Version 2025.2.19

Service solutions
Programmer's Guide

Introduction

Any code snippets in this documentation are demos only. These examples serve no other purpose than to illustrate the use of FastReport Cloud, FastReport Corporate Server, and FastReport Publisher. Fast Reports Inc. does not warrant the accuracy or completeness of these examples and shall not be liable for any direct or indirect damages that may result from them.

The documentation may contain hyperlinks to various Internet resources. Such links are relevant at the time of publishing the documentation. Fast Reports Inc. shall not be liable for their availability at the time of reading the document or any damage caused by them.

Guides

This section is dedicated to guides for working and integrating FastReport Cloud, FastReport Corporate Server, and FastReport Publisher into the application. At the moment the guides are available for:

- REST API
- C#/.NET

The guides for use with other programming languages are in the plans.

Contact us if you are interested in a specific programming language or platform. Contact!

Development tools

The SDKs are available for the following languages:

- Haskell
- JavaScript
- C++
- Python
- Java
- Go
- C#/.NET and ASP.NET

Please note, that the development tools are posted as sources on GitHub. To use them, you will need to create a package or library from the sources yourself.

REST API

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This section provides step-by-step instructions and guidelines for performing typical tasks for using Service solutions without being bound to a specific programming language.

Getting started

You will need the following tools and facilities:

1. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

2. FastReport Community Designer.

   In some guides, you will need the designer to create the report.

3. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

4. Internet access.

> Please note: These guides assume that you already know how to use curl or another REST client.

> Note. The items above describe the recommended tools.

What next?

We advise you to start reading the following articles:

- Status codes.
- Authentication and authorization.

Status codes

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article lists all status codes that can be returned by FastReport Cloud, FastReport Corporate Server, and FastReport Publisher services.

Request Successfully Processed — 2xx

- 200 OK - the request was processed successfully. It is used in most cases when the response body is returned along with the result.
- 201 Created - the request was processed successfully. As a result of its execution, a new entity was created.
- 204 No Content - the request was processed successfully. An empty response body is returned.

User Error — 4xx

- 400 Bad Request - the request contains errors, for example:
  - There is no parameter required to run the request.
  - The entity `Id` format is different from Hex 24.
  - The negative skip or take parameter is in the request.
- 401 Unauthorized - the request was sent by an unauthorized user.
- 402 Payment Required - the requested resource is limited by subscription plan.
- 403 Forbidden - the requested resource is found, but the user does not have permission to perform the action.
- 404 Not Found - the requested resource is not found.
- 413 Request Entity Too Large - request body exceeds the limits in the service settings.

Server Error — 5xx

- 500 Internal Server Error - the request has passed all validity checks, but an Exception occurred during its execution. In this case, please inform us by emailing info@fast-report.com

Other

Intermediate services between Service solution and the user may return other status codes.

Authentication and authorization

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

The process of user data authentication and granting user certain rights in Service solutions is carried out in one of two available ways:

1. Via JWT token.

   In this case, authentication must be done manually, and the token will only be valid for 5 minutes, during which time the user must log in to their application. When connecting to the server, the browser will redirect to the authentication server and then generate an access token. We restrict the possibility of retrieving the JWT token to the user personally from a security point of view.

   > If the user has not logged in to the application within 5 minutes, the authentication must be re-authenticated. If the user has logged in, re-authentication is not required.

2. Via API key.

   In this case, the obtaining of access rights is performed for server applications. To get an API key, a user must be present. However, the key itself can be valid for a long time, for example, a year.

Retrieve the first API key

To get the first API key, access the user panel. If for some reason you do not have access to the user panel, you can request a key as described below.

1. Open the link in your browser: <{host_name}/account/signin?r={host_name}/api/manage/v1/ApiKeys>.

   If you click on this link, it will direct you to the automatic browser authentication process.

2. Now when authentication has passed, you need to request a new key.

   Press `F12` or `Ctrl+Shift+I` to open the developer panel. The key combinations may differ from the default, in that case open the developer panel via the browser menu

3. Copy and run the code in the JavaScript console.

   This code will make a `POST` request to the URL `{host_name}/api/manage/v1/ApiKeys` to create a new access key valid till 2030.

4. Refresh the browser page and get the result.

```
{
  "apiKeys": [
    {
      "value": "cc355oeu1z5d5wncayo33me6c1g5junqdqk4pkupid7t8ynjshey",
      "description": "Generated by js develop panel",
      "expired": "2030-01-01T07:41:23.399Z"
    }
  ],
  "count": 1
}
```

You can also create a key by opening the Api keys tab.

Now you can use the API key. In the example above, `cc355oeu1z5d5wncayo33me6c1g5junqdqk4pkupid7t8ynjshey` was used.

There is no need to retrieve a new API key through the browser again.

How to use the API key

The key should be passed with each request in the header `Authorization: Basic`. Use `apikey` as the username and the key value as the password. For example:

```
Authorization: Basic Base64Encode(apikey:cc355oeu1z5d5wncayo33me6c1g5junqdqk4pkupid7t8ynjshey);
```

Where `Base64Encode` is a function for converting a string to `base64` when using `UTF8` encoding.

Retrieve new API key

To get the new key, execute the `POST` request to the `{host_name}/api/manage/v1/ApiKeys` entry point and pass JSON in the request body according to the scheme below.

```
{
   "description": "string",
   "expired": "string($date-time)"
}
```

Example request:

```
curl -X POST "{host_name}/api/manage/v1/ApiKeys" -H "accept: text/plain" -H "authorization: Basic YXBpa2V5OmNjMzU1b2V1MXo1ZDV3bmNheW8zM21lNmMxZzVqdW5xZHFrNHBrdXBpZDd0OHluanNoZXk=" -H "Content-Type: application/json-patch+json" -d "{ \"description\": \"Generated by js develop panel\", \"expired\": \"2030-01-01T07:41:23.399Z\"}"
```

Response scheme:

```
{
  "value": "string",
  "description": "string",
  "expired": "2020-12-02T08:47:43.270Z"
}
```

You can also get a new key through the user panel.

What next?

- Upload new template.
- Work with groups.
- Add new users to the workspace.

Upload new template

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

One of the main features of Service solutions is storing templates, reports, and other data in the cloud. This article will describe how to upload your own template to service solution template storage.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Report template.

   It can be created in the free FastReport Community Designer program.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

5. Internet access.

Instruction

1. You need to get the root folder ID of the workspace. This identifier will never change, so you can save it and you do not need to request the ID every time before uploading a new template.

   To request the root directory, execute the `GET` request to `{host_name}/api/rp/v1/Templates/Root`.

   ```
   curl -X GET "{host_name}/api/rp/v1/Templates/Root" -H "accept: text/plain"
   ```

   In this case, the directory for the default workspace will be returned. You can specify a particular workspace by passing the `subscriptionId` parameter.

   ```
   curl -X GET "{host_name}/api/rp/v1/Templates/Root?subscriptionId=your_workspace_id" -H "accept: text/plain"
   ```

   Example answer:

```
{
  "name": "RootFolder",
  "parentId": null,
  "tags": [],
  "icon": "",
  "type": "Folder",
  "size": 16384,
  "subscriptionId": "5fa919fa292a8300019349bc",
  "status": "None",
  "id": "5fa919f9292a8300019349b9",
  "createdTime": "2020-11-09T10:29:13.928Z",
  "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
  "editedTime": "2020-11-13T15:58:45.69Z",
  "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
}
```


Directory ID from the example above: `5fa919f9292a8300019349b9`.

2. To upload the desired template, execute the `POST` request to `{host_name}/api/rp/v1/Templates/Folder/{id}/File`, where the directory ID should be substituted for `{id}` and JSON should be passed in the request body according to the scheme below.

```
{
  "name": "string",
  "content": "base64encoded(template.frx)"
}
```

   - `name` — name of the file to be displayed in the user panel.

     Add the `.frx` extension to the file name manually.

   - `content` — file content encoded in base64.

   Example request:

```
curl -X POST "{host_name}/api/rp/v1/Templates/Folder/5fa919f9292a8300019349b9/File" -H "accept: text/plain" -H
"Content-Type: application/json-patch+json" -d "{ \"name\": \"template.frx\", \"content\":
\"77u/PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4NCjxSZXBvcnQgU2NyaXB0TGFuZ3VhZ2U9IkNTaGF
ycCIgUmVwb3J0SW5mby5DcmVhdGVkPSIxMi8wNC8yMDIwIDEwOjU4OjU3IiBSZXBvcnRJbmZvLk1vZGlmaWVkPSIxMi8w
NC8yMDIwIDExOjAwOjIwIiBSZXBvcnRJbmZvLkNyZWF0b3JWZXJzaW9uPSIyMC4yMC40LjEiPg0KICA8RGljdGlvbmFyeS8+D
QogIDxSZXBvcnRQYWdlIE5hbWU9IlBhZ2UxIiBXYXRlcm1hcmsuRm9udD0iQXJpYWwsIDYwcHQiPg0KICAgIDxSZXBvcnRU
aXRsZUJhbmQgTmFtZT0iUmVwb3J0VGl0bGUxIiBXaWR0aD0iNzE4LjIiIEhlaWdodD0iMzcuOCIvPg0KICAgIDxQYWdlSGVhZ
GVyQmFuZCBOYW1lPSJQYWdlSGVhZGVyMSIgVG9wPSI0MSIgV2lkdGg9IjcxOC4yIiBIZWlnaHQ9IjI4LjM1Ii8+DQogICAgPER
hdGFCYW5kIE5hbWU9IkRhdGExIiBUb3A9IjcyLjU1IiBXaWR0aD0iNzE4LjIiIEhlaWdodD0iNzUuNiI+DQogICAgICA8VGV4dE9
iamVjdCBOYW1lPSJUZXh0MSIgV2lkdGg9IjcxOC4yIiBIZWlnaHQ9Ijc1LjYiIFRleHQ9IkhlbGxvLCBGYXN0UmVwb3J0IENsb3Vk
ISEhIiBIb3J6QWxpZ249IkNlbnRlciIgVmVydEFsaWduPSJDZW50ZXIiIEZvbnQ9IkFyaWFsLCAxMHB0Ii8+DQogICAgPC9EYXR
hQmFuZD4NCiAgICA8UGFnZUZvb3RlckJhbmQgTmFtZT0iUGFnZUZvb3RlcjEiIFRvcD0iMTUxLjM1IiBXaWR0aD0iNzE4LjIiIE
hlaWdodD0iMTguOSIvPg0KICA8L1JlcG9ydFBhZ2U+DQo8L1JlcG9ydD4NCg==\"}"
```

   Example answer:

```
{
 "reportInfo": {
  "author": null,
  "created": "2020-12-04T10:58:57",
  "creatorVersion": "20.20.4.1",
  "description": null,
  "modified": "2020-12-04T11:00:20",
  "name": null,
  "picture": null,
  "previewPictureRatio": 0,
  "saveMode": "All",
  "savePreviewPicture": false,
  "tag": null,
  "version": null
 },
 "name": "template.frx",
 "parentId": "5fa919f9292a8300019349b9",
 "tags": null,
 "icon": null,
 "type": "File",
 "size": 17159,
 "subscriptionId": "5fa919fa292a8300019349bc",
 "status": "Success",
 "id": "5fc9ece6b792c90001d94b13",
 "createdTime": "2020-12-04T08:01:42.7087229+00:00",
 "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
 "editedTime": "2020-12-04T08:01:42.7087112+00:00",
 "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
}
```


3. Now this template can be used to prepare (create) a report and export it.

What next?

- Access rights management by template example.
- Create report.

Add JSON data source

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article will cover how to create a new JSON-based data source in Service solutions.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. JSON file.

4. JSON scheme.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

Create a data source

To create a data source, send `CREATE` request to `{host_name}/api/data/v1/DataSources` .

Example request body:

```
{
  "name": "fakeAPI",
  "connectionString":
"Json=aHR0cHM6Ly9qc29ucGxhY2Vob2xkZXIudHlwaWNvZGUuY29tL3Bvc3Rz;JsonSchema=eyJ0eXBlIjoiYXJyYXkiLCJpdGVtcyI
6eyJ0eXBlIjoib2JqZWN0IiwicHJvcGVydGllcyI6eyJ1c2VySWQiOnsidHlwZSI6Im51bWJlciJ9LCJpZCI6eyJ0eXBlIjoibnVtYmVyIn0sInR
pdGxlIjp7InR5cGUiOiJzdHJpbmcifSwiYm9keSI6eyJ0eXBlIjoic3RyaW5nIn19fX0=;Encoding=utf-8",
  "subscriptionId": "604f52e1261a3c19104c0e25",
  "connectionType": "JSON"
}
```

Where

- `name` — data source name (it will be displayed in the designer when selected).
- `connectionString` — connection string, in case of JSON, it consists of 3 elements:
  - `Json` — JSON file or http/https link encoded in base64;
  - `JsonSchema` — schema describing JSON file structure encoded in base64;
  - `Encoding` — encoding, it should always be passed to dfg `utf-8`.
- `subscriptionId` — ID of the workspace (subscription) to which the data source will be attached.
- `connectionType` — connection type, this guide uses `JSON`.

Example request:

```
curl -X POST "{host_name}/api/data/v1/DataSources" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d "{ \"name\": \"fakeAPI\", \"connectionString\": \"Json=aHR0cHM6Ly9qc29ucGxhY2Vob2xkZXIudHlwaWNvZGUuY29tL3Bvc3Rz;JsonSchema=eyJ0eXBlIjoiYXJyYXkiLCJpdGVtcyI6eyJ0eXBlIjoib2JqZWN0IiwicHJvcGVydGllcyI6eyJ1c2VySWQiOnsidHlwZSI6Im51bWJlciJ9LCJpZCI6eyJ0eXBlIjoibnVtYmVyIn0sInRpdGxlIjp7InR5cGUiOiJzdHJpbmcifSwiYm9keSI6eyJ0eXBlIjoic3RyaW5nIn19fX0=;Encoding=utf-8\", \"subscriptionId\": \"604f52e1261a3c19104c0e25\", \"connectionType\": \"JSON\"}"
```

Example answer:


```
{
  "id": "60648953db44d83f9c6da98f",
  "name": "fakeAPI",
  "connectionType": "JSON",
  "connectionString":
"Json=aHR0cHM6Ly9qc29ucGxhY2Vob2xkZXIudHlwaWNvZGUuY29tL3Bvc3Rz;JsonSchema=eyJ0eXBlIjoiYXJyYXkiLCJpdGVtcyI6eyJ0eXBlIjoib2JqZWN0IiwicHJvcGVydGllcyI6eyJ1c2VySWQiOnsidHlwZSI6Im51bWJlciJ9LCJpZCI6eyJ0eXBlIjoibnVtYmVyIn0sInRpdGxlIjp7InR5cGUiOiJzdHJpbmcifSwiYm9keSI6eyJ0eXBlIjoic3RyaW5nIn19fX0=;Encoding=utf-8",
  "dataStructure": "<JsonDataSourceConnection
ConnectionString=\"rijcmlqrcq6OJBTPt0pNFvRuRtDUSHSHLQy/QlNolifTTaTjsrExzdbf1ifpPblp655sducwkD1lEVzxVZF8qRuE0NT6UkyTr7kwjGItFOwh7DBsOyL6QkQY4FOZ2ki8Al2R30gpXs6nMUGg1BRwCF0rj3+QvmXbj+2t8x5RerR5y7inP1R+oCuo0wvfcTeOMfYfZrjdE3whziFh5Qn3mR7vaevmV9peDWQ3LYyK2ec3KpGVeEXSqM+10WyL4ahY7EHuQIzIZROGFGKfW50cUYwdillhKy24gNdsUzi5klG66DDQtCKEOLbNutDvA0xqCTW3MvRNORSbvckL6g3gM+cStJ5PQ2XUjF9yz9zdwmramnXI6k+MK8V9lrMkc0XFkDMHOxDlfG2jHhkFuUTgmiKp7hQMg==\">\r\n  <JsonTableDataSource Name=\"JSON\"
DataType=\"FastReport.Data.JsonConnection.JsonParser.JsonArray\" Enabled=\"true\" TableName=\"JSON\">\r\n    <Column Name=\"index\" DataType=\"System.Int32\"/>\r\n    <Column Name=\"item\" DataType=\"FastReport.JsonBase\">\r\n
<Column Name=\"userId\" DataType=\"System.Double\"/>\r\n      <Column Name=\"id\" DataType=\"System.Double\"/>\r\n
<Column Name=\"title\" DataType=\"System.String\"/>\r\n      <Column Name=\"body\" DataType=\"System.String\"/>\r\n
</Column>\r\n    <Column Name=\"array\" DataType=\"FastReport.JsonBase\"/>\r\n
</JsonTableDataSource>\r\n</JsonDataSourceConnection>\r\n",
  "subscriptionId": "604f52e1261a3c19104c0e25",
  "editedTime": "2021-03-31T14:38:10.5792982Z",
  "editorUserId": "2df79f83-07f1-41ba-96b5-7757bbf377df",
  "createdTime": "0001-01-01T00:00:00",
  "creatorUserId": "2df79f83-07f1-41ba-96b5-7757bbf377df",
  "isConnected": true
}
```

Access rights management by template example

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

Restricting access to private resources is a very important feature of Service solutions. The flexible access system allows you to restrict or grant rights to each resource separately, specifying the range of people who can access it

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Report template.

