массив чисел с плавающей точкой. Элемент массива задает максимальное количество строк текста в многострочной ячейке. По умолчанию ограничения нет.
- **align** - массив строк. Элемент массива задает горизонтальное выравнивание в ячейке.
- **vert_align** - массив строк. Элемент массива задает вертикальное выравнивание в ячейке.
- **font** - массив объектов описателей шрифта. Элемент массива задает параметры шрифта в ячейке. Набор полей каждого описателя аналогичен параметрам команды **SetPrintFont**. Значение по умолчанию - null, что означает использования настроек шрифта, установленного на предыдущем вызове **SetPrintFont**.
- **h** - число с плавающей точкой. Высота одной строки текста в ячейке (по умолчанию = 5). Если в ячейке однострочный текст, то высота прямоугольника ячейки равна высоте одной строки текста, а иначе высота прямоугольника ячейки будет складываться из высот ее строк текста.
- **min_height** число с плавающей точкой. Минимальная высота ячейки прямоугольника. Ноль означает, что параметр не используется. Значение по умолчанию - 0. Если данный параметр равен 0, то высота ячейки прямоугольника будет равна высоте сумме высот строк в ней.
- **fill** - логическое значение. Указывает заполнять ли ячейку текущим цветом для заполнения. См. `setColor4Filling`. Значение по умолчанию - false.
- **clipping** - логическое значение. Указывает обрезать ли текст по границам ячейки. Значение по умолчанию - false.
- **no-linebreak** - логическое значение. Отменяет перевод на новую строку после отрисовки, что позволяет выполнить несколько команд в рамках одной горизонтали. Значение по умолчанию - false.

Пример:

```

{
  "name": "Row_Print",
  "params": {
    "h": 4.5,
    "min_height": 27,
    "width": [85, 95],
    "data": [
      "1.4. Средства (часть средств) материнского (семейного) капитала",
      "0,00"
    ],
    "align": ["L", "L"],
    "vert_align": ["T", "T"],
    "border": ["0", "0"],
    "font": [
      {
        "size": 6,
        "font_id": "Arial-Bold"
      },
      {
        "size": 6,
        "font_id": "Arial"
      }
    ],
    "fill": true,
    "clipping": false
  }
}

```

PrintText

Выводит текст по заданным координатам.

Параметры

- **x** - число с плавающей точкой. X координата начала текста
- **y** - число с плавающей точкой. Y координата начала текста
- **text** - текст
- **position** - строковый параметр, определяет направление текста относительно X,Y. Поддерживаемые значения:
 - right - текст выводится справа от точки X,Y (по умолчанию)
 - middle - центр текста соответствует точке X,Y
 - left - текст выводится слева от точки X,Y

Пример:

```

{
  "name": "PrintText",
  "params": {
    "x": 10,
    "y": 40,
    "text": "Список документов:",

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "position": "middle"
  }
}

```

underline

Подчеркивает текст, выведенный с помощью команды **PrintText**.

Параметры аналогичны **PrintText**, значения параметров должны совпадать.

```

{
  "name": "underline",
  "params": {
    "x": 10,
    "y": 40,
    "text": "Список документов:",
    "position": "middle"
  }
}

```

LineBreak

Разрыв строки. Переносит курсор на начало следующей строки.

Параметры

- **h** - число с плавающей точкой. Отступ от текущей координаты Y курсора.

Line

Рисует прямую линию между двумя точками на странице.

Параметры

- **x1** - число с плавающей точкой. X координата начала линии.
- **y1** - число с плавающей точкой. Y координата начала линии.
- **x2** - число с плавающей точкой. X координата конца линии.
- **y2** - число с плавающей точкой. Y координата конца линии.
- **color** строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «color»:«#ff0000». По умолчанию используется цвет, предварительно заданный командой **setColor4Drawing**.
- **width** - число с плавающей точкой. Толщина линии. По умолчанию используется значение, предварительно заданное командой **setLineWidth**.

Пример:

```

{
  "name": "Line",
  "params": {
    "x1": 100,
    "y1": 230,
    "x2": 150,
    "y2": 230,

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "color": "#000000",
    "width": 0.2
  }
}

```

Circle

Рисует окружность на странице.

Параметры

- **x** - число с плавающей точкой. X координата центра окружности.
- **y** - число с плавающей точкой. Y координата центра окружности.
- **r** - число с плавающей точкой. Радиус окружности.
- **draw** - логическое значение. Управляет отрисовкой контура окружности. Значение по умолчанию - true.
- **fill** - логическое значение. Управляет заливкой цветом внутренней области окружности. Значение по умолчанию - false.
- **draw_color** цвет контура окружности. Строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «color»:«#ff0000». По умолчанию используется цвет, предварительно заданный командой **setColor4Drawing**.
- **fill_color** цвет контура окружности. строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «color»:«#ff0000». По умолчанию используется цвет, предварительно заданный командой **setColor4Filling**.
- **linewidth** - число с плавающей точкой. Толщина контура окружности. По умолчанию используется значение, предварительно заданное командой **setLineWidth**.

Пример:

```

{
  "name": "Circle",
  "params":
  {
    "x": 135,
    "y": 250,
    "r": 15,
    "draw": false,
    "fill": false,
    "draw_color": "#ff0000",
    "fill_color": "#ff0000",
    "linewidth": 0.1
  }
}

```

Rect

Рисует на странице прямоугольник/ прямоугольник со скругленными углами.

Параметры

- **x** - число с плавающей точкой. X координата левого верхнего угла прямоугольника.
- **y** - число с плавающей точкой. Y координата левого верхнего угла прямоугольника.

- **w** - число с плавающей точкой. Ширина прямоугольника.
- **h** - число с плавающей точкой. Высота прямоугольника.
- **r** - число с плавающей точкой. Радиус скругления углов.
- **draw** - логическое значение. Управляет отрисовкой контура окружности. Значение по умолчанию - true.
- **fill** - логическое значение. Управляет заливкой цветом внутренней области окружности. Значение по умолчанию - false.
- **draw_color** цвет контура окружности. Строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «color»:“#ff0000”. По умолчанию используется цвет, предварительно заданный командой **setColor4Drawing**.
- **fill_color** цвет контура окружности. строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «color»:“#ff0000”. По умолчанию используется цвет, предварительно заданный командой **setColor4Filling**.
- **linewidth** - число с плавающей точкой. Толщина контура окружности. По умолчанию используется значение, предварительно заданное командой **setLineWidth**.

Пример:

```
{
  "name": "Rect",
  "params":
  {
    "draw": true,
    "draw_color": "#ba03fc",
    "fill": true,
    "fill_color": "#d16ef5",
    "h": 10,
    "w": 20,
    "x": 140,
    "y": 105
  }
},
```

putImage

Параметры

- **name** - строка. Имя изображения или ID изображения. Если имя совпадает с указанным на предыдущем вызове, используется изображение, указанное в том же вызове.
- **data** - строка. Закодированный в base64 файл изображения. Если имя **name** совпадает с указанным на предыдущем вызове, данный параметр не указывается, чтобы не повторять в запросе блок base64 несколько раз.
- **x** - число с плавающей точкой. X координата изображения.
- **y** - число с плавающей точкой. Y координата изображения.
- **w** - число с плавающей точкой. Ширина изображения в единицах, заданных командой **setCoordinateMode**. Если параметр не указан, изображение выводится в натуральную ширину, которая высчитывается на основе плотности пикселей PDF-документа (72 точки на дюйм) и ширины изображения согласно файлу изображения. *Пример:* изображение 72x72 точки будет иметь размер 1 дюйм на 1 дюйм в PDF-документе.
- **h** - число с плавающей точкой. Высота изображения. Единицы измерения: аналогично параметру w.

- **link** - строка. URL или ID внутренней ссылки. В текущей версии не поддерживается.

Пример:

```
{
  "name": "putImage",
  "params": {
    "name": "img",
    "data": "base64 encoded image",
    "x": 100,
    "y": 205,
    "w": 50,
    "h": 37.5
  }
}
```

также в поле `data` можно передать имя тега изображения из раздела **input-data** (пример `embedded_template_and_image.json`)

```
"input-data":
{
  "IMG1": "iVBORw0KGgoAAAANSUhEUg2VDEy ... gg=="
},
...
{
  "name": "putImage",
  "params":
  {
    "h": 50,
    "w": 50,
    "x": 20,
    "y": 185,
    "data": "[IMG1]",
    "name": "img"
  }
}
```

setPageTemplate

Параметры

- **template** - объект в формате

```
{
  "uri": "local/embedded",
  "id": "template_example"
}
```

где идентификатор шаблона - это путь к документу pdf относительно директории **templates/pdf** без расширения либо идентификатор встроенного шаблона из раздела `templates`.

- **page** - целое число. Номер страницы шаблона для вывода. Диапазон от единицы до количества страниц в шаблоне.

Пример команды для многостраничного шаблона:

```
{
  "name": "setPageTemplate",
  "params": {
    "template": {
      "id": "example_multipage_template",
      "uri": "local"
    },
    "page": 1
  }
}
```

i Примечание

Происходит дублирование ресурсов шрифтов в отчете при использовании команды **setPageTemplate**, когда шрифт полностью совпадает с загруженным командой **SetPrintFont**. Может увеличить размер файла отчета.

setTemplate

Данная команда устанавливает весь PDF-файл в качестве шаблона. Данная команда используется для случаев, когда для генерации документа используется один многостраничный шаблон, и итоговый документ равен шаблону по количеству страниц. Таким образом, при вызове команды `newPage` автоматически применяется шаблон следующей страницы из файла шаблона.

