Documentation

Release 5.0.0.0

XSQUARE - REPORTS

Mar. 04, 2025

Table of contents

CHAPTER 1

General information

1.1 Introduction to the report server

XSQUARE-REPORTS (**XREPORTS**) is a report server that enables document creation based on template-documents using HTTP requests. The files in DOCX, XLSX, PDF formats are used as template documents. The client requests use JSON and XML formats.

The main functions of the report server are:

- generation of reports in DOCX, XLSX, PDF formats,
- merging PDF documents,
- generation of a printed PDF form with a stamp based on a previously generated PDF document,
- export (conversion) of data from XLSX and XLS (Microsoft Office Excel Binary 2003) to JSON, XML, CSV formats.

1.2 Report server modules

The report server includes the following modules:

- *Service for generating reports from a template document*
- *PDF report generator*
- *PDF merge service*
- *Print form generation service*
- *Export (conversion) service of XLSX and XLS files to JSON, XML, CSV*
- *Code generation service (QR code)*

1.3 Installing the report server

The installation of the report server is described in the *Installation* section.

1.4 Examples of use

The examples of requests and templates can be found in the *Examples* section.

1.5 Creating the first report

The process of creating a simple report using the report server is described in *My first report*.

CHAPTER 2

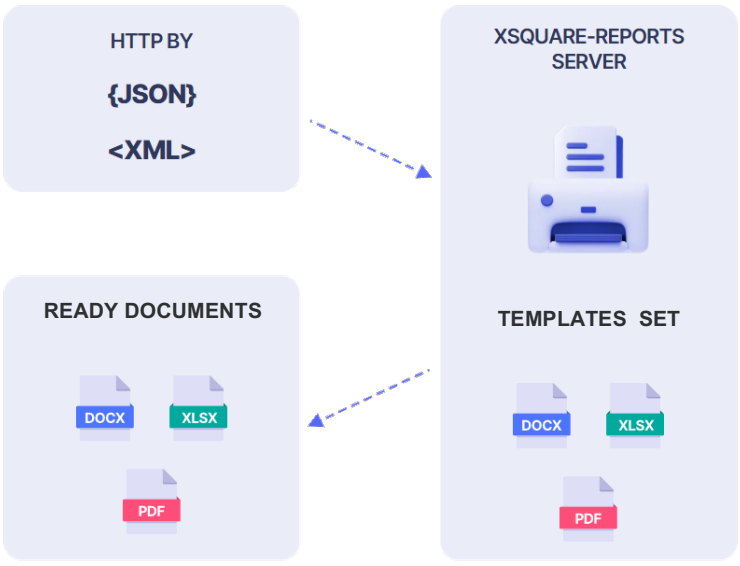
Architecture and system requirements

2.1 Architecture

The application architecture is a web server that processes client HTTP requests by generating reports based on template documents and returns the result to the client.

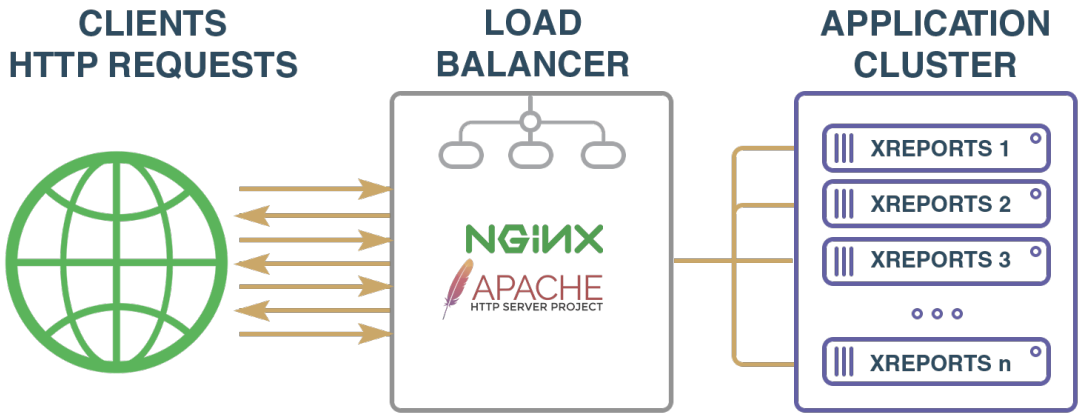
2.2 Basic architecture

The principle or basic architecture of the application is as follows for an industrial environment:



2.3 Architecture for highly loaded systems

The principle or basic architecture of an application for highly loaded systems:



2.4 Performance environment

Supported architecture:

- x86-64
- ARM
- Loongson

Supported OS:

- DEB-based - any
- RPM-based - any
- Debian 12 - recommended

2.5 System requirements

Minimum system requirements:

- CPU - 1 Core
- RAM - 100 MB
- HDD - 100 MB+ Logs

The DOCX/XLSX/PDF generation modules are independent and operate completely autonomously.

Libre Office open source project components are used to convert DOCX/XLSX documents to PDF.

The installation of virtualization/containerization system and the operating system is carried out, if necessary, as per the needs of the Organization and at the discretion of the Administrator.

CHAPTER 3

Installation

3.1 Installing the report server

With Debian as an example, below we review the installation of the report server.

All commands should be run with root privileges.

1. Create a directory for the PGHS distribution

```
mkdir /root/xsquare
```

   Go to the directory

```
cd /root/xsquare
```

2. Download/retrieve the distribution to the created directory

```
wget https://lcdp.xsquare.dev/files/xreports/rpm_dep/5.0.0.3/deb/xsquare.
↪xreports_5.0.0.3.deb
```

3. Install xreports

```
dpkg -i xsquare.xreports_5.0.0.3.deb
```

4. Check the status of the report server

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

3.1.1 Installing fonts for RPM-based OS

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

3.1.2 Installing Libre Office

To convert DOCX/XLSX documents to PDF, one needs to install Libre Office components.

> **Note**
>
> If DOCX/XLSX to PDF conversion is not required, Libre Office installation can be skipped.

1. Download the Libre Office distribution

```
wget https://lcdp.xsquare.dev/files/libreoffice/LibreOffice_24.8.4_Linux_
↪x86-64_deb.tar.gz
```

2. Install 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 Configuration file .json

For the report server to operate, the config.json configuration file must be present in the directory with the service. The configuration file contains 3 sections:

1. The App descriptor where one can define the basic service settings

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

- **port** - string. It specifies the number of the network port on which the service will be started (default is 8886)
- **debug** - Boolean value. It enables debugging mode, with a detailed log of request processing available, and all requests and finished documents are saved in the local directory **reports_debug of** the service (similar to the *enable-debug-report-save* flag in the request properties). The default value is **false**.
- **data-directory** - string. It defines the absolute path to the directory with report server data (assets, templates, reports_debug)
- **save-file-directory** - string. It specifies the absolute path to the directory for saving reports when the report server is running in the file server mode and returns links to finished reports
- **save-file-path-mask** string. It specifies a file mask for saving finished reports. The mask supports templated parameters that can be used when composing the path
  - %DOMAIN% - domain, specified in the request options
  - %YYYYYY% - current year

- %MM% - current month
- %DD% - current day
- %HH% - current hour
- %GUID% - random identifier
- %FILENAME% - file name

2. FormatConversion descriptor where the settings for the soffice application from the Libre Office suite are defined:

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

- **format-conversion-dir** - string. It specifies the directory for storing temporary files (by default - "/tmp")
- **soffice-max-process-count** - number. It defines the number of processor cores that soffice can use for conversion (default is "0", use all available cores).
- **soffice-path** - string. It specifies the path to the soffice executable file (by default - the path written to environment variables when Libre Office is installed)

If after Libre Office installation soffice startup is not available by name - it is necessary to write the full path in the "soffice-path" parameter.

For example:

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

One must restart the report server to apply the new configuration settings.

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

CHAPTER 4

Operation

This section describes how to keep the application operational and the order of loading the components.

4.1 Starting the report server

To download xsquare.xreports, the user needs to make sure they have a properly configured configuration file.

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

The command should display the correct configuration file described in the "Installation" section.

Next, the report server should be started by executing the command:

```
systemctl start xsquare.xreports
```

Afterwards, check the status of the application server:

```
systemctl status xsquare.xreports
```

If errors occur, they will be logged. One can check error messages by executing the command:

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

A status service handler is also available, which can be accessed to retrieve the status, version, and license settings of the report server:

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

CHAPTER 5

Modules

5.1 Service for generating reports from a template document

5.1.1 General description of the service

The service generates a report in OOXML format (.docx and .xlsx documents) from a template document by HTTP request in XML or JSON format. Optionally, the report can be converted to PDF.

The algorithm of the service: tags of the template document are replaced with specific data given in the request.

Main features of the service

1. Generating reports in DOCX format based on a template document. It is possible to generate:
    - simple reports in DOCX format,
    - multi-reports in DOCX format (reports from one template based on multiple input data).
2. Generating reports in XLSX format based on a template document.
3. Converting reports to PDF format.

5.1.2 Working with template documents

To generate a report, one needs to create a template in DOCX or XLSX format.

Location of template files

Template documents in DOCX and XLSX formats are stored locally in the *templates* directory located in the service application directory or in the directory specified by the data-directory option.

Template file format

To create template documents, one should use a word processor (MS Word/Excel, LibreOffice Writer/Calc, Google Docs, etc.).

Important:

- Templates for generating documents in DOCX format must be saved in DOCX format.
- Templates for generating documents in XLSX format must be saved in XLSX or XLSM format.
- Templates for subsequent conversion to PDF should be saved in DOCX format (preferably using Libre Office, as the resulting PDF will be fully consistent with the visual representation of the document in Libre Office).

Using tags in the template

The document templates can contain tags that will be replaced with input data from the query. The tag is specified in square brackets. Example: [debt].

In the body of the request, the input data for the tags is contained in **input-data**. No brackets should be used in the request (see examples from the section "Request Structure", the subsections "Generating a DOCX Report" and "Generating an XLSX Report").

Working with images

The principle of working with images:

- an image is added to the template to be later replaced
- when forming a document, it is replaced by an image file, which is passed in the request. The new content of the image file encoded in BASE64 or data for dynamic image creation (e.g. QR code) is passed in the request

The image tags in the template are set in the image property "Alt text" without square brackets.

Supported formats: JPEG, PNG.

Limitations for images

1. The image format (JPEG, PNG) in the request must match the format of the image being replaced in the template.
2. If it is necessary to replace several images of the template with different images from the input data, the images in the template must be also different. This is due to the fact that the image in the document body refers to its content (file), which means that replacing the content in the document will lead to changing all the images that refer to this content.

Examples of templates

The example templates can be found in the *Examples archive*.

5.1.3 Generating a report in DOCX format

Service description

Service name	Service for generating reports from a template document in **DOCX** format
Service path	Simple reports in DOCX format: • [host]:[port]/word_report_json • [host]:[port] /word_report_xml Multi-reports in DOCX format: • [host]:[port]/word_multi_report_1_N_json • [host]:[port]/word_multi_report_1_N_xml
Method	POST
Parameters	The request body must contain an object in JSON or XML format. Read more about the structure of the request body in the subsection "Request body structure". In response, the service returns a file in DOCX format encoded in base64. When converting a report to PDF format, the service returns a base64-encoded PDF file.
Purpose	The service is designed to generate reports in **DOCX** format from a document-template. Conversion to PDF is possible.

Request body structure

The body of the request contains an object in JSON or XML format that includes:

1. Template descriptor.
2. Input data to be substituted into the template instead of tags.
3. Request options.
4. Response format.

Template descriptor

The element (the object) of template descriptor *template* contains:

- **uri** - string. It specifies the location of the template document file depending on how the **id** parameter is interpreted.

  Supported values:

  - **local** - location of the template in the *templates* directory.
  - **embedded** - the template is included in the request as a base64 encoded string. The value of the string must be given in the *value* parameter.
  - **remote** - the template is located on a remote file server. Reference to the template must be specified in the *value* parameter.
- **id** - string. The template identifier. The path to the template document file relative to the *templates* directory. It is also used to write a report file in debug mode and to identify a request in the log.
- **value** - string. The value is used in embedded and remote modes.
- **type** - string. This parameter is used to specify a non-standard template type. Supported value: xlsm,

Examples for XML format:

```
<template uri="local" id="Information letter"/>
```

```
<template uri="embedded" id="Letter" value="UEsDBBQACAA ... nJlbHOtkt1KA0EMh"/>
```

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

Example for JSON format:

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

```
"template":
{
  "uri": "embedded",
  "id": "Letter",
  "value": "UEsDBBQACAgIAPyFilcAAAAAAAAAAAAAAAA...LAAAAX3JlbHMvLnJlbHOtkt1KA0EMh"
}
```

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

Input data

The element (object) **input-data** contains:

- Simple string data to substitute into the template instead of tags
- Values for conditional expressions
- Table descriptors
- List descriptors
- Data to be substituted into image tags
- Block descriptors

> **Note**
>
> The difference between the data structure for JSON and XML is that instead of XML elements, data is represented as objects and lists of objects. Tag names are specified by the field names of the objects. Lists, tables, images are child objects in relation to the **input-data** object.

> ® **Note**
>
> For multi-reports a list (array**) input-data-array** containing elements (objects) with the same structure **as input-data** is used. For more details on multi-reports, please refer to the subsection 'Working with a multi-report document'.

Simple string data to be substituted into the template instead of tags

For XML format

The element **tags** contains a list of child **tag** elements with the **name** attribute and the desired specific value to which the tag in the template will be replaced when generating the report. The **name** attribute matches the tag in the template

Example:

```
<input-data><tags>
    <tag name="ORGANIZATION">«xxxxyyyy» LLC</tag>
</tags></input-data>
```

For JSON format

Tag names are given by the names of the objects within the **input-data**.

Example:

```
"input-data":
{
  "ORGANIZATION":  "«xxxxyyyy» LLC"
}
```

Description of tables

For XML format

The element of tables descriptor contains a list of child *table* elements with the *name* attribute. The *name* attribute matches the tag in the template. The tag in the template will be replaced by the table created according to the table description in the request.