   To learn how to upload a report template, refer to the Upload a new template article.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

5. Internet access.

Instruction

1. To view the current rights to the resource, execute the `GET` request to `{host_name}/api/rp/v1/Templates/File/{id}/permissions`, where the template ID should be used instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/api/rp/v1/Templates/File/5fc9ece6b792c90001d94b13/permissions" -H "accept: text/plain"
   ```

   Example answer:

```
{
 "permissions": {
  "ownerId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
  "owner": {
   "create": "All",
   "delete": "All",
   "execute": "All",
   "get": "All",
   "update": "All",
   "administrate": "All"
  },
  "groups": null,
  "other": {
   "create": "All",
   "delete": "All",
   "execute": "All",
   "get": "All",
   "update": "All",
   "administrate": "All"
  },
  "anon": null
 }
}
```

In this example, the owner and other members of the workspace have full access to the report template.

2. To change the permissions, execute the `POST` request to `{host_name}/api/rp/v1/Templates/File/{id}/permissions`, where the template ID should be used instead of `{id}`.

   Example model:

```
{
 "newPermissions": {
  "ownerId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
  "owner": {
   "create": -1,
   "delete": -1,
   "execute": -1,
   "get": -1,
   "update": 0,
   "administrate": -1
  },
  "other": {
   "create": 0,
   "delete": 0,
   "execute": 0,
   "get": 0,
   "update": 0,
   "administrate": 0
  }
 },
 "administrate": "Owner,Other"
}
```

   Important! In the request to update the rights in the `administrate` property, the values `Owner` and `Other` were passed in, in this case, only the specified rights categories, i.e., `other` and `owner` will be updated, and the categories `anon` and `groups` will not be changed.

   Example request that restricts the owner’s right to update the report (e.g., editing in the online designer), as well as all workspace participants’ rights to the file:

```
curl -X POST "{host_name}/api/rp/v1/Templates/File/5fc9ece6b792c90001d94b13/permissions" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d "{ \"newPermissions\": { \"ownerId\": \"5af5a8dc-8cb0-40f9-ac99-ca2533fa4491\", \"owner\": { \"create\": -1, \"delete\": -1, \"execute\": -1, \"get\": -1, \"update\": 0, \"administrate\": -1 }, \"other\": { \"create\": 0, \"delete\": 0, \"execute\": 0, \"get\": 0, \"update\": 0, \"administrate\": 0 } }, \"administrate\": \"Owner,Other\"}
```

Example answer:

```
200 OK
```

3. Now you can request the permissions again to make sure they are indeed updated.

   Example request:

```
curl -X GET "{host_name}/api/rp/v1/Templates/File/5fc9ece6b792c90001d94b13/permissions" -H "accept: text/plain"
```

   Example answer:

```
{
 "permissions": {
   "ownerId": "2df79f83-07f1-41ba-96b5-7757bbf377df",
 "owner": {
     "create": "All",
     "delete": "All",
     "execute": "All",
  "get": "All",
     "update": "None",
  "administrate": "All"
   },
   "groups": null,
   "other": {
     "create": "None",
     "delete": "None",
     "execute": "None",
     "get": "None",
     "update": "None",
     "administrate": "None"
   },
   "anon": null
 }
}
```

What next?

- Create report.
- Work with groups.
- Add new users to the workspace.

Create report

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article describes the process of creating a report from a template using the Service solutions report processor.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Report template.

   To learn how to upload a report template, refer to the Upload a new template article.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

5. Internet access.

Instruction

1. To create a template, you need its ID. Execute `GET` request to `{host_name}/api/rp/v1/Templates/Root` to retrieve the root directory.

   Example request:

   ```
   curl -X GET "{host_name}/api/rp/v1/Templates/Root" -H "accept: text/plain"
   ```

   Example answer:

   ```
   {
    "name": "RootFolder",
    "parentId": null,
    "tags": [],
    "icon": "",
    "type": "Folder",
    "size": 16384,
    "subscriptionId": "5fa919fa292a8300019349bc",
    "status": "None",
    "id": "5fa919f9292a8300019349b9",
    "createdTime": "2020-11-09T10:29:13.928Z",
    "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
    "editedTime": "2020-11-13T15:58:45.69Z",
    "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
   }
   ```

   Directory ID from the example above: `5fa919f9292a8300019349b9`.

2. To get a list of files in the directory, execute `GET` request to `{host_name}/api/rp/v1/Templates/Folder/{id}/ListFiles?skip=0&take=10`, where the directory ID should be used

instead of `{id}`.

Example request:

```
curl -X GET "{host_name}/api/rp/v1/Templates/Folder/5fa919f9292a8300019349b9/ListFiles?skip=0&take=10" -H "accept: text/plain"
```

Example answer:


```
[
  {
    "reportInfo": {
      "author": null,
      "created": "2020-12-04T10:58:57Z",
      "creatorVersion": "20.20.4.1",
      "description": null,
      "modified": "2020-12-04T11:00:20Z",
      "name": null,
      "picture": null,
      "previewPictureRatio": 0,
      "saveMode": "All",
      "savePreviewPicture": false,
      "tag": null,
      "version": null
    },
    "name": "template.frx",
    "parentId": "5fa919f9292a8300019349b9",
    "tags": null,
    "icon": null,
    "type": "File",
    "size": 17159,
    "subscriptionId": "5fa919fa292a8300019349bc",
    "status": "Success",
    "id": "5fc9ece6b792c90001d94b13",
    "createdTime": "2020-12-04T08:01:42.708Z",
    "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
    "editedTime": "2020-12-04T08:01:42.708Z",
    "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
  }
]
```


Template ID from the example above: `5fc9ece6b792c90001d94b13`.

3. To create a report, you will need a directory to store the report in. Request the root report directory by executing `GET` request to `{host_name}/api/rp/v1/Reports/Root`.

   Example request:

   ```
   curl -X GET "{host_name}/api/rp/v1/Reports/Root" -H "accept: text/plain"
   ```

   Example answer:

```
{
 "name": "RootFolder",
 "parentId": null,
 "tags": null,
 "icon": null,
 "type": "Folder",
 "size": 16384,
 "subscriptionId": "5fa919fa292a8300019349bc",
 "status": "None",
 "id": "5fa919f9292a8300019349ba",
 "createdTime": "2020-11-09T10:29:13.993Z",
 "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
 "editedTime": "0001-01-01T00:00:00Z",
 "editorUserId": null
}
```

Directory ID from the example above: `5fa919f9292a8300019349ba`.

4. To create a report, execute the `POST` request to `{host_name}/api/rp/v1/Templates/File/{id}/Prepare`, where the template ID should be used instead of `{id}`.

   In the request body, pass JSON according to the scheme below.

```
{
 "name": "string",
 "folderId": "string",
 "reportParameters": {
   "additionalProp1": "string",
   "additionalProp2": "string",
   "additionalProp3": "string"
 }
}
```

   - `folderId` — ID of the directory where the report will be placed.
   - `name` — name of the resulting file. Add `.fpx` extension manually.
   - `reportParameters` — parameters that are specified in the report itself via the report designer, they are not needed in this example.

> If you do not specify folderId, the generated report will be saved to the root folder.

   Example request:

```
curl -X POST "{host_name}/api/rp/v1/Templates/File/5fc9ece6b792c90001d94b13/Prepare" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d "{ \"name\": \"awesome_report.fpx\", \"folderId\": \"5fa919f9292a8300019349ba\"}"
```

   Example answer:

```
{
 "templateId": "5fc9ece6b792c90001d94b13",
 "reportInfo": null,
 "name": "awesome_report.fpx",
 "parentId": "5fa919f9292a8300019349ba",
 "tags": null,
 "icon": null,
 "type": "File",
 "size": 16384,
 "subscriptionId": "5fa919fa292a8300019349bc",
 "status": "InQueue",
 "id": "5fe4614bcd7c55000148e4c6",
 "createdTime": "2020-12-24T09:37:15.7169531+00:00",
 "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
 "editedTime": "2020-12-24T09:37:15.7169582+00:00",
 "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
}
```

Pay attention to the status of the `InQueue` file, it means that the task for generation has been created and then it has been added to the builder's queue. Already at this stage the report has received its ID for work: `5fe4614bcd7c55000148e4c6`.

You should wait a while for the report to be generated. You can call the method of getting the report in a loop every few milliseconds and check the report status.

5. For information about the report, execute the `GET` request to `{host_name}/api/rp/v1/Reports/File/{id}`, where the report ID should be used instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/api/rp/v1/Reports/File/5fe4614bcd7c55000148e4c6" -H "accept: text/plain"
   ```

   Example answer:

   ```
   {
    "templateId": "5fc9ece6b792c90001d94b13",
    "reportInfo": null,
    "name": "awesome_report.fpx",
    "parentId": "5fa919f9292a8300019349ba",
    "tags": null,
    "icon": null,
    "type": "File",
    "size": 16927,
    "subscriptionId": "5fa919fa292a8300019349bc",
    "status": "Success",
    "id": "5fe4614bcd7c55000148e4c6",
    "createdTime": "2020-12-24T09:37:15.716Z",
    "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
    "editedTime": "2020-12-24T09:37:15.716Z",
    "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
   }
   ```

   Note the status of the `Success` file. The report was successfully generated.

6. To download the report, execute the `GET` request to `{host_name}/download/r/{id}`, where the report ID should be passed instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/download/r/5fe4614bcd7c55000148e4c6" -H "accept: text/plain"
   ```

   The file will be received in the response.

What next?

- Export report to PDF.

> Note! In the examples there may be no Authorization header because a cookie-based authentication model is used. For more information about authorization, read the Authentication and authorization section.

Report parameters

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of passing parameters to a report, which are a dictionary of parameters provided when generating a report. More information on this can be found in the FastReport .NET guide in the Report parameters section.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Report template.

   It can be created in the free FastReport Community Designer program.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

5. Internet access.

Pass parameters to the report

Parameters can be passed when:

- preparing a report;
- exporting from a template;
- working with tasks.

Let’s consider parameter passing on the example of report generation. Detailed instructions on how to create a report can be found in the Create report section.

To create a report, execute the `POST` request to the `{host_name}/api/rp/v1/Templates/File/{id}/Prepare`, where the template ID should be used instead of `{id}`.

In the request body, pass JSON according to the scheme below.

```
{
  "name": "string",
  "folderId": "string",
  "reportParameters": {
    "additionalProp1": "string",
    "additionalProp2": "string"
  }
}
```

- `folderId` — ID of the directory where the report will be placed.
- `name` — name of the resulting file. Add `.fpx` extension manually.
- `reportParameters` — parameters that are specified in the report itself using the report designer.

> If you do not specify folderId, the generated report will be saved to the root folder.

Example request:

```
curl -X POST "{host_name}/api/rp/v1/Templates/File/5fc9ece6b792c90001d94b13/Prepare"
-H "accept: text/plain"
-H "Content-Type: application/json-patch+json"
-d "{
 "name": "awesome_report.fpx",
 "folderId": "5fa919f9292a8300019349ba",
 "reportParameters": {
   "Parameter1": "Value1",
   "Parameter2": "Value2"
   }
 }"
```

Example answer:

```
{
  "templateId": "5fc9ece6b792c90001d94b13",
  "name": "awesome_report.fpx",
  "parentId": "5fa919f9292a8300019349ba",
  "type": "File",
  "size": 16384,
  "subscriptionId": "5fa919fa292a8300019349bc",
  "status": "InQueue",
  "statusReason": "None",
  "id": "5fe4614bcd7c55000148e4c6",
  "createdTime": "2020-12-24T09:37:15.7169531+00:00",
  "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
  "editedTime": "2020-12-24T09:37:15.7169582+00:00",
  "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
}
```

Export report to PDF

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of exporting a report using the Service solutions report processor.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions. How to retrieve and use API key you can learn in the Authentication and authorization article.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Report template.

   To learn how to upload a report template, refer to the Upload a new template article.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

5. Internet access.

Annotation

Note, that you can export the report directly from the template without saving the report. To do this, run the same commands for the report template, replacing `Report` in the query strings with `Template`, and use the template ID rather than the report ID.

Instruction

1. You will need the report ID to export to PDF. Execute the `GET` request to `{host_name}/api/rp/v1/Report/Root` to retrieve the root directory.

   Example request:

   ```
   curl -X GET "{host_name}/api/rp/v1/Reports/Root" -H "accept: text/plain"
   ```

   Example answer:

```
{
  "name": "RootFolder",
  "parentId": null,
  "tags": null,
  "icon": null,
  "type": "Folder",
  "size": 16384,
  "subscriptionId": "5fa919fa292a8300019349bc",
  "status": "None",
  "id": "5fa919f9292a8300019349ba",
  "createdTime": "2020-11-09T10:29:13.993Z",
  "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
  "editedTime": "0001-01-01T00:00:00Z",
  "editorUserId": null
}
```

Directory ID from the example above: `5fa919f9292a8300019349ba`.

2. Retrieve a list of the files in the directory by executing `GET` request to `{host_name}/api/rp/v1/Reports/Folder/{id}/ListFiles?skip=0&take=10`, where the directory ID should be used instead of `{id}`.

   Example request:

```
curl -X GET "{host_name}/api/rp/v1/Reports/Folder/5fa919f9292a8300019349ba/ListFiles?skip=0&take=10" -H "accept: text/plain"
```

   Example answer:

```
[
  {
    "templateId": "5fc9ece6b792c90001d94b13",
    "reportInfo": null,
    "name": "awesome_report.fpx",
    "parentId": "5fa919f9292a8300019349ba",
    "tags": null,
    "icon": null,
    "type": "File",
    "size": 16927,
    "subscriptionId": "5fa919fa292a8300019349bc",
    "status": "Success",
    "id": "5fe4614bcd7c55000148e4c6",
    "createdTime": "2020-12-24T09:37:15.716Z",
    "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
    "editedTime": "2020-12-24T09:37:15.716Z",
    "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
  }
]
```

   Report ID from the example above: `5fe4614bcd7c55000148e4c6`.

3. To export the report, you will need a directory in which to place the export file.

   Retrieve the root directory of the exports; to do this, execute `GET` request to `{host_name}/api/rp/v1/Exports/Root`.

   Example request:

```
curl -X GET "{host_name}/api/rp/v1/Exports/Root" -H "accept: text/plain"
```

   Example answer:

```
{
 "name": "RootFolder",
 "parentId": null,
 "tags": null,
 "icon": null,
 "type": "Folder",
 "size": 16384,
 "subscriptionId": "5fa919fa292a8300019349bc",
 "status": "None",
 "id": "5fa919fa292a8300019349bb",
 "createdTime": "2020-11-09T10:29:14.002Z",
 "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
 "editedTime": "0001-01-01T00:00:00Z",
 "editorUserId": null
}
```

Directory ID from the example above: `5fa919fa292a8300019349bb`.

4. To export the report, execute the `POST` request to `{host_name}/api/rp/v1/Reports/File/{id}/Export`, where report ID should be used instead of `{id}`.

   In the request body, pass JSON according to the scheme below.