Параметры

- **template** - объект в формате

```
{
  "uri": "local/embedded",
  "id": "template_example"
}
```

где идентификатор шаблона - это путь к документу pdf относительно директории **templates/pdf** без расширения либо идентификатор встроенного шаблона из раздела `templates`.

Пример:

```
{
  "name": "setTemplate",
  "params": {
    "template": {
      "id": "/example_logo_multi_page",
      "uri": "local"
    }
  },
  {
    "name": "NewPage"
  },
  {
    "name": "PrintText",
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

"params":
{
  "text": "Тест глобального шаблона",
  "x": 80,
  "y": 80
}
}

```

Примечание

для отмены использования шаблона для страниц необходимо вызвать команду `setTemplate` со значением `null` для параметра `template`.

```

"name": "setTemplate",
"params":
{
  "template": null
}

```

if

Команда позволяет выполнить набор подкоманд при истинности условия непосредственно в момент выполнения команды (в отличие от команды `register_auto_new_page_commands`).

Поддерживаемый набор команд для использования в качестве условия: **checkPageBreak**, **checkInputParam**.

Параметры

- **condition** - объект, описывающий команду, возвращающую логическое значение.
- **commands** - массив команд для выполнения в случае, если команда указанная в параметре **condition** вернула `true`.

Пример:

```

{
  "name": "if",
  "params": {
    "condition": {
      "name": "checkPageBreak",
      "params": {
        "h": 4.5,
        "newpage": true
      }
    }
  },
  "commands": [
    {
      "name": "setPageTemplate",
      "params": {
        "template": {
          "id": "example_template",
          "uri": "local"
        }
      }
    }
  ]
}

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    }
  },
  {
    "name": "setCurrentXY",
    "params": {
      "x": 10,
      "y": 30
    }
  }
]
}
}

```

Типичный вариант использования: перед попыткой вывода элемента или набора элементов в нижнюю часть страницы, когда есть риск выхода за нижний отступ страницы. Значение параметра **h** для проверки обычно берётся из размера элемента, который нужно вывести на экран.

i Примечание

Команда **if** работает непосредственно по месту вызова, с текущей координатой Y. То есть, если ее вызвать в момент, когда на странице есть вертикальное пространство высотой **h**, то список подкоманд исполнен не будет.

i Примечание

Не рекомендуется использовать команду **if** для вывода строк таблиц. Она может быть полезна для вывода уникального элемента, когда отрисовка происходит ближе к нижней части страницы. В случае таблиц лучше использовать **register_auto_new_page_commands**.

checkPageBreak

Если добавление указанной высоты к текущей координате Y приведет к переполнению страницы (то есть выходу за пределы нижнего отступа страницы), команда добавляет новую страницу и возвращает true (возвращаемое значение имеет значение только в команде **if**)

Параметры

- **h** - число с плавающей точкой. Высота для проверки на переполнение страницы.
- **newpage** - логическое значение. Указывает создавать ли новую страницу в документе при переполнении. Значение по умолчанию - true.

checkInputParam

Команда проверяет соответствие входных параметров из input-data указанным значениям и в случае полного соответствия возвращает true, а иначе false (возвращаемое значение имеет значение только в команде **if**).

Пример:

```

"name": "if",
  "params":
  {

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

"condition":
{
  "name": "checkInputParam",
  "params":
  {
    "GENDER": "Мужской",
    "PHONE": null,
    "NO": null,
    "DOC_DATE": "01.01.2000",
    "NOT_NULL": "notNull"
  }
},

```

Примечание

Для того, чтобы проверить параметр на **not null** необходимо указать строковую константу «notNull».

register_auto_new_page_commands

Позволяет зарегистрировать набор подкоманд для выполнения в случае, когда исполнение команд типа **Row_Print** приводит к автоматическому созданию новой страницы документа. Может быть полезно для рендера заголовка таблицы. Действует с момента вызова. Отмена возможна вызовом этой же команды с пустым списком подкоманд.

Параметры

- **commands** - массив команд для выполнения.

Пример:

```

{
  "name": "register_auto_new_page_commands",
  "params": {
    "commands": [
      {
        "name": "setPageTemplate",
        "params": {
          "template": {
            "id": "example_template",
            "uri": "local"
          }
        }
      },
    ],
    {
      "name": "setCurrentXY",
      "params": {
        "x": 10,
        "y": 30
      }
    }
  ]
}

```

Отмена команды:

```
{
  "name": "register_auto_new_page_commands",
  "params": {
    "commands": []
  }
}
```

PutLink

Команда позволяет создать область с ссылкой на странице. Если параметр

Параметры

- **w** - ширина области ссылки.
- **h** - высота области ссылки.
- **link** - ссылка.

Пример:

```
{
  "name": "PutLink",
  "params": {
    "w": 50,
    "h": 10,
    "link": "https://xsquare.ru/"
  }
}
```

5.2.5 Координаты

Команды, принимающие координаты и размеры в качестве параметров, трактуют значения на основании вызова команды **setCoordinateMode**. По умолчанию используются миллиметры.

Все подобные параметры могут быть заданы в числовом или текстовом варианте.

В текстовом виде поддерживаются выражения и специальные теги **PDF:currentX/****PDF:currentY**

Пример:

```
{
  "name": "Line",
  "params": {
    {
      "x1": "[PDF:currentX]",
      "y1": "[PDF:currentY]",
      "x2": "[PDF:currentX] + 50",
      "y2": "[PDF:currentY]",
      "color": "#ff0000",
      "width": 0.1
    }
  }
}
```

5.2.6 Теги

Команды, которые работают с текстом (**PrintText**, **PrintCell** и др.), могут содержать теги, которые будут заменены входными данными из запроса.

Входные данные содержатся в теле запроса с объекте **“input-data”: {...}** (см. пункт “Входные данные” подраздела “Структура запроса”).

Формат тегов

Тег задается в квадратных скобках. Например, [ORGANIZATION].

Пример:

```
{
  "name": "PrintMultiLineCell",
  "params": {
    "h": 4,
    "w": 77,
    "ln": 2,
    "txt": "[ORGANIZATION], в лице генерального директора Иванова Ивана Ивановича с
→одной стороны, и [CLINAME], именуемый в дальнейшем «Участник», с другой стороны,
→заключили настоящее Соглашение о нижеследующем:",
    "fill": false,
    "align": "J",
    "border": "0",
    "indent": 15,
    "clipping": false
  }
}
```

В запросе скобки не указываются (см. пример из пункта “Входные данные” подраздела “Структура запроса”).

Встроенные теги

Некоторые теги встроены в генератор PDF.

1. **PDF:currPageNum**, номер текущей страницы.
2. **PDF:currentX/PDF:currentY**, X и Y координаты курсора.
3. **PDF:savedX/PDF:savedY**, доступно после вызова команд **saveX/saveY**.
4. **[SAVED:ключ]**, доступно после команды **saveCoordinate**.

Пример:

```
{
  "name": "Line",
  "params": {
    "x1": "[PDF:savedX]",
    "x2": "[SAVED:endLineX]",
    "y1": "[PDF:currentY]",
    "y2": "[PDF:currentY]",
    "color": "#E8E8E8",
    "width": 0.8
  }
},
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
{
  "name": "PrintText",
  "params": {
    "x": "[SAVED:pageNumX]",
    "y": "[SAVED:pageNumY]",
    "text": "Номер страницы: [PDF:currPageNum]"
  }
}
```

5.2.7 Работа со штрихкодом

Возможен вывод информации в виде штрихкода посредством использования шрифтов семейства Libre Barcode. Шрифты доступны в репозитории [librebarcode](#). Каждый шрифт представляет собой определенный формат штрихкода. Поддерживаются следующие форматы:

- Code-128
- Code-39
- EAN13

Шрифт может включать оригинальный текст непосредственно под штрихкодом (шрифты с суффиксом Text).

Рассмотрим применение на примере формата Code-128.

Code-128

Шрифты для помещения в папку **assets/fonts** сервиса:

- LibreBarcode128Text-Regular.ttf
- LibreBarcode128-Regular.ttf

Как использовать

В генераторе команд выбрать нужный шрифт, например:

```
{
  "name": "SetPrintFont",
  "params": {
    {
      "font_id": "LibreBarcode128Text-Regular",
      "size": 30
    }
  }
}
```

Тестовый запрос

```
curl --request POST --data-binary "@templates/examples/pdf/embedded-report-generator/
↪request_barcode_code-128.json" http://localhost:8087/pdf_report_json
```

Объект описателя генератора PDF **report-generator** содержит:

- **uri** - строка. Задаёт положение источника команд генератора в зависимости от того, как трактуется параметр **id**. Поддерживаемые значения: **local** и **embedded**.
- **id** - строка. ID генератора.

Возможны два варианта использования генератора PDF:

- генератор как часть запроса (все команды генератора размещаются внутри объекта **embedded-report-generator** тела запроса);
- генератор как часть сервиса (все команды генератора размещаются внутри файла JSON в специальной папке **pdf_report_gen** на сервере).