Example:

```
<input-data>
    <tables>
        <table name="TABLE-FORMATTED">
            <rows>
                <row>
                    <cell tag="order number">1</cell>
                    <cell tag="contract_number">Example number 1</cell>
                    <cell tag="district ">Example district 1</cell>
                    <cell tag="enterprise">Example enterprise 1</cell>
                    <cell tag="date_off">Example date 1</cell>
                    <cell tag="address"> Example address 1</cell>
                    <cell tag="complex_field" Example Sum1</cell>
                </row>
            </rows>
        </table>
        <table name="TABLE-NO-FORMAT">
            <header>
```

(continued on next page)

(continued from previous page)

```
                <cell>Number of item</cell>
                <cell>Case No. in Arbitration Court</cell>,
                <cell>Debt period</cell>,
                <cell>Main debt, USD incl. VAT</cell>,
                <cell>Interest, USD</cell>,
                <cell>State duty, USD</cell>,
                <cell> Court-ordered penalty</cell>,
                <cell>Total repayment amount, USD</cell>,
                <cell>Maturity date, not later</cell>
            </header>
            <rows>
                <row>
                    <cell>1</cell>,
                    <cell>A12-1234/2030</cell>,
                    <cell>October 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>
```

For JSON format

Table descriptors are child objects in relation to the **input-data** object. Field names of table objects are the same as tags in the template.

Example:

```
"input-data":
{
  "TABLE-NO-FORMAT":
  {
    "header":
    [
        “No. of item”,
        “Case No. in Arbitration Court”,
        “Debt period”,
        “Principal debt, USD, including VAT”,
        “Interest, USD”,
        ‘State duty, USD,
        “Court-ordered penalty”,
        “Total repayment amount, USD”,
        “Maturity date, not later”
    ],
    "rows":
    [
      [
```

(continued on next page)

(continued from previous page)

```
        "1",
        "A12-1234/2030",
        "October 2024,
        "12 345,58",
        null,
        "23 456,00",
        "78 912,41",
        null, "31.08.2022"
      ]
    ]
  },
  "TABLE-FORMATTED":
  {
    "rows": [
      {
        "No of item": "1",
        "contract_number": "Example number 1",
        "district": "Example of district 1",
        "enterprise": "Example of enterprise 1",
        "cutoff_date": "Example of date 1",
        “address”: "Example of address 1",
        "complex_field": "Example of amount 1"
      }
    ]
  }


}
```

List Descriptors

For XML format

The **list** descriptor element contains a list of child **list** elements with the *name* attribute. The *name* attribute matches the tag in the template. The tag in the template will be replaced by a paragraph formatted as a list with element data from the list descriptor in the request.

Example:

```
<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>
```

(continued on next page)

(continued from previous page)

```
            </items>
        </list>
    </lists>
</input-data>
```

For JSON format

The list descriptors are arrays within the **input-data** object. The field names of the list objects match the tags in the template.

Example:

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

Data to be substituted into image tags

For XML format

The **images** element contains a list of child **image** elements with the *name* attribute. The *name* attribute matches the tag in the template. The images in the template will be replaced by the image data from the request.

Example:

```
<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>
```

For JSON format

The data to be substituted into the image tags is inside the **images** object, which is a child of the **input-data** object.

Example:

```
"input-data":
{
  "images":
  {
```

(continued on next page)

(continued from previous page)

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

The **data** descriptor can also contain data for code generation (e.g. QR code). Read more in *Working with graphical codes*.

Block Descriptors

For XML format

The **blocks** descriptor element contains a list of **block** child elements with the **block-template** attribute (matches the name of the block template in the template document). A block has the same elements as **input-data** except for image and block elements. The block templates in the template document will be used to insert the needed number of block instances with different data sets into the document. The insertion locations are determined by the tags of the template instance in the template document. Details can be found in the subsection "Working with the document → Blocks".

For JSON format

The list (array) of **blocks** descriptors is inside the **input-data** object.

Example:

```
"input-data":
{
  "blocks":
  [
    {
      "block-template": "block1",
      "data":
      {
        "table1":
        {
          "rows":
          [
            {
              "days": "670",
              "share": "130",
              "penalties": "12564,46",
              "date_from": "11.02.2023",
              "period": "Implementation 0000165 dated 01/31/2022 23:59:59",
              "rate": "7.5",
              "date_until": "12/12/2024",
              "formula": "32505.07 * 670 * * 0.08 1/130",
              "debt": "32505.07"
            }
```

(continued on next page)

(continued from the previous page)


```
          ]
        },
        "amount_penalty": "12345.67",
        "amount_debt": "78910.23",
        "contract_number": "1"
      }
    },
    {
      "block-template": "block2",
      "data":
      {
        "table1":
        {
          "rows":
          [
            {
              "days": "670",
              "penalties": "12564.46",
              "date_from": "11.02.2023",
              "period": "Implementation 0000165 from 31.01.2022 23: 59:59",
              "rate": "7.5",
              "date_until": "12.12.2024",
              "debt": "32505.07"
            }
          ]
        },
        "amount_penalty": "76543.21",
        "amount_debt": "987654.32",
        "contract_number": "2"
      }
    }
  ]
}
```


Request options

The request **options** element (object) contains:

- **output-mode** - string. It determines the form in which the result is presented.

  Supported Values:

  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

  Note: The *enable-binary-output* flag is considered outdated but remains in use to support older reports.

- **report-format** - string. It indicates the format of the report file if it is different from the format mentioned in the request url. The only non-empty supported value: **pdf**. Default value: empty string. See "Convert report to PDF" below.

- **formatting - formatting options object. Supported only for word requests.**
  - Object field: **tables** - object of table formatting options.
  - Object field: **enable-cells-auto-merge** - Boolean value. It specifies whether to merge consecutive cells with the same value in the same column vertically. Default value **- true**.
- **domain-name** - string. It specifies the domain name to form the path for saving the report file when using a file server (providing a report by link).
- **file-name** - string. It specifies the file name to form the path for saving the report file when using a file server (providing a report by link).
- **enable-debug-report-save** - Boolean value. It specifies whether to create a copy of the document report and a copy of the request (file name - template ID) in the local **reports_debug** directory of the service. The default value - **false**.

Example for XML format:

```
<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>
```

Example for JSON format:

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

Response format

The **response-format** element (object) can accept **json** and **xml** values. One can read more about how

the response format is set in the subsection " Response Structure".

Example for XML format:

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

Example for JSON format:

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

Examples of requests

Examples for XML format

Example 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>>
```

Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
   <template uri="local" id="template_example_1"/>
   <input-data>
       <tags>
           <tag name="ORGANIZATION"> «xxxxyyyy» LLC</tag>
           <tag name="TAG_IN_HEADER_TEST">Top footer example</tag>
           <tag name="TAG_IN_FOOTER_TEST">Footer footer example</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="number of items">1</cell>
                       <cell tag="contract_number">Example number 1</cell>
                       <cell tag="district">Example district 1</cell>
                       <cell tag="enterprise">Example enterprise 1</cell>
                       <cell tag="cutoff_date ">Example date 1</cell>
                       <cell tag="address">Example address 1</cell>
                       <cell tag="complex_field" Example >Sum1</cell>
                   </row>
               </rows>
           </table>
           <table name="TABLE-NO-FORMAT">
               <header>
                   <cell>Number of item </cell>
```

(continued on next page)

(continued from previous page)

```
                    <cell Case No. in Arbitration Court </cell>,
                    <cell>Debt period</cell>,
                    <cell>Main debt, USD including VAT</cell>,
                    <cell>Interest, USD</cell>,
                    <cell>State duty, USD</cell>,
                    <cell> Court-ordered penalty</cell>,
                    <cell>Total repayment amount, USD</cell>,
                    <cell>Maturity date, not later</cell>
                </header>
                <rows>
                    <row>
                        <cell>1</cell>,
                        <cell>A12-1234/2030</cell>,
                        <cell>October 2024</cell>,
                        <cell>12  345.58</cell>,
                        <cell/>
                        <cell>23  456.00</cell>,
                        <cell>78  912.41</cell>,
                        <cell/>
                        <cell>31.08.2022</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>
```

Example for JSON format:

```
                {
  "template":
  {
    "uri":  "local",
    "id":  "template_example_1"
  },
  "input-data":
  {
    "ORGANIZATION":  " «xxxxyyyy» LLC",
    "TAG_IN_HEADER_TEST": "An example of a header"",
    "TAG_IN_FOOTER_TEST": "An example of a footer",
```

(continued on next page)

(continued from previous page)


```
    CONDITIONAL_TAG_TRUE": "true",
    "CONDITIONAL_TAG_FALSE": "false",
    "TABLE-NO-FORMAT":
    {

      "header":
      [
         “No. of items”,
         “Case No. in Arbitration Court”,
         “Debt period”,
         “Principal debt, USD, including VAT”,
         “Interest, USD”,
         ‘State duty, USD,
         “Court judgment penalty”,
         “Total repayment amount, USD”,
         “Maturity date, not later”
      ],
      "rows":
      [
        [
          "1",
          "A12-1234/2030",
          "October 2024,
          "12 345,58",
          null,
          "23 456,00",
          "78 912,41",
          null,
          "31.08.2022"
        ]
      ]
    },
    "TABLE-FORMATTED":
    {
      "rows":
      [
        {
          "number_of item": "1",
          "contract_number": "Example number 1",
          "district": "Example of district 1",
          "enterprise": "Example of enterprise 1",
          "cutoff_date": "Example of date 1",
          "address": "Example of address 1",
          "complex_field": "Example of amount 1"
        }
      ]
    },
    "images":
    {
      "IMAGE-1-JPEG":
      {
        "data": "/9j/4AAQSkZJR ... oAKKACigAooAKKACigD//Z"
      },
```


(continued on next page)

(continued from previous page)

```
        "IMAGE-2-JPEG":
        {
          "data": "/9j/4AAQQSkZJR ... CigAooAKKACigAooA//9k="
        }
      }
    },
    "options":
    {
      "enable-debug-report-save": true,
      "formatting":
      {
        "tables":
        {
          "enable-cells-auto-merge": true
        }
      }
    }
}
```

The examples of requests in XML and JSON formats can be also found in the *Examples archive*.

Example of a service call

For XML format

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

For JSON format

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

Response structure

The response format (JSON or XML) can be specified in two ways:

1. in the HTTP header Accept. Supported values: **application/json** and **application/xml** .
2. in the request body **response-format** element (object). Supported values: **json** and **xml**.

Priority is given to the format specified in the request body.

The service response contains an object in JSON or XML format that includes:

1. Error description (error code, error message). In case of successful service response, the value **null** is returned.
2. Result (base64 encoded file of the report document in docx/pdf format or a link to the report document file).

Response format

```
{
  "error":
  {
```

(continued from previous page)

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

Response example

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

Example for the report provided by the link

Part of the request:

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

Response:

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

Working with the document

General information

The following is available for documents in DOCX format:

- tags that will be replaced by the input data from the request,
- tables,
- lists,
- images,
- conditional expressions,

- creating a single report from a single template based on multiple input data (multi-reports),
- blocks.

Working with tables

The following table types are supported in the template:

1. Table without formatting. It is defined by tag only. The table is generated to the width of the page with equal column widths.
2. Table with formatting. It is defined by a table with the following parameters:
    - Optional custom header.
    - The first cell of the row following the header contains a table tag. This row is deleted during generation.
    - The row following the tag line contains column tags, their formatting will be applied to the generated rows. This row is deleted during generation.
    - Optional fixed rows with any tags (replacement data source - main tags).

> ® **Note**
>
> Tables in document footers are supported.

Cell merge management

The basic merge algorithm is automatic merging of cells with the same content vertically.

One can change the merge algorithm using the **formatting.tables.enable-cells-auto-merge** request option.

For JSON-formatted requests, advanced vertical and horizontal merge options are supported. An object describing a table row has an optional **formatting** object. Example:

```
"formatting": {
   "column tag value": {
        "vertical_merge":  "restart",
        "column_span":  2
   }
}
```

The field names of this object are the names of the table columns for which it is necessary to apply the merge option. Field values: merge option object. This object has the following fields:

- **vertical_merge** – row that controls the vertical merge of a cell. Supported values: “restart” - prevents merging by a neighboring cell located in the previous row of the same column; “continue”- force merge with a cell higher in the column; “unset” - the merge option will be inherited from the cell higher in the column.
- **column_span** – integer that controls the horizontal merge of the cell. The value specifies the number of cells to merge to the right of the current cell (inclusive).

> ® **Note**
>
> This is a direct control of merge settings in a document. To quickly understand what values to set for certain cells, one should first configure the desired behavior in the text editor, save the docx document, unzip it as a zip archive and see what values are applied in the .xml file.

Lists

Single-level token-type and numbered lists are supported.

Example for XML format:

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

A detailed example of a request using lists (lists.json) and its template (lists.docx) can be found in the *Examples archive.*

Adding items is possible in any place of the list in the template. In case of an empty list in the input data, the list is deleted from the report.

Conditional expressions

Parts of the template text can be excluded from the report depending on the truth of the condition in the conditional tag. Format of the conditional tag:

```
[#condition]conditional template [/condition]text
```

The opening tag starts with a # character followed by a condition. The closing tag starts with a / followed by the condition of the opening tag.

> ® **Note**
>
> Conditional expressions in footers are not supported.

In the request, the data to be substituted into conditional expression tags is located inside the **input-data** element (object). Example for XM format:

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

Example for JSON format:

```
"input-data":
{
  "ORGANIZATION": "xxxxyyyyyyy LLC",
  "TAG_IN_HEADER_TEST"": "Example header",
  "TAG_IN_FOOTER_TEST"": Example footer",
  "CONDITIONAL_TAG_TRUE": "true",
  "CONDITIONAL_TAG_FALSE": "false"
}
```

A detailed example of a request using conditional expressions (example_from_json.json) and its template (example_from_json.docx) can be found in the *Examples archive*.

Blocks

A block is a report fragment containing text/tables/lists (everything except images in the current version). It is created from a template-block (a certain fragment of a template document) a necessary number of times in a necessary place of the template with different sets of input data.

The block template in a template document is defined by the tags.

[block-template:custom name of the block template]

The block templates are completely removed from the template document before replacing tags with input data.

Block insertion locations are defined by the block template instance tags.

[block-instances:comma separated list of block template names]

of the template document

The block template content is inserted into the document (with tags replaced in the document) at the location of the template instance tag.

Each block template instance tag specifies which templates should be taken as the basis for creating a block in the report (in other words, instantiate the block template). All block inputs are considered in their sequence order. When replacing tags in the block template, all tag values are taken only from the block **data** object (relevant in the current version).

In the request, block data are defined by the **blocks** array, which contains **block** objects with **block-template** fields (coincides with the block template name in the template document) and **data** object field. The **data** object has the same child objects as the **input-data** object, except for images and blocks.