```
{
 "fileName": "awesome_result.pdf",
 "folderId": "5fa919fa292a8300019349bb",
 "locale": "en-GB",
 "format": "Pdf",
 "exportParameters": {
  "additionalProp1": {},
  "additionalProp2": {},
  "additionalProp3": {}
 }
}
```

   - `folderId` — ID of the directory where the export will be placed. If left blank, the export will be placed in the root folder for exports in the workspace.
   - `fileName` — name of the resulting file. If you do not specify the extension, or specify it incorrectly, the server will replace it by itself.
   - `locale` — localization of the exported report. This option will change the date and number formats to those corresponding to the selected ISO code (e.g., French (Switzerland) looks like “fr-CH”). If you leave this field empty or specify a non-existent country, the default locale from the subscription or English (USA) will be substituted in case the default locale is not specified.
   - `format` — export format.
   - `exportParameters` — export parameters. They are set similarly to the export parameters from the FastReport .NET library. A more detailed description is stored in the user documentation in the export parameters section.

> If you do not specify folderId, the generated report will be saved to the root folder.

   Example request:

```
curl -X POST "{host_name}/api/rp/v1/Reports/File/5fe4614bcd7c55000148e4c6/Export" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d "{ \"fileName\": \"awesome_result.pdf\", \"folderId\": \"5fa919fa292a8300019349bb\", \"locale\":\"ru-RU\", \"format\": \"Pdf\"}"
```

   Example answer:

```
{
 "format": "Pdf",
 "reportId": "5fe4614bcd7c55000148e4c6",
 "name": "awesome_result.pdf",
 "parentId": "5fa919fa292a8300019349bb",
 "tags": null,
 "icon": null,
 "type": "File",
 "size": 16384,
 "subscriptionId": "5fa919fa292a8300019349bc",
 "status": "InQueue",
 "id": "5fe46a33cd7c55000148e4c7",
 "createdTime": "2020-12-24T10:15:15.8039648+00:00",
 "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
 "editedTime": "2020-12-24T10:15:15.8039697+00:00",
 "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
}
```


Please note the status of the `InQueue` file, it means that the task for export has been created and it has been added to the builder's queue. Already at this stage the file has received its ID for work `5fe46a33cd7c55000148e4c7`.

You should wait a while for the export to be completed.

5. For information about the file, execute the `GET` request to `{host_name}/api/rp/v1/Exports/File/{id}`, where the export ID should be used instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/api/rp/v1/Exports/File/5fe46a33cd7c55000148e4c7" -H "accept: text/plain"
   ```

   Example answer:


   ```
   {
    "format": "Pdf",
    "reportId": "5fe4614bcd7c55000148e4c6",
    "name": "awesome_result.pdf",
    "parentId": "5fa919fa292a8300019349bb",
    "tags": null,
    "icon": null,
    "type": "File",
    "size": 41142,
    "subscriptionId": "5fa919fa292a8300019349bc",
    "status": "Success",
    "id": "5fe46a33cd7c55000148e4c7",
    "createdTime": "2020-12-24T10:15:15.803Z",
    "creatorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
    "editedTime": "2020-12-24T10:15:15.803Z",
    "editorUserId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
   }
   ```


   Note the status of the `Success` file. The report was successfully exported.

6. To download a report, execute the `GET` request to `{host_name}/download/e/{id}`, where the report ID should be passed instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/download/e/5fe46a33cd7c55000148e4c7" -H "accept: text/plain"
   ```

   The file will be received in the response.

What next?

- Work with groups.
- Add new users to the workspace.

> Note! In the examples there may be no Authorization header because a cookie-based authentication model is used. For more information about authorization, read the Authentication and authorization section.

Add new users to the workspace

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of adding a new user to a workspace, retrieving the list of users in the workspace, and removing a user from it.

> Each workspace always has a subscription associated with it. They have a common ID.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher. The subscription must have at least two user slots.

4. Internet access.

Instruction

There are two ways to add a user to a workspace:

- Directly by specifying the user ID.
- Using the invitation system.

Add a user via invitation

1. To create an invitation, you need a workspace ID.

   Retrieve the workspace ID by executing `GET` request to `{host_name}/api/manage/v1/Subscriptions?skip=0&take=10`.

   Example request:

   ```
   curl -X GET "{host_name}/api/manage/v1/Subscriptions?skip=0&take=10" -H "accept: text/plain"
   ```

   Example answer:

```
{
 "subscriptions": [
  {
   "id": "604f52e1261a3c19104c0e25",
   "name": "iwantpizza",
   "locale": null,
   "current": {
    "startTime": "2021-03-15T12:28:17.773Z",
    "endTime": "2121-03-15T12:28:17.773Z",
    "plan": {
     "id": "5f89b4d722d2d823440b6d10",
     "isActive": false,
     "displayName": "unlimited",
     "timePeriodType": "Year",
     "timePeriod": 100,
     "readonlyTimeLimitType": "Second",
     "readonlyTimeLimit": 0,
     "templatesSpaceLimit": 1048576000,
     "reportsSpaceLimit": 1048576000,
     "exportsSpaceLimit": 1048576000,
     "fileUploadSizeLimit": 1073741824,
     "dataSourceLimit": 10,
     "maxUsersCount": 10,
     "groupLimit": 5,
     "onlineDesigner": true,
     "isDemo": false,
     "urlToBuy": "https://www.fast-report.com/",
     "unlimitedPage": true,
     "pageLimit": 0
    }
   },
   "old": null,
   "invites": null,
   "templatesFolder": {
    "folderId": "604f52e1261a3c19104c0e22",
    "bytesUsed": 241247
   },
   "reportsFolder": {
    "folderId": "604f52e1261a3c19104c0e23",
    "bytesUsed": 16384
   },
   "exportsFolder": {
    "folderId": "604f52e1261a3c19104c0e24",
    "bytesUsed": 8059419
   }
  }
 ],
 "count": 1,
 "skip": 0,
 "take": 10
}
```


Workspace (subscription) ID from the example above: `604f52e1261a3c19104c0e25`.

2. To create an invitation, execute the `POST` request to `{host_name}/api/manage/v1/Subscriptions/{subscriptionId}/invite`, where the workspace ID will be specified instead of `{subscriptionId}`.

   By specifying the invitation properties in the request body.

   Example body:

```
{
 "usages": 1,
 "durable": true,
 "expiredDate": "2021-03-25T11:53:29.351Z"
}
```

- `usages` - possible number of uses;
- `durable` - if it is set to true, it indicates that the number of uses is not limited;
- `expiredDate` - it indicates the expiration date of the invitation (the invitation without this field is valid forever);

  If you leave the body empty, it will create a one-time invitation valid until the next day.

Example request:

```
curl -X POST "{host_name}/api/manage/v1/Subscriptions/604f52e1261a3c19104c0e25/invite" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d "{ \"usages\": 1, \"durable\": true, \"expiredDate\": \"2021-03-25T11:53:29.351Z\"}"
```

Example answer:

```
{
 "id": "604f52e1261a3c19104c0e25",
 "name": "iwantpizza",
 "locale": null,
 "current": {
  "startTime": "2021-03-15T12:28:17.773Z",
  "endTime": "2121-03-15T12:28:17.773Z",
  "plan": {
   "id": "5f89b4d722d2d823440b6d10",
   "isActive": false,
   "displayName": "unlimited",
   "timePeriodType": "Year",
   "timePeriod": 100,
   "readonlyTimeLimitType": "Second",
   "readonlyTimeLimit": 0,
   "templatesSpaceLimit": 1048576000,
   "reportsSpaceLimit": 1048576000,
   "exportsSpaceLimit": 1048576000,
   "fileUploadSizeLimit": 1073741824,
   "dataSourceLimit": 10,
   "maxUsersCount": 10,
   "groupLimit": 5,
   "onlineDesigner": true,
   "isDemo": false,
   "urlToBuy": "https://www.fast-report.com",
   "unlimitedPage": true,
   "pageLimit": 0
  }
 },
 "old": null,
 "invites": [
  {
   "usages": 1,
   "durable": true,
   "accessToken": "fj3534g341ir7dyytfiaap9z1r",
   "expiredDate": "2021-03-25T11:53:29.351Z",
   "addedUsers": [],
   "creatorUserId": "2df79f83-07f1-41ba-96b5-7757bbf377df"
  }
 ],
 "templatesFolder": {
  "folderId": "604f52e1261a3c19104c0e22",
  "bytesUsed": 241247
 },
 "reportsFolder": {
  "folderId": "604f52e1261a3c19104c0e23",
  "bytesUsed": 16384
 },
 "exportsFolder": {
  "folderId": "604f52e1261a3c19104c0e24",
  "bytesUsed": 8059419
 }
}
```


The invitation with access token appeared in the subscription: `fj3534g341ir7dyytfiaap9z1r`.

3. You need to pass the link to accept the invitation (or the access token itself) to the user you want to invite (in any way you like). The final link will look like this: `{host_name}/api/manage/v1/Subscriptions/{subscriptionId}/invite/{accessToken}/accept`, where the workspace ID should be specified instead of `{subscriptionId}`, and the access token should be specified instead of `{accessToken}`.

4. Now the invited user can click on this link directly in the browser or the execute `GET` request to

`{host_name}/api/manage/v1/Subscriptions/{subscriptionId}/invite/{accessToken}/accept`, where instead of `{subscriptionId}` the workspace ID should be specified, and the access token should be specified instead of `{accessToken}`.

Example request:

```
curl -X GET "https://xn--80ab2acne.xn--e1aflibyb2b0b.xn--p1ai/api/manage/v1/Subscriptions/6051f2a06c07a10001737394/invite/to9kxrxdz4iwbfyz3pq4fktdcr/accept" -H "accept: text/plain"
```

Delete the invitation

- When an invitation runs out of uses or expires, it is not automatically deleted. You will need to do this manually by sending `DELETE` request to `{host_name}/api/manage/v1/Subscriptions/{subscriptionId}/invite/{accessToken}`.

  Example request:

  ```
  curl -X DELETE "https://xn--80ab2acne.xn--e1aflibyb2b0b.xn--p1ai/api/manage/v1/Subscriptions/6051f2a06c07a10001737394/invite/to9kxrxdz4iwbfyz3pq4fktdcr" -H "accept: text/plain"
  ```

  An empty message with the code `NO CONTENT 204` will be received in response.

Add a user directly

1. To add a new user to a workspace, a workspace ID is required.

   Retrieve the subscription ID by executing `GET` request to `{host_name}/api/manage/v1/Subscriptions?skip=0&take=10`.

   Example request:

   ```
   curl -X GET "{host_name}/api/manage/v1/Subscriptions?skip=0&take=10" -H "accept: text/plain"
   ```

   Example answer:

```
{
 "subscriptions": [
  {
   "id": "5fa919fa292a8300019349bc",
   "name": "Awesome Corp",
   "current": {
    "startTime": "2020-11-17T10:22:58.584Z",
    "endTime": "2025-11-17T10:22:58.584Z",
    "plan": {
     "id": "5f43924b0231500001225686",
     "isActive": false,
     "displayName": "The greatest power",
     "timePeriodType": "Year",
     "timePeriod": 5,
     "readonlyTimeLimitType": "Second",
     "readonlyTimeLimit": 0,
     "templatesSpaceLimit": 1048576000,
     "reportsSpaceLimit": 1048576000,
     "exportsSpaceLimit": 1048576000,
     "fileUploadSizeLimit": 1048576000000,
     "dataSourceLimit": 10,
     "maxUsersCount": 10,
     "groupLimit": 5,
     "onlineDesigner": true,
     "isDemo": false,
     "urlToBuy": "https://www.fast-report.com",
     "unlimitedPage": true,
     "pageLimit": 15
    }
   },
   "old": [],
   "templatesFolder": {
    "folderId": "5fa919f9292a8300019349b9",
    "bytesUsed": 1668491
   },
   "reportsFolder": {
    "folderId": "5fa919f9292a8300019349ba",
    "bytesUsed": 6085990
   },
   "exportsFolder": {
    "folderId": "5fa919fa292a8300019349bb",
    "bytesUsed": 8336710
   }
  }
 ],
 "count": 1,
 "skip": 0,
 "take": 10
}
```


Workspace (subscription) ID from the example above: `5fa919fa292a8300019349bc`.

2. To add a new user, execute `PUT` request to `{host_name}/api/manage/v1/Subscriptions/{subscriptionId}/users/{userId}`, where the workspace ID should be passed instead of `{subscriptionId}`, and the user ID should be passed instead of `{userId}`.

   Example request:

   ```
   curl -X PUT "{host_name}/api/manage/v1/Subscriptions/5fa919fa292a8300019349bc/users/5af5a8dc-8cb0-40f9-ac99-ca2533fa4492" -H "accept: text/plain"
   ```

   In response you will receive an empty message with the `OK 200` status code.

3. To retrieve the list of users, execute `GET` request to `{host_name}/api/manage/v1/Subscriptions/{id}/users?skip=0&take=10`, where the workspace ID should be used instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/api/manage/v1/Subscriptions/5fa919fa292a8300019349bc/users?skip=0&take=10" -H "accept: text/plain"
   ```

   Example answer:

   ```
   {
    "users": [
     {
      "userId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491"
     },
     {
      "userId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4492"
     }
    ],
    "count": 2,
    "take": 10,
    "skip": 0
   }
   ```

4. To remove a user from the workspace, execute `DELETE` request to `{host_name}/api/manage/v1/Subscriptions/{subscriptionId}/users/{userId}`, where instead of `{subscriptionId}` the workspace ID should be used, and instead of `{userId}` - user ID.

   Example request:

   ```
   curl -X DELETE "{host_name}/api/manage/v1/Subscriptions/5fa919fa292a8300019349bc/users/5af5a8dc-8cb0-40f9-ac99-ca2533fa4492" -H "accept: text/plain"
   ```

   In response you will receive an empty message with the `OK 200` status code.

What next?

- Work with groups
- Help and feedback

Work with groups

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of creating a new group, adding a user to the group, and retrieving the list of users in it.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher. The subscription must have at least two user slots.

4. Internet access.

Annotation

Important! You can add a user to a group only if the user exists in the workspace.

Important! It is possible to add a user to a group only by their user ID.

Instruction

1. To create a new group, you need a workspace ID and the name of the new group.

   Retrieve the workspace ID by executing `GET` request to `{host_name}/api/manage/v1/Subscriptions?skip=0&take=10`.