Соответственно:

- При значении **uri**, равному **embedded**, **id** используется только для диагностических сообщений и идентификации запроса. Генератор должен находиться в корневом объекте **embedded-report-generator** запроса. См. пункт «Генераторы как часть запроса» ниже.
- При значении **uri**, равному **local**, **id** - это имя файла в директории сервиса **pdf_report_gen**. См. пункт «Генераторы как часть сервиса» ниже.

5.3 Сервис объединения PDF-документов

Сервис объединяет документы в формате PDF в один документ по HTTP-запросу в формате JSON.

Примечание

Формат XML не поддерживается в текущей версии.

Наименование сервиса	Сервис объединения PDF-документов
Путь к сервису	[host]:[port]/pdf_merge_json
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт файл документа в формате PDF, закодированный в base64.
Назначение	Сервис предназначен для объединения нескольких документов в формате PDF в один.

Тело запроса содержит объект в формате JSON, который включает:

1. Идентификатор запроса.
2. Входные данные: список описателей документов для объединения.
3. Опции запроса.

Пример:

```
{
  "request-id": "...",
  "input-data": {
    "document-descriptors": [
      {...},
      {...}
    ]
  },
  "options": {...}
}
```

Идентификатор запроса **request-id** служит для записи объединенного файла в отладочном режиме и для идентификации запроса в логе.

Объект входных данных **input-data** содержит:

Список (массив) **document-descriptors** описателей документов для объединения.

Пример:

```
"input-data": {
  "document-descriptors": [
    {...},
    {...}
  ]
}
```

Объект описателя входного документа поддерживает поля:

- **id** - строка. Идентификатор документа. Используется для ссылок на входной параметр в сообщениях об ошибках и диагностическом логе.
- **content-type** - строка. Указывает способ трактовки параметра **content**. Поддерживаемое значение: base64.
- **content** - строка. При значении типа контента base64 - закодированный в base64 PDF-документ. Примечание: Ограничения на количество страниц в документе нет.

Пример:

```
{
  "id": "doc1",
  "content-type": "base64",
  "content": "[doc 1 in base64 format]"
}
```

Объект опций запроса **options** имеет поля:

- **output-mode** - строка. Определяет форму предоставления результата. Поддерживаемые значения:
 - **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
 - **binary** - результат будет представлен в виде бинарного потока.
 - **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

Примечание: флаг `enable-binary-output` считается устаревшим, но для поддержки старых отчетов, продолжает использоваться.

- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **enable-debug-merged-doc-save** - логическое значение. Указывает создавать ли копию отчета документа и копию запроса (имя файла - идентификатор шаблона) в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.
- **enable-debug-pdf-log** - логическое значение. Включает расширенный диагностический лог создания PDF. Значение по умолчанию - **false**.

Пример:

```
"options": {
  "enable-debug-merged-doc-save": true,
  "output-mode": "base64",
  "enable-debug-pdf-log": true
}
```

```
{
  "request-id": "pdf_merge_example1",
  "input-data": {
    "document-descriptors": [
      {
        "id": "doc1",
        "content-type": "base64",
        "content": "[doc 1 in base64 format]"
      },
      {
        "id": "doc2",
        "content-type": "base64",
        "content": "[doc 2 in base64 format]"
      }
    ]
  },
  "options": {
    "enable-debug-merged-doc-save": true,
    "enable-debug-pdf-log": true
  }
}
```

Примеры запросов находятся в директории **examples/pdf/pdf_merge** в *архиве примеров*.

```
curl --request POST --data-binary "@templates/examples/pdf/pdf_merge/request_example.
↪json" http://localhost:8886/pdf_merge_json
```

В зависимости от **output-mode** ответ сервиса может быть либо потоком бинарных данных, либо ответом в формате JSON.

Ответ сервиса содержит объект в формате JSON, который включает:

1. Описание ошибки (код ошибки, сообщение об ошибке). В случае успешного ответа сервиса возвращается значение **null**.
2. Результат (закодированный в base64 PDF-файл объединенного документа или ссылка).

```
{
  "error":
  {
    "code": "",
    "message": ""
  },
  "result": "[base64 encoded docx/pdf file] or [url]"
}
```

```
{
  "error": null,
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="
}
```

i Примечание

Объединенный документ может быть большего размера, чем сумма объединяемых документов. Связано с дублированием одинаковых ресурсов (например, шрифтов), используемых во входящих документах.

5.4 Сервис формирования печатной формы

Сервис формирует печатную форму документа в формате PDF по HTTP-запросу в формате JSON:

- На последней странице документа генерируется таблица подписантов.
- На промежуточных страницах отображается колонтитул с краткой информацией о документе.

i Примечание

Формат XML не поддерживается в текущей версии.

Наименование сервиса	Сервис формирования печатной формы
Путь к сервису	[host]:[port]/print_form_pdf_json
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт файл печатной формы документа в формате PDF, закодированный в base64.
Назначение	Сервис предназначен для формирования печатной формы документа в формате PDF, включающей колонтитулы с краткой информацией о документе и таблицу подписантов на последней странице.

Тело запроса содержит объект в формате JSON, который включает:

1. Идентификатор запроса.
2. Входные данные:
 - Описатель входного документа для создания печатной формы (оригинал документа PDF для создания печатной формы).
 - Описатель таблицы подписантов и колонтитула промежуточных страниц (данные для подстановки в таблицу подписантов и колонтитул промежуточных страниц с данными о подписываемом документе).
 - Опции вида генерируемых таблиц (цвет/размер шрифтов).
3. Опции запроса.

Пример:

```
{
  "request-id": "...",
  "input-data":
  {
    "input-document": {...},
    "signatures-info": {...},
    "draw-options": {...}
  },
  "options": {...}
}
```

Идентификатор запроса **request-id** служит для записи файла в отладочном режиме и для идентификации запроса в логе.

Объект входных данных **input-data** содержит:

- Описатель входного документа для создания печатной формы.
- Описатель таблицы подписантов и колонтитула промежуточных страниц.
- Опции вида генерируемых таблиц.

Пример:

```
"input-data":
{
  "input-document": {...},
  "signatures-info": {...},
  "draw-options": {...}
}
```

Объект описателя входного документа для создания печатной формы **input-document** имеет поля:

- **id** - строка. Идентификатор документа. Используется для ссылок на входной параметр в сообщениях об ошибках и диагностическом логе.
- **content-type** - строка. Указывает способ трактовки параметра content. Поддерживаемое значение: base64.
- **content** - строка. При значении типа контента base64 - закодированный в base64 PDF-документ. Примечание: Ограничения на количество страниц в документе нет.

Пример:

```
"input-document":
{
  "id": "doc1",
  "content-type": "base64",
  "content": "[base64 encoded pdf file]"
}
```

Объект описателя таблицы подписантов и колонтитула промежуточных страниц **signatures-info** имеет поля:

- **signature-summary-text** - строка. Общее описание таблицы подписантов для вывода перед данными подписантов.
- **table-header-texts**- массив строк. Список заголовков таблицы подписантов. Примечание: в текущей версии может быть строго 4 столбца (ограничение элемента массива **signatures**, описанного ниже).

- **table-header-column-width-list** - массив чисел с плавающей точкой. Управляет шириной столбцов таблицы. Сумма всех нормализованных ширин должна равняться единице. Размер массива должен совпадать с размером поля **table-header-texts**.
- **signatures** - массив объектов с данными подписантов. Детали см. в пункте “Описатель данных подписанта” ниже.
- **intermediate-page-footer-info** - данные для вывода в колонтитуле на всех страницах документа, кроме последней, где находится таблица подписантов. Детали см. в пункте “Описатель колонтитула промежуточных страниц с данными о подписываемом документе” ниже.
- **logo-asset-name** - строка. Опциональный параметр. Имя файла изображения (с расширением) для вывода в правом верхнем углу таблицы подписантов. Путь задается относительно директории **assets/images/print_forms** в рабочем каталоге сервиса.

Пример:

```
"signatures-info": {
  "signature-summary-text": "...",
  "table-header-texts": [...],
  "table-header-column-width-list": [...],
  "signatures": [...],
  "intermediate-page-footer-info": {...},
  "logo-asset-name": "..."
}
```

Элемент массива описателя данных подписанта **signatures** имеет поля:

- **description** - строка. Общее описание принадлежности подписи. Текст выводится в первый столбец таблицы подписантов.
- **certificate-owner** - объект со строковыми полями **organization** и **employee**, описывающий организацию, ФИО и должность владельца сертификата. Текст выводится во второй столбец.
- **certificate** - объект со строковыми полями **serial-number** и **validity**, описывающий серийный номер и период действия сертификата, используемого для подписи. Текст выводится в третий столбец.
- **signing-timestamp** - строка. Описание точного времени подписи. Текст выводится в четвертый столбец.

Пример:

```
{
  "description": "Подпись отправителя",
  "certificate-owner": {
    "organization": "ООО \"ФИ-КРУГ\"",
    "employee": "Петров Иван Сидорович, Генеральный Директор"
  },
  "certificate": {
    "serial-number": "11255807011ABCBBE4DBF93ECCCB96C45",
    "validity": "с 01.02.2001 18:48 по 01.02.2031 18:48 GMT+03:00"
  },
  "signing-timestamp": "01.02.2021 19:00 GMT+03:00"
}
```

Объект описателя колонтитула промежуточных страниц с данными о подписываемом документе **intermediate-page-footer-info** имеет поля:

- **lines** - массив строк. Произвольные данные, описывающие подписываемый документ. Выводятся в виде таблицы с невидимыми границами. Последней строкой будет описание номера текущей страницы, если это

включено полем **print-page-number**.

