Using Adobe FrameMaker Publishing Server
# Contents

## Getting Started
- Resources .................................................. 1
- System requirements and licensing ............................. 1
  - System requirements .............................................. 1
  - Installation .................................................. 2
  - License activation ............................................ 2
- Help and support ............................................. 2
  - Community Help ................................................ 2
  - Other resources .............................................. 3

## Introduction
- FrameMaker Publishing Server .................................... 4
- What's new in FrameMaker Publishing Server ...................... 4
- Other features in FrameMaker Publishing Server .................. 5

## Installation and setup
- System requirements ............................................... 8
- Plan the deployment architecture .................................. 8
  - Vertical scaling ............................................... 9
  - Horizontal scaling ......................................... 10
  - Mixed scaling ............................................... 11
- Install and configure MongoDB .................................... 11
- Install FrameMaker Publishing Server ............................ 13
- Configure the server component .................................. 14
  - Stop the server component ................................... 14
  - Configure the server_url.txt file .............................. 15
  - Configure the development.json file .......................... 15
  - Launch the server using the run.bat file ..................... 18
- Configure the client components .................................. 18
  - Stop the client component ................................... 19
  - Configure the server.ini file ................................ 19
  - Configure the FrameMaker instance ............................ 20
**REST APIs for working with tasks**

- Create a task .................................................. 46
- Delete a task .................................................. 46
- Get all tasks ................................................... 55
- Get a specific task ............................................. 56
- Get the status of a task of a specific user .................. 57
- Update a task ................................................... 58
- Upload a post-publish script file ............................. 59
- Upload a prepublish script file ............................... 60

**REST APIs to work with scheduling tasks**

- Schedule a task ............................................... 62
- Get ID of a running task ...................................... 62
- Stop a running task ............................................ 63
- Get running object details for a GUID ....................... 64
- Get all objects present in the running task queue .......... 65
- Get the details of a specific queued object .................. 66

**Other REST APIs to work with the server and logs**

- Get the server status ......................................... 67
- Get server connection parameters .......................... 67
- Enable or disable logging and change log level .......... 68
- Get maximum number of configured logs .................. 68
- Get last $n$ number of logs for a specific task .......... 70
- Get logs for a specific task by its run ID ................. 70
- Get all clients (or workers) .................................. 71

**Access API documentation** ................................ 72

**FMPS troubleshooting** ........................................ 73

**Legal notices** ..................................................... 76
Getting Started

Adobe FrameMaker Publishing Server (FMPS) is an enterprise software that allows you to automate your multichannel publishing process. Using the horizontal and vertical scaling architecture, FMPS can run on one or multiple remote systems. You can automate output generation in multiple formats: PDF, Responsive HTML5, EPUB, Kindle, Microsoft HTML Help, Basic HTML, and Mobile Apps. Leverage out-of-the-box support for leading Content Management Systems (CMS), or use the web APIs to smoothly integrate with other CMSs.

Resources

Before you begin working with Adobe FrameMaker Publishing Server, take a few moments to read an overview of activation and the many resources available to you. You have access to instructional videos, plug-ins, templates, user communities, seminars, tutorials, RSS feeds, and much more.

System requirements and licensing

To review complete system requirements and recommendations for your Adobe FrameMaker Publishing Server software, see the FrameMaker Publishing Server product page at www.adobe.com/products/fmserver.

System requirements

FrameMaker Publishing Server is available in English only. Before you install FrameMaker Publishing Server ensure that your computer meets the following minimum system requirements:

Hardware
   64-bit 4 vCPU with Core i5 or faster processor

Operating System
   64-bit Microsoft® Windows® Server 2019 or 2016

RAM
   8 GB or higher

Hard Disk Space
   6 GB of available hard-disk space; additional free space required during installation (cannot install on a volume that uses a case-sensitive file system or on removable flash storage devices).

NOTE: If you copy your source documents to the server that hosts FrameMaker Publishing Server, allocate additional disk space.
Screen Resolution
Maximum supported screen resolution – 4K (3480 x 2160, 8.3 megapixels, aspect ratio 16:9).

Activation
FrameMaker Publishing Server does not operate without activation. Internet connection and registration are required for software activation, validation of subscriptions, and access to Online Services. Phone activation is not available.

Pre-requisite
You must have MongoDB Community Server version 4.0.19 or 3.6.19 installed and configured on your system before installing FrameMaker Publishing Server. For downloading MongoDB, visit https://www.mongodb.com/try/download/community.

Browsers
- Google Chrome
- Firefox Mozilla
- Microsoft Edge

Installation
You can install FMPS on a machine that meets the System requirements and then access it over the network using other machines. For more information on installation, see Installation and setup.

License activation
During the installation process, your Adobe software contacts an Adobe server to complete the license activation process. No personal data is transmitted. For more information on product activation, visit the Adobe website at www.adobe.com/go/activation.

Help and support
Understand the various Help resources for you to get started with FMPS and other related products like FrameMaker.

Community Help
Community Help is an integrated environment on Adobe.com that gives you access to community-generated content moderated by Adobe and industry experts. Comments from users help guide you to an answer. Search the FrameMaker Community Help to find the best content on the web about Adobe products and technologies, including these resources:
- Videos, tutorials, tips and techniques, blogs, articles, and examples for designers and developers. Check the publishing workflow videos on the FrameMaker video hub page.
• Complete online Help, which is updated regularly and may contain more information than the Help delivered with FrameMaker Publishing Server.
• All other content on Adobe.com, including knowledge-base articles, downloads and updates, Adobe Developer Connection, and more.

Other resources

Online Help also include links to the complete, updated PDF and HTML versions of Help for FMPS and other related products.
Visit the Adobe Support website at www.adobe.com/support to learn about free and paid technical support options.
Introduction

FrameMaker Publishing Server

With centralized and automated publications, it is easy to integrate and synchronize publication of technical documents with product build schedules. With support for scalable architecture, you can deploy FMPS on one or multiple systems. In a scalable setup, the server component is configured to communicate with the FrameMaker instances deployed on one or multiple systems.

Multiple users can remotely access FMPS simultaneously and run various operations such as creating presets and creating and scheduling publishing tasks. FMPS leverages out-of-the-box integration with leading Content Management Systems (CMS) like OpenText Documentum, Microsoft SharePoint, DitaExchange and Adobe Experience Manager. Using the FMPS APIs, you can smoothly integrate with other CMSs as well to enable you to manage publication tasks and enforce version control.

With ExtendScript Toolkit integrated with FMPS, you can create custom publication tasks that work with source files in XML/DITA and FrameMaker formats. By creating reusable publication task settings, you can quickly set up custom publication environments and schedule the build processes to run daily, weekly, or at any defined intervals.

Watch this video Overview of FrameMaker Publishing Server for more information about the latest release.

What’s new in FrameMaker Publishing Server

Horizontal and vertical server scaling

Adobe FrameMaker Publishing Server is a very powerful solution that helps enterprises automate their large-scale publishing tasks. This solution has now been given extra strength by allowing you to scale it up to multiple systems. With the much-improved scalable architecture, you can now deploy FrameMaker Publishing Server in one of the following modes:

- Horizontal scaling
- Vertical scaling
- Mixed scaling or a combination of horizontal and vertical scaling

For more information, see Plan the deployment architecture.

WYSIWYG PDF for DITA content

Creating WYSIWYG PDF output is even more simpler now. With the settings file saving and controlling all publishing parameters, you can leverage the new DITA publishing engine to create WYSIWYG PDFs. Use FrameMaker to make changes to the PDF output settings, and then import the settings file in FrameMaker Publishing Server to generate output for multiple DITA maps and book-maps.
With the new capabilities to generate PDF/A- and PDF/X-compliant PDFs, you can add more flavors to your PDF output. Moreover, you no longer have to create separate templates and associate them with the publishing presets. The entire publishing template information is also stored in the settings file, which saves your configuration and publishing time.

**Reimagined architecture**

FrameMaker Publishing Server has been rewritten from scratch. With the completely new architecture that support horizontal and vertical scaling, superior performance and queue management, and new server components, your large-scale publishing tasks couldn't have been much easier and simpler.

The newly designed architecture also removes the dependency on Internet Information Server. Now, FMPS uses open source technologies to deliver its functionalities. In place of Windows-based file system, now FMPS uses MongoDB to store all your publishing tasks and configurations.

**Scripting support for automation**

The scripting support in FrameMaker Publishing Server has been further enhanced to let you create and use ExtendScript or batch script to publish the output. You can define scripts to execute before publishing the output, such as for applying conditional text settings, or even after the output is generated. Using the pre- and post-publishing scripts give you more control on what output you want to generate and how you want to generate it.

**Other features in FrameMaker Publishing Server**

**Customize DITA templates for PDF**

FMPS can now store and use DITA templates for PDF output. You can store your PDF output settings in a settings (.sts) file and easily make changes to it. The same settings file can then be exported and imported into a new publishing task.

**Responsive HTML5 layouts**

Create modern Responsive HTML5 layout to deliver a superior navigation experience. Also, the responsive HTML5 output comes with first-of-its kind search experience. The newly redesigned responsive HTML5 output now displays predictive search results based on a few characters that you type in the search box. This reduces the search time and improves the usability of your Help system.

**Basic HTML output**

You can now publish in Basic HTML format from FMPS. The Basic HTML output format takes each topic file within your book or DITA map and generates a corresponding HTML file with the similar look-and-feel controlled through a CSS file. The generated output does not contain any JavaScript or custom layouts that can be re-purposed or customized for the intended audience.

**Enhanced native integration with AEM**

XML Documentation Add-on for Adobe Experience Manager is an end-to-end enterprise-class component content management solution (CCMS). The XML Documentation Add-on integrates with FMPS that allows you to publish DITA content to formats like Responsive HTML5, PDF, EPUB, Kindle and more.
Enhanced template-based publishing solution

Quickly automate document assembly and formatting by defining multiple output template presets for generating output. The latest version of FMPS comes with HTML5 template that helps you generate HTML5 output without investing time and effort in creating a new template.

Bidirectional language publishing

Publish documents with bidirectional content, including XML/DITA content, from within FrameMaker in