   Example request:

   ```
   curl -X GET "{host_name}/api/manage/v1/Subscriptions?skip=0&take=10" -H "accept: text/plain"
   ```

   Example answer:

```
{
 "subscriptions": [
  {
   "id": "5fa919fa292a8300019349bc",
   "name": "Awesome Corp",
   "current": {
    "startTime": "2020-11-17T10:22:58.584Z",
    "endTime": "2025-11-17T10:22:58.584Z",
    "plan": {
     "id": "5f43924b0231500001225686",
     "isActive": false,
     "displayName": "The greatest power",
     "timePeriodType": "Year",
     "timePeriod": 5,
     "readonlyTimeLimitType": "Second",
     "readonlyTimeLimit": 0,
     "templatesSpaceLimit": 1048576000,
     "reportsSpaceLimit": 1048576000,
     "exportsSpaceLimit": 1048576000,
     "fileUploadSizeLimit": 1048576000000,
     "dataSourceLimit": 10,
     "maxUsersCount": 10,
     "groupLimit": 5,
     "onlineDesigner": true,
     "isDemo": false,
     "urlToBuy": "https://www.fast-report.com/",
     "unlimitedPage": true,
     "pageLimit": 15
    }
   },
   "old": [],
   "templatesFolder": {
    "folderId": "5fa919f9292a8300019349b9",
    "bytesUsed": 1668491
   },
   "reportsFolder": {
    "folderId": "5fa919f9292a8300019349ba",
    "bytesUsed": 6085990
   },
   "exportsFolder": {
    "folderId": "5fa919fa292a8300019349bb",
    "bytesUsed": 8336710
   }
  }
 ],
 "count": 1,
 "skip": 0,
 "take": 10
}
```


Workspace (subscription) ID from the example above: `5fa919fa292a8300019349bc`.

2. To create a new group, execute the `POST` request to `{host_name}/api/manage/v1/Groups`, pass JSON into the request body according to the scheme below.

```
{
 "name": "string",
 "subscriptionId": "string id"
}
```

Example request:

```
curl -X POST "{host_name}/api/manage/v1/Groups" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d "{ \"name\": \"My first group\", \"subscriptionId\": \"5fa919fa292a8300019349bc\"}"
```

Example answer:

```
{
 "id": "5fe5d7866882ca0001760fcb",
 "name": "My first group",
 "subscriptionId": "5fa919fa292a8300019349bc"
}
```

Group ID from the example above: `5fe5d7866882ca0001760fcb`.

3. To add a new user to the group, execute `PUT` request to `{host_name}/api/manage/v1/Groups/{groupId}/Users/{userId}`, where the group ID should be specified instead of `{groupId}`, and user ID - instead of `{userId}`.

   Example request:

   ```
   curl -X PUT "{host_name}/api/manage/v1/Groups/5fe5d7866882ca0001760fcb/Users/5af5a8dc-8cb0-40f9-ac99-ca2533fa4492" -H "accept: text/plain"
   ```

   In response you will receive an empty message with the `OK 200` code.

4. To retrieve a list of users in the group, execute `GET` request to `{host_name}/api/manage/v1/Groups/{id}/Users?skip=0&take=10`, where group ID should be specified instead of `{id}`.

   Example request:

   ```
   curl -X GET "{host_name}/api/manage/v1/Groups/5fe5d7866882ca0001760fcb/Users?skip=0&take=10" -H "accept: text/plain"
   ```

   Example answer:

   ```
   {
    "users": [
      {
       "userId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4491",
       "userId": "5af5a8dc-8cb0-40f9-ac99-ca2533fa4492"
      }
    ],
    "count": 2,
    "take": 10,
    "skip": 0
   }
   ```

What next?

- Help and feedback

> Note! In the examples there may be no Authorization header because a cookie-based authentication model is used. For more information about authorization, read the Authentication and authorization section.

Work with tasks

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

Tasks in Service solutions are actions to convert and deliver documents to consumers. They are described in detail in the Tasks section.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. Curl tool.

   Any other REST client will work, but the examples will be built for curl.

3. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher. The subscription must have at least two user slots.

4. Internet access.

General Information

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the types of tasks and how to work with them.

Task Classification

Tasks in Service solutions are divided into 2 types by launch method:

1. Stored tasks;
2. Tasks executed from request body.

Stored tasks are tasks that are saved in the database and can be run later. They allow executing tasks on demand or at a scheduled time. Such tasks are displayed in the "Tasks" section of the Service solutions web interface.

Tasks executed from request body are not saved in the database and are executed immediately. They are useful for one-time requests. For example, when building a PDF document and sending it via email through the API from a third-party application. This way you can quickly add report and document creation and sending functionality to your application written in any programming language.

Stored Tasks

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

As mentioned in the previous section, Service solutions provides the ability to create tasks that are saved in the database and can be run later. These tasks are called stored tasks.

Let's look in detail at how to create, update, and run stored tasks using the REST API.

About ViewModels and Types

When working, you will pass different ViewModels in JSON format. It is important that the first field of any ViewModel should be "$t": "Name". Detailed examples will be provided below in this article.

View Models for Creating Tasks

View Models for creating transformer tasks:

- CreatePrepareTaskVM
- CreateExportTemplateTaskVM
- CreateExportReportTaskVM

View Models for creating transport tasks:

- CreateEmailTaskVM
- CreateWebhookTaskVM
- CreateFTPUploadTaskVM
- CreateS3UploadTaskVM

View Models for creating other tasks:

- CreateFetchTaskVM

View Models for Updating Properties of Stored Tasks

View Models for updating transformer tasks:

- UpdatePrepareTaskVM
- UpdateExportTemplateTaskVM
- UpdateExportReportTaskVM

View Models for updating transport tasks:

- UpdateEmailTaskVM
- UpdateWebhookTaskVM
- UpdateFTPUploadTaskVM
- UpdateS3UploadTaskVM

View Models for updating other tasks:

- UpdateFetchTaskVM

View Model for Updating Task Permissions:

- UpdateTaskPermissionsVM

Creating a Task

To create tasks, you should use the CreateTask method. It accepts a View Model for creating tasks.

Let's look at creating a report export task followed by email sending:

```
curl -X 'POST' \
 'https://{host_name}/api/tasks/v1/tasks' \
 -H 'accept: application/json' \
 -H 'Content-Type: application/json' \
 -d '{
 "$t": "CreateExportTemplateTaskVM",
 "subscriptionId": "23e0134c816935c1e11b3737",
 "name": "SendViaWebhookExport",
 "inputFile": {
  "entityId": "61e0134c816935c1e11b3787"
 },
 "transports": [
  {
     "$t": "CreateEmailTaskVM",
   "name": "Email sending task",
   "from": "admin@company.com",
   "to": ["user@company.com", "seller@company.com"],
   "subject": "Important report",
      "body": "An important report is attached to this email.",
   "EnableSsl": true,
   "username": "admin@company.com",
   "password": "parol123",
   "server": "smtp.company.com",
   "port": 587
  }
 ],
 "format": "Pdf"
}'
```

In the EntityId and FolderId fields, you should enter real object identifiers. Otherwise, the task will be terminated with an error.

> Note! If you don't specify OutputFile, it will be saved to a temporary folder. If you specify an empty OutputFile - to the root folder. If you specify a folder id - to that folder.

A list of all available ViewModels is available at the link - https://{host_name}/api/swagger/index.html

Running Tasks on Schedule

To configure task execution on a schedule, you need to specify the following fields:

- `cronExpression`
- `startsOn`
- `ends`

`cronExpression` - a string containing a cron expression that defines the task execution schedule.

`startsOn` - the date and time from which scheduled task execution begins (in user's timezone).

`ends` - an object containing the date and time until which the task will be executed on schedule (`on`) or the number of repetitions (`after`). The `on` field is specified in user's timezone. If neither the `after` nor the `on` field is specified, the task will run indefinitely.

Example of creating a task that runs on schedule:

```
curl -X 'POST' \
 'https://{host_name}/api/tasks/v1/tasks' \
 -H 'accept: application/json' \
 -H 'Content-Type: application/json' \
 -d '{
 "$t": "CreateExportTemplateTaskVM",
 "subscriptionId": "23e0134c816935c1e11b3737",
 "name": "SendViaWebhookExport",
 "inputFile": {
  "entityId": "61e0134c816935c1e11b3787"
 },
 "transportIds": [
            "68767aef184dcf225ca085e1"
         ],
 "format": "Pdf",
 "cronExpression": "*/5 * * * *",  # Run every 5 minutes
 "startsOn": "2025-10-01T00:00:00Z",  # Start from October 1, 2025
 "ends": {
  "on": "2025-12-31T23:59:59Z"  # End on December 31, 2025
 }
}'
```

In this example, the task will run every 5 minutes, starting from October 1, 2025 and ending on December 31, 2025.

To make the task execute, for example, 10 times, you need to replace the `on` field with the `after` field:

```
"cronExpression": "*/5 * * * *",  # Run every 5 minutes
 "startsOn": "2025-10-01T00:00:00Z",  # Start from October 1, 2025
 "ends": {
  "after": 10  # Execute 10 times"
 }
```

Getting Task List

```
// Getting the first ten tasks from the workspace
curl -X 'GET' \
 'https://{host_name}/api/tasks/v1/tasks?skip=0&take=10' \
 -H 'accept: application/json'
```

Editing Task

After a task is created, you can change some of its parameters.

```
// Editing task
curl --location --request PUT 'https://{host_name}/api/tasks/v1/Tasks/{task id}' \
--header 'accept: application/json' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic {base64(apikey:api-key-value)}' \
--data '{
   "$t": "UpdateExportTemplateTaskVM",
   "reportParameters": {
      "param1": "val1",
      "param2": "val2"
   },
   "transportIds" : ["{transport task id (email sending, FTP upload, etc.)}"]
}'
```

Note that when updating a task, you need to specify `transportIds` - a list of transport task identifiers (Email, FTP, Webhook, S3), not the tasks themselves. The task itself can be created in advance either separately or when

creating a transformer task. For more details on creating them, read the section Creating Transport Tasks.

Executing Task by Specified Identifier

```
curl -X 'POST' \
 'https://{host_name}/api/tasks/v1/tasks/42d134ae3130aaad37by345f/run' \
 -H 'accept: */*' \
 -d ''
```

Deleting Task from Storage

```
curl -X 'DELETE' \
  'https://{host_name}/api/tasks/v1/tasks/42d134ae3130aaad37by345f' \
  -H 'accept: */*'
```

Tasks Executed from Request Body

Products: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

As mentioned in the previous section, Service solutions provides the ability to run tasks without prior saving. To do this, you should make a POST request to the path `/api/tasks/v1/Tasks/run`, passing all the necessary data in the request.

For example, to build a PDF document and send it via FTP, you need to pass data such as the template identifier (`entityId`) or the template itself (`content`) in the `inputFile` object, as well as data for connecting to the FTP server.

Let's look in detail at how to work with such tasks using the REST API.

About ViewModels and Types

When working, you will pass different ViewModels in JSON format. It is important that the first field of any ViewModel should be "$t": "Name". Detailed examples will be provided below in this article.

View Models for running Tasks On-the-Fly (without saving)

View Models for running transformer tasks on-the-fly:

- RunPrepareTaskVM
- RunExportTemplateTaskVM
- RunExportReportTaskVM

View Models for running transport tasks on-the-fly:

- RunEmailTaskVM
- RunWebhookTaskVM
- RunFTPUploadTaskVM
- RunS3UploadTaskVM

View Models for running other tasks on-the-fly:

- RunFetchTaskVM

Running a Task on-the-Fly (from request body)

To run a task on-the-fly, you should use the RunTask method.

Let's look at running a report export task followed by saving to S3 storage:

```
 curl --location 'https://{host_name}/api/tasks/v1/tasks/run' \
--header 'accept: application/json' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic {base64(apikey:api-key-value)}' \
--data '{
  "$t": "RunExportTemplateTaskVM",
  "subscriptionId": "{workspace id}",
  "inputFile": {
    "$t": "RunInputFileVM",
    "entityId": "{template id}",
    "type" : "Template",
    "fileName": "Template.frx"
  },
  "format": "Pdf",
  "transports": [
    {
    "$t": "RunS3UploadTaskVM",
    "accessKey": "{public_access_key}",
    "secretKey": "{secret_access_key}",
 "bucketName": "storage",
 "destinationFolder": "test",
 "useAws": true,
 "enableSsl": true,
 "region": "us-east-1"
    }
  ]
}'
```

In the EntityId field, you should enter real object identifiers. Otherwise, the task will be terminated with an error.

Also, instead of it, in the `inputFile` object you can pass the template content in the `content` field as a Base64 string.

A list of all available ViewModels is available at the link - https://{host_name}/api/swagger/index.html

C# .NET

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This section provides step-by-step instructions and guides for performing typical Service solutions application tasks using the C#/.NET programming language and FastReport.Cloud.SDK.

Getting started

You will need the following tools and facilities.

1. .NET SDK.
2. C# code editor or text editor, e.g., Visual Studio Code.
3. FastReport Community Designer.

   For some guides, you will need a designer to create the report.
4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.
5. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Install and configure

To connect the SDK, you should install the `FastReport.Cloud.SDK` package of the latest stable version. This can be done by running the following command.

```
dotnet add package FastReport.Cloud.SDK
```

To add ASP.NET support, install the package `FastReport.Cloud.SDK.Web`.

```
dotnet add package FastReport.Cloud.SDK.Web
```

You should also add a line to the `Startup` class.

```
services.AddFastReportCloud();
```

What next?

We advise you to start reading the following articles:

- Status codes.
- Authentication and authorization.

Authentication and authorization

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

The process of user data authentication and granting user certain rights in Service solutions is carried out in one of two available ways:

1. Via JWT token.

   In this case, authentication must be done manually, and the token will only be valid for 5 minutes, during which time the user must log in to their application. When connecting to the server, the browser will redirect to the authentication server and then generate an access token. We restrict the possibility of retrieving the JWT token to the user personally from a security point of view.

   > If the user has not logged in to the application within 5 minutes, the authentication must be re-authenticated. If the user has logged in, re-authentication is not required.

2. Via API key.

   In this case, the obtaining of access rights is performed for server applications. To get an API key, a user must be present. However, the key itself can be valid for a long time, for example, a year.

Retrieve the first API key

To get the first API key, access the user panel. If for some reason you do not have access to the user panel, you can request a key as described below.

The easiest way to retrieve a key is to open the Api keys tab and create one on this page.

Option 2:

1. Open the link in your browser: <{host_name}/account/signin?r={host_name}/api/manage/v1/ApiKeys>.

   If you click on this link, it will direct you to the automatic browser authentication process.

2. Now when authentication has passed, you need to request a new key.

   Press `F12` or `Ctrl+Shift+I` to open the developer panel. The key combinations may differ from the default, in that case open the developer panel via the browser menu

3. Copy and run the code in the JavaScript console.

   This code will make a `POST` request to the URL `{host_name}/api/manage/v1/ApiKeys` to create a new access key valid till 2030.

4. Refresh the browser page and get the result.

```
{
  "apiKeys": [
    {
      "value": "cc355oeu1z5d5wncayo33me6c1g5junqdqk4pkupid7t8ynjshey",
      "description": "Generated by js develop panel",
      "expired": "2030-01-01T07:41:23.399Z"
    }
  ],
  "count": 1
}
```

Now you can use the API key. In the example above, `cc355oeu1z5d5wncayo33me6c1g5junqdqk4pkupid7t8ynjshey` was used.

There is no need to retrieve a new API key through the browser again.

How to use API key

The key should be passed with each request in the `Authorization: Basic`. You should use apikey as the username and the key value as the password. For example:

```
Authorization: Basic Base64Encode(apikey:cc355oeu1z5d5wncayo33me6c1g5junqdqk4pkupid7t8ynjshey);
```

Where `Base64Encode` is a function to encode a string into base64.

For `FastReport.Cloud.SDK` there is a special class that allows you to add a key to the request header <xref:FastReport.Cloud.FastReportCloudApiKeyHeader>.

To add the necessary header, create a new `HttpClient`.

```
HttpClient httpClient = new HttpClient();
httpClient.BaseAddress = new Uri({host_name});
httpClient.DefaultRequestHeaders.Authorization = new FastReportCloudApiKeyHeader(apiKey);
```

Now this `HttpClient` can be used for all requests.