- **logo-asset-name** - строка. Опциональный параметр. Имя файла изображения (с расширением) для вывода в правой части данных о документе. Путь задается относительно директории **assets/images/print_forms** в рабочем каталоге сервиса.
- **print-page-number** - логическое значение. Указывает выводить ли номер текущей страницы документа по данным о документе.

Пример:

```
{
  "lines": [
    "Передан через XXX 01.02.2003 15:20 GMT+03:00",
    "abcdef00-0000-1111-2222-abcdefabcdef"
  ],
  "logo-asset-name": "golang-gopher.png",
  "print-page-number": true
}
```

Объект опций вида генерируемых таблиц **draw-options** имеет поля:

- **font-color** - строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «font-color»:«#ff0000».
- **font-size** - целое число. Размер шрифта данных подписантов в таблице подписантов. Единица равна 1/72 дюйма.
- **large-font-size** - целое число. Размер шрифта вывода текста **signature-summary-text** в таблице подписантов. Единица равна 1/72 дюйма.
- **table-border-color** - строка в формате «#rrggbb», где rgb - шестнадцатеричные компоненты RGB цвета в диапазоне от 0 до ff. Пример красного цвета: «font-color»:«#ff0000». Цвет границ таблицы подписантов.

Пример:

```
{
  "font-color": "#0054BE",
  "font-size": 8,
  "large-font-size": 14,
  "table-border-color": "#0054BE"
}
```

Объект опций запроса **options** имеет поля:

- **output-mode** - строка. Определяет форму предоставления результата. Поддерживаемые значения:
 - **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
 - **binary** - результат будет представлен в виде бинарного потока.
 - **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

Примечание: флаг **enable-binary-output** считается устаревшим, но для поддержки старых отчетов, продолжает использоваться.

- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).

- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **enable-debug-doc-save** - логическое значение. Указывает создавать ли копию отчета документа и копию запроса (имя файла - идентификатор шаблона) в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.
- **enable-binary-output** - логическое значение. Указывает, что сервис будет выдавать результат в виде бинарных данных, без кодирования в base64. Значение по умолчанию - **false**.
- **enable-debug-pdf-log** - логическое значение. Активирует расширенный диагностический лог создания PDF. Значение по умолчанию - **false**.

Пример:

```
{
  "options": {
    "enable-debug-doc-save": true,
    "output-mode": "base64",
    "enable-debug-pdf-log": true
  }
}
```

```
{
  "request-id": "print_form_pdf_example1",
  "input-data": {
    "input-document": {
      "id": "doc1",
      "content-type": "base64",
      "content": "JVBERi0xLjcKJYGBgYEKcjYgMCMCB...VmCjI1ODI5CiU1RU9G"
    },
    "signatures-info": {
      "signature-summary-text": "Документ подписан и передан через оператора АО_
↔«Пример»",
      "table-header-texts": [
        "",
        "Владелец сертификата: организация, сотрудник",
        "Сертификат: серийный номер, период действия",
        "Дата и время подписания"
      ],
      "table-header-column-width-list": [
        0.15,
        0.33,
        0.32,
        0.2
      ],
      "signatures": [
        {
          "description": "Подпись отправителя",
          "certificate-owner": {
            "organization": "ООО \"ФИ-КРУГ\"",
            "employee": "Петров Иван Сидорович, Генеральный Директор"
          },
          "certificate": {
            "serial-number": "11255807011ABCBBE4DBF93ECCCB96C45",

```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

        "validity": "с 01.02.2001 18:48 по 01.02.2031 18:48 GMT+03:00"
    },
    "signing-timestamp": "01.02.2021 19:00 GMT+03:00"
},
{
    "description": "Подпись получателя",
    "certificate-owner": {
        "organization": "ООО «СНАБЖЕНИЕ КОНЦЕССИЙ»",
        "employee": "Сидоров Петр Иванович, Директор"
    },
    "certificate": {
        "serial-number": "3610D7230004000503BF",
        "validity": "с 04.05.2006 16:07 по 31.12.2021 23:59 GMT+03:00"
    },
    "signing-timestamp": "02.02.2021 13:00 GMT+03:00"
}
],
"intermediate-page-footer-info": {
    "lines": [
        "Передан через XXX 01.02.2003 15:20 GMT+03:00",
        "8f109134-403b-4de5-aa09-6d10462ec071"
    ],
    "logo-asset-name": "golang-gopher.png",
    "print-page-number": true
},
"logo-asset-name": "golang-gopher.png"
},
"draw-options": {
    "font-color": "#0054BE",
    "font-size": 8,
    "large-font-size": 14,
    "table-border-color": "#0054BE"
}
},
"options": {
    "enable-debug-pdf-log": true
}
}

```

Примеры запросов находятся в директории **examples/pdf/print_form** в *архиве примеров*.

```
curl --request POST --data-binary "@templates/examples/pdf/print_form/request_
->example1.json" http://localhost:8886/print_form_pdf_json
```

В зависимости от **output-mode** ответ сервиса может быть либо потоком бинарных данных, либо ответом в формате JSON.

Ответ сервиса содержит объект в формате JSON, который включает:

1. Описание ошибки (код ошибки, сообщение об ошибке). В случае успешного ответа сервиса возвращается значение **null**.
2. Результат (закодированный в base64 PDF-файл печатной формы документа или ссылка).

```
{
  "error":
  {
    "code": "",
    "message": ""
  },
  "result": "[base64 encoded docx/pdf file] or [url]"
}
```

```
{
  "error": null,
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="
}
```

Изображения для использования в качестве логотипа, находятся в директории `assets/images/print_forms`, расположенной в директории приложения сервиса.

5.5 Сервис заполнения PDF-форм

Сервис позволяет заполнить интерактивные формы PDF документа на основе данных запроса в формате JSON.

Наименование сервиса	Сервис заполнения PDF-форм
Путь к сервису	[host]:[port]/pdf_fill_form
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт документ в формате PDF, закодированный в base64 (по умолчанию)
Назначение	Сервис позволяет заполнить интерактивные формы PDF документа на основе данных запроса в формате JSON.

Тело запроса содержит объект в формате JSON, который включает:

1. Описатель шаблона.
2. Входные данные:
 - Имена интерактивных элементов форм и их значения.
3. Опции запроса.

Пример:

```
{
  "template":
  {
    "uri": "...",
    "id": "..."
  },
  "input-data":
  {
    "..."
  },
}
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
"options": {...}
}
```

Элемент (объект) описателя шаблона `template` содержит:

- **uri** - строка. Задаёт расположение файла документа-шаблона в зависимости от того, как трактуется параметр `id`.

Поддерживаемые значения:

- **local** - расположение шаблона в каталоге `templates`.
- **embedded** - шаблон включен в запрос в виде закодированной base64 строки. Значение строки должно быть записано в параметр `value`.
- **remote** - шаблон расположен на удаленном файловом сервере. Ссылка на шаблон должна быть указана в параметре `value`.
- **id** - строка. Идентификатор шаблона. Путь к файлу документа-шаблона относительно директории `templates`. Также используется для записи файла отчета в отладочном режиме и для идентификации запроса в логге.
- **value** - строка. Значение используется в режимах `embedded` и `remote`.

Пример для формата JSON:

```
"template":
{
  "uri": "local",
  "id": "forms_example_1"
}
```

```
"template":
{
  "uri": "embedded",
  "id": "forms2",
  "value": "UESDBBQACAgIAPyFilcAAAAAAAAAAAAAAAAAA...LAAAAX3JlbHMvLnJlbHOtkt1KA0EMh"
}
```

```
"template":
{
  "uri": "remote",
  "id": "forms3",
  "value": "http://directlink/templates/forms3.pdf"
}
```

Элемент (объект) входных данных **input-data** содержит:

- Имена однострочных и многострочных элементов ввода и текст для заполнения
- Имена `radiobuttons` и значение состояния (On/Off или Yes/No, в зависимости от стандарта PDF-документа)
- Имена `checkboxes` и значение состояния (On/Off или Yes/No, в зависимости от стандарта PDF-документа)

Элемент (объект) опций запроса **options** содержит:

- **output-mode** - строка. Определяет форму представления результата.

Поддерживаемые значения:

- **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
- **binary** - результат будет представлен в виде бинарного потока.
- **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.
- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **enable-debug-doc-save** - логическое значение. Указывает создавать ли копию отчета документа и копию запроса (имя файла - идентификатор шаблона) в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.
- **enable-debug-pdf-log** - логическое значение. Включает расширенный диагностический лог создания PDF. Значение по умолчанию - **false**.
- **finalize** - логическое значение. Указывает включение/отключение интерактивности, т.е. при значении true документ будет финализирован и формы будут в режиме read-only.