Example:

```
"blocks":
[
  {
    "block-template": "block1",
    "data":
    {
      "table1":
      {
        "rows":
        [
          {}
        ]
      },
      "amount_penalty": "12345,67",
```

(continued on next page)

(continued from previous page)


```
      "amount_debt":  "78910,23",
      "contract_number": "1"
    }
  },
  {
    "block-template": "block2",
    "data":
    {
      "table1":
      {
        "rows":
        [
          {}
        ]
      },
      "amount_penalty":  "76543,21",
      "amount_debt":  "987654,32",
      "contract_number": "2"
    }
  },
  {
    "block-template": "block3",
    "data":
    {
      "contract_number": "1",
      "contract_name": "Contract name 1",
      "date_from": “01/01/2000”,
      "date_until": "01/01/2100"
    }
  },
  {
    "block-template": "block3",
    "data":
    {
      "contract_number": "2",
      "contract_name”: "contract name 2",
      "date_from": “02.02.2020",
      "date_until": "02.02.2080"
    }
  }
]
```


A detailed example of the request using blocks (example_multiblocks.json) and its template (example_multiblocks.docx) can be found in the *Examples archive*.

Working with multi-reports

General information

It is possible to create a single report from a single template based on multiple inputs.

® Note

- The current version does not support numbered lists.
- The current version does not support creating a single report from multiple templates.
- Tags in footers need to be used wisely, as there is no way to have custom footers for each report.
- Image replacement is only supported globally. All sub-reports will have the set of images applied the last time. This is a limitation of the current version of the service.

Examples of requests

Example for XML format

```
<request>
  <template uri="local" id="multi_report_1-N"/>
  <input-data-array>
    <input-data>
      <tags>
        <tag name= "subscriber"> subscriber1</tag>
        <tag name= "subscriber_address"> subscriber_address1</tag>
        <tag  name= "period"> period1</tag>
        <tag  name= "contracts"> contracts1</tag>
        <tag name= "total_amount_debt"> total_amount_debt1</tag>
        <tag name= "amount_indebtedness1"> amount_indebtedness1_1</tag>
        <tag name= "amount_indebtedness2"> amount_indebtedness2_1</tag>
        <tag name= "date_penalty"> date_penny1</tag>
      </tags>
      <tables>
        <table  name="TABLE-FORMATTED">
          <rows>
            <row>
              <cell tag= "item_number"> 1</cell>
              <cell tag= "contract_number"> Example number 1</cell>
              <cell tag= "district"> Example of district 1</cell>
              <cell tag= "enterprise"> Example of enterprise 1</cell>
              <cell tag= "date_off"> Example date 1</cell>
              <cell tag= "address"> Example address 1</cell>
              <cell tag= "complex_field"> Example amount 1</cell>
            </row>
          </rows>
        </table>
      </tables>
    </input-data>
    <input-data>
      <tags>
        <tag name= "subscriber"> subscriber2</tag>
        <tag name= "subscriber_address"> subscriber_address2</tag>
        <tag  name= "period"> period2</tag>
        <tag  name= "contracts"> contracts2</tag>
        <tag name= "total_amount_debt"> total_amount_debt2</tag>
        <tag name= "amount_indebtedness1"> amount_indebtedness1_2</tag>
```

(continued on next page)

(continued from previous page)

```
        <tag name= "amount_indebtedness2"> amount_indebtedness2_2</tag>
      </tags>
      <tables>
        <table name= "TABLE-FORMATTED">
          <rows>
            <row>
              <cell tag= "item_number"> 1</cell>
              <cell tag= "contract_number"> Example number 3</cell>
              <cell tag= "district"> Example of district 3</cell>
              <cell tag= "enterprise"> Example of enterprise 3</cell>
              <cell tag= "date_off"> Example date 3</cell>
              <cell tag= "address"> Example address 3</cell>
              <cell tag= "complex_field"> Example amount 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>
```

Example for JSON format

```
{
  "template":
  {
    "uri": "local",
    "id": "template_example_1"
  },
  "input-data-array": [
    {
      "ORGANIZATION": "xxxxyyyyyy1 LLC",
      "CLINAME": " MUNICIPAL ENTERPRISE OF THE CITY            \XXXXXXXXXXXXX\",
      "CTRNUMBER": "1234",
      "TABLE-NO-FORMAT":
      {
        "header":
        [
          "Item number1",
          "Case No. in Arbitration Court1",
          "Debt period"1,
          "Principal debt, USD including VAT1",
          "Interest, USD1",
```

(continued on next page)

(continued from previous page)


```
        "State duty, USD1",
        "Court-ordered penalty1",
        "Total repayment amount, USD1",
        "Maturity, not later1"
      ],

      "rows":
      [
        [
          "1_1",
          "A12-1234/2030_1",
          "October 2024_1",
          "12,345.58_1",
          null,
          "23 456,00_1",
          "78 912,41_1",
          null,
          "31.08.2022_1"
        ]
      ]
    },
    "TABLE-FORMATTED":
    {
      "rows":
      [
        {
          "item number": "1_1",
          "contract_number": "Example of number 1_1",
          "district": "Example of district 1_1",
          "enterprise": "Example of enterprise  1_1",
          "cutoff_date": "Example of date 1_1",
          "address": "Example of address 1_1",
          "complex_field": "Example of amount 1_1"
        }
      ]
    },
    "images":
    {
      "IMAGE-1-JPEG":
      {
        "data":  "/9j/4AAQSkZJRgABAQEAYABgAAD...AKKACigD//Z"
      },
      "IMAGE-2-JPEG":
      {
        "data":  "/9j/4AAQSkZJRgABAQEAYABg...oAKKACigAooA//9k="
      }
    }
  },
  {
    "ORGANIZATION": "xxxxyyyyyy_2 LLC",
    "CLINAME": "CITY ENTERPRISE 'PASSENGER AUTOMOBILE TRANSPORT_2'",
    "CTRNUMBER": "1234_2",
    "RESTPENY": null,
```


(continued on next page)

(continued from previous page)


```
    "TAG_IN_HEADER_TEST": "Example header_2",
    "TAG_IN_FOOTER_TEST": "Example footer_2",
    "CONDITIONAL_TAG_TRUE": "true",
    "CONDITIONAL_TAG_FALSE":  "false",
    "TABLE-NO-FORMAT":
    {

      "header":
      [
        "Item number_2",
        "Case No. in Arbitration Court_2",
        "Debt period_2",
        "Principal debt, USD incl. VAT_2",
        "Interest, USD_2",
        "State duty, USD_2",
        "Court-ordered penalty_2",
        "Total repayment amount, USD_2",
        "Maturity, not later than_2"
      ],
      "rows":
      [
        [
          "1_2",
          "A12-1234/2030_2",
          "October 2017_2",
          "12  345,58_2",
          null,
          "23  456,00_2",
          "78  912,41_2",
          null,
          "31.08.2019_2"
        ]
      ]
    },
    "TABLE-FORMATTED":
    {
      "rows":
      [
        {
          "item number": "1_2",
          "contract_number": "Example number 1_2",
          "district":  "Example of district 1_2",
          "Enterprise": "Example of enterprise 1_2",
          "cutoff_date":  "Example of date 1_2",
          "address": "Example of address 1_2",
          "complex_field": "Example of amount 1_2"
        }
      ]
    }
  }
],
"options":
{
```


(continued on next page)

(continued from previous page)

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

A detailed example of a request to generate a multi report (multi_report_1-N_json.json) and its template (multi_report_1-N_json.docx) can be found in the *Examples archive*.

Examples of service call

For XML format

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

For JSON format

```
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 Report generation in XLSX format

Service Description

Name of service	Service for generating multi-reports from a template document in **XLSX** format
Path to service	• [host]:[port]/excel_report_json • [host]:[port]/excel_report_xml • [host]:[port]/excel_report_json/v2
Method	POST
Parameters	The request body must contain a JSON or XML object. Read more about the structure of the request body in the subsection "Request Body Structure". In response the service gives a document file in XLSX format encoded in BASE64. When converting a report to PDF format, the service returns a PDF document file encoded in BASE64.
Assignment	The service is designed to generate multi-reports from a template document in XLSX format. Conversion to PDF is possible.

Request body structure

The body of the request contains an object in JSON or XML format that includes:

1. Template descriptor.
2. Input data to be substituted into the template instead of tags.
3. Request options.

4. Response format.

Template descriptor

The element (object) of the **template** descriptor contains:

- **uri** – string that specifies the location of the template document file depending on how the id parameter is interpreted.
  Supported Values:
  - **local** - template location in the *templates* directory
  - **embedded** - the template is included in the request as a base64 encoded string. The value of the string must be written to the value parameter.
  - **remote** - the template is located on a remote file server. Reference to the template must be specified in the value parameter.
  - **inner** - inner template, used to generate a table with a header.
- **id** - string. The template identifier. The path to the template document file relative to the **templates** directory. It is also used to write a report file in debug mode and to identify a request in the log.
- **value** - string. The value is used in embedded and remote modes.
- **type** - string. This parameter is used to specify a non-standard template type. The supported value: xlsm.

Example for XML format:

```
<template uri= "local" id= "Information letter"/>
```

Example for JSON format:

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

Input data

The element (object) **input-data** contains:

- Data to substitute into the template instead of tags: strings, numbers, formulas, links.
- Table descriptors.
- Block descriptors.
- Data to be substituted into image tags.

> **Note**
>
> The difference between the data structure for JSON and XML is that instead of XML elements, data is represented as objects and lists of objects. Tag names are specified by object field names. Tables and images are child objects in relation to the **input-data** object.

Data to be substituted in the template instead of tags

Strings

For XML format

The **tags** element contains a list of child **tag** elements with the **name** attribute and the desired specific value to which the tag in the template will be replaced when generating the report. The **name** attribute is the same as the tag in the template.

Example:

```
<input-data>
    <tags>
        <tag name="ORGANIZATION">«xxxxyyyyyy» LLC</tag>
    </tags>
</input-data>
```

For JSON format

Tag names are given by the names of the objects within the **input-data.**

Example:

```
"input-data":
{
    "ORGANIZATION": "«xxxxyyyyyy» LLC"
}
```

Numbers

For XML format:

The element **tag** also supports the **type** attribute. If the type attribute is **num**, the tag value will be interpreted as a floating point number. In this case a numeric value will be substituted into the template, this allows, for example, to apply formulas.

Example:

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

For JSON format:

To interpret the tag value as a number, one must specify the value without quotation marks, with a period as the separator of the integer and fractional parts.

Example:

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

Formulas

Instead of a tag, a formula can be inserted into the template to calculate and display values.

The formula in the request is specified by an object with fields:

- **type** - string, supported value: "formula".

- **value** - string, values are taken from the table editor used, without equal sign.

For the XML format, type is the attribute of the tag element and value is its value:

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

For JSON format:

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

No row number conversions are performed. It is assumed that the row numbers are relevant for the final report, i.e. after inserting the table rows that come before the row containing the formula. The row numbers can be calculated based on the table row numbers in the template document and the number of table rows at the time of generating the row with the formula in the query.

Example request: **formula.json** It can also be found in the *Examples archive*.

References

Instead of a tag, a hyperlink can be inserted into the template.

The reference in the request is specified by an object with fields:

- **type** - string, supported value: "link".
- **text** - string, the name of the link displayed in the document.
- **value** - string, the value of the link to jump to.

For the XML format, type and text are attributes of the tag element, and value is its value:

```
<tag name="http://link1" type="link" text="company «XSQUARE»">
    "http://xsquare.dev"
</tag>
```

For JSON format:

```
"http://link1":
{
  "type": "link",
  "text": "«XSQUARE» company",
  "value": "http://xsquare.dev"
}
```

> ® **Note**
>
> in the document template, tags for links should be in hyperlink format. To do this, start the tag name with http:// (or https://, ftp://)

Example request: **hyperlinks.json.** It can be also found in the *Example archive*.

Table descriptors

For XLSX reports, it is also possible to generate tables based on data from the request.

Difference from the DOCX format

- tags are set in the **cell-tags** element (object) once for the whole table,
- cell tag maps the column tag to the cell index in the array specifying the table row.

For XML format

The **tables** descriptor element contains a list of child **table** elements with the **name** attribute. The **name** attribute matches the tag in the template. The tag in the template will be replaced by the table created according to the table description in the request.

Example:

```
<input-data>
    <tables>
        <table  name= "table_insured_individuals">
            <cell-tags>
                <cell-tag name="item_number" index="0"/>
                <cell-tag name="social_security_number" index="1"/>
                <cell-tag name="gender" index="2"/>
                <cell-tag name="full_name" index="3"/>
                <cell-tag name="date_birth" index="4"/>
                <cell-tag name="date_contract" index="5"/>
                <cell-tag name="contract_number" index="6"/>
            </cell-tags>
            <rows>
                <row>
                    <cell>1</cell>
                    <cell>001-02-0034</cell>
                    <cell>F</cell>
                    <cell>Catherine Smith</cell>
                    <cell>01/01/1970</cell>
                    <cell>01/01/1988</cell>
                    <cell>ABC-001-002-003-00</cell>
                </row>
            </rows>
        </table>
    </tables>
</input-data>
```

For JSON format

Table descriptors are child objects in relation to the **input-data** object.

Example:

```
"input-data":
{
  "table_insured_individuals":
  {
    "cell-tags":
```

(continued on next page)

(continued from previous page)

```
    {
      "item number": 0,
      "social_security_number": 1,
      "gender": 2,
      "full name": 3,
      "date_birth": 4,
      "date_contract": 5,
      "contract_number":  6
    },
    "rows":
    [
      [
        "1",
        "001-02-0034",
        "F",
        "Catherine Smith",
        "01/01/1970",
        "01/01/1988",
        "ABC-001-002-003-00".
      ]
    ]
  }
}
```

Cell values in a table can also be formulas and links:

Example JSON:

```
        "input-data":
{
  "table_1":
  {
    "rows":
    [
      [
        "1",
        "Heat Supply Company",
        "Enterprise 1",
        161975.87,
        null,
        161975.87,
        12985.51,
        "15/08/2023",
        {
          "type": "link",
          "text": "example link in table 1",
          "value": "http://xsquare.dev"
        }
      ],
      [
        null,
        null,
        "TOTAL FEDERAL BUDGET",
```

(continued on next page)

(continued from previous page)