Retrieve new API key

To retrieve a new key, call the following method <xref:FastReport.Cloud.Management.ApiKeysClient.CreateApiKeyAsync(FastReport.Cloud.CreateApiKeyVM)>

```
CreateApiKeyVM model = new CreateApiKeyVM()
{
   Description = "Created by FastReport.Cloud.SDK",
   Expired = DateTime.UtcNow.AddYears(1)
};

IApiKeysClient apiKeysClient = new ApiKeysClient(httpClient);
await apiKeysClient.CreateApiKeyAsync(model);
```

> Whenever possible, use asynchronous method analogs instead of synchronous methods.

This function will result in a <xref:FastReport.Cloud.ApiKeyVM> model.

What next?

- Upload new template.
- Work with groups.
- Add new users to the workspace.

Upload new template

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

One of the main features of Service solutions is storing templates, reports, and other data in the cloud. This article will describe how to upload your own template to service solution template storage.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Report template.

   It can be generated in the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Instruction

1. You need to get the root folder ID of the workspace. This ID will never change, so you can save it and not have to request it each time before loading a new template.

   To request the root directory, call the method <xref:FastReport.Cloud.ITemplateFoldersClient.GetRootFolderAsync(System.String,System.Threading.CancellationToken)>.

   ```
   public async Task<string> GetTemplatesRoot(HttpClient httpClient, string subscriptionId = null)
   {
      ITemplateFoldersClient templateFoldersClient = new TemplateFoldersClient(httpClient);

      FileVM result = await templateFoldersClient.GetRootFolderAsync(subscriptionId);

      return result.Id;
   }
   ```

   In this example, the `subscriptionId` parameter specifies the workspace (subscription) ID, if it is not specified (equal to null), the default user workspace ID will be returned.

2. To upload the necessary template, use the following method <xref:FastReport.Cloud.ITemplatesClient.UploadFileAsync(System.String,FastReport.Cloud.TemplateCreateVM,System.Threading.CancellationToken)>.

```
public async Task<string> UploadFrx(HttpClient httpClient, string folderId, string filePath)
{
   ITemplatesClient templatesClient = new TemplatesClient(httpClient);

   byte[] bytes = File.ReadAllBytes(filePath);

   TemplateCreateVM model = new TemplateCreateVM()
   {
      Name = Path.GetFileName(filePath),
      Content = Convert.ToBase64String(bytes)
   };

   TemplateVM result = await templatesClient.UploadFileAsync(folderId, model);

   return result.Id;
}
```

In this example, the function retrieves a file from disk and uploads it into the folderId directory. Learn more about the parameters:

- `folderId` — template directory ID.
- `model.Name` — file name, so it will be displayed inside FastReports Corporate Server.

  The file name should have the extension `.frx`, if it is not there, then add it manually.
- `model.Content` — Base64 encoded file content.

The method returns the identifier of the uploaded template.

Now this template can be used to prepare (create) a report and export it.

What next?

- Access rights management by template example.
- Create report.

Access rights management by template example

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

Restricting access to private resources is a very important feature of Service solutions. The flexible access system allows you to restrict or grant rights to each resource separately, specifying the range of people who can access it.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Report template.

   To learn how to upload a report template, see the Upload new template article.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Instruction

1. To view the current rights to the resource, use the following method <xref:FastReport.Cloud.ITemplatesClient.GetPermissionsAsync(System.String,System.Threading.CancellationToken)>.

   ```
   public async Task<FilePermission> GetOwnerPermissions(HttpClient httpClient, string templateId)
   {
    ITemplatesClient templatesClient = new TemplatesClient(httpClient);

      FilePermissions permissions = await
         templatesClient.GetPermissionsAsync(templateId);

      return permissions.Owner;
   }
   ```

   In this example, the method returns the permissions for the template owner.

   > Important! Classes <xref:FastReport.Cloud.FilePermissions> and <xref:FastReport.Cloud.FilePermission> differ by the letter `s` at the end, the first contains a full description of access rights to the entity, the second contains only a description for one of the categories.

2. To change the permissions, use the following method <xref:FastReport.Cloud.ITemplatesClient.UpdatePermissionsAsync(System.String,FastReport.Cloud.UpdateFilePermissionsVM,System.Threading.CancellationToken)>.

```
public async Task<FilePermission> ShareTemplate(HttpClient httpClient, string templateId)
{
   ITemplatesClient templatesClient = new TemplatesClient(httpClient);

   UpdateFilePermissionsVM viewModel = new UpdateFilePermissionsVM()
   {
      NewPermissions = new FilePermissions() {
 Anon = new FilePermission() { Get = FilePermissionGet.Entity | FilePermissionGet.Download }
      },
      Administrate = UpdateFilePermissionsVMAdministrate.Anon
   };

   var result = await templatesClient.AddPermissionAsync(templateId, viewModel);

   return result.Other;
}
```

   In this example, the function allows anonymous users to view information about the template and download it.

   Important! In this example, we pass only the permissions we want to edit and specify them in the `Administrate` property.

   If you need to change several categories of rights at once, you can list them using the bitwise operator OR (`|`).

What next?

- Create report.
- Work with groups.
- Add new users to the workspace.

Create report

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article describes the process of creating a report from a template using the Service solutions report processor.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Report template.

   To learn how to upload a report template, see the Upload new template article.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Instruction

1. You need the template ID to create the template. To get it, use the following method <xref:FastReport.Cloud.ITemplatesClient.GetFilesListAsync(System.String,System.Nullable{System.Int32},System.Nullable{System.Int32},System.Threading.CancellationToken)>

   ```
   public async Task<string> GetTemplateId(HttpClient httpClient)
   {
      ITemplateFoldersClient templateFoldersClient =
         new TemplateFoldersClient(httpClient);
      ITemplatesClient templatesClient = new TemplatesClient(httpClient);

      FileVM rootFolder = await templateFoldersClient.GetRootFolderAsync(null);

      IEnumerable<TemplateVM> templates =
         await templatesClient.GetFilesListAsync(rootFolder.Id, 0, 10);

      TemplateVM template = templates.First();

      return template.Id;
   }
   ```

   In this example, the function requests the root directory of the user's default workspace, then requests 10 templates and returns the first one.

2. To generate a report, you will need a directory to place the report in. Request the report root directory by using the following method <xref:FastReport.Cloud.IReportFoldersClient.GetRootFolderAsync(System.String,System.Threading.CancellationToken)>.

```
public async Task<string> GetReportsRoot(HttpClient httpClient,
                                string subscriptionId = null)
{
   IReportFoldersClient reportFoldersClient = new ReportFoldersClient(httpClient);

  FileVM result = await reportFoldersClient.GetRootFolderAsync(subscriptionId);

   return result.Id;
}
```

In this example, the function requests the root directory, the workspace ID can be omitted. In this case, the root directory for the user's default workspace will be returned.

3. To create a report, use the following method <xref:FastReport.Cloud.ITemplatesClient.PrepareAsync(System.String,FastReport.Cloud.PrepareTemplateTaskVM,System.Threading.CancellationToken)>.

```
public async Task<string> BuildReport(HttpClient httpClient,
                                string folderId,
                                string templateId,
                                string fileName)
{
   ITemplatesClient templatesClient = new TemplatesClient(httpClient);

   PrepareTemplateVM task = new PrepareTemplateVM()
   {
      Name = Path.ChangeExtension(fileName, ".fpx"),
      FolderId = folderId
   };

   ReportVM result = await templatesClient.PrepareAsync(templateId, task);

   return result.Id;
}
```

In this example, the function creates a task to create a report.

Important! The report has not been generated yet, but it has already been assigned an identifier. After some time, the builder's turn will come to this task and the report will be created.

> If you do not specify FolderId, the generated report will be saved to the root folder.

4. For information about the report, use the following method <xref:FastReport.Cloud.IReportsClient.GetFileAsync(System.String,System.Threading.CancellationToken)>.

```
public async Task<ReportVMStatus> CheckStatus(HttpClient httpClient, string reportId)
{
   IReportsClient reportsClient = new ReportsClient(httpClient);

   ReportVM result = await reportsClient.GetFileAsync(reportId);

   return result.Status.GetValueOrDefault();
}
```

In this example, the function requests a report by its ID and returns the status. You must wait for the <xref:FastReport.Cloud.FileStatus.Success> status, check it every few seconds.

5. Check the status in the loop and download the file.

```
int tries = 10;
FileStatus status;
do
{
    status = await CheckStatus(httpClient, reportId);
    tries--;
} while (status != FileStatus.Success && tries > 0);

var report = await DownloadReport(httpClient, reportId);
```

6. To download the report, use the following method <xref:FastReport.Cloud.IDownloadClient.GetReportAsync(System.String,System.Threading.CancellationToken)>.

```
public async Task<byte[]> DownloadReport(HttpClient httpClient, string reportId)
{
   IDownloadClient downloadClient = new DownloadClient(httpClient);

 FileResponse file = await downloadClient.GetReportAsync(reportId);

   using(MemoryStream ms = new MemoryStream())
   {
      file.Stream.CopyTo(ms);

      return ms.ToArray();
   }
}
```

   In this example, the function requests a file and copies it to memory.

What next?

- Export report to PDF.

Report parameters

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of passing parameters to a report, which are a dictionary of parameters provided when generating a report. More information on this can be found in the FastReport .NET guide in the Report parameters section.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Report template.

   It can be generated in the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Pass parameters to the report

Parameters can be passed when:

- preparing a report;
- exporting from a template;
- working with tasks.

Let’s consider parameter passing by example of report generation. Detailed instructions on how to create a report can be found in the Create report section.

```
public async Task PassingReportParameters(HttpClient httpClient,
                        string folderId,
                        string templateId,
                        string fileName)
{
   ITemplatesClient templatesClient = new TemplatesClient(httpClient);
   PrepareTemplateVM task = new PrepareTemplateVM()
   {
    Name = Path.ChangeExtension(fileName, ".fpx"),
    FolderId = folderId,
    ReportParameters = new Dictionary<string, string>
    {
       { "Parameter1", "Value1" },
       { "Parameter2", "Value2" }
    }
};

// Add new parameter to dictionary
task.ReportParameters.Add("Parameter3", "Value3");

ReportVM result = await templatesClient.PrepareAsync(templateId, task);}
```

Export report to PDF

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of exporting a report using the Service solutions report processor.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Report.

   You can learn how to create a report in the Create report article.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Annotation

You can export a report directly from the template, without intermediate saving of the report. To do this run the same commands for the report template, replacing `Report` in methods with `Template` use the template ID, not the report ID.

Instruction

1. You will need a report ID to export to PDF. To retrieve it, use the following method <xref:FastReport.Cloud.ITemplatesClient.GetFilesListAsync(System.String,System.Nullable{System.Int32},System.Nullable{System.Int32},System.Threading.CancellationToken)>.

```
public async Task<string> GetReportId(HttpClient httpClient)
{
   IReportFoldersClient reportFoldersClient = new ReportFoldersClient(httpClient);
   IReportsClient reportsClient = new ReportsClient(httpClient);

   FileVM rootFolder = await reportFoldersClient.GetRootFolderAsync(null);

   IEnumerable<ReportVM> reports =
      await reportsClient.GetFilesListAsync(rootFolder.Id, 0, 10);

   ReportVM report = reports.First();

   return report.Id;
}
```

In this example, the function requests the root directory of the user's default workspace, then requests 10 reports and returns the first one.

2. To export the report, you will need a directory to save the export to.

   Retrieve the root directory of exports by using the method <xref:FastReport.Cloud.IExportFoldersClient.GetRootFolderAsync(System.String,System.Threading.CancellationToken)>.

```
public async Task<string> GetExportsRoot(HttpClient httpClient,
                                 string subscriptionId = null)
{
   IExportFoldersClient exportFoldersClient = new ExportFoldersClient(httpClient);

   FileVM result = await exportFoldersClient.GetRootFolderAsync(subscriptionId);

   return result.Id;
}
```

   In this example, the function requests the root directory, the workspace ID can be omitted. In this case, the root directory for the user's default workspace will be returned.

3. To export the report, use the following method <xref:FastReport.Cloud.IReportsClient.ExportAsync(System.String,FastReport.Cloud.ExportReportTaskVM,System.Threading.CancellationToken)>.

```
public async Task<string> ExportReport(HttpClient httpClient,
                                       string folderId,
                                       string reportId,
                                       string fileName)
{
   IReportsClient reportsClient = new ReportsClient(httpClient);

   ExportReportVM task = new ExportReportVM()
   {
      FileName = Path.ChangeExtension(fileName, ".pdf"),
      FolderId = folderId,
      Format = ExportReportTaskVMFormat.Pdf,
      ExportParameters = new Dictionary<string, string>
      {
         { "additionalProp1", ""},
         { "additionalProp2", ""},
         { "additionalProp3", ""}
      }
   };

   ExportVM result = await reportsClient.ExportAsync(reportId, task);

   return result.Id;
}
```

- `FileName` — name of the resulting file. If you do not specify the extension, or specify it incorrectly, the server will replace it by itself.
- `FolderId` — ID of the directory where the export will be placed. If left blank, the export will be placed in the root folder for exports in the workspace.
- `Format` — export format.
- `ExportParameters` — export parameters. They are set similarly to the export parameters from the FastReport .NET library. A more detailed description is stored in the user documentation in the export parameters section.

In this example, the function creates a task to export a report.

Important! The report has not been exported yet, but an ID has already been assigned to the export. After some time, the builder's turn will come to this task and the report will be exported.

> If you do not specify FolderId, the export will be saved to the root folder.

4. For information about the file, use the following method <xref:FastReport.Cloud.IExportsClient.GetFileAsync(System.String,System.Threading.CancellationToken)>.

```
public async Task<ExportVMStatus> CheckStatus(HttpClient httpClient, string exportId)
{
   IExportsClient exportsClient = new ExportsClient(httpClient);

   ExportVM result = await exportsClient.GetFileAsync(exportId);

   return result.Status.GetValueOrDefault();
}
```

In this example, the function requests an export by its ID and returns the status. You must wait for the <xref:FastReport.Cloud.FileStatus.Success> status. Check it every few seconds.

5. Check the status in the loop and download the export.

```
int tries = 10;
FileStatus status;
do
{
status = await CheckStatus(httpClient, reportId);
   tries--;
} while (status != FileStatus.Success && tries > 0);
var report = await DownloadExport(httpClient, reportId);
```

6. Use the following method to download the report <xref:FastReport.Cloud.IDownloadClient.GetExportAsync(System.String,System.Threading.CancellationTok n)>.

```
public async Task<byte[]> DownloadExport(HttpClient httpClient, string exportId)
{
   IDownloadClient downloadClient = new DownloadClient(httpClient);

   FileResponse file = await downloadClient.GetExportAsync(exportId);

   using (MemoryStream ms = new MemoryStream())
   {
      file.Stream.CopyTo(ms);

      return ms.ToArray();
   }
}
```

   In this example, the function requests a file and copies it to memory.

What next?

- Work with groups.
- Add new users to the workspace.

Add new users to the workspace

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of adding a new user to a subscription, retrieving the list of users, and removing a user from it.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher. The subscription must have at least two user slots.

5. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Annotation

Important! It is only possible to add a user to a workspace by user ID.

Instruction

1. To add a new user to a workspace, a workspace (subscription) ID is required.

   To retrieve the workspace ID, use the following method <xref:FastReport.Cloud.ISubscriptionsClient.GetSubscriptionsAsync(System.Nullable{System.Int32},System.Nullable{System.Int32},System.Threading.CancellationToken)>.

   ```
   public async Task<string> GetSubscriptionId(HttpClient httpClient)
   {
      ISubscriptionsClient subscriptionsClient = new SubscriptionsClient(httpClient);
    SubscriptionsVM subscriptions =
          await subscriptionsClient.GetSubscriptionsAsync(0, 10);
    SubscriptionVM subscription = subscriptions.Subscriptions.First();
      return subscription.Id;
   }
   ```

   In this example, the function requests the first 10 workspaces (subscriptions) from the user's list of workspaces, selects the first workspace, and returns its ID.