```
"options": {
  "enable-debug-doc-save": false,
  "output-mode": "binary",
  "enable-debug-pdf-log": true,
  "finalize": false
}
```

```
{
  "template":
  {
    "uri": "local",
    "id": "examples/pdf/fill_form/simple_form"
  },
  "input-data":
  {
    "checkbox_4ljvs": "On",
    "radio_group_3yqbn": "On",
    "text_1hfnc": "новое значение текстового поля1",
    "text_2vxyw": "new value of text field2"
  },
  "options":
  {
    "enable-debug-doc-save": false,
    "output-mode": "binary",
    "enable-debug-pdf-log": true,
    "finalize": false
  }
}
```

Примеры запросов находятся в директории **examples/pdf/fill_form** в *архиве примеров*.

```
curl --request POST --data-binary "@templates/examples/pdf/fill_form/forms.json" -
↳http://localhost:8886/pdf_fill_form
```

В зависимости от **output-mode** ответ сервиса может быть либо потоком бинарных данных, либо ответом в формате JSON.

Ответ сервиса содержит объект в формате JSON, который включает:

1. Описание ошибки (код ошибки, сообщение об ошибке). В случае успешного ответа сервиса возвращается значение **null**.
2. Результат (закодированный в base64 PDF-файл или ссылка).

```
{
  "error":
  {
    "code": "",
    "message": ""
  },
  "result": "[base64 encoded pdf file] or [url]"
}
```

```
{
  "error": null,
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="
}
```

5.6 Сервис конвертации XLSX документов в JSON, XML, CSV

Сервис конвертирует XLSX документ в форматы JSON, XML, CSV по HTTP-запросу в формате JSON или по HTTP-запросу в формате multipart/form-data.

В ответ сервис отдает содержимое входного документа представленное в одном из целевых форматов.

Наименование сервиса	Сервис конвертации XLSX документов в JSON, XML, CSV
Путь к сервису	Для multipart/form-data запросов [host]:[port]/xlsx_convert Для json запросов [host]:[port]/xlsx_convert_json
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON или объекты в формате multipart/form-data. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт содержимое документа в целевом формате.
Назначение	Сервис предназначен для конвертирования XLSX документа в форматы JSON, XML, CSV

В общем виде запросы в формате JSON или multipart/form-data включают:

- Описатель документа.
- Опции запроса.

Пример для JSON запроса:

```
{
  "document":
  {
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

    "name": "",
    "data": ""
  },
  "options": {...}
}

```

Описатель документа для формата JSON имеет поля:

- **name** - строка. Имя - идентификатор документа. Используется для ссылок на входной параметр в сообщениях об ошибках и диагностическом логге.
- **data** - строка. Закодированный в BASE64 XLSX-документ.

Описатель документа для формата multipart/form-data состоит из параметра `xlsx=>@[путь к XLSX-документу]`

Например:

```

curl --request POST -F xlsx="@convert_example.xlsx" -F output-format="xml" http://
↳localhost:8886/xlsx_convert

```

Запрос может содержать следующие опции запроса:

- **output-mode** - строка. Определяет форму предоставления результата. Поддерживаемые значения:
 - **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
 - **binary** - результат будет представлен в виде бинарного потока.
 - **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

Примечание: флаг `enable-binary-output` считается устаревшим, но для поддержки старых отчетов, продолжает использоваться.

- **output-format** - строка. Целевой формат конвертации. Поддерживаемые значения: "JSON", "XML", "CSV". Значение по умолчанию - **JSON**.
- **domain-name** - строка. Определяет имя домена для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **file-name** - строка. Определяет имя файла для формирования пути сохранения файла отчета при использовании файлового сервера (предоставление отчета по ссылке).
- **process-sheets-numbers** - массив целых чисел. Определяет порядковые номера страниц для конвертации из входного документа (Например: [1,2,3] - для обработки первых трех страниц). Нумерация начинается с "1". Значение по умолчанию - **все страницы** для JSON и XML, **первая страница** для CSV.
- **process-from-start-of-sheet** - логическое значение. Указывает начать обработку документа с самой первой ячейки (A1) независимо есть ли в ней данные или нет. Иначе обработка будет производиться с первой ячейки с данными. Значение по умолчанию - **false**.
- **no-extend-line** - логическое значение. Данная опция отвечает за дополнение каждой строки исходного документа до максимальной длины строки в документе с целью получения двумерного массива данных (именно такое поведение конвертора предусмотрено по умолчанию). При значении `true` опция указывает не дополнять строки и на выходе будет массив массивов данных. Значение по умолчанию - **false**.
- **process-start-cell** - строка. Данная опция позволяет указать стартовую ячейку для обработки (в формате именованной ячеек, например "E5"), то есть задает левую верхнюю границу обработки по строке-столбцу,

что позволяет конвертировать только интересующий диапазон данных. Значение по умолчанию - **первая ячейка с данными** (будет определена автоматически).

- **process-end-cell** - строка. Данная опция позволяет указать конечную ячейку для обработки (в формате именованной ячейки, например "G8"), то есть задает правую нижнюю границу обработки по строке-столбцу, что позволяет конвертировать только интересующий диапазон данных. Значение по умолчанию - **последняя ячейка с данными** (будет определена автоматически).
- **ignore-empty-rows** - логическое значение. Указывает пропускать при экспорте строки без данных. Данная опция доступна только для экспорта в JSON, XML, так как эти форматы содержат нумерацию строк. Значение по умолчанию - **false**.
- **encoding** - строка. Данная опция позволяет указать целевую кодировку результата обработки. Поддерживаемые значения: WIN1250, WIN1251, WIN1252, WIN1253, WIN1254, WIN1255, WIN1256, WIN1257, WIN874, WIN866, KOI8R, ISO_8859_5. Значение по умолчанию - **UTF8**.
- **enable-debug-report-save** - логическое значение. Указывает создавать ли копию запроса в локальной директории **reports_debug** сервиса. Значение по умолчанию - **false**.

Для запросов в формате JSON опции передаются в объекте **options**

Пример:

```
"options":
{
  "output-format": "json",
  "no-extend-line": false,
  "process-sheets-numbers": [1, 2],
  "process-from-start-of-sheet": false,
  "process-start-cell": "C2",
  "process-end-cell": "G5",
  "enable-debug-report-save": true
}
```

Для запросов в *multipart/form-data* формате каждая опция передается как отдельный параметр, все параметры задаются как строки.

Пример:

```
curl --request POST -F "xlsx=@convert_example.xlsx" -F output-format="json" -F no-
↪extend-line="false" -F process-sheets-numbers="1,2" -F process-from-start-of-sheet=
↪"false"
-F process-start-cell="C2" -F process-end-cell="G5" http://localhost:8886/xlsx_convert
```

Примеры запросов находятся в директории **examples/xlsx** в *архиве примеров*.

Для запросов в формате JSON:

```
curl --request POST --data-binary "@convert_example.json" http://localhost:8886/xlsx_
↪convert_json > convert_example_from_json.json
```

Для запросов в формате *multipart/form-data*:

```
curl --request POST -F "xlsx=@convert_example.xlsx" -F output-format="xml" http://
↪localhost:8886/xlsx_convert > convert_example1.xml

curl --request POST -F "xlsx=@convert_example.xlsx" -F output-format="csv" -F process-
↪from-start-of-sheet="false" -F process-start-cell="C2" -F process-end-cell="G5"
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

http://localhost:8886/xlsx_convert > convert_example2.csv

curl --request POST -F "xlsx=@convert_example.xlsx" -F output-format="json" -F no-
↪extend-line="true" -F process-sheets-numbers="1,2" -F process-from-start-of-sheet=
↪"false"
http://localhost:8886/xlsx_convert > convert_example2.json

```

Ответ сервиса содержит текстовый поток данных в целевой кодировке

Рассмотрим в качестве примера конвертацию диапазона D2:G4 второй страницы документа `convert_example.xlsx` из *архива примеров*:

	A	B	C	D	E	F	G	H
1								
2		№ п/п	Номер страхового свидетельства обязательного пенсионного страхования застрахованного лица	Пол застрахованного лица	Фамилия, имя, отчество (при наличии) застрахованного лица	Число, месяц, год рождения застрахованного лица	Дата заключения договора об обязательном пенсионном страховании	Номер договора об обязательном пенсионном страховании
3		1	001-002-003 00	Ж	Горбунова Екатерина Васильевна	01.01.1970	01.01.1988	ABC-001-002-003-00
4		2	001-002-004 00	М	Харитонов Степан Эдуардович	02.01.1970	02.01.1988	ABC-001-002-004-00
5		3	001-002-005 00	Ж	Шестакова Анна Львовна	03.01.1970	03.01.1988	ABC-001-002-005-00
6								

Запрос в формате `multipart/form-data`:

```

curl --request POST -F "xlsx=@convert_example.xlsx" -F output-format="csv" -F process-
↪sheets-numbers="2" -F process-start-cell="D2" -F process-end-cell="G4"
http://localhost:8886/xlsx_convert

```

Результат конвертации:

```

Пол застрахованного лица;Фамилия, имя, отчество (при наличии) застрахованного лица;
↪Число, месяц, год рождения застрахованного лица;Дата заключения договора об
↪обязательном пенсионном страховании
Ж;Горбунова Екатерина Васильевна;01.01.1970;01.01.1988
М;Харитонов Степан Эдуардович;02.01.1970;02.01.1988

```

5.7 Сервис конвертации XLS документов в JSON, XML, CSV

Сервис конвертирует XLS (Microsoft Office Excel Binary 2003) документ в форматы JSON, XML, CSV по HTTP-запросу в формате JSON или по HTTP-запросу в формате `multipart/form-data`.

В ответ сервис отдает содержимое входного документа представленное в одном из целевых форматов.

Наименование сервиса	Сервис конвертации XLS документов в JSON, XML, CSV
Путь к сервису	Для multipart/form-data запросов [host]:[port]/xls_convert Для json запросов [host]:[port]/xls_convert_json
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON или объекты в формате multipart/form-data. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис отдаёт содержимое документа в целевом формате.
Назначение	Сервис предназначен для конвертирования XLS документа в форматы JSON, XML, CSV

Работа сервиса, функционал и опции полностью аналогичны сервису конвертации XLSX, за исключением того, что вместо параметра `xlsx` в multipart/form-data запросе используется параметр `xls`.

Пример:

```
curl --request POST -F "xls=@convert_xls.xls" -F output-format="json" http://localhost:8886/xls_convert
```

5.8 Сервис генерации кодов (QR code)

Сервис генерирует графические коды в форматах JPG, PNG по HTTP-запросу в формате JSON. В данной версии сервис позволяет генерировать QR code с гибкой настройкой параметров результирующего изображения.

Наименование сервиса	Сервис генерации кодов
Путь к сервису	[host]:[port]/code
Метод	POST
Параметры	Тело запроса должно содержать объект в формате JSON. Подробнее про структуру тела запроса можно прочитать в подразделе “Структура тела запроса”. В ответ сервис возвращает содержимое графического файла в формате JPG или PNG.
Назначение	Сервис предназначен для генерации графических кодов в формате JPG, PNG.

Пример:

```
curl --request POST --data-binary @"qr-code" http://localhost:8886/code
```

Тело запроса содержит объект в формате JSON, который включает:

1. Идентификатор запроса.
2. Входные данные: параметры кода.
3. Опции запроса.

Минимальный запрос для создания QR code должен содержать тип запроса (type) и кодируемые данные (content).

Пример:

```
{
  "id": "qr-code",
  "input-data":
  {
    "type": "qr-code",
    "content": "https://xsquare.ru/"
  },
  "options":
  {
    "output-mode": "binary"
  }
}
```

Все остальные параметры в данном случае будут иметь значения по умолчанию, что позволяет сгенерировать «классический» QR code в формате JPG:



Идентификатор запроса **id** используется для записи результата в отладочном режиме, а также для идентификации запроса в логах.

Объект входных данных **input-data** содержит описатель создаваемого графического кода:

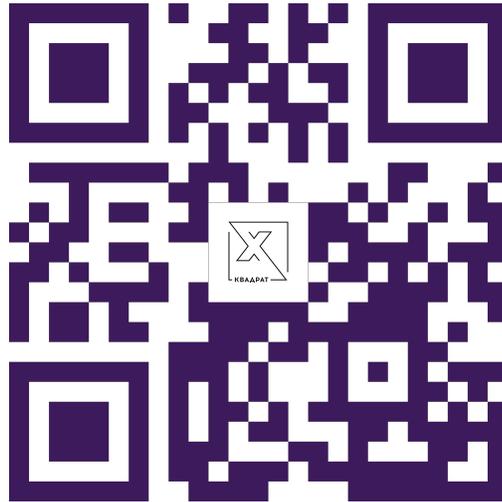
- **type** – строка. Обязательный параметр. Тип генерируемого кода. Поддерживаемые значения: «qr-code».
- **content** – строка. Обязательный параметр. Кодированные данные.
- **qr-color** – строка. Цвет точек кода в шестнадцатеричном формате записи. По умолчанию - черный.
- **bg-color** – строка. Цвет фона кода в шестнадцатеричном формате записи. По умолчанию - белый.
- **bg-transparent** – флаг. Определяет прозрачность фона для кодов в формате PNG. По умолчанию – false.
- **dot-size** – число. Размер точки кода в пикселях.
- **format** – строка. Формат результата. Возможные варианты JPG (по умолчанию), PNG.
- **logo** – строка. Закодированное в base64 изображение логотипа для вставки в центр QR code. Поддерживаются форматы JPG, PNG. Размер закодированного логотипа не должен превышать 1/5 размера кода.

Пример:

```

{
  "id": "qr-code",
  "input-data":
  {
    "type": "qr-code",
    "content": "https://xsquare.ru/",
    "qr-color": "#3d1a5c",
    "bg-transparent": true,
    "dot-size": 40,
    "format": "png",
    "logo": "iVBORw0KGgoAAAANSUhEUgAAAMg ... e3o7DbGAH/
↪LAAAAABJRU5ErkJggg=="
  },
  "options":
  {
    "output-mode": "binary"
  }
}

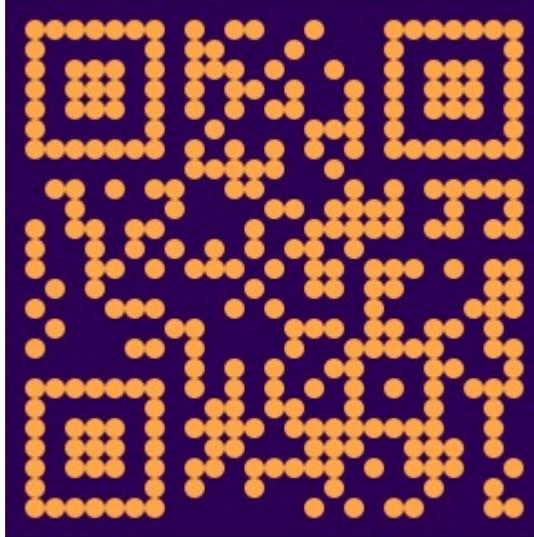
```



- **style** – строка. Стиль QR code.

Варианты стилей:

- **dot**



– rounded_rectangle



– rectangle_with_border



Если стиль не указан - используется стиль по умолчанию.



Таким образом параметры позволяют гибко настроить внешний вид результирующего кода, а используя определенный формат содержимого можно создавать коды для передачи различного рода информации.

Пример VCARD QR code:

```
{
  "id": "qr-code",
  "input-data":
  {
    "type": "qr-code",
    "content": "BEGIN:VCARD\nVERSION:3.0\nFN:Иван Иванов\nN:Иванов;Иван;;; \
↪nORG:Компания ХИ-КВАДРАТ\nTEL;TYPE=WORK,VOICE:+7 (499) 703-38-99\nTEL;TYPE=CELL,
↪VOICE:+7 (499) 703-38-99\nEMAIL:info@xsquare.ru\nADR;TYPE=WORK;;; Нижняя
↪Красносельская ул., дом 35 стр. 9;Москва;;105066;Россия\nURL:http://xsquare.ru\
↪nEND:VCARD",
    "qr-color": "#2bab00",
    "bg-color": "#150157",
  }
}
```

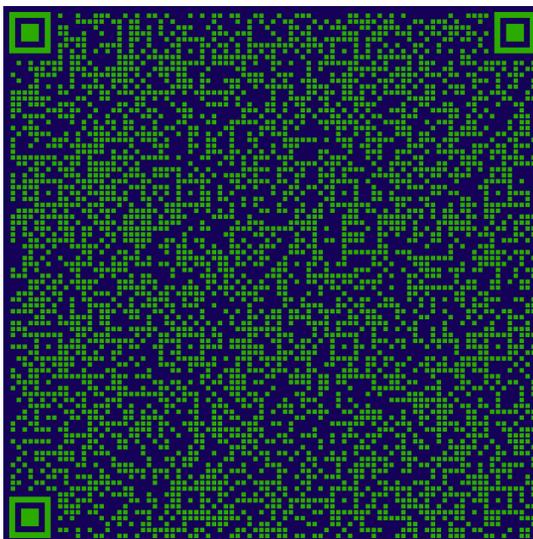
(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```

"style": "rectangle_with_border",
"format": "png"
},
"options":
{
  "output-mode": "binary"
}
}

```



Элемент (объект) опций запроса **options** содержит:

- **output-mode** - строка. Определяет форму представления результата.

Поддерживаемые значения:

- **base64** - результат будет представлен в виде закодированной в base64 строки. Данное значение используется по умолчанию.
- **binary** - результат будет представлен в виде бинарного потока.
- **url** - результат будет представлен в виде ссылки на файл отчета, который можно скачать с сервера отчетов.

5.9 Сервис конвертации документов DOCX/XLSX в формат PDF

Сервис позволяет сконвертировать документы DOCX/XLSX в формат PDF с помощью Libre Office.

Наименование сервиса	Сервис конвертации документов DOCX/XLSX в формат PDF
Путь к сервису	[host]:[port]/code
Метод	POST
Параметры	multipart/form-data запрос с параметром file - путь к документу В ответ сервис возвращает содержимое PDF файла.
Назначение	Сервис позволяет сконвертировать документы DOCX/XLSX в формат PDF с помощью Libre Office.

Пример:

```
curl -X POST -F "file=@/путь_к_документу" localhost:8886/2pdf > выходной_файл.pdf
```

Свойства готовых отчетов (метаданные)

XREPORTS позволяет заполнить свойства готового документа (метаданные), такие как заголовок, автор, тема, создатель, продюсер.