```
        {
          "type":  "formula",
          "format":  "raw",
          "value": "SUM(D11:D11)"
        },
        null,
        161975.87,
        12985.51,
        null,
        null
      ]
    ],
    "cell-tags":
    {
      "title": 1,
      "item number": 0,
      "link": 8,
      "due_time": 7,
      "indebtedness_total": 3,
      "indebtedness_current": 4,
      "organization_name":  2,
      "outstanding_indebtedness": 5,
      "balance_of_indebtedness_from_outstanding": 6
    }
  }
}
```

XML :Example

```
<tables>
   <table name= "table_insured_individuals">
     <cell-tags>
       <cell-tag name="item number" index="0" />
       <cell-tag name="individual insurance account number" index="1" />
       <cell-tag name="http://link" index="2" />
     </cell-tags>
     <rows>
       <row>
         <cell>1</cell>
         <cell>001-02-0034</cell>
         <cell type="link" text="link text1">http://table1.com</cell>
       </row>
       <row>
         <cell>2</cell>
         <cell> 001-02-0034</cell>
         <cell type="link" text="link text2">http://table2.com</cell>
       </row>
     </rows>
  </table>
</tables>
```

An example request: **example_with_links.xml.** It can be also found in the *Example archive*.

Block descriptors

For XLSX reports, it is possible to generate data blocks in a document based on a template block and data from the request.

A block template is a sequence of rows in the document between cells with the following tags **[block_name] - [/block_name].**

A block-template can be repeated many times consecutively in the document. One can also optionally add a page break after each block, which is useful when printing documents.

The **blocks** descriptor element contains a list of child **block** elements with the **template** attribute (matches the name of the block template in the template document).

The block has the same elements as **input-data**, except for image and block elements. The data for the block is located in the **data** descriptor, and for each block you can also specify a page break at the end of the block.

The page break is set by specifying **true** in the **page-break** descriptor.

The block templates in a template document will be used to insert any number of instances of blocks with different data sets. The blocks are inserted sequentially one after another.

Example XML:

```
<blocks>
    <block template="block_1">
        <data>
            <tags>
                <tag name="date">01/02/2024</tag>
            </tags>
        </data>
    </block>
    <block template="block_1">
        <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>
```

For requests in JSON format, the name of the block template is specified in the **block-template** descriptor.

Example JSON:

```
    "blocks":
[
  {
    "block-template": "block_1",
    "no-page-break": true,
```

(continued on next page)

(continued from previous page)


```
    "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"
           ]
         ]
       }
     }
  },
  {
    "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",
            "11/22/2022"
          ]
        ]
```


(continued on next page)

(continued from previous page)

```
      }
    }
  }
]
```

An example request: **blocks.json**. It can be also found in the *Examples archive*.

> **Note**
>
> Rows with empty cells are not allowed in the block-template. To make cells not empty, it is enough to fill cells of the block with any color and then cancel the filling and save the template file.

Data to be substituted into image tags

For XML format

The **images** element contains a list of child **image** elements with the **name** attribute. The **name** attribute matches the tag in the template. The images in the template will be replaced with the image data from the query. The new BASE64 encoded image file content is passed in the request.

Example:

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

For JSON format

The data to be substituted into the image tags are inside the child objects of the **input-data** object.

Example:

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

The **data** descriptor can also contain data for code generation (e.g. QR code). Read more in *Working with graphical codes*.

Deleting images

To remove an image from the template, one must pass **null** instead of the new content. Example:

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

Request options

The request **options** element (object) contains:

- **output-mode – string that defines the form of output delivery.**
  Supported values:
  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

  Note: The *enable-binary-output* flag is considered outdated but is still in use to support older reports.

- **report-format** – string that indicates the format of the report file if it is different from the format mentioned in the request url. The only non-empty supported value: "**pdf**". Default value: empty string. See "Convert report to PDF" below.
- **clean-data-by-null** - Boolean value that defines the way of processing tags with null values for xlsx, xlsm templates. If the option value is *true*, the cell corresponding to the tag will be completely cleared of any content. If the value of the option is *false*, the string tags will be filled with empty rows, and for other types - cleared. The default value is *false*.
- **domain-name** – string that specifies the domain name to form the path for saving the report file when using a file server (providing the report by link).
- **file-name** – string that specifies the file name to form the path for saving the report file when using a file server (providing the report by link).
- **enable-debug-report-save** - Boolean value that specifies whether to create a copy of the document report and request copy (file name - template identifier) in the local **reports_debug** directory of the service. The default value is **false**.

> ® **Note**
>
> Set print areas (“Set Print Area”) in the document template, according to them the conversion to PDF will be performed.

Example for XML format:

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

Example for JSON format:

"options":

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

Response format

The response-format element (object) can accept **json** and **xml** values.

One can read more on how the response format is set in the subsection "Response structure".

Example for XML format:

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

Example for JSON format:

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

Examples of requests

Example for XML format

```
<request>
  <response-format value="xml"/>
  <template uri="local" id="template_example"/>
  <input-data>
    <tags>
      <tag name="organization">«Success» LLC </tag>
      <tag name="day">25</tag>
      <tag name="month">December</tag>
      <tag name="year">2018</tag>
      <tag name="document_number">01</tag>
      <tag name="number_of_contracts">3</tag>
      <tag name="employee">Jane Doe</tag>
      <tag name="tel_employee">+1(123)456-7890</tag>
```

(continued on next page)

(continued from previous page)

```
      <tag   name="tag1">tag1_value</tag>
      <tag   name="tag2">tag2_value</tag>
    </tags>
    <tables>
      <table   name="table_insured_individuals">
        <cell-tags>
          <cell-tag name="item_number" index="0"/>
          <cell-tag name="social_security_number" index="1"/>
          <cell-tag name="gender" index="2"/>
          <cell-tag name="full_name" index="3"/>
          <cell-tag name="date_birth" index="4"/>
          <cell-tag name="date_contract" index="5"/>
          <cell-tag name="contract_number" index="6"/>
        </cell-tags>
        <rows>
          <row>
            <cell>1</cell>
            <cell>001-02-0034</cell>
            <cell>F</cell>
            <cell>Catherine Smith</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>
    </images>
  </input-data>
  <options>
    <enable-debug-report-save>true</enable-debug-report-save>
  </options>
</request>
```

Example for JSON format

```
{ "response-format": "json",
  "template": {
     "uri": "local",
     "id": "template_example"
  },
  "input-data": {
     organization": "«Success» LLC",
     "day": "25",
     "month": "December",
     "year": "2018",
     "document_number": "01",
     "number_contracts": "3",
```

(continued on next page)

(continued from previous page)

```
    "employee": “Jane Doe”,
    "tel_employee": “+1(123)456-7890",
    “tag1": “tag1_value",
    "tag2": tag2_value",
    “table_insured_individuals": {
      "cell-tags": {
        "item_number": 0,
        "social_security_number": 1,
        "gender": 2,
        "full_name": 3,
        "date_birth": 4,
        "date_contract": 5,
        "contract_number":  6
      },
      "rows": [ [
          "1",
          "001-02-0034",
          "F",
          "Catherine Smith",
          "01/01/1970",
          "01/01/1988",
          "ABC-001-002-003-00"
        ]
      ]
    },
    "images":{
      "IMAGE-1-JPEG": {
        "data":  "/9j/4AAQSkZJRgABAQEAYABgAAD/4QFQR0  ..."
      },
      "IMAGE-2-JPEG": {
        "data":  "/9j/4AAQSkZJRgABAQEAYABgAAD/4QFQR  ..."
      }
    }
  },
  "options":{
    "enable-debug-report-save": true
  }
}
```

Example of a service call

For XML format

```
curl --request POST --data-binary "@templates/examples/xlsx/base_example_xml.xml"␣
↪http://localhost:8886/excel_report_xml
```

For JSON format

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

Response structure

The response format (JSON or XML) can be specified in two ways:

1. In HTTP header Accept. Supported values: **application/json** and **application/xml**.
2. In the body of the request in the **response-format** element (object). Supported values: json and xml.

The format specified in the request body has priority.

The service response contains an object in JSON or XML format that includes:

1. Error description (error code, error message). In case of successful service response, the value **null** is returned.
2. Result (BASE64 encoded report document file in xlsx format).

Response format

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

Example response

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

An example for the report provided by the link:

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

Response:

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

Error codes

The field **code** can accept the following values:

"**1**" - an error in the request,

"**2**" - report generation error.

Working with the document

General information

For documents in XLSX format, the following are available:

- tags that will be replaced by the input data from the query,
- tables,
- images.

> **Note**
>
> - The input cell data format is text and numbers (integer or real).
> - Cell references in formulas are not corrected during report generation. Formulas may become incorrect if rows are added/deleted to the report.
> - Conditional expressions are not supported in the current version.

Working with tables

There are two options of working with tables.

v1 - API: /excel_report_json

A formatted table is defined as follows:

- the first cell of the row contains a tag with the table name. This row is deleted during generation.
- “tag string”, the row(s) following the table tag string contains the column tags, their formatting will be applied to the generated rows (if no formatting string is specified, see below). This row(s) is removed during generation.
- optional: a string indicating that the next row contains a formatting string. It contains a single cell with the text “[cell format]”. Used for tables where numeric data should be in cells with number (or general) format, not rows (otherwise it is easier to specify formatting in the tag line).
- optional: “format string”, a string that specifies the format of cells instead of the tag string. Cell values are ignored. The format is applied to the generated strings. It must be a copy of the tag string in terms of the number of cells and their order.

The ability to map a single row of input data to multiple rows in a document is supported. This may be needed for tables where a single logical row has cells with vertical merging. The number of template rows per input data row is defined as the maximum number of vertically merged rows in any cell from the first row after the tagged row of the table itself (see the function **getNumberOfSheetRowsPerInputDataRow**).

An example template can be found in the *Example archive*.

> ® **Note**
>
> currently, applying formatting to generated rows does not include font settings. Background color and cell border settings (e.g. horizontal merging of cells) are supported.

v2 - API: /excel_report_json/v2

A formatted table is defined as follows:

- the first cell of the row contains a tag with the table name. This row is deleted during generation.
- "tag string", the row following the tag string contains column tags, their formatting will be applied to the generated rows (if no formatting row is specified, see below). This row is deleted during generation.
- optional: a string indicating that the next row contains a formatting string. It contains a single cell with the text "[cell format]". It is used for tables where numeric data should be in cells with number (or general) format, and not rows (otherwise it is easier to set the formatting in the tag string).
- optional: "format string", a string that specifies the format of cells instead of the tag string. Cell values are ignored. The format is applied to the generated rows. It must be a complete copy of the tag string in terms of the number of cells and their order.

The example templates can be found in the *Example archive*.

- **cell_format_options.xlsx**
- **cell_format_options_typed_cells.xlsx**

> ® **Note**
>
> currently, applying formatting to generated rows does not include font settings. Background color and cell border settings (e.g. horizontal merging of cells) are supported.

Cell merge management

The basic merging algorithm is to apply horizontal merging of generated cells based on merging of template table cells.

Supports advanced vertical merge options. Defined by an object with a single field:

- **vertical_span** is an integer. It controls the vertical merging of a cell. The value specifies the number of cells to merge down from the current cell (inclusive).

Options can be mapped to table columns in the template, specified by index

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

or to cell tags (this method has priority, which is convenient for selective correction of merge options for some of the columns):

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

Adding rows to the end of a template document

The **rows** specified by the **rows-to-add** list will be added to the end of the template. The added rows can contain different number of cells and different column data, that is, each added cell is independent of the others. See the example request **add_rows_to_end.json** in the *Examples archive*.

When using the **rows-to-add** mode, the cell values can be:

- **null**
- **lines**
- **numbers**
- **formulas**
- **hyperlinks**

Example:

```
"input-data": {
  "rows-to-add": [
    [
      "1",
      "001-02-0034",
      "F",
      "Catherine Smith",
      "01/01/1970",
      null,
      "ABC-001-002-003-00",
      30
    ],
    [
      2,
      "001-02-0034",
      "M",
      "John Doe",
      "02/03/1970",
      "04/07/1988",
      "ABC-001-002-003-00",
      40,
      {
        "type": "link",
        "text": "example link 1",
        "value": http://xsquare.dev"
      }
    ],
    [
```

(continued on next page)

(continued from previous page)

```
        "3",
        "001-02-0034",
        null,
        "Peter Parker",
        "04/05/1970",
        "06/07/1988",
        "ABC-004-005-006-00",
        50,
        {
          "type": "link",
          "text": "example link 2",
          "value": http://xsquare.dev"
        }
      ],
      [
      ],
      [
        null,
        null,
        null,
        null,
        null,
        null,
        "Amount of contributions:",
        {
          "type": "formula",
          "value": "SUM(H2:H4)"
        }
      ]
  ]
}
```

See the example request **add_rows_to_end_complex.json** in the *Example archive*.

If it is necessary to specify style, formatting and cell type of added rows, you can use the last row in the document as a template. To do this, specify the **use-last-row-style** flag in the request options. In this mode, the last row will serve as a source of style, formatting and type for each cell when adding new rows using the **rows-add** method. The last row-template will be removed from the resulting document. In the request, the type of data to be substituted must match the cell type in the last row.

> **® Note**
>
> Dates and times are passed as numbers, not strings.

Example:

```
{
  "template":
  {
    "uri": "local",
    "id": "examples/xlsx/add_rows_to_end_with_style"
```

(continued on next page)

(continued from previous page)

```
},
"options":
{

  "use-last-row-style":  true
},
"input-data":
{
  "rows-to-add":
  [
    [
      12,
      "001-02-0034",
      "F",
      "Catherine Smith",
      34021,
      null,
      "no link."
    ],
    [
      2.56,
      "001-02-0034",
      "M",
      "John Doe ", 34022,
      0.6458,
      {
        "type": "link",
        "text": "example link 1",
        "value": “http://xsquare.dev"
      }
    ],
    [
      3.4555555,
      "001-02-0034",
      null,
      "Peter Parker",
      34025,
      0.3,
      {
        "type": "link",
        "text": "example link 2",
        "value":
      } http://xsquare.dev"
    ],
    [],
    [

      4,
      null,
      "M",
      "Anthony Kohl",
      null,
      null,
```

(continued on next page)

(continued from previous page)

```
        null
      ]
    ]
  }
}
```

Generating a table without a template

The report server enables generating a document as a table without a template. To do this, **inner** is used as the **uri** value of the template descriptor. The request must contain a global **table** descriptor, which contains a **header** descriptor with column names and a **rows** descriptor with an array of rows.