   > A workspace always is associated with the one subscription, so they have the same ID.

2. To add a new user, use the following method

<xref:FastReport.Cloud.ISubscriptionUsersClient.AddUserAsync(System.String,System.String,System.Threa
ing.CancellationToken)>.

```
public async Task AddUser(HttpClient httpClient, string subscriptionId, string userId)
{
   ISubscriptionUsersClient subscriptionUsersClient =
       new SubscriptionUsersClient(httpClient);
   await subscriptionUsersClient.AddUserAsync(subscriptionId, userId);
}
```

In this example, the function adds the user with the `userId` ID to the workspace with the `subscriptionId` ID.

3. To retrieve the list of workspace users, use the following method <xref:FastReport.Cloud.ISubscriptionUsersClient.GetUsersAsync(System.String,System.Nullable{System.Int
32},System.Nullable{System.Int32},System.Threading.CancellationToken)>.

```
public async Task<IEnumerable<string>> GetUsers(HttpClient httpClient, string subscriptionId)
{
   ISubscriptionUsersClient subscriptionUsersClient =
       new SubscriptionUsersClient(httpClient);
 SubscriptionUsersVM users =
       await subscriptionUsersClient.GetUsersAsync(subscriptionId, 0, 10);
 return users.Users.Select(m => m.UserId);
}
```

In this example, the function requests the first 10 users from a workspace with the `subscriptionId` ID.

4. To remove a user from the workspace, use the following method <xref:FastReport.Cloud.ISubscriptionUsersClient.RemoveUserAsync(System.String,System.String,System.Th
reading.CancellationToken)>.

```
public async Task RemoveUser(HttpClient httpClient, string subscriptionId, string userId)
{
   ISubscriptionUsersClient subscriptionUsersClient =
       new SubscriptionUsersClient(httpClient);
 await subscriptionUsersClient.RemoveUserAsync(subscriptionId, userId);
}
```

In this method, the function removes the user with the `userId` ID from the workspace with the `subscriptionId` ID.

What next?

- Work with groups
- Help and feedback

Work with groups

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This article covers the process of creating a new group, adding a user to the group, and getting a list of the group's users.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher. The subscription must have at least two user slots.

5. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

Annotation

Important! Adding a user to a group is only possible if the user exists in the workspace.

Important! It is possible to add a user to a group only by their ID.

Instruction

1. To create a new group, you need a workspace ID and the name of the new group.

   To retrieve the workspace ID, use the following method <xref:FastReport.Cloud.ISubscriptionsClient.GetSubscriptionsAsync(System.Nullable{System.Int32},System.Nullable{System.Int32},System.Threading.CancellationToken)>.

   ```
   public async Task<string> GetSubscriptionId(HttpClient httpClient)
   {
      ISubscriptionsClient subscriptionsClient = new SubscriptionsClient(httpClient);

      SubscriptionsVM subscriptions =
       await subscriptionsClient.GetSubscriptionsAsync(0, 10);

      SubscriptionVM subscription = subscriptions.Subscriptions.First();

      return subscription.Id;
   }
   ```

   In this example, the function requests the first 10 workspaces from the user's list of workspaces, selects the first workspace, and returns its ID.

> The ID of a workspace and a subscription are the same because one subscription is associated with each workspace.

2. To create a new group, use the following method <xref:FastReport.Cloud.IGroupsClient.CreateGroupAsync(FastReport.Cloud.CreateGroupVM,System.Threading.CancellationToken)>.

```
public async Task<string> CreateGroup(HttpClient httpClient,
                                      string subscriptionId,
                                      string groupName)
{
   IGroupsClient groupsClient = new GroupsClient(httpClient);

   CreateGroupVM viewModel = new CreateGroupVM()
   {
      Name = groupName,
      SubscriptionId = subscriptionId
   };

   GroupVM group = await groupsClient.CreateGroupAsync(viewModel);

   return group.Id;
}
```

   In this example, the function creates a new group with the `groupName` name for the workspace with the `subscriptionId` ID, as a result the function will return the ID of the created group.

3. To add a new user to the group, use the following method <xref:FastReport.Cloud.IGroupUsersClient.AddUserToGroupAsync(System.String,System.String,System.Threading.CancellationToken)>.

```
public async Task AddUser(HttpClient httpClient, string groupId, string userId)
{
   IGroupUsersClient groupUsersClient = new GroupUsersClient(httpClient);

   await groupUsersClient.AddUserToGroupAsync(groupId, userId);
}
```

   In this example, the function adds a user with the `userId` ID to the group with `groupId` ID.

4. To retrieve the list of users in the group, use the following method <xref:FastReport.Cloud.IGroupUsersClient.GetUsersInGroupAsync(System.String,System.Nullable{System.Int32},System.Nullable{System.Int32},System.Threading.CancellationToken)>.

```
public async Task<IEnumerable<string>> GetUsers(HttpClient httpClient, string groupId)
{
   IGroupUsersClient groupUsersClient = new GroupUsersClient(httpClient);

   GroupUsersVM users =
    await groupUsersClient.GetUsersInGroupAsync(groupId, 0, 10);

 return users.Users.Select(m => m.UserId);
}
```

   In this example, the function requests the first 10 users from the group with the `groupId`.

What next?

- Help and feedback

Work with tasks

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

Tasks in Service solutions are actions to convert and deliver documents to consumers. They are described in detail in the Tasks section.

Getting started

You will need the following tools and facilities:

1. Know how to use API key in Service solutions.

   This article will skip over additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, e.g., Visual Studio Code.

4. Report template.

   It can be created in the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Important! These guides assume that you already know how to develop your application in the C# programming language.

> Note. The items above describe recommended tools.

General Information

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article we will cover task types and how to work with them.

Getting Started

You will need the following tools and capabilities:

1. Know how to use API key in Service solutions.

   This article will skip additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, such as Visual Studio Code.

4. Report template.

   It can be prepared using the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

> Note! This guide assumes that you already know how to develop your application in the C# programming language.

> Note! The points above describe the recommended tools.

First, you need to create an HttpClient that we will use later:

```
var httpClient = new HttpClient();
httpClient.BaseAddress = new Uri("https://fastreport.cloud");
httpClient.DefaultRequestHeaders.Authorization = new FastReportCloudApiKeyHeader(ApiKey);
```

Next, let's get the subscription that we will work with tasks:

```
var subscriptions = new SubscriptionsClient(httpClient);
var subscription = (await subscriptions.GetSubscriptionsAsync(0, 10)).Subscriptions.FirstOrDefault();
```

Now we need to create a client for working with tasks:

```
TasksClient tasksClient = new TasksClient(httpClient);
```

Now you can proceed directly to working with tasks.

Stored Tasks

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

As mentioned in the previous section, Service solutions has the ability to create tasks that are stored in the database and can be run later. These tasks are called stored tasks.

Let's take a closer look at how to create, update, and run stored tasks using the C# SDK.

About ViewModels and Types

When working, you will pass different ViewModels (View Models) to SDK methods. Let's consider which view models can be used.

View Models for Creating Tasks

View Models for creating transformer tasks:

- CreatePrepareTaskVM
- CreateExportTemplateTaskVM
- CreateExportReportTaskVM

View Models for creating transport tasks:

- CreateEmailTaskVM
- CreateWebhookTaskVM
- CreateFTPUploadTaskVM
- CreateS3UploadTaskVM

View Models for creating other tasks:

- CreateFetchTaskVM

View Models for Updating Properties of Saved Tasks

View Models for updating transformer tasks:

- UpdatePrepareTaskVM
- UpdateExportTemplateTaskVM
- UpdateExportReportTaskVM

View Models for updating transport tasks:

- UpdateEmailTaskVM
- UpdateWebhookTaskVM
- UpdateFTPUploadTaskVM
- UpdateS3UploadTaskVM

View Models for updating other tasks:

- UpdateFetchTaskVM

View Model for Updating Task Permissions:

- UpdateTaskPermissionsVM

Creating a Task

To create tasks, you should use the CreateTask or CreateTaskAsync method. It can accept any class that inherits from TaskBaseVM:

- CreatePrepareTaskVM
- CreateExportTemplateTaskVM
- CreateExportReportTaskVM
- CreateFetchTaskVM
- CreateEmailTaskVM
- CreateWebhookTaskVM
- CreateFTPUploadTaskVM
- CreateS3UploadTaskVM.

Let's consider creating a report preparation task, saving it to a folder and subsequent export to PDF with saving to a folder:

```
await tasksClient.CreateTaskAsync(new CreatePrepareTaskVM
        {
           Name = "My First Task",
           Type = TaskType.Prepare,
           InputFile = new InputFileVM
           {
              EntityId = "{templateId}"
           },
           OutputFile = new OutputFileVM
           {
              FileName = "My First Report Generated by Task.fpx",
              FolderId = "{report folder identifier}"
           },
           Exports = new List<CreateExportReportTaskVM>
           {
              new CreateExportReportTaskVM
              {
                 Format = ExportFormat.Pdf,
                 OutputFile = new OutputFileVM
                 {
                    FileName = "pdfFromFpxFromFrx.pdf",
                    FolderId = "{export folder identifier}"
                 }
              }
           }
        });
```

Real object identifiers should be entered in the EntityId and FolderId fields. Otherwise, the task will be aborted with an error.

> Note! If you don't specify OutputFile, it will be saved to a temporary folder. If you specify an empty OutputFile - to the root folder. If you specify a folder id - to it.

Next, let's consider creating a report export task with subsequent email sending:

```
var currentTask = await tasksClient.CreateTaskAsync(new CreateExportTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "PDF Email Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   ReportParameters = new Dictionary<string, string>
   {
      { "param1", "val1" },
      { "param2", "val2" },
      { "param3", "val3" }
   },
   Format = ExportFormat.Pdf,
   ExportParameters = new Dictionary<string, string>
   {
      { "PrintOptimized", "true" },
      { "JpegQuality", "90" }
   },
   Transports = new List<CreateTransportTaskBaseVM>
   {
      new CreateEmailTaskVM
      {
         Name = "Email Distribution Task",
         From = "admin@company.com",
         To = new List<string> { "user@company.com","seller@company.com" },
         Subject = "Important Report",
         IsBodyHtml = false,
         Username = "admin@company.com",
         Password = "parol123",
         Server = "smtp.company.com",
         Body = "An important report is attached to this email.",
         Port= 587,
         EnableSsl = true
      }
   }
});
```

Real object identifiers should be entered in the EntityId and FolderId fields. Otherwise, the task will be aborted with an error.

> Note! If you don't specify OutputFile, it will be saved to a temporary folder. If you specify an empty OutputFile - to the root folder. If you specify a folder id - to it.

Running Tasks on Schedule

To configure scheduled task execution, you need to specify the following fields:

- `CronExpression`
- `StartsOn`
- `Ends`

`CronExpression` - a string containing a cron expression that defines the task execution schedule.

`StartsOn` - the date and time from which the scheduled task execution begins (in user's timezone).

`Ends` - an object containing the date and time until which the task will be executed (`On`) or the number of repetitions (`After`). The `On` field is specified in user's timezone. If neither the `After` nor the `On` field is specified, the task will run indefinitely.

Example of creating a scheduled task:

```
        var currentTask = await tasksClient.CreateTaskAsync(new CreateExportTemplateTaskVM
        {
           SubscriptionId = subscriptionId,
           Name = "PDF Email Sending Task",
           InputFile = new InputFileVM
           {
              EntityId = template.Id
           },
           Format = ExportFormat.Pdf,
           CronExpression = "*/5 * * * *",
           StartsOn = DateTime.Now.AddMinutes(1),
           Ends = new CreateTaskEndVM
           {
              On = DateTime.Now.AddDays(1)
           }
        });
```

In this example, the task will run every 5 minutes, starting from the task creation date and will finish execution after one day.

To make the task execute, for example, 10 times, you need to replace the `On` field with the `After` field:

```
CronExpression = "*/5 * * * *",
           StartsOn = DateTime.UtcNow,
           Ends = new CreateTaskEndVM
           {
              After = 10
           }
```

Getting Task List

```
// Get the first 100 tasks from the workspace
var tasks = await tasksClient.GetListAsync(0, 100, subscription.Id);
```

Editing a Task

After a task is created, you can change some of its parameters.

```
// Task editing
var editedTask = await tasksClient.UpdateTaskAsync(currentTaskId, new UpdateExportTemplateTaskVM
{
   Format = ExportFormat.Docx,
   ReportParameters = new Dictionary<string, string>
   {
      { "param1", "val1" },
      { "param2", "val2" }
   },
   TransportIds = new List<string> { "{transportId}" }
});
```

Note that when updating a task, you need to specify `TransportIds` - a list of transport task identifiers (Email, FTP, Webhook, S3), not the tasks themselves. The task itself can be created in advance either separately or when creating a transformer task. For more details on their creation, read the section Creating Transport Tasks.

Executing a Task by Specified Identifier

```
// Run task by identifier
await tasksClient.RunTaskByIdAsync(id);
```

Deleting Tasks from Storage

```
// Delete all tasks
foreach(var t in tasks.Tasks)
{
    await tasksClient.DeleteTaskAsync(t.Id);
}
```

Tasks Executed from Request Body

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

As mentioned in the previous section, Service solutions has the ability to run tasks without prior saving. To do this, you should use the RunTask method, passing a model inherited from UpdateTaskBaseVM into it.

For example, to build a PDF document and send it via FTP, you need to pass data such as template identifier (`EntityId`) or the template itself (`Content`) to the `InputFile` object in the RunTask method, as well as data for connecting to the FTP server.

Let's take a closer look at how to work with such tasks using the C# SDK.

About ViewModels and Types

When working, you will pass different ViewModels (View Models) to SDK methods. Let's consider which view models can be used.

View Models for Running Tasks On-the-Fly (Without Saving)

View Models for running transformer tasks on-the-fly:

- RunPrepareTaskVM
- RunExportTemplateTaskVM
- RunExportReportTaskVM

View Models for running transport tasks on-the-fly:

- RunEmailTaskVM
- RunWebhookTaskVM
- RunFTPUploadTaskVM
- RunS3UploadTaskVM

View Models for running other tasks on-the-fly:

- RunFetchTaskVM

Running Task On-the-Fly (from Request Body)

Let's consider running a report export task with subsequent email sending:

```
var runTask = await tasksClient.RunTaskAsync(new RunExportTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   InputFile = new RunInputFileVM
   {
      EntityId = template.Id,
      FileName = "Document"
      //Content = new byte[] { }
   },
   Format = ExportFormat.Docx,
   Transports = new List<RunTransportTaskBaseVM>
   {
      new RunEmailTaskVM
      {
         From =  "admin@company.com",
         To = ["user@company.com", "seller@company.com"],
         Subject = "Important Report",
         Body = "An important report is attached to this email.",
         EnableSsl = true,
         Username = "admin@company.com",
         Password = "password123",
         Server =  "smtp.company.com",
         Port = 587
      }
   }
});
```

Real object identifiers should be entered in the EntityId field. Otherwise, the task will be aborted with an error.

Also, instead of it, you can pass the template content in the `Content` field as a byte array in the `InputFile` object.

Export Template Task

Products: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider working with the template export task (ExportTemplateTask). General instructions on how to work with tasks are described in the General Information section.

It should be noted that you can export a template not only using the task system, but also directly through a request to the templates controller. The advantage of using the task system is that it allows not only saving the document to a folder, but also transferring it to external systems (via Email, FTP, S3, Webhook).