6.1 Поддерживаемые форматы

Функция заполнения свойств доступна для следующих форматов:

- DOCX
- XLSX
- PDF

6.2 Доступные метаданные

- Title - заголовок
- Author - автор
- Subject - тема
- Creator - создатель
- Producer - продюсер

6.3 Структура запроса

В запрос добавлено новое поле metadata, содержащее свойства документа в виде пар «ключ-значение».

Пример JSON-запроса:

```
{
  "input-data":
  {
    "DATE": "01.01.2023",
    "EMP": "Иванов Иван Иванович",
    "ORGANIZATION": "АО «Пример»"
  },
  "metadata":
  {
    "author": "автор",
    "creator": "создатель",
    "subject": "тема",
    "title": "заголовок"
  },
  "options":
  {
    "output-mode": "binary"
  },
  "template":
  {
    "id": "examples/docx/metadata",
    "uri": "local"
  }
}
```

Примеры запросов metadata.json в *архиве примеров*.

Генератор запросов

7.1 Описание сервиса

Сервис представляет собой веб-интерфейс для генерации запросов в формате JSON на основе документов-шаблонов в форматах DOCX/XLSX/PDF-forms.

Генератор запросов позволяет:

- автоматизировать и ускорить создание запросов на основе разработанных шаблонов.
- проводить проверку запросов благодаря быстрой генерации отчетов прямо в интерфейсе генератора запросов.
- быстро разобраться со структурой и параметрами запросов на практике.

7.2 Путь к сервису

Для работы с сервисом необходимо в адресной строке браузера открыть страницу генератора запросов:

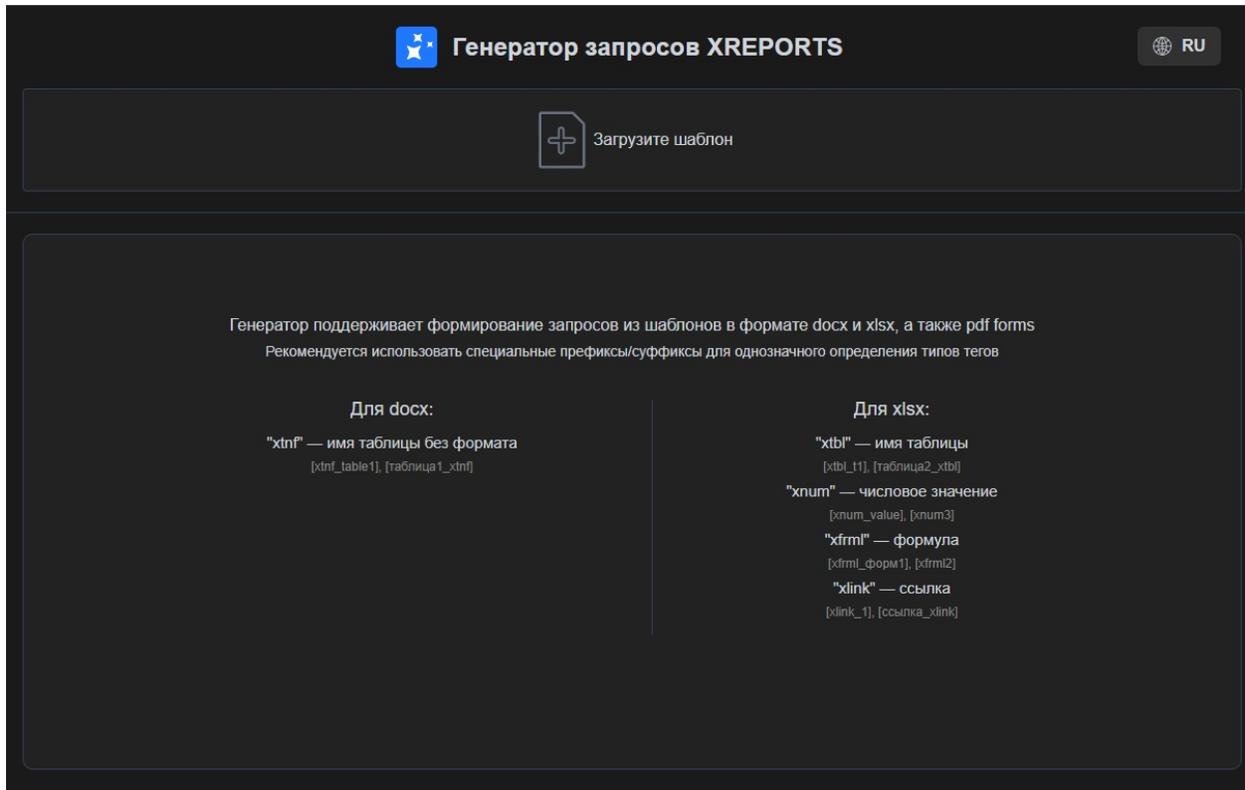
[host]:[port]/xreports/req_gen

Например:

```
http://localhost:8886/xreports/req_gen
```

7.3 Интерфейс

Стартовая страница генератора запроса состоит из блока для загрузки шаблона и блока со справочной информацией.



Основная страница генератора запросов состоит из:

- блока для загрузки шаблона (drag&drop либо выбор файла)
- блока curl, который содержит команду для выполнения запроса к xreports.
- блока запроса, который содержит JSON запроса и элементы управления запросом.
- блока параметров, который содержит параметры для формирования запроса в зависимости от его типа.

7.4 Описание работы

Генератор запросов поддерживает загрузку шаблонов документов в формате DOCX, XLSX и PDF-форм. После загрузки файла шаблона сервис сканирует шаблон на наличие тегов и формирует json запроса с тестовыми подстановочными данными.

Пользователь может отредактировать тестовые данные вручную, а также настроить опции запроса. После того, как запрос готов, пользователь может скопировать/скачать запрос или сгенерировать отчет на основе текущего запроса.

Для того, чтобы начать работу с новым шаблоном, необходимо загрузить новый документ через блок выбора файла.

7.5 Именованные теги в шаблонах

Генератор запросов различает большинство тегов из документов шаблонов, но некоторые объектные теги неотличимы от обычных тегов данных по структуре шаблона, поэтому для корректного распознавания необходимо придерживаться определенных правил именования тегов.

Для DOCX:

В шаблонах DOCX тег таблицы без формата должен содержать константу **xtnf** в начале или конце названия тега:

```
[xtnf_table1], [таблица1_xtnf], [xtnf], [тблxtnf]
```

Для XLSX:

В шаблонах XLSX тег таблицы должен содержать константу **xtbl** в начале или конце названия тега:

```
[xtbl_t1], [таблица2_xtbl]
```

Все теги подстановки по умолчанию распознаются как строковые, таким образом тестовые значения в запросе будут строковые. Для того, чтобы генератор подставил тестовые значения тегов правильного типа нужно в тегах использовать следующие константы:

- **xnum** — тег с числовым значением. Например: [xnum_value], [xnum3]
- **xfrml** — тег с формулой. Например: [xfrml_форм1], [xfrml2]
- **xlink** — тег с ссылкой. Например: [xlink_1], [ссылка_xlink]

7.6 API генератора

Генератор запросов имеет API, который позволяет создать свой интерфейс генератора запросов и внедрить его в свои системы.

Ниже приведено описание доступных методов API.

Метод	Путь	Описание	Параметры / Тело	Тело	Ответ
POST	/xreports/api/request	Загрузка шаблона документа	Тип: multipart/form-data Поле: template_file (файл шаблона)		200 OK cookies: session_id
GET	/xreports/api/data	Получение тела запроса в JSON	cookies: session_id		200 OK Тело: JSON запроса
POST	/xreports/api/cmd	Установка опции (cmd)	cookies: session_id Тело: команда		204 No Content
GET	/xreports/api/curl	Получение curl для запроса	cookies: session_id		200 OK Тело: JSON с curl
POST	/xreports/api/report	Генерация текущего отчета	cookies: session_id Тело: JSON запроса		200 OK Тело: Бинарный файл отчета

При установке опций с помощью POST /xreports/api/cmd в теле запроса может быть передана одна из следующих команд:

Команда	Описание
uri_local	Локальный URI шаблона
uri_remote	Удаленный URI шаблона
uri_embedded	Встроенный шаблон (base64)
out_base64	Установить output-mode = base64
out_binary	Установить output-mode = Бинарный поток
out_url	Установить output-mode = получение результата по ссылке
debug_on	Включить отладку
debug_off	Выключить отладку
pdf_conv_on	Включить конвертацию в PDF
pdf_conv_off	Выключить конвертацию в PDF
cells_automerge_on	Включить автоматическое объединение ячеек
cells_automerge_off	Выключить автоматическое объединение ячеек
tt_multi	Включить мультидок режим для docx, для xlsx включить запрос v2
tt_single	Выключить мультидок режим, для xlsx включить запрос v1
iter_values_on	Включить итерацию тестовых значений
iter_values_off	Выключить итерацию тестовых значений
readonly_on	Включить финализацию PDF-форм
readonly_off	Выключить финализацию PDF-форм

XREPORTS также предоставляет документацию API в формате [OpenAPI 3.0](#).

OpenAPI Spec (YAML) - Машиночитаемая спецификация API в формате YAML. Может быть использована для генерации клиентов, валидации и интеграций.