```
{
  "template": {
    "uri": "inner",
      "id": "table_example"
  },
  "input-data": {
    "table": {
      "header": [
        "Column 1",
        "Column 2",
        "Column 3",
        "Column 4",
        "Column 5",
        "Column 6",
        "Column 7"
      ],
      "rows": [
        [
          2,
          "001-02-0034",
          "M",
          "Michael Green",
          "01.01.1970",
          "02.03.1989",
          "ABC-001-002-005-00"
        ]
      ]
    }
  }
}
```

You can use the **column_width** descriptor to specify the **column** width, by default the column widths will be set up automatically based on the content.

See the example request **table_custom_column_column_width.json** in the *Example archive*.

Using XLSM format as a template

The XLSM format - spreadsheet files with macro support - can be used as templates. To use this type of template it is necessary to specify the "**type**" parameter in the **template** descriptor:
"xlsm".

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

See the **example_macros.json** request in the *Example archive*.

5.1.5 Convert report to PDF

Activation

Specify the report format "pdf" in the request options.

Example:

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

Dependencies

The Libre Office package must be installed. The **soffice** program must be available for invocation. It is possible to specify the path to it in the command line parameters of the report server or via a configuration file.

Principle of operation

The report server saves the report to a file and calls the **soffice** utility with the following parameters, for example

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

then sends the resulting PDF file instead of the original report.

By default

The path to the **soffice** utility is searched in the directories specified by the **PATH** environment variable. One can specify a direct path in the reporting server command line arguments or through a configuration file.

File exchange with **soffice** goes through the temporary folder of the system. One can specify the working directory in the command line arguments of the report server or via a configuration file.

See the output of the command

```
xreports --help
```

Performance

By default, the command execution time

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

for a blank document - 2.3 seconds (for cpu i7-3615M, ssd). Most of the time is spent on initialization of Libre Office.

To speed up the conversion, one can use the **quick start** mode, i.e. pre-execute command.

```
soffice --quickstart
```

In this mode, Libre Office is in RAM the whole time, which saves about 2 seconds.

> **Note**
>
> for Libre Office 6.3.4.2 in Ubuntu 19.10 **soffice -quickstart** causes the main window to show. The parameters -**minimized** and -**headless** do not solve this problem, as they do not allow to use acceleration after a single conversion. In Windows 10, LO is only visible in the tray when started this way.

5.1.6 Barcode operation

It is possible to output a line of text as a barcode. To operate correctly in Word and Excel reports, Libre Barcode family fonts must be installed in the operating system. Available in the librebarcode repository.

Each font represents a specific barcode format. The following formats are supported:

- Code-128.
- Code-39
- EAN13

The font can include the original text directly below the barcode (fonts with Text suffix). Find below the example of using the Code-128 format.

Code-128

Fonts to install:

- **LibreBarcode128Text-Regular.ttf**, will be available as "Libre Barcode 128 Text"
- **LibreBarcode128-Regular.ttf**, will be available as "Libre Barcode 128"

How to use

In the template for a regular tag, select a font. For example, "Libre Barcode 128 Text".

Test requests

Document in DOCX format

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

Document in XLSX format

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

5.1.7 Working with graphic codes (QR code)

To substitute dynamically created graphic codes in place of the image, it is necessary to pass the code descriptor to the parameter **data**.
Example:

```
{
  "template":
  {
    "uri": "local",
    "id": "examples/docx/qr-code"
  },
  "input-data":
  {
    "images":
    {
      "qr_link":
      {
        "data":
        {
          "type": "qr-code",
          "content": "https://xsquare.dev",
          "qr-color": "#ffa747",
          "bg-color": "#2b0054",
          "dot-size": 40,
          "style": "dot",
          "format": "png"
        }
      }
    }
  },
  "options":
  {
    "enable-debug-report-save": false,
    "output-mode": "binary"
  }
}
```

Parameters for generating graphic codes are similar to parameters in the module of code generation, detailed description of parameters can be found in the section *Code generation service (QR code)* .

5.2 PDF report generator

5.2.1 General description of the service

The service generates a report in PDF format by HTTP-request in JSON format.

The PDF document is generated using special generator commands.

5.2.2 Document generation using PDF generator

Service description

Name of service	PDF report generator
Path to service	[host]:[port]/pdf_report_json
Method	POST
Parameters	The request body must contain an object in JSON format. One can read more about the structure of the request body in the subsection “Request body structure”. In response, the service gives a base64-encoded PDF document file.
Purpose	The service is designed to generate documents in PDF format.

Request body structure

The body of the request contains an object in JSON format that includes:

1. PDF generator descriptor.
2. Input data to be substituted into the template instead of tags.
3. Optional: PDF generator.
4. Request options.

Example:

```
{
  "report": "report-generator",
  "uri":{
    "uri": "local",
    "id": "example_1"
  },
  "input-data": {
    "ORGANIZATION": "«Success» LLC",
    "CLINAME: "Peter Parker"
  },
  "options": {
    "enable-debug-report-save": true
  }
}
```

PDF generator descriptor

The PDF **report-generator** descriptor object contains:

- **uri** – string that specifies the position of the generator command source depending on how the **id** parameter is interpreted. Supported values: **local** and **embedded**.
- **id** - string. The ID of the generator.

There are two possible ways to use the PDF generator:

- generator as part of the request (all generator commands are placed inside the **embedded-report-generator** object of the request body)
- generator as a part of the service (all generator commands are placed inside JSON file in special **pdf_report_gen** folder on the server).

Accordingly:

- When is **uri** set to **embedded**, **id is** only used for diagnostic messages and request identification. The generator must be in the root **embedded-report-generator** object of the request. See "Generators as part of the request" below.
- When **uri** is set to **local**, **id** is the name of the file in the **pdf_report_gen** service directory. See "Generators as part of the service" below.

Generator as part of the request

Optionally, the generator can be integrated into the request.

In this case, all generator commands are specified in the **embedded-report-generator** object.

Example:

```
{
  "report-generator": {
       "id":"example",
      "uri":"embedded"
  },
  "embedded-report-generator":  {
    "commands":  [
      {
        "name":  "NewPage"
      },
      {
        "name":  "setCoordinateMode",
        "params": {
          "mode":  "millimeters"
        }
      },
      {
        "name":  "setCellMargin",
        "params": {
          "margin":  1
        }
      }
    ]
  },
  "options": {
    "enable-debug-report-save":  false
  }
}
```

Generator as part of the service

Goal: not to pass command data within requests unless it changes from one request to another.

In this case, all generator commands are specified in a JSON format file, which must be located in the **pdf_report_gen** directory located in the service application directory.

Generator format inside the file in the **pdf_report_gen** directory:

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

The example generator in the body of the request:

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

where **id** is the name of the file in the **pdf_report_gen** service directory.

An example of a simple request for the generator inside the example.json file in the **pdf_report_gen** directory:

```
{
   "report": "report-generator",
   "uri":{
      "uri": "local",
      "id": "example"
   },
   "options":{
      "enable-debug-report-save": true
   }
}
```

Input data

The **input-data** object contains the input data that is substituted into the tags.

> ® **Note**
>
> PDF generator commands that work with text may contain tags that will be replaced with input data from the request. One can read more about using tags in the "Tags" section below.

Example:

```
    "input-data":
{
    "ORGANIZATION": “«Success» LLC”,
    "CLINAME: "Peter Parker"
}
```

Optional: PDF generator

The PDF **embedded-report-generator** object contains generator commands. It is used optionally when **uri** is set to **local**.

It is described in detail in the subsection "Generators as part of the request".

Request options

The **options** request object has fields:

- **output-mode – string that specifies the form of output delivery. Supported values:**
  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

  Note: The *enable-binary-output* flag is considered outdated but is still in use to support older reports.
- **domain-name** – string that specifies the domain name to form the path for saving the report file when using a file server (providing the report by link).
- **file-name** – string that specifies the file name to form the path for saving the report file when using a file server (providing the report by link).
- **enable-debug-pdf-log** - Boolean value that enables extended diagnostic log of PDF creation. Default value is **false**.
- **enable-debug-report-save** - Boolean value that specifies whether to create a copy of the document report and request copy (file name - template identifier) in the local **reports_debug** directory of the service. The is default value **false**.

Example:

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

5.2.3 Example of a service call

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

Response structure

The service response contains an object in JSON format that includes:

1. Error description (error code, error message). In case of a successful service response, the value **null** is returned.
2. Result (base64 encoded generated PDF file of the document).

Response format

```
{
  "error":
```

(continued on next page)

(continued from previous page)

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

Response example

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

Using page templates

One can use ready-made PDF page templates to generate a document using the PDF generator.

There is a special command **setPageTemplate** to set the template of a particular page. One can read more about the command in the subsection “Generator commands”.

> **Note**
>
> Multi-page templates are supported.

To position text and other elements on a template page, one must set coordinates. Read more about specifying coordinates of document elements in the subsection "Coordinates".

Page templates in PDF format are stored locally in the **templates/pdf** directory located in the service application directory.

Service directories

PDF generation scripts are stored in the **pdf_report_gen** directory located in the service application directory**.**

Page templates in PDF format are stored locally in the **templates/pdf** directory located in the service application directory.

Example requests can be found in the directory **query_examples/pdf.**

Fonts to be used in the generator are located in the **assets/fonts** directory located in the service application directory. They are loaded at service startup.

> **Note**
>
> When adding new font files, one must restart the service.

Examples of requests and templates

Example requests can be found in the **examples/pdf/embedded-report-generator** and **examples/pdf/report-generator** directories in the *Examples archive*.

Example templates can be found in the **examples/pdf** directory of the *Examples archive*.

5.2.4 Generator commands

The PDF document is generated using special generator commands.

The commands are located inside the **embedded-report-generator** object if the generator is embedded in the request (see "Generator as part of the request" above).

Commands can be also located in a JSON format file, which is located in the **pdf_report_gen** directory in the service application directory (see "Generator as part of the service" above).

Commands are described inside the **commands** array.

Example:

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

The **commands** array can include the commands described below.

NewPage

Creates a new page in the document.

Parameters

- **orientation** - string. Supported values:
  - "L" - landscape orientation
  - "P" - portrait orientation

The default value is “P”.

Example:

```
{
  "name":"NewPage",
  "params":{
```

(continued on next page)

(continued from previous page)

```
        "orientation":"L"
    }
}
```

setCoordinateMode

Specifies the interpretation of coordinates and dimensions specified in subsequent commands. The default mode is "in millimeters".

Parameters

- **mode -** Supported values: "millimeters", "percent", "normalized".

setPageMargins

Sets the indentation from the page borders.

Parameters

left, **right**, **top**, **bottom** - floating point numbers. Indentation from the left, right, top, bottom edges of the page. When the bottom margin is reached in commands of **Row_Print** type, a new page is automatically created

Example:

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

setCurrentX

Sets the X coordinate of the cursor.

Parameters

- **value** - floating point number. X coordinate

setCurrentY

Sets the Y coordinate of the cursor.

Parameters

- **value** - floating point number. Y coordinate

setCurrentXY

Sets the coordinates of the cursor.

Parameters

- **x** is a floating point number. X.coordinate

- **y** is a floating point number. Y coordinate

saveX

Saves the X coordinate for later use in the command coordinate parameters via the tag **[PDF:savedX]**.

Parameters

- **value** - optional parameter, by default equal to the current X coordinate. It can be a floating point number or a string-expression with references to embedded tags. Example expression: **"[PDF:currentX] - 50"**

saveY

Saves the Y coordinate for later use in the command coordinate parameters via the tag **[PDF:savedY]**. Warning: the value may become irrelevant when creating a new page later.

Parameters

- **value** - optional parameter, by default equal to the current Y coordinate. It can be a floating point number or a string-expression with references to embedded tags. Example expression: **"[PDF:currentY] - 50"**

saveCoordinate

Saves a necessary coordinate value for later use in the coordinate parameters of commands via the **[SAVED:key]** tag. Supplements **saveX/saveY** commands in cases where multiple values need to be accessed.

Parameters

- **key** - string. The arbitrary key for use in the [SAVED:key] tag
- **value** - can be a floating point number or a string-expression with references to embedded tags. Example expression: **"[PDF:currentY] - 50"**

Example:

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

setRotate

Sets the rotation of all subsequent objects.

Parameters

- **angle** is a floating point number. Angle of rotation in degrees.
- x **is** a floating point number. X coordinate of the rotation point. If null, then X is the cursor coordinate
- **y** is a floating point number. Y coordinate of the rotation .point. If null, then Y is the cursor coordinate

Example:

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

Command cancellation:

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

startOpacity

Specifies the degree of transparency for all subsequent objects. The action is canceled by the **endOpacity** command.

Parameters

- **val** - floating point number. Valid range is from 0.0 to 1.0, where zero is full transparency.

endOpacity

Cancels the effect of **the startOpacity** command.

No parameters.

SetPrintFont

Sets the font type and size for subsequent text commands.

Parameters

- **font_id** Supported values: names of font files from **assets/fonts** directory without extension. Fonts are loaded at service startup.
- **size** - the **size** of the font in points in the user space of the pdf document.

> ® **Note**
>
> The only supported charset is cp1251.

setColor4Text

Sets the text color for subsequent text commands.

Input options:

- **r**, **g**, **b** integer decimal components of RGB color in the range from 0 to 255. Example of red color: "r":255, "g":0, "b":0.
- **color** string in the format "#rrggbb", where rgb are hexadecimal components of RGB color in the range from 0 to ff. Example of red color: "color": "#ff0000".

setColor4Drawing

Sets the border color for objects output after this command is executed. Examples of objects: table cells, circles, lines, rectangles, etc.

Parameters: similar to the **setColor4Text** command.

setColor4Filling

Sets the fill color of the internal areas for objects output after this command is executed. Examples of objects: table cells, circles, rectangles, etc.

Parameters: similar to the **setColor4Text** command.

setLineWidth

Sets the thickness of lines to be output after execution of this command. Examples of objects: table cells borders, lines, rectangles, etc.

Parameters

- **width** - floating point number. Line thickness.

setCellMargin

Sets the table cell content indentation from the left, top, right, and bottom borders.

Parameters

- **margin** is a floating point number.

setCellBottomMargin

Sets the table cell content indentation from the bottom border.

Parameters

- **margin** is a floating point number.

setCellLeftMargin

Sets the table cell content indentation from the left border.

Parameters

- **margin** is a floating point number.

setCellTopMargin

Sets the table cell content indentation from the top border. Parameters

- **margin** is a floating point number.

setCellRightMargin

Sets the table cell content indentation from the right border. Parameters

- **margin** is a floating point number.

PrintCell

Outputs a rectangular cell with text inside. One can specify the color and if there are borders.

Parameters

- **w** - floating point number. The width of the cell rectangle.
- **h** is a floating point number. Height of the cell rectangle.
- **txt** - text
- **border** - string. It sets the visible cell borders. Valid values: 0 - no border, 1 - outer rectangle border, L - left, T - top, R - right, B - bottom border. A combination of L, T, R and B values is allowed.
- **align** - string. Horizontal alignment of text. Values: L – left edge, R - right edge, C – center edge, J – justified alignment across the cell width
- **vert_align** - string. Vertical alignment of the text. Values: T – top edge. B - bottom edge, C - center. Default value: C.
- **fill** - Boolean value that indicates whether to fill the cell with the current color. See **setColor4Filling**. The default value is *false*.
- **link** - string. A link, such as a URL or an internal link. Not supported in the current version.
- **clipping** - Boolean value that specifies whether to clip the text on the cell borders. Default value - *false*.
- **ln** - integer. Cursor position after drawing the cell. Valid values: 0 - next to the cell, 1 - new line, 2 - under the cell. The default value is 0.

Example:

```
{
  "name":"PrintCell",
  "params": {
     "w":120,
     "h":7,
     "txt": "Personal Data",
     "border":"0",
     "align":"L",
     "vert_align":"C",
     "fill": true,
     "clipping":false,
     "ln": 2
   }
}
```

PrintMultiLineCell

Outputs a multi-line cell

Parameters

- **w** is a floating point number. Width of the rectangle.
- **h** is a floating point number. Height of the cell rectangle.
- **txt** - text

- **border** - string. Sets the visible cell borders. Valid values: 0 - no border, 1 - outer rectangle border, L - left, T - top, R - right, B - bottom border. A combination of L, T, R and B values is allowed.
- **align** - string. Horizontal alignment of text. The values L - by the left edge. R – by the right edge, C - center, J – justified alignment across the cell width.
- **vert_align** - string. Vertical alignment of the text. Values: T - top edge. B - bottom edge, C - center. Default value is C.
- **fill** - Boolean value that indicates whether to fill the cell with the current color. See. **setColor4Filling**. The default value is *false*.
- **maxline** - array of floating point numbers. The array element specifies the maximum number of text lines in a multiline cell.
- **link** - string. A link, such as a URL or an internal link. Not supported in the current version.
- **clipping** - Boolean value that specifies whether to clip the text on the cell borders. Default value - *false*.
- **indent** - floating point number. Indentation for the first line of text. The default value is 0.
- **ln** - integer. Cursor position after drawing the cell. Valid values: 0 - next to the cell, 1 - new line, 2 - under the cell. The is default value 0.

Example:

```
{
  "name":"PrintMultiLineCell",
  "params":{
     "h":5,
     "w":95,
     "txt": "Amount received for the entire period",
     "border":"LTR",
     "align":"L",
     "vert_align":"T",
     "fill":true,
     "indent":0,
     "clipping":false,
     "ln":0
   }
}
```

Row_Print

Prints a line in a PDF document. A line consists of multi-line cells (see **PrintMultiLineCell**). The height of the line is determined by the cell with the maximum height.

Parameters

- **data** - array of strings. The array element specifies the cell text in the string.
- **input_data_tag** - alternative to data, the name of the array of string-value cells in the request input data.
- **width** is an array of floating point numbers. The array element specifies the cell width.
- **border** - array of strings. The array element describes the visible borders of the cell. See the description of the command **PrintMultiLineCell**. All borders are visible by default.

- **maxline** is an array of floating point numbers. The array element specifies the maximum number of text lines in a multiline cell. There is no limit by default.
- **align** - array of rows. The array element specifies the horizontal alignment in the cell.
- **vert_align** - array of strings. The array element specifies the vertical alignment in the cell.
- **font** - an array of font descriptor objects. The array element specifies font parameters in a cell. The set of fields of each descriptor is similar to the parameters of the **SetPrintFont** command. The default value is null, which means that the font settings set in the previous **SetPrintFont** call are used.
- **h** - floating point number. The height of the cell. The default value is 5 units.
- **fill** - Boolean value that indicates whether to fill the cell with the current color. See **setColor4Filling**. The default value is *false*.
- **min_height** is a floating point number. The minimum height of the row. Zero means that the parameter is not used. The default value is 0.
- **clipping** - Boolean value that specifies whether to clip the text on the cell borders. The default value - *false*.

Example:

```
{
 "name":"Row_Print",
 "params":{
    "h":4.5,
    "min_height":  27,
    "width":[85, 95],
    "data":[
       "1.4 Funds (part of funds) of the maternity (family) capital",
       "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

Outputs the text at the specified coordinates.

Parameters

- **x** is a floating point number. X coordinate of the beginning of the text

- y **is** a floating point number. Y coordinate of the beginning of the text
- **text** - text

Example:

```
{
  "name":"PrintText",
  "params":{
    "x":10,
    "y":40,
    "text": "List of documents:"
   }
}
```

underline

Underlines the text output by the command **PrintText**.

The parameters are the same as for **PrintText**, the parameter values must match.

```
{
  "name":"underline",
  "params":{
     "x":10,
     "y":40,
     "text": "List of documents:"
   }
}
```

LineBreak

Line break. It moves the cursor to the beginning of the next line.

Parameters

- **h** - floating point number. Indentation from the current Y coordinate of the cursor.

Line

Draws a straight line between two points on the page. Parameters

- **x1** is a floating point number. X coordinate of the line start.
- **y1** is a floating point number. Y coordinate of the line start.
- **x2** is a floating point number. X coordinate of the end of the line.
- **y2** is a floating point number. Y coordinate of the end of the line.
- **color** string in the format "#rrggbb", where rgb are hexadecimal components of RGB color in the range from 0 to ff. An example of red color: "color": "#ff0000". By default, the color previously set by **setColor4Drawing** command is used.
- **width** - floating point number. Line thickness. By default, the value previously set by **setLineWidth** command is used.

Example:

```
{
  "name":"Line",
  "params":{
        "x1":100,
        "y1":230,
        "x2":150,
        "y2":230,
        "color":"#000000",
        "width":0.2
    }
}
```

Circle

Draws a circle on the page.

Parameters

- **x** is a floating point number. X coordinate of the center of the circle.
- **y** is a floating point number. Y coordinate of the center of the circle.
- **r** is a floating point number. The radius of the circle.
- **draw** - Boolean value that controls drawing of the circle outline. The default value is true.
- **fill** - Boolean value that controls the fill color of the inner area of the circle. The default value is *false*.
- **draw_color** the color of the circle outline. The string in the format "#rrggbb", where rgb is hexadecimal components of RGB color in the range from 0 to ff. An example of a red color is “color”: "#ff0000". By default, the color previously set by **setColor4Drawing** command is used.
- **fill_color** the color of the circle outline. The string in the format "#rrggbb", where rgb is the hexadecimal components of RGB color in the range from 0 to ff. An example of a red color is “color”: "#ff0000". By default, the color previously set by **set with the setColor4Filling** command is used.
- **linewidth** is a floating point number. The thickness of the circle outline. By default, the value previously **set** by **setLineWidth** command is used.

Example:

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

putImage

Parameters

- **name** - string. The name of the image or image ID. If the name matches the one specified in the previous call, the image specified in that same call is used.
- **data** - string. Base64 encoded image file. If **name** is the same as the one specified in the previous call, this parameter is not specified.
- **x** is a floating point number. X coordinate of the image.
- **y** is a floating point number. Y coordinate of the image.
- **w** - floating point number. Image width in units specified by the command **setCoordinateMode**. If the parameter is not specified, the image is displayed in actual width, which is calculated based on the pixel density of the PDF document (72 dots per inch) and the image width according to the image file. *Example*: a 72x72 dots image will be 1 inch by 1 inch in a PDF document.
- **h** - floating point number. Image height. Units: similar to the parameter w.
- **link** - string. URL or ID of internal link. Not supported in the current version.

Example:

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

setPageTemplate

Parameters

- **template** - object in the format

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

where template id is the path to the pdf document relative to the **templates/pdf** directory without extension.

- **page** - integer. Template page number for output. Range from one to the number of pages in the template.

Example command for a multi-page template:

```
{
 "name":"setPageTemplate",
 "params":{
    "template":{
```

(continued on next page)

(continued from previous page)

```
        "id":"example_multipage_template",
        "uri":"local"
      },
     "page":1
     }
}
```

® Note

Duplicate font resources in the report when using the **setPageTemplate** command when the font is the same as the one loaded by the **setPrintFont** command. May increase the size of the report file.

checkPageBreak

If adding the specified height to the current Y coordinate results in a page overflow (i.e., going beyond the bottom indent of the page), the command adds a new page and returns *true* (the return value has a value only in the **if** command).

Parameters

- **h** - floating point number. The height to check for page overflow.
- **newpage** - Boolean value that specifies whether to create a new page in the document when overflow occurs. The default value is *true*.

if

The command allows to execute a set of subcommands if the condition is true immediately at the moment of command execution (unlike **register_auto_new_page_commands**).

Supported set of commands to use as a condition: **checkPageBreak**.

Parameters

- **condition** - object describing the command that returns a logical value.
- **commands** - an array of commands to be executed in case the command specified in the **condition** parameter returned *true*.

Example:

```
{
  "name": "if",
  "params": {
    "condition": {
      "name": "checkPageBreak",
      "params": {
        "h": 4.5,
        "newpage": true
      }
    },
    "commands": [
      {
        "name": "setPageTemplate",
        "params": {
```

(continued on next page)

(continued from previous page)

```
          "template": {
            "id": "example_template",
            "uri":   "local"
          }
        }
      },
      {
        "name": "setCurrentXY",
        "params": {
          "x": 10,
          "y": 30
        }
      }
    ]
  }
}
```

Typical use case: before attempting to display an element or set of elements at the bottom of the page, when there is a risk of going beyond the bottom margin of the page. The value of the **h** parameter to check is usually taken from the size of the element to be rendered.

> ® **Note**
>
> The **if** command works directly in the place where it is called, with the current Y coordinate. That is, if it is called at the moment when there is a vertical space of height **h** on the page, the list of subcommands will not be executed.

> ® **Note**
>
> It is not recommended to use the **if** command to output table rows. It can be useful for outputting a unique element when rendering is closer to the bottom of the page. In case of tables, it is better to use **register_auto_new_page_commands**

register_auto_new_page_commands

Allows registering a set of subcommands to be executed when execution of commands of **Row_Print** type leads to automatic creation of a new document page. It can be useful for rendering a table header. It is effective from the moment it is called. It can be canceled by calling the same command with an empty list of subcommands.

Parameters

- **commands** - an array of commands to execute.

Example:

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

(continued on next page)

(continued from previous page)

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

Cancellation of the command:

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

5.2.5 Coordinates

Commands that accept coordinates and dimensions as parameters interpret the values based on the call of the command **setCoordinateMode**. Millimeters are used by default.

All such parameters can be specified numerically or textually.

In text form, expressions and special tags **PDF:currentX/PDF:currentY** are supported.

Example:

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

5.2.6 Tags

Commands that work with text (**PrintText**, **PrintCell**, etc.) can contain tags that will be replaced with input data from the request.

The input data is contained in the body of the request with the "**input-data**" object: {...} (see "Input data" in the subsection "Request structure").

Tag format

The tag is specified in square brackets. For example, [ORGANIZATION].

Example:

```
{
  "name":"PrintMultiLineCell",
  "params":{
      "h":4,
      "w":77,
      "ln":2,
      "txt":"[ORGANIZATION], represented by CEO John Smith, as one party, and [CLINAME],
            hereinafter referred to as 'Participant', as the other party, have entered into this
            Agreement as follows:",
      "fill":false,
      "align":"J",
      "border":"0",
      "indent":15,
      "clipping":false
  }
}
```

No brackets are specified in the request (see the example from "Input data" in the subsection "Request structure").

Embedded tags

Some tags are embedded in the PDF generator.

1. **PDF:currPageNum**, current page number.
2. **PDF:currentX/PDF:currentY**, X and Y coordinates of the cursor.
3. **PDF:savedX/PDF:savedY**, available after calling **saveX/saveY** commands.
4. **[SAVED:key]**, available after the **saveCoordinate** command.

Example:

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

(continued on next page)

(continued from previous page)

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

5.2.7 Barcode operation

It is possible to output information as a barcode by using the Libre Barcode family of fonts. The fonts are available in the librebarcode repository. Each font represents a specific barcode format. The following formats are supported:

- Code-128.
- Code-39
- EAN13

The font can include the original text directly below the barcode (fonts with Text suffix). Find below the example of the Code-128 format.

Code-128.

Fonts to be placed in the **assets/fonts** folder of the service:

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

How to use

In the command generator, select the desired font, for example:

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

Test request

```
curl --request POST --data-binary "@templates/examples/pdf/embedded-report-generator/
↪request_barcode_code-128.json" http://localhost:8886/pdf_report_json
```

The PDF **report-generator** descriptor object contains:

- **uri** – string that specifies the position of the generator command source depending on how the **id** parameter is interpreted. Supported values: **local** and **embedded**.
- **id** - string. The generator ID.

There are two possible ways to use the PDF generator:

- generator as part of the request (all generator commands are placed inside the **embedded-report-generator** object of the request body),
- generator as a part of the service (all generator commands are placed inside JSON file in special **pdf_report_gen** folder on the server).

Therefore:

- When is **uri** set to **embedded**, **id is** only used for diagnostic messages and request identification. The generator must be in the root **embedded-report-generator** object of the request. See "Generators as part of the request" below.
- When **uri** is is the set to **local**, **id** is the name of the file in the **pdf_report_gen** service directory. See "Generators as part of the service" below.

5.3 PDF merge service

5.3.1 General description of the service

The service combines documents in PDF format into one document by HTTP request in JSON format.

> ® **Note**
>
> XML format is not supported in the current version.

5.3.2 Merging PDF documents

Service Description

Name of service	PDF merge service
Path to service	[host]:[port]/pdf_merge_json
Method	POST
Parameters	The request body must contain an object in JSON format. You can read more about the structure of the request body in the “Request Body Structure” subsection. In response, the service gives a base64-encoded PDF document file.
Purpose	The service is designed to merge several PDF documents into One document.