Creating a Task

Let's consider creating a template export task to PDF:

```
// Creating a task
var currentTask = await tasksClient.CreateTaskAsync(new CreateExportTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "PDF Email Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   ReportParameters = new Dictionary<string, string>
   {
      { "param1", "val1" },
      { "param2", "val2" },
      { "param3", "val3" }
   },
   Format = ExportFormat.Pdf,
   ExportParameters = new Dictionary<string, string>
   {
      { "PrintOptimized", "true" },
      { "JpegQuality", "90" }
   },
   OutputFile = new OutputFileVM
   {
      FileName = "Report"
   }
});
```

Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

Since no folder for saving is specified in OutputFile, when the task runs, the document will be saved to the root folder.

Let's specify a different folder for saving:

```
 "OutputFile" : {
   "FolderId": "61cb2226d24478b172103075",
    "FileName": "Report"
 }
```

Running Task by Identifier

Now when the task runs, the PDF file will be saved to the specified folder.

```
await tasksClient.RunTaskByIdAsync(currentTask.Id);
```

Sending Document After Export

To send the document after export, you can use the Email, FTP, S3, or Webhook sending task.

Let's consider an example of creating a PDF export task with subsequent sending of the finished document via Email.

```
// Creating a task
var currentTask = await tasksClient.CreateTaskAsync(new CreateExportTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "PDF Email Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   ReportParameters = new Dictionary<string, string>
   {
      { "param1", "val1" },
      { "param2", "val2" },
      { "param3", "val3" }
   },
   Format = ExportFormat.Pdf,
   ExportParameters = new Dictionary<string, string>
   {
      { "PrintOptimized", "true" },
      { "JpegQuality", "90" }
   },
   Transports = new List<CreateTransportTaskBaseVM>
   {
      new CreateEmailTaskVM
      {
         Name = "Email Distribution Task",
         From = "admin@company.com",
         To = new List<string> { "user@company.com","seller@company.com" },
         Subject = "Important Report",
         IsBodyHtml = false,
         Username = "admin@company.com",
         Password = "parol123",
         Server = "smtp.company.com",
         Body = "An important report is attached to this email.",
         Port= 587,
         EnableSsl = true
      }
   }
});
```

Editing a Task

After a task is created, you can change some of its parameters.

```
// Task editing
var editedTask = await tasksClient.UpdateTaskAsync(currentTask.Id, new UpdateExportTemplateTaskVM
{
   Format = ExportFormat.Docx,
   ReportParameters = new Dictionary<string, string>
   {
      { "param1", "val1" },
      { "param2", "val2" }
   },
   TransportIds = new List<string> { "{transportId}" }
});
```

Note that when updating a task, you need to specify `TransportIds` - a list of transport task identifiers (Email, FTP, Webhook, S3), not the tasks themselves. The task itself can be created in advance either separately or when creating a transformer task. For more details on their creation, read the section Creating Transport Tasks.

Executing Task from Request Body

```
var runTask = await tasksClient.RunTaskAsync(new RunExportTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   InputFile = new RunInputFileVM
   {
      EntityId = template.Id,
      FileName = "Document"
      //Content = new byte[] { }
   },
   Format = ExportFormat.Docx,
   Transports = new List<RunTransportTaskBaseVM>
   {
      new RunEmailTaskVM
      {
         From =  "admin@company.com",
         To = ["user@company.com", "seller@company.com"],
         Subject = "Important Report",
         Body = "An important report is attached to this email.",
         EnableSsl = true,
         Username = "admin@company.com",
         Password = "parol123",
         Server =  "smtp.company.com",
         Port = 587
      }
   }
});
```

Thus, as a result of executing this request, a PDF document will be created and sent via email.

> Note! In this case, the creation of the PDF document and its sending via email will be performed directly by this request and the task will not be saved in the database.

Export Report Task

Products: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider working with the report export task (ExportReportTask). General instructions on how to work with tasks are described in the General Information section.

It should be noted that you can export a template not only using the task system, but also directly through a request to the reports controller. The advantage of using the task system is that it allows not only saving the document to a folder, but also transferring it to external systems (via Email, FTP, S3, Webhook).

Creating a Task

Let's consider creating a report export task to PDF:

```
// Creating a task
var currentTask = await tasksClient.CreateTaskAsync(new CreateExportReportTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "PDF Email Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   Format = ExportFormat.Pdf,
   ExportParameters = new Dictionary<string, string>
   {
      { "PrintOptimized", "true" },
      { "JpegQuality", "90" }
   },
   OutputFile = new OutputFileVM
   {
      FileName = "Report"
   }
});
```

Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

Since no folder for saving is specified in OutputFile, when the task runs, the document will be saved to the root folder.

Let's specify a different folder for saving:

```
"OutputFile" : {
  "FolderId": "61cb2226d24478b172103075",
   "FileName": "Report"
}
```

Running Task by Identifier

Now when the task runs, the PDF file will be saved to the specified folder.

```
await tasksClient.RunTaskByIdAsync(currentTask.Id);
```

Sending Document After Export

To send the document after export, you can use the Email, FTP, S3, or Webhook sending task.

Let's consider an example of creating a PDF export task with subsequent sending of the finished document via Email.

```
// Creating a task
var currentTask = await tasksClient.CreateTaskAsync(new CreateExportReportTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "PDF Email Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   Format = ExportFormat.Pdf,
   ExportParameters = new Dictionary<string, string>
   {
      { "PrintOptimized", "true" },
      { "JpegQuality", "90" }
   },
   Transports = new List<CreateTransportTaskBaseVM>
   {
      new CreateEmailTaskVM
      {
         Name = "Email Distribution Task",
         From = "admin@company.com",
         To = new List<string> { "user@company.com","seller@company.com" },
         Subject = "Important Report",
         IsBodyHtml = false,
         Username = "admin@company.com",
         Password = "parol123",
         Server = "smtp.company.com",
         Body = "An important report is attached to this email.",
         Port= 587,
         EnableSsl = true
      }
   }
});
```

Editing a Task

After a task is created, you can change some of its parameters.

```
// Task editing
var editedTask = await tasksClient.UpdateTaskAsync(currentTask.Id, new UpdateExportReportTaskVM
{
   Format = ExportFormat.Docx,
   TransportIds = new List<string> { "{transportId}" }
});
```

Note that when updating a task, you need to specify `TransportIds` - a list of transport task identifiers (Email, FTP, Webhook, S3), not the tasks themselves. The task itself can be created in advance either separately or when creating a transformer task. For more details on their creation, read the section Creating Transport Tasks.

Executing Task from Request Body

```
var runTask = await tasksClient.RunTaskAsync(new RunExportReportTaskVM
{
   SubscriptionId = subscriptionId,
   InputFile = new RunInputFileVM
   {
      EntityId = template.Id,
      FileName = "Document"
      //Content = new byte[] { }
   },
   Format = ExportFormat.Docx,
   Transports = new List<RunTransportTaskBaseVM>
   {
      new RunEmailTaskVM
      {
         From =  "admin@company.com",
         To = ["user@company.com", "seller@company.com"],
         Subject = "Important Report",
         Body = "An important report is attached to this email.",
         EnableSsl = true,
         Username = "admin@company.com",
         Password = "parol123",
         Server =  "smtp.company.com",
         Port = 587
      }
   }
});
```

Thus, as a result of executing this request, a PDF document will be created and sent via email.

> Note! In this case, the creation of the PDF document and its sending via email will be performed directly by this request and the task will not be saved in the database.

Prepare Report Task

Products: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider working with the report preparation task (PrepareTemplateTask). General instructions on how to work with tasks are described in the General Information section.

It should be noted that you can prepare a report not only using the task system, but also directly through a request to the templates controller. The advantage of using the task system is that it allows not only saving the document to a folder, but also transferring it to external systems (via Email, FTP, S3, Webhook).

Creating a Task

Let's consider creating a report preparation task:

```
// Creating a task
var currentTask = await tasksClient.CreateTaskAsync(new CreatePrepareTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "Report Preparation Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   OutputFile = new OutputFileVM
   {
      FileName = "Report.fpx"
   }
});
```

Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

Since no folder for saving is specified in OutputFile, when the task runs, the document will be saved to the root folder.

Let's specify a different folder for saving:

```
 "OutputFile" : {
  "FolderId": "61cb2226d24478b172103075",
   "FileName": "Report"
 }
```

Running Task by Identifier

Now when the task runs, the PDF file will be saved to the specified folder.

```
await tasksClient.RunTaskByIdAsync(currentTask.Id);
```

Exporting Report After Preparation

To export the report after preparation, you can use the export task.

Let's consider an example of creating a report preparation task with subsequent export to PDF.

```
// Creating a task
var currentTask = await tasksClient.CreateTaskAsync(new CreatePrepareTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   Name = "Report Preparation Task",
   InputFile = new InputFileVM
   {
      EntityId = template.Id
   },
   OutputFile = new OutputFileVM
   {
      FileName = "Report.fpx"
   },
   Exports = new List<CreateExportReportTaskVM>()
   {
      new CreateExportReportTaskVM
      {
         SubscriptionId = subscriptionId,
         Name = "PDF Export Task",
         Format = ExportFormat.Pdf,
         OutputFile = new OutputFileVM
         {
            FileName = "Document.pdf"
         }
      }
   }
});
```

Editing a Task

After a task is created, you can change some of its parameters.

```
// Task editing
var editedTask = await tasksClient.UpdateTaskAsync(currentTask.Id, new UpdatePrepareTemplateTaskVM
{
   ReportParameters = new Dictionary<string, string>
   {
      { "param1", "val1" },
      { "param2", "val2" }
   },
   TransportIds = new List<string> { "{transportId}" }
});
```

Note that when updating a task, you need to specify `TransportIds` - a list of transport task identifiers (Email, FTP, Webhook, S3), not the tasks themselves. The task itself can be created in advance either separately or when creating a transformer task. For more details on their creation, read the section Creating Transport Tasks.

Executing Task from Request Body

```
var runTask = await tasksClient.RunTaskAsync(new RunPrepareTemplateTaskVM
{
   SubscriptionId = subscriptionId,
   InputFile = new RunInputFileVM
   {
      EntityId = template.Id,
      FileName = "Document"
      //Content = new byte[] { }
   },
   Transports = new List<RunTransportTaskBaseVM>
   {
      new RunEmailTaskVM
      {
         From =  "admin@company.com",
         To = ["user@company.com", "seller@company.com"],
         Subject = "Important Report",
         Body = "An important report is attached to this email.",
         EnableSsl = true,
         Username = "admin@company.com",
         Password = "parol123",
         Server =  "smtp.company.com",
         Port = 587
      }
   }
});
```

Thus, as a result of executing this request, a report will be prepared and sent via email.

> Note! In this case, the creation of the report and its sending via email will be performed directly by this request and the task will not be saved in the database.

Sending via Email

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider the method of sending a report via email. Instructions on how to work with tasks are described in the General Information section.

Getting Started

You will need the following tools and capabilities:

1. Knowledge of using API key in Service solutions.

   This article will skip additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, such as Visual Studio Code.

4. Report template.

   It can be prepared using the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

7. Configured and accessible mail server.

> Note! This guide assumes that you already know how to develop your application in the C# programming language.

> Note! The points above describe the recommended tools.

> Note! This guide assumes that you have experience configuring mail server connections.

Creating a Task

Let's consider creating an email template sending task:

```
// Object initialization
CreateEmailTaskVM emailTaskVM = new CreateEmailTaskVM
{
   Name = "Email Sending Task",
   SubscriptionId = "{workspace identifier}",
   InputFile = new InputFileVM
   {
      EntityId = "{template identifier}",
      Type = FileKind.Template
   },
   From = "admin@company.com",
   To = ["user@company.com", "seller@company.com"],
   Subject = "Important Report",
   Body = "An important report is attached to this email.",
   EnableSsl = true,
   Username = "admin@company.com",
   Password = "parol123",
   Server = "smtp.company.com",
   Port = 587
};

 // Creating a task
 TaskBaseVM emailTask = await tasksClient.CreateTaskAsync(emailTaskVM);
```

```
 // Run task by identifier
 await tasksClient.RunTaskByIdAsync(emailTask.Id);
```

Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

Executing Task from Request Body

```
RunEmailTaskVM emailTaskVM = new RunEmailTaskVM
{
   SubscriptionId = "{workspace identifier}",
   InputFile = new RunInputFileVM
   {
      EntityId = "{template identifier}",
      Type = FileKind.Template
   },
   From = "admin@company.com",
   To = ["user@company.com", "seller@company.com"],
   Subject = "Important Report",
   Body = "An important report is attached to this email.",
   EnableSsl = true,
   Username = "admin@company.com",
   Password = "parol123",
   Server = "smtp.company.com",
   Port = 587
};

// Task execution
TaskMessageIdVM emailTask = await tasksClient.RunTaskAsync(emailTaskVM);
```

> Note! In this case, the file sending via email will be performed directly by this request and the task will not be saved in the database. Also, you can pass reports and ready documents in various formats in the content field, for example, PDF, XLSX, etc.

> Note! The message identifier will be returned in emailTask.

Saving to FTP Server

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider the method of sending a report to an FTP server. Instructions on how to work with tasks are described in the General Information section.

Getting Started

You will need the following tools and capabilities:

1. Knowledge of using API key in Service solutions.

   This article will skip additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, such as Visual Studio Code.

4. Report template.

   It can be prepared using the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

7. Configured and accessible FTP server.

> Note! This guide assumes that you already know how to develop your application in the C# programming language.

> Note! The points above describe the recommended tools.

> Note! This guide assumes that you have experience configuring and setting up an FTP server to accept files from external sources.

Creating a Task

Let's consider creating a template sending task to FTP server:

```
// Object initialization
 CreateFTPUploadTaskVM ftpUploadTaskVM = new CreateFTPUploadTaskVM
 {
    Name = "FTP Sending Task",
    InputFile = new InputFileVM
    {
       EntityId = "{template identifier}",
       Type = FileKind.Template
    },
    FtpHost = "{FTP server address}",
    FtpPort = 21,
    FtpUsername = "{FTP server username}",
    FtpPassword = "{password}",
    UseSFTP = false,
    DestinationFolder = "{/path_to_folder/}",
    Archive = false,
    ArchiveName = "Archive Name",
    SubscriptionId = "{workspace identifier}"
 };

 // Creating a task
 TaskBaseVM ftpUploadTask = await tasksClient.CreateTaskAsync(ftpUploadTaskVM);

 // Run task by identifier
 await tasksClient.RunTaskByIdAsync(ftpUploadTask.Id);
```

Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

Executing Task from Request Body

```
// Run task from request body
 await tasksClient.RunTaskAsync(new RunFTPUploadTaskVM
 {
    InputFile = new RunInputFileVM
    {
       EntityId = "{template identifier}",
       Type = FileKind.Template
    },
    FtpHost = "{FTP server address}",
    FtpPort = 21,
    FtpUsername = "{FTP server username}",
    FtpPassword = "{password}",
    UseSFTP = false,
    DestinationFolder = "{/path_to_folder/}",
    Archive = false,
    ArchiveName = "Archived Template",
    SubscriptionId = "{workspace identifier}"
 });
```

> Note! In this case, the FTP sending will be performed directly by this request and the task will not be saved in the database.

Updating Task by Identifier

```
await tasksClient.UpdateTaskAsync("{old task identifier}", new UpdateFTPUploadTaskVM()
{
    InputFile = new RunInputFileVM
    {
        EntityId = "{updated template identifier}",
        Type = FileKind.Template
    },
    FtpHost = "{Updated FTP server address}",
    FtpPort = 21,
    FtpUsername = "{updated FTP server username}",
    FtpPassword = "{updated password}",
    UseSFTP = false,
    DestinationFolder = "/updated_path_to_folder",
    Archive = false,
    ArchiveName = "Updated Archive Name"
});
```

Saving via Webhook

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider the method of sending a report via webhook. Instructions on how to work with tasks are described in the General Information section.