Адрес: `http://<host>:<port>/xreports/swagger.yaml`

8.1 Архив примеров

В данном разделе можно ознакомиться с примерами запросов и шаблонов для сервиса.

Архив примеров необходимо распаковать в директорию **templates** в корне сервиса. Распакованный архив содержит следующие каталоги:

- **pdf** - содержит PDF - шаблоны страниц, используемых в *генераторе отчетов PDF*
- **examples** - содержит каталоги code, docx, pdf, xlsx

Каталог **code** содержит примеры для генерации QR-кодов.

Каталоги **docx**, **xlsx** содержат набор запросов (в форматах json/xml), документов - шаблонов (.docx и .xlsx) и скриптов (.sh) для получения отчетов.

Каталог **pdf** содержит следующие каталоги:

- **embedded-report-generator** -запросы (.json) и скрипты (.sh) для формирования отчетов в формате .pdf через *Генератор отчетов PDF*.
- **pdf_merge** -запросы (.json) и скрипты (.sh) для объединения PDF-документов.
- **print_form** -запросы (.json) и скрипты (.sh) для *формирования печатной формы*.
- **report-generator** -запросы (.json) и скрипты (.sh) для формирования отчетов через *Генератор отчетов PDF* с примером генератора как часть сервиса. Каталог **pdf_report_gen** с примерами генераторов необходимо разместить в корневом каталоге сервиса.
- **fill-form** -запросы (.json) и скрипты (.sh) для *заполнения PDF-форм*

8.2 Формирование отчета из архива примеров

Рассмотрим генерацию отчета из каталога docx на основе запроса lists.json

```

{
  "template": {
    "uri": "local",
    "id": "examples/docx/lists"
  },
  "input-data": {
    "CONDITIONAL_TAG_TRUE": "true",
    "CONDITIONAL_TAG_FALSE": "false",
    "BULLET_LIST": [
      "bullet item 1",
      "bullet item 2",
      "bullet item 3"
    ],
    "NUMBERED_LIST": [
      "numbered item 1",
      "numbered item 2",
      "numbered item 3"
    ],
    "EMPTY_BULLET_LIST": [
    ],
    "EMPTY_NUMBERED_LIST": [
    ]
  },
  "options": {
    "enable-debug-report-save": false,
    "output-mode": "binary",
    "formatting": {
      "tables": {
        "enable-cells-auto-merge": true
      }
    }
  }
}

```

В результате запуска скрипта `lists.sh`

```

curl --request POST --data-binary "@lists.json" http://localhost:8886/word_report_
↪json --output "lists_report.docx"

```

в каталоге `templates/examples/docx` был записан файл отчета `lists_report.docx`

Примечание

скриптам может понадобиться назначить права на исполнение. Это можно сделать следующей командой:
`chmod +x *.sh`

Мой первый отчет

9.1 Общее описание

В данном подразделе будет рассмотрен процесс создания простого отчета с помощью сервера отчетов. Для работы необходимо удостовериться, что установка сервера отчетов прошла успешно.

9.2 Первый шаблон

Для формирования отчета необходимо создать шаблон документа в формате DOCX или XLSX с помощью MS Word/Excel, LibreOffice Writer/Calc, Google Docs и т.п.

Шаблоны документов могут содержать теги, которые будут заменены входными данными из запроса. Тег задается в квадратных скобках. Пример: [задолженность]. Подробнее про создание шаблонов в формате DOCX и XLSX можно прочитать в разделе Сервис формирования отчетов из документа-шаблона.

Для первого отчета создадим простой шаблон в формате DOCX, который содержит несколько тегов: `first_report_template.docx`. Примеры более сложных шаблонов можно найти в архиве примеров.

Созданный шаблон размещаем на сервере в директории **templates**, расположенной в директории приложения сервиса.

Примечание

Шаблоны страниц в формате PDF хранятся в директории **templates/pdf**, расположенной в директории приложения сервиса.

9.3 Первый запрос

Далее формируем HTTP-запрос.

Запрос для первого отчета в формате JSON будет выглядеть следующим образом: `first_report.json`. Документ-шаблон, созданный ранее, указан в атрибуте **id** элемента **template**. Примеры более сложных запросов можно

найти в архиве примеров.

Примечание. Также для запросов в формате DOCX или XLSX может быть использован формат XML.

Для получения файла отчета используем Сервис формирования отчетов из документа-шаблона.

Для получения первого отчета на основе созданного шаблона DOCX используем URI http://localhost:8886/word_report_json.

9.3.1 Описание

Наименование сервиса	Сервис формирования отчетов из документа-шаблона
Путь к сервису	[host]:[port]/word_report_json
Метод	POST
Параметры	Тело запроса должно содержать JSON объект: <div data-bbox="576 661 1356 1050" style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"> <pre>{ "template": { "uri": "local", "id": "examples/docx/first_report_template" }, "input-data": { "ORGANIZATION": "АО «Пример»", "DATE": "01.01.2023", "EMP": "Иванов Иван Иванович" } }</pre> </div>

Объект **template** имеет атрибуты (поля):

- **uri** - строка. Задаёт расположение файла документа-шаблона в зависимости от того, как трактуется параметр **id**. Поддерживаемое значение: **local**.
- **id** - строка. Идентификатор шаблона. Путь к файлу документа-шаблона относительно директории сервиса **templates**. Также используется для записи файла отчета в отладочном режиме и для идентификации запроса в логге.

продолжается на следующей странице

Таблица 1 – продолжение с предыдущей страницы

input-data - входные данные для подстановки в шаблон.

Объект **input-data** может содержать:

- теги простых строковых данных;
- теги описателей таблиц;
- теги описателей списков;
- теги изображений;
- теги блоков.

options - опции запроса.

Подробнее про структуру JSON можно прочитать в разделе Сервис формирования отчетов из документа-шаблона. В ответ сервис отдаёт файл документа отчета в формате DOCX, закодированный в BASE64. В случае, если необходимо получить результат без кодирования в BASE64, в опциях запроса необходимо указать флаг `enable-binary-output`, равный `true`.

```
{
  «template»: {
    «uri»: «local», «id»: «examples/docx/first_report_template»
  }, «input-data»: {
    «ORGANIZATION»: «АО «Пример»», «DATE»:
    «01.01.2023», «EMP»: «Иванов Иван Иванович»
  }, «options»: {
    «enable-binary-output»: true
  }
}
```

9.3.2 Пример запроса

```
curl -X POST \
'http://localhost:8886/word_report_json' \
-H 'Content-Type: application/json' \
-d '{
  "template":
  {
    "uri": "local",
    "id": "examples/docx/first_report_template"
  },
  "input-data":
  {
    "ORGANIZATION": "АО «Пример»",
    "DATE": "01.01.2023",
    "EMP": "Иванов Иван Иванович"
  }
}'
```

9.3.3 Пример запроса для сохранения результата в файл без кодирования в base64

```
curl -X POST \
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
'http://localhost:8886/word_report_json' \  
-H 'Content-Type: application/json' \  
-d '{  
  "template":  
  {  
    "uri": "local",  
    "id": "examples/docx/first_report_template"  
  },  
  "input-data":  
  {  
    "ORGANIZATION": "АО «Пример»",  
    "DATE": "01.01.2023",  
    "EMP": "Иванов Иван Иванович"  
  },  
  "options":  
  {  
    "output-mode": "binary"  
  }  
}'  
-o first_report.docx
```

9.4 Получение первого ответа

9.4.1 Формат возвращаемого ответа

```
{  
  error: {  
    code:  
    message:  
  }  
  result: "[base64 encoded docx/pdf file]"  
}
```

9.4.2 Пример ответа

```
{  
  "error": null,  
  "result": "UESDBBQACAAIAPdKo...JwAAAAA="  
}
```

9.5 Завершение

В итоге получаем файл документа первого отчета в формате DOCX, закодированный в base64, или сообщение об ошибке.

10.1 Быстрая установка

```
echo "Install XREPORTS"
apt -y install unzip vim wget curl open-vm-tools zip
mkdir /root/xsquare
cd /root/xsquare
wget https://lcdp.xsquare.ru/files/xreports/rpm_dep/6.0.0.3/deb/xsquare.xreports_6.0.
↪0.3.deb
dpkg -i xsquare.xreports_6.0.0.3.deb

##Install Libre Office. We recommend 7.4.7.2 or 24.8.4
echo "Install Libre Office"

apt -y install libxinerama1 libcairo2 libcups2 default-jre
cd /root/xsquare
wget https://lcdp.xsquare.ru/files/libreoffice/LibreOffice_24.8.4_Linux_x86-64_deb.
↪tar.gz
tar -xvzf LibreOffice_24.8.4_Linux_x86-64_deb.tar.gz
dpkg -i ./LibreOffice_24.8.4.2_Linux_x86-64_deb/DEBS/*.deb

#set "soffice-path" in config
soffice_path=`find / -name "soffice"`
echo $soffice_path
old_soffice_path="soffice-path": "";
new_soffice_path="soffice-path": "'$soffice_path'";
echo $new_soffice_path
sed -i -e "s#$old_soffice_path#$new_soffice_path#g" /usr/local/xsquare.xreports/
↪config.json
cat /usr/local/xsquare.xreports/config.json
systemctl restart xsquare.xreports.service
```