Request body structure

The body of the request contains an object in JSON format that includes:

1. Request ID.
2. Input data: list of document descriptors to merge.
3. Request options.

Example:

```
{
    "request-id": "...",
```

(continued on next page)

(continued from previous page)

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

Request ID

The **request-id** is used to write the merged file in debug mode and to identify the request in the log.

Input data

The **input-data** object contains:

A list (array) of **document-descriptors** to merge.

Example:

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

Input document descriptor

The input document descriptor object supports fields:

- **id** - string. The document identifier that is used for references to the input parameter in error messages and diagnostic log.
- **content-type** – string that indicates how the **content** parameter is interpreted. Supported value: base64.
- **content** - string. If the content type is base64 – this means base64 encoded PDF document. Note: There is no limitation on the number of pages in the document.

Example:

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

Request options

The **options** request object has fields:

- **output-mode – string that specifies the form of output delivery.**
  Supported values:

  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

  Note: The *enable-binary-output* flag is considered outdated but is still in use to support older reports.

- **domain-name** – string that specifies the domain name to form the path for saving the report file when using a file server (providing the report by link).
- **file-name** – string that specifies the file name to form the path for saving the report file when using a file server (providing the report by link).
- **enable-debug-merged-doc-save** - Boolean value that specifies whether to create a copy of the document report and a copy of the request (file name - template ID) in the local **reports_debug** directory of the service. The default value is **false**.
- **enable-debug-pdf-log** - Boolean value that enables extended diagnostic log of PDF creation. Default value is **false**.

Example:

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

Example of the request body

```
    {
  "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
  }
}
```

The example requests can be found in the **examples/pdf/pdf/pdf_merge** directory in the *Examples archive*.

Example of a service call

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

Response structure

The service response contains an object in JSON format that includes:

1. Error description (error code, error message). In case of a successful service response, the value **null** is returned.
2. Result (base64 encoded PDF file of the merged document).

Response format

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

Example of the response

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

> **Note**
>
> The merged document may be larger than the sum of the merged documents. This is due to duplication of identical resources (e.g., fonts) used in incoming documents.

5.4 Print form generation service

5.4.1 General description of the service

The service generates a printed form of a document in PDF format by HTTP-request in JSON format:

- A table of signers is generated on the last page of the document.
- Intermediate pages display a footer with brief information about the document.

> **Note**
>
> XML format is not supported in the current version.

5.4.2 Generating a printed form of the document in PDF format

Service Description

Name of service	Print form generation service
Path to service	[host]:[port]/print_form_pdf_json
Method	POST
Parameters	The request body must contain an object in JSON format. One can read more about the structure of the request body in the subsection “Request body structure”. In response, the service gives a base64-encoded PDF file of the printed form of the document.
Purpose	The service is designed to generate a printed form of a document in PDF format, including columns with brief information about the document and a table of signers on the last page.

Request body structure

The body of the request contains an object in JSON format that includes:

1. Request ID.
2. Inputs:
   - Descriptor of the input document to create the printed form (the original PDF document to create the printed form).
   - Descriptor of the table of signers and footer of intermediate pages (data to be substituted in the table of signers and footer of intermediate pages with data on the document to be signed).
   - Options for the appearance of generated tables (color/size of fonts).
3. Request options.

Example:

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

Request ID

The request-**id** is used to write the file in debug mode and to identify the request in the log.

Input data

The **input-data** object contains:

- Descriptor of the input document to create the printed form.
- Descriptor of the signers table and footer of the intermediate pages.

- Options for the appearance of generated tables.

Example:

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

Input document descriptor

The **input-document** descriptor object for creating a printable input-document form has fields:

- **id** - string. The document identifier. It is used for references to the input parameter in error messages and diagnostic log.
- **content-type** – string that indicates how the content parameter is interpreted. Supported value: base64.
- **content** - string. If the content type is base64 – this means base64 encoded PDF document. Note: There is no limitation on the number of pages in the document.

Example:

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

Descriptor of the signers table and footer of intermediate pages

The descriptor object of the signers table and the footer of intermediate pages **signatures-info** has fields:

- **signature-summary-text** - string. A general description of the signer table to output before the signer data.
- **table-header-texts**- array of strings. A list of signer table headers. Note: in the current version there can only be 4 columns (limitation of the **signatures** array element described below).
- **table-header-column-width-list** is an array of floating point numbers that controls the width of the table columns. The sum of all normalized widths must equal one. The size of the array must match the size of the **table-header-texts** field.
- **signatures** - array of objects with the data on signers. See "Signer data descriptor" below for details.
- **intermediate-page-footer-info** - data to be displayed in the footer on all pages of the document except for the last page, where the table of signatories is located. See " Descriptor of the intermediate pages footer with data on the document to be signed " below for details.
- **logo-asset-name** - string. The optional parameter. Name of the image file (with extension) to be displayed in the upper right corner of the table of signers. The path is set relative to the directory **assets/images/print_forms** in the working directory of the service.

Example:

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

Signer data descriptor

The array element of the signers' data **signatures** data descriptor has fields:

- **description** - string. A general description of who the signature belongs to. The text is displayed in the first column of the signatures table.
- **certificate-owner** - object with string fields **organization** and **employee**, describing organization, full name and position of the certificate-owner. The text is displayed in the second column.
- **certificate** - object with string fields **serial-number** and **validity** describing the serial number and validity period of the certificate used for signing. The text is displayed in the third column.
- **signing-timestamp** - string. Description of the exact time of signing. The text is displayed in the fourth

column.

Example:

```
{
  "description": "Sender's signature",
  "certificate-owner": {
    "organization": "\"PHI-CIRCLE\" LLC",
    "employee": "Peter Parker, CEO"
  },
  "certificate": {
    "serial-number":  "11255807011ABCBBE4DBF93ECCCB96C45",
    "validity": "from 01/02/2001 18:48 to 01/02/2031 18:48 GMT+01:00"
  },
  "signing-timestamp": "01.02.2021 19:00 GMT+01:00"
}
```

Descriptor of the intermediate pages footer with data on the document to be signed

The descriptor object **intermediate-page-footer-info** has fields:

- **lines** - array of lines. Arbitrary data describing the document to be signed. It is output as a table with invisible borders. The last line will be a description of the current page number, if enabled by the **print-page-number** field.
- **logo-asset-name** - string. The optional parameter. Name of the image file (with extension) to be displayed in the right part of the document data. The path is set relative to the directory **assets/images/print_forms** in the working directory of the service.
- **print-page-number** - Boolean value that specifies whether to print the current page number of the document under the document data.

Example:

```
{
  "lines": [
    "Transmitted via XXX on 01/02/2003 15:20 GMT+01:00",
    "abcdef00-0000-1111-2222-abcdefabcdef"
  ],
  "logo-asset-name": "golang-gopher.png",
  "print-page-number": true
}
```

Generated table appearance options

The **draw-options** object of the generated table appearance options has fields:

- **font-color** - a string in the format "#rrggbb", where rgb - hexadecimal components of RGB color in the range from 0 to ff. The example of red color: "font-color": "#ff0000".
- **font-size** - integer. The font size of the signer data in the signer table. The unit is equal to 1/72 of an inch.
- **large-font-size** is an integer. The size of the font **signature-summary-text** output in the signer table. The unit is equal to 1/72 of an inch.
- **table-border-color** - a string in the format "#rrggbb", where rgb - hexadecimal components of RGB color in the range from 0 to ff. An example of red color: "font-color": "#ff0000". The color of the signer table borders.

Example:

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

Request options

The **options** request object has fields:

- **output-mode – string that specifies the form of output delivery. Supported values:**
  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

  Note: The *enable-binary-output* flag is considered outdated but is still in use to support older reports.
- **domain-name** – string that specifies the domain name to form the path for saving the report file when using a file server (providing the report by link).
- **file-name -** string that specifies the file name to form the path for saving the report file when using a file server (providing the report by link).

- **enable-debug-doc-save** - Boolean value that specifies whether to create a copy of the document report and a copy of the request (file name - template identifier) in the local **reports_debug** directory of the service. The default value is **false**.
- **enable-binary-output** - Boolean value that indicates that the service will output the result as binary data, without base64 encoding. The default value is **false**.
- **enable-debug-pdf-log** - Boolean value that enables extended diagnostic log of PDF creation. The default value is **false**.

Example:

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

Example of the request body

```
{
  "request-id":  "print_form_pdf_example1",
  "input-data":{
    "input-document": {
      "id": "doc1",
      "content-type": "base64",
      "content":   "JVBERi0xLjcKJYGBgYEKCjYgMCB....VmCjI1ODI5CiUlRU9G"
    },
    "signatures-info":{
      "signature-summary-text":"The document is signed and transmitted via the operator of JSC
                                «Example»."
      "table-header-texts": [
        "",
        "Certificate owner: organization, employee",
        "Certificate: serial number, validity period",
        "Date and time of signing"
      ],
      "table-header-column-column-width-list": [
        0.15,
        0.33,
        0.32,
        0.2
      ],
      "signatures": [
        {
          "description": "Sender's signature",
          "certificate-owner":  {
            "organization": "\"PHI-CIRCLE\" LLC",
            "employee": Peter Parker, CEO"
          },
         "certificate": {
            "serial-number":  "11255807011ABCBBE4DBF93ECCCB96C45",
```

(continued on next page)

(continued from previous page)

```
            "validity": from 01.02.2001 18:48 to 01.02.2031 18:48 GMT+01:00"
          },
          "signing-timestamp": "01.02.2021 19:00 GMT+01:00"
        },
        {
          "description": "Signature of recipient",
          "certificate-owner:  {
            "organization": "«SUPPLY CONCESSION» LLC",
            "employee": "John Green, CEO"
          },
          "certificate": {
            "serial-number":  "3610D7230004000503BF",
            "validity": "from 04.05.2006 16:07 to 31.12.2021 23:59 GMT+01:00"
          },
          "signing-timestamp": "02.02.2021 13:00 GMT+01:00"
        }
      ],
      "intermediate-page-footer-info":
        "lines": [
          "Transmitted via XXX on 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
  }
}
```

The example requests can be found in the **examples/pdf/print_form** directory in the *Examples archive*.

Service call example

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

Response structure

The service response contains an object in JSON format that includes:

1. Error description (error code, error message). In case of a successful service response, the value **null** is returned.
2. Result (base64 encoded PDF file of the printed form of the document).

Response format

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

Response example

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

Service directories

Images to be used as the logo that are located in the **assets/images/print_forms** directory in the service application directory**.**

5.5 Service for converting XLSX documents to JSON, XML, CSV

5.5.1 General description of the service

The service converts XLSX document to JSON, XML, CSV formats by HTTP-request in JSON format or by HTTP-request in multipart/form-data format.

In response, the service gives the content of the input document presented in one of the target formats.

Service Description

Name of service	Service for converting XLSX documents to JSON, XML, CSV
Path to service	For multipart/form-data requests [host]:[port]/xlsx_convert For json requests [host]:[port]/xlsx_convert_json
Method	POST
Parameters	The request body must contain an object in JSON format or objects in multipart/form-data format. One can read more about the structure of the request body in the subsection “Request body structure”. In response, the service returns the document content in the target format.
Purpose	The service is designed to convert XLSX document to JSON, XML, CSV formats

Request body structure

In general, JSON or multipart/form-data queries include:

- Document descriptor.
- Request options.

Example for JSON request:

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

Document descriptor

The document descriptor for the JSON format has fields:

- **name** - string. The name is a document identifier. It is used for references to the input parameter in error messages and diagnostic log.
- **data** - string. BASE64 encoded XLSX-document.

The document descriptor for the multipart/form-data format consists of the parameter **xlsx="@[path to XLSX-document]**

For example:

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

Request options

The request can contain the following options:

- **output-mode – string that specifies the form of output delivery. Supported values:**
  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

  Note: The *enable-binary-output* flag is considered outdated but is still in use to support older reports.
- **output-format** - string. The target format of conversion. Supported values: "JSON", "XML", "CSV". The default value is **JSON**
- **domain-name** – string that specifies the domain name to form the path for saving the report file when using a file server (providing the report by link).
- **file-name** - string. Specifies the file name to form the path for saving the report file when using a file server (providing the report by link).
- **process-sheets-numbers** - array of integers that defines sequential numbers of pages to be converted from the input document (For example: [1,2,3] - for processing the first three pages). The numbering starts with "1". Default value - **all pages** for JSON and XML, **first page** for CSV.
- **process-from-start-of-sheet** - Boolean value that specifies to start processing the document from the very first cell (A1) regardless of whether it contains data or not. Otherwise, processing will be performed from the first cell containing the data. The default value is **false**.

- **no-extend-line** - Boolean value. This option is responsible for extending each line of the source document to the maximum line length in the document to get a two-dimensional data array (this is the default behavior of the converter). If true, the option specifies not to append strings, and the output will be an array of data arrays. The default value is **false**.
- **process-start-cell** - string. This option allows you to specify the starting cell for processing (in cell naming format, e.g. "E5"), i.e. it specifies the left upper boundary of processing by row-column, which allows to convert only the data range of interest. The default value is the **first cell with data** (will be defined automatically).
- **process-end-cell** - string. This option allows you to specify the final cell to be processed (in cell naming format, e.g. "G8"), i.e. it specifies the right lower boundary of processing by row-column, which allows you to convert only the data range of interest. The default value is the **last cell with data** (will be defined automatically).
- **ignore-empty-rows** - Boolean value that specifies to skip strings without data when exporting. This option is available only for exporting to JSON, XML, as these formats contain row numbering. The default value is **false**.
- **encoding** - string. This option allows you to specify the target encoding of the processing result. Supported values: WIN1250, WIN1251, WIN1252, WIN1253, WIN1254, WIN1255, WIN1256, WIN1257, WIN874, WIN866, KOI8RISO_8859_5. The default value is **UTF8**.
- **enable-debug-report-save** - Boolean value that specifies whether to create a copy of the request in the local **reports_debug** directory of the service. The default value is **false**.

For requests in JSON format, the options are passed in the **options** object.

Example:

```
"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
}
```

For *multipart/form-data* requests, each option is passed as a separate parameter, all parameters are specified as strings.

Example:

```
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
```

The example requests can be found in the **examples/xslx** directory in the *Examples archive*.

Service call example

For requests in JSON format:

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

For multipart/form-data requests:

```
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"
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
```

Response structure

The service response contains a text stream of data in the target encoding.

Conversion example

As an example, this is the conversion range D2:G4 of the second page of the document **convert_example.xlsx** from the *Example archive*:

	A	B	C	D	E	F	G	H
1								
2		№	Number of the insurance certificate of compulsory pension insurance of the insured person	Sex of the insured person	Full name of the insured person	Day, month, year of birth of the insured person	Date of the contract on compulsory pension insurance contract.	Number of the compulsory pension insurance agreement
3		1	001-002-003 00	F	Kate Wilson	01.01.1970	01.01.1988	ABC-001-002-003-00
4		2	001-002-004 00	M	Stephan Brandt	02.01.1970	02.01.1988	ABC-001-002-004-00
5		3	001-002-005 00	F	Anna Schedding	03.01.1970	03.01.1988	ABC-001-002-005-00
6								

The request is in multipart/form-data format:

```
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
```

Conversion result:

```
Gender of the insured person;Surname, first name, middle name (if available) of the insured person;␣
↪Day, month, year of birth of the insured person;the date of signing the contract on␣
↪compulsory pension insurance
F;Kate Wilson;01.01.1970;01.01.1988
M;Stephan Brandt;02.01.1970;02.01.1988
```

5.6 Service for converting XLS documents to JSON, XML, CSV

5.6.1 General description of the service

The service converts XLS (Microsoft Office Excel Binary 2003) document into JSON, XML, CSV formats by HTTP-request in JSON format or by HTTP-request in multipart/form-data format.

In response, the service gives the content of the input document presented in one of the target formats.

Service Description

Name of service	Service for converting XLS documents to JSON, XML, CSV
Path to service	For multipart/form-data requests [host]:[port]/xls_convert For json requests [host]:[port]/xls_convert_json
Method	POST
Parameters	The request body must contain an object in JSON format or objects in multipart/form-data format. One can read more about the structure of the request body in the subsection “Request body structure”. In response, the service gives the content of the document in the target format.
Purpose	The service is designed to convert XLS document to JSON, XML, CSV formats

The service operation, functionality and options are the same as for the XLSX conversion service, except that instead of the **xlsx** parameter in multipart/form-data request, the **xls** parameter is used.

Example:

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

5.7 Code generation service (QR code)

5.7.1 General description of the service

The service generates graphic codes in JPG, PNG formats by HTTP-request in JSON format. In this version, the service allows one to generate QR code with flexible setting of parameters of the resulting image.

Service Description

Name of service	Code generation service
Path to service	[host]:[port]/code
Method	POST
Parameters	The body of the request must contain an object in JSON format. More details about the structure of the request body can be found in the subsection "Request body structure". In response, the service returns the contents of the graphic file in the JPG or PNG format.
Purpose	The service is designed to generate graphic codes in JPG and PNG formats.

Example:

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

Request body structure

The body of the request contains an object in JSON format that includes:

1. Request ID.
2. Input data: code parameters.
3. Request options.

The minimum request to create a QR code must contain the type of request (type) and the data to be encoded (content). Example:

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

All other parameters in this case will have default values, which allows generating the "classic" QR code in JPG format:

Request ID

The request **id** is used to record the result in debug mode, as well as to identify the request in the logs.

Input data

The **input-data** object contains a descriptor of the graphic code to be created:

- **type** - string. A mandatory parameter. The type of the generated code. Supported values: "qr-code".
- **content** - string. A mandatory parameter. Encoded data.
- **qr-color** - string. A color of code points in hexadecimal record format. The default color is black.
- **bg-color** - string. A code background color in hexadecimal record format. The default color is white.
- **bg-transparent** – flag that defines background transparency for codes in PNG format. By default - false.
- **dot-size** - number. Code dot size in pixels.
- **format** - string. The resulting format. The possible options are JPG (default), PNG.
- **logo** - string. Base64 encoded logo image for inserting into the center of QR code. JPG, PNG formats are supported. The size of the encoded logo should not exceed 1/5 of the code size.

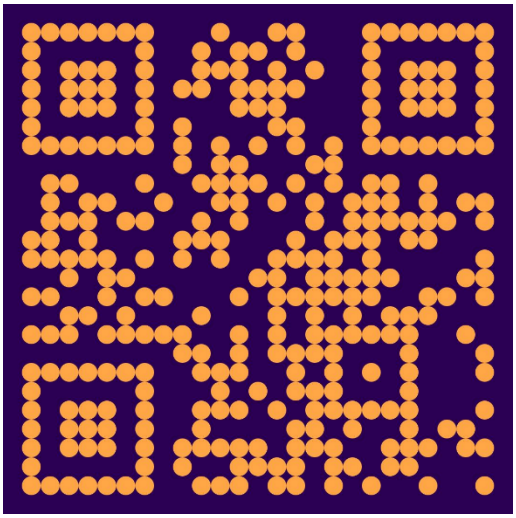
  Example:

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

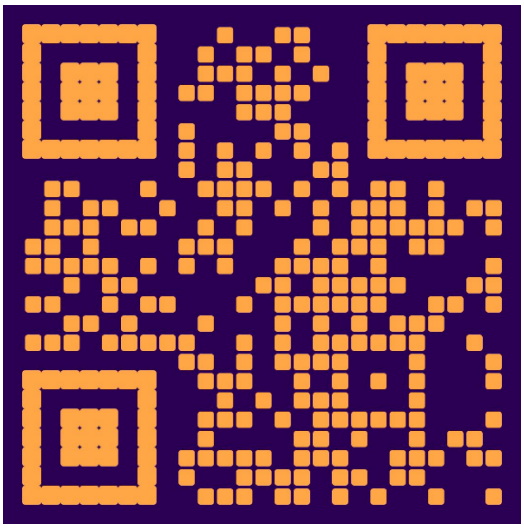
- **style** - string. QR code style.

  Style options:

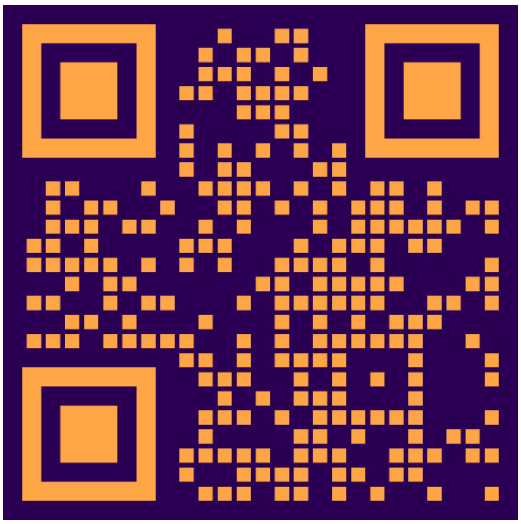
  – **dot**



  – **rounded_rectangle**



– **rectangle_with_border**

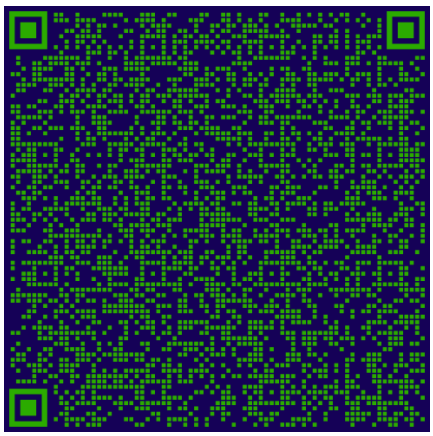


If no style is specified, the default style is used.

Thus, the parameters allow to flexibly customize the appearance of the resulting code, and using a specific content format, one can create codes to transmit various kinds of information.

VCARD QR code example:

```
{
  "id": "qr-code",
  "input-data":
  {
       "type": "qr-code",
       "content": "BEGIN:VCARD\nVERSION:3.0\nFN:John Smith\nN:Smith, John;;;\
↪nORG:Company XSQUARE\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:;;Example␣
↪Street 5, building 2;London;;105066;United Kingdom\nURL:http://xsquare.ru\
↪nEND:VCARD",
       "qr-color": "#2bab00",
       "bg-color": "#150157",
    "style": "rectangle_with_border",
    "format": "png"
  },
  "options":
  {
    "output-mode": "binary"
  }
}
```



Request options

The element (object) of request **options** contains:

- **output-mode** – string that defines the form of the result presentation.

  Supported Values:

  - **base64** - the result will be represented as a base64 encoded string. This value is used by default.
  - **binary** - the result will be represented as a binary stream.
  - **url** - the result will be presented as a link to the report file that can be downloaded from the report server.

CHAPTER 6

Examples

6.1 Examples archive

In this section, one can find example requests and templates for the service.

The archive of examples should be unpacked to the **templates** directory in the root of the service. The unpacked archive contains the following directories:

- pdf - contains PDF - page templates used in the *PDF report generator*
- examples - represents the following directory tree:

```
├──────docx
├──────pdf
│       ├──────embedded-report-generator
│       ├──────pdf_merge
│       ├──────print_form
│       └──────report-generator
│                └──────pdf_report_gen
├──────code
└──────xlsx
```

The **docx, xlsx** directories contain a set of requests (in json/xml formats), documents - templates (.docx and .xlsx) and scripts (.sh) for receiving reports.

The **code** directory contain a set of requests (in json format) and scripts (.sh) for receiving QR-code image (.png or .jpg).

The pdf directory contains the following directories:

- embedded-report-generator -requests (.json) and scripts (.sh) to generate .pdf reports via the *PDF report generator*.
- pdf_merge -requests (.json) and scripts (.sh) to merge PDF documents.
- print_form - requests (.json) and scripts (.sh) to *generate the print form*

- report-generator - requests (.json) and scripts (.sh) for generating reports via *PDF report generator* with an example of generator as part of the service. The pdf_report_gen directory with example generators should be placed in the root directory of the service.

6.2 Generating a report from the archive of examples

Examine below generating a report from a docx directory based on a lists.json request

```
{
  "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
      }
    }
  }
}
```

As a result of running the lists.sh script,

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

The report file lists_report.docx was written in the directory templates/examples/docx

> **Note**
>
> scripts may need to be assigned execution rights. This can be done by the following command:
> chmod +x *.sh

CHAPTER 7

My first report

7.1 General description

This subsection will cover the process of creating a simple report using the report server. Before starting, one needs to make sure that the installation of the report server was successful.

7.2 First template

To generate a report, one needs to create a document template in DOCX or XLSX format using MS Word/Excel, LibreOffice Writer/Calc, Google Docs, etc.

The document templates can contain tags that will be replaced with input data from the request. The tag is specified in square brackets. Example: [debt]. Read more about creating templates in DOCX and XLSX format in the section 'Service for generating reports from a template document'.

For the first report, we create a simple template in DOCX format that contains a few tags: first_report_template.docx. Examples of more complex templates can be found in the examples archive.

The created template is placed on the server in the **templates** directory located in the application directory of the service.

> ® **Note**
>
> The page templates in in the PDF format are stored **templates/pdf** directory located in the application directory of the service.

7.3 First request

Next, we will make an HTTP request.

The request for the first report in JSON format will look like this: first_report.json. The template document created earlier is specified in the **id** attribute of the **template** element. The examples of more complex requests can be

found in the examples archive.

Note: The XML format can also be used for requests in DOCX or XLSX format. To get a report file, use the 'service for report generation from a template document'.

To get the first report based on the DOCX template created, we use the URI http://localhost:8886/word_report_json.

7.3.1 Description

Name of service	Print form generation service
Path to service	[host]:[port]/print_form_pdf_json
Method	POST
Parameters	The body of the request must contain a JSON object:

```
{
  "template": {
    "uri": "local",
    "id": "first_report_template"
  },
  "input-data": {
    "ORGANIZATION": "Example LLC",
    "DATE": "01/01/2023",
    "EMP": "john Smith"
  },
  "options": {
    "formatting": {
      "tables": {
        "enable-cells-auto-merge": true
      }
    }
  }
}
```

A **template** object has attributes (fields):

- **uri** - string that specifies location of the template file document depending on how the **id** parameter is interpreted. Supported value: **local**.
- **id** - string. The template identifier. The path to the template document file relative to the **templates** service directory. It is also used to write a report file in debug mode and to identify the request in the log.

continues on the next page

Table 1 - continued from previous page

input-data - input data to be substituted into the template.

An **input-data** object can contain:

- tags of simple string data,
- table descriptor tags,
- list descriptor tags,
- image tags,
- block tags.

options - request options.

One can read more about JSON structure in the section 'Report generation service from the template document'. In response, the service gives the report document file in DOCX format encoded in BASE64. If it is necessary to get the result without BASE64 encoding, one should specify the enable-binary-output flag as *true* in the request options.

```
{
    "template": {
        "uri": "local", "id": "first_report_template"
    }, "input-data": {
        "ORGANIZATION": "«Example» LLC", "DATE":
        "01.01.2023", "EMP": "John Smith".
    }, "options": {
        "enable-binary-output": true, "formatting": {
            "tables": {
                "enable-cells-auto-merge": true
            }
        }
    }
}
```

Purpose	The service is designed to generate a printed form of the document in PDF format, including columns with brief information about the document and a table of signers on the last page.

7.3.2 Example request

```
curl -X POST \
'http://localhost:8886/word_report_json' \
-H 'Content-Type: application/json' \
-d '{
    "template":
    {
        "uri": "local",
        "id": "first_report_template"
    },
    "input-data":
    {
        "ORGANIZATION": "Example JSC",
        "DATE": "01/01/2023",
        "EMP": "John Smith"
    },
```

(continued on next page)

(continued from previous page)

```
    "options":
    {
        "formatting": {
            "tables":  {
                "enable-cells-auto-merge": true
            }
        }
    }
}'
```

7.3.3 Example of a request to save the result to a file without base64 encoding

```
curl -X POST \
'http://localhost:8886/word_report_json'    \
-H 'Content-Type:  application/json'  \
-d '{
    "template":
    {
        "uri": "local",
        "id": "first_report_template"
    },
    "input-data":
    {
        "ORGANIZATION": "Example LLC",
        "DATE": "01/01/2023",
        "EMP": “John Smith"
    },
    "options":
    {
        "output-mode": "binary",
        "formatting":  {
            "tables": {
                "enable-cells-auto-merge":  true
            }
        }
    }
}'
-o  first_report.docx
```

7.4 Receiving the first response

7.4.1 The format of the returned response

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

(continued on next page)

(continued from previous page)

```
}
```

7.4.2 Example answer

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

7.5 Completion

As a result, a base64-encoded DOCX file of the first report document or an error message is received.

CHAPTER 8

Appendix

8.1 Quick installation

```
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/5.0.0.3/deb/xsquare.xreports_5.0.0.3.deb
dpkg -i xsquare.xreports_5.0.0.3.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
```