Getting Started

You will need the following tools and capabilities:

1. Knowledge of using API key in Service solutions.

   This article will skip additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, such as Visual Studio Code.

4. Report template.

   It can be prepared using the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

7. Client application that accepts requests.

> Note! This guide assumes that you already know how to develop your application in the C# programming language.

> Note! The points above describe the recommended tools.

> Note! This guide assumes that you have a client application that accepts and processes incoming webhooks.

Creating a Task

Let's consider creating a template sending task via webhook and its subsequent execution:

```
// Object initialization
CreateWebhookTaskVM webhookTaskVM = new CreateWebhookTaskVM
{
   Name = "Webhook Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = "{template identifier}",
      Type = FileKind.Template
   },
   Url = new Uri("https://example.com/"),
   Headers = new Dictionary<string, string> {
      { "Authorization", "Bearer <token>" },
      { "Content-Length", "1238" }
   },
   SubscriptionId = "{workspace identifier}"
};

 // Creating a task
 TaskBaseVM webhookTask = await tasksClient.CreateTaskAsync(webhookTaskVM);

 // Run task by identifier
 await tasksClient.RunTaskByIdAsync(webhookTask.Id);
```

> Note! Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

> Note! EntityId can be passed template, report, and export identifier.

Executing Task from Request Body

```
// Run task from request body
 await tasksClient.RunTaskAsync(new RunWebhookTaskVM
 {
    InputFile = new RunInputFileVM
    {
       EntityId = "{template identifier}",
       Type = FileKind.Template
    },
    Url = new Uri("https://example.com/"),
    Headers = new Dictionary<string, string> {
       { "Authorization", "Bearer <token>" },
       { "Content-Type", "multipart/form-data" }
    },
    SubscriptionId = "{workspace identifier}"
 });
```

> Note! In this case, the webhook sending will be performed directly by this request and the task will not be saved in the database.

Updating Task by Identifier

```
await tasksClient.UpdateTaskAsync("{old task identifier}", new UpdateWebhookTaskVM()
 {
    InputFile = new RunInputFileVM
    {
       EntityId = "{updated template identifier}",
       Type = FileKind.Template
    },
    Url = new Uri("{Updated address}"),
    Headers = new Dictionary<string, string> {
       { "{Updated header}", "{Updated value}" },
    }
 });
```

Example of a request that will come to the webhook endpoint:


```
{
  "startedDateTime": "2024-06-07 08:37:09",
  "request": {
    "method": "POST",
    "url": "https://example.com/6e259560-25e5-482b-a7b2-8c5267ba6ae3/",
    "headers": [
      {
        "name": "connection",
        "value": "close"
      },
      {
        "name": "content-length",
        "value": "1421"
      },
      {
        "name": "content-type",
        "value": "multipart/form-data; boundary=\"62ec6524-9360-4e91-834f-e9531e2d2c30\""
      },
      {
        "name": "host",
        "value": "example.com"
      }
    ],
    "bodySize": 0,
    "postData": {
      "mimeType": "application/json",
      "text": ""
    }
  },
  "response": {
    "status": 200,
    "httpVersion": "HTTP/1.1",
    "headers": [
      {
        "name": "Content-Type",
        "value": "text/html"
      }
    ],
    "content": {
      "size": 145,
      "text": "This URL has no default content configured.",
      "mimeType": "text/html"
    }
  }
}
```

Saving to S3 Storage

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

In this article, we will consider the method of sending a report to S3 storage. Instructions on how to work with tasks are described in the General Information section.

Getting Started

You will need the following tools and capabilities:

1. Knowledge of using API key in Service solutions.

   This article will skip additional information on authentication and authorization.

2. .NET SDK.

3. C# code editor or text editor, such as Visual Studio Code.

4. Report template.

   It can be prepared using the free FastReport Community Designer program.

5. Active subscription to one of the following products: FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.

6. Internet access.

7. Configured and accessible S3 storage.

> Note! This guide assumes that you already know how to develop your application in the C# programming language.

> Note! The points above describe the recommended tools.

> Note! This guide assumes that you have configured S3 storage to accept files from external sources.

Creating a Task

Let's consider creating a template sending task to S3 storage:

```
// Object initialization
 CreateS3UploadTaskVM s3UploadTaskVM = new CreateS3UploadTaskVM
 {
   Name = "S3 Sending Task",
   InputFile = new InputFileVM
   {
      EntityId = "{template identifier}",
      Type = FileKind.Template
   },
   AccessKey = "{public_access_key}",
   SecretKey = "{secret_access_key}",
   Url = "https://example.com",
   BucketName = "{bucket_name}",
   UseAws = true,
   EnableSsl = true,
   Region = "us-east-1",
   DestinationFolder = "{/path_to_folder/}",
   SubscriptionId = "{workspace identifier}"
 };

 // Creating a task
 TaskBaseVM s3UploadTask = await tasksClient.CreateTaskAsync(s3UploadTaskVM);

 // Run task by identifier
 await tasksClient.RunTaskByIdAsync(s3UploadTask.Id);
```

Real object identifiers should be entered in the EntityId and SubscriptionId fields. Otherwise, the task will be aborted with an error.

Executing Task from Request Body

```
// Run task from request body
 await tasksClient.RunTaskAsync(new RunS3UploadTaskVM
 {
    InputFile = new RunInputFileVM
    {
       EntityId = "{template identifier}",
       Type = FileKind.Template
    },
    AccessKey = "{public_access_key}",
    SecretKey = "{secret_access_key}",
    Url = "https://example.com",
    BucketName = "{bucket_name}",
    UseAws = true,
    EnableSsl = true,
    Region = "us-east-1",
    DestinationFolder = "{/path_to_folder/}",
    SubscriptionId = "{workspace identifier}"
 });
```

> Note! In this case, the S3 sending will be performed directly by this request and the task will not be saved in the database.

Updating Task by Identifier

```
await tasksClient.UpdateTaskAsync("{task identifier}", new UpdateS3UploadTaskVM()
{
   InputFile = new RunInputFileVM
   {
      EntityId = "{updated template identifier}",
      Type = FileKind.Template
   },
   AccessKey = "{Updated public_access_key}",
   SecretKey = "{Updated secret_access_key}",
   Url = "Updated S3 storage address",
   BucketName = "{Updated bucket_name}",
   UseAws = true,
   EnableSsl = true,
   Region = "{Updated region}",
   DestinationFolder = "{/Updated_path_to_folder/}"
});
```

Static Preview Integration Guide

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

This guide provides comprehensive instructions for integrating Service solutions reports into your web applications using iframes.

Basic Iframe Setup

Service solutions Static Preview service is designed to be easily embedded in your web applications using iframes. This allows you to integrate report previews seamlessly into your existing user interfaces.

```
<iframe
  src="https://yourdomain.com/staticpreview/t/your-template-id?apikey=your-api-key"
  width="100%"
  height="600px"
  frameborder="0"
  style="border: 1px solid #ccc;">
</iframe>
```

URL Structure

The iframe source URL follows this pattern:

```
https://yourdomain.com/staticpreview/{type}/{report-id}?{parameters}
```

Where:

- `{type}`: Either `t` for templates or `r` for reports
- `{report-id}`: Your template or report identifier
- `{parameters}`: Query string parameters for authentication and configuration

Authentication Parameters

API Key Authentication

```
<!-- For API Key authentication -->
<iframe src="https://yourdomain.com/staticpreview/t/template-123?apikey=your-api-key-here"></iframe>
```

Parameter: `apikey`

- **Required:** Yes (for API key auth)
- **Description:** Your API key for FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.
- **Example:** `apikey=dw3ized7i7i97qjsrg4mesn3n8r1bpi54wbyhywpfier7er3nqjo`

Access Key Authentication

```
<!-- For Access Key authentication -->
<iframe src="https://yourdomain.com/staticpreview/r/report-456?accessKey=your-access-key"></iframe>
```

Parameter: `accessKey`

- **Required:** Yes (for access key auth)
- **Description:** Temporary access key for specific report access
- **Example:** `accessKey=temp-key-abc123def456`

Report Parameters

For templates that require parameters, use the `rp` parameter format:

```
<iframe src="https://yourdomain.com/staticpreview/t/sales-report?apikey=your-key&rp=StartDate:2023-01-01&rp=EndDate:2023-12-31&rp=Region:North"></iframe>
```

Parameter: `rp` (Report Parameters)

- **Format:** `rp=ParameterName:ParameterValue`
- **Multiple:** Use multiple `rp` parameters for different values
- **Description:** Sets initial values for report parameters
- **Examples:**
  - Date: `rp=StartDate:2023-01-01`
  - String: `rp=CustomerName:John Doe`
  - Number: `rp=MinAmount:1000`
  - Boolean: `rp=IncludeInactive:true`

> The parameters must be specified inside the report template.

Localization of UI

The service automatically detects locale from:

1. Server configuration (if available)
2. Browser language settings

Localization of Report

The report localization will be used from the report parameters itself. Or from the workspace parameters.

Responsive Iframe Examples

Full-width Responsive

```
<div style="position: relative; width: 100%; height: 0; padding-bottom: 75%;">
  <iframe
    src="https://yourdomain.com/staticpreview/t/template-123?apikey=your-key"
    style="position: absolute; top: 0; left: 0; width: 100%; height: 100%; border: none;"
    allowfullscreen>
  </iframe>
</div>
```

Modal/Dialog Integration

```
<div class="modal" id="reportModal">
  <div class="modal-content" style="width: 90%; height: 90%;">
    <iframe
      id="reportFrame"
      src="https://yourdomain.com/staticpreview/t/template-123?apikey=your-key"
      style="width: 100%; height: 100%; border: none;">
    </iframe>
  </div>
</div>

<script>
function openReport(templateId, apiKey, parameters = {}) {
  const baseUrl = 'https://yourdomain.com/staticpreview/t/' + templateId;
  const params = new URLSearchParams({ apikey: apiKey });

  // Add report parameters
  Object.entries(parameters).forEach(([key, value]) => {
    params.append('rp', `${key}:${value}`);
  });

  document.getElementById('reportFrame').src = baseUrl + '?' + params.toString();
  document.getElementById('reportModal').style.display = 'block';
}

// Usage
openReport('sales-report', 'your-api-key', {
  StartDate: '2023-01-01',
  EndDate: '2023-12-31',
  Region: 'North America'
});
</script>
```

Recommended Iframe Dimensions

- **Minimum Width:** 400px (for mobile compatibility)
- **Minimum Height:** 300px (to show at least one page)
- **Recommended:** 800px × 600px or larger
- **Full-screen:** Use responsive design for best experience

Security Considerations

1. **API Key Protection:** Never expose API keys in client-side code for production
2. **Access Keys:** Use temporary access keys when possible
3. **HTTPS:** Always use HTTPS for iframe sources in production
4. **CSP Headers:** Configure Content Security Policy to allow iframe sources

```
<iframe src="..."></iframe>
```

Parameter Reference

PARAMETER	TYPE	REQUIRED	DESCRIPTION	EXAMPLE
`apikey`	String	Yes*	API key for FastReport Cloud, FastReport Corporate Server, or FastReport Publisher.	`apikey=your-api-key`
`accessKey`	String	Yes*	Temporary access key	`accessKey=temp-key-123`

PARAMETER	TYPE	REQUIRED	DESCRIPTION	EXAMPLE
`rp`	String	No	Report parameter in `Name:Value` format	`rp=StartDate:2023-01-01`

* Either `apikey` or `accessKey` is required depending on authentication method.

User Security and CORS

> **Products**: FastReport Cloud, FastReport Corporate Server, FastReport Publisher

FastReport Cloud, FastReport Corporate Server, FastReport Publisher takes user security seriously and implements a strict CORS (Cross-Origin Resource Sharing) policy to protect data and prevent potential attacks. This article details how CORS and Origin work in the context of FastReport Cloud and explains browser security principles.

What is CORS and why is it important?

CORS (Cross-Origin Resource Sharing) is a browser security mechanism that determines whether a web page can request resources from a different domain. Without CORS, browsers block cross-domain requests by default according to the Same-Origin Policy.

Same-Origin Policy

A browser considers two URLs to belong to the same origin if they match:

- Protocol (https/http)
- Domain name
- Port

For example:

- `https://app.fastreport.cloud` and `https://api.fastreport.cloud` are different origins
- `https://example.com` and `https://subdomain.example.com` are different origins

CORS Policy in FastReport Cloud, FastReport Corporate Server, FastReport Publisher

FastReport Cloud, FastReport Corporate Server, FastReport Publisher implements a differentiated approach to CORS depending on the user's authorization state:

Unauthorized Requests

For unauthorized users, the CORS policy is the most lenient:

- Cross-domain requests to the cluster are allowed
- The Origin header is not required
- This allows the use of public APIs and functions without restrictions

Authorized Requests with Origin

Strict rules apply to authorized users:

- Basic Authentication (`Authorization: Basic ...`)
- Bearer Token (`Authorization: Bearer ...`)
- The Origin header must be specified to receive permission from the browser

Cookie-based Authorization

To enhance security, the use of cookies in cross-domain requests (including when loading content into an iframe from a different origin) is restricted. This prevents automatic sending of session cookies in potentially unwanted requests. To embed reports via iframe, it is recommended to use an ApiKey, which allows bypassing

these restrictions.

Browser Security Mechanisms

FastReport Cloud, FastReport Corporate Server, FastReport Publisher correctly handles requests and returns the appropriate CORS headers.

CORS Headers

The server returns the following headers:

```
Access-Control-Allow-Origin: https://your-domain.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: *
Access-Control-Allow-Methods: GET, PUT, POST, DELETE, PATCH, OPTIONS
```

Setup and Usage

Origin Configuration

To ensure the system works correctly, you need to:

1. Specify the list of allowed Origins (Domains) in the workspace
    - Maximum of 10 domains
    - Only the domain needs to be specified (e.g., `your-domain.com`). FastReport Cloud, FastReport Corporate Server, FastReport Publisher will automatically match the incoming `Origin` header against this list. Wildcards (e.g., `*.example.com`) are not supported; specific domains must be listed.
    - The protocol (https) does not need to be specified, as the protocol will also be determined automatically.
2. Using the Origins (Domains) list
    - The list of allowed domain names is configured by the administrator.
    - When a request is received from an authorized user, the cluster checks the request's `Origin` header against this list and, if there is a match, returns the appropriate CORS headers.

Configuration Examples

For web applications:

```
Origin: https://your-app.com
```

For stand-alone applications and integrations:

Only pass the header if you implement standard browser security.

For API requests with authorization:

Practical Recommendations

For Developers

1. Always specify the Origin header for authorized requests (actually, the browser does this automatically for cross-domain requests)
2. Check the list of allowed Origins in the workspace settings
3. Use HTTPS for all domains in the Origins list
4. Test CORS requests in the browser, not just via command-line tools

For Administrators

1. Regularly review the Origins (Domains) list
2. Remove unused domains
3. Monitor for domain name changes
4. Update Origins (Domains) when migrating applications

Protection Against Common Attacks

CSRF (Cross-Site Request Forgery)

The CORS policy prevents malicious requests from being executed on behalf of an authorized user from other sites.

XSS (Cross-Site Scripting)

Limiting Origins reduces the risk of using XSS vulnerabilities to steal data via cross-domain requests.

Clickjacking

Using proper `X-Frame-Options` and CORS headers helps prevent clickjacking attacks.

Conclusion

The CORS policy in FastReport Cloud, FastReport Corporate Server, FastReport Publisher provides a balanced solution that ensures:

- Maximum security for authorized users
- Ease of use for unauthorized requests
- Flexible configuration for various usage scenarios
- Compatibility with modern browsers and their security mechanisms

Proper configuration of Origins and adherence to CORS rules allows FastReport Cloud, FastReport Corporate Server, FastReport Publisher to be securely integrated into any web application, ensuring user data protection and compliance with modern web application security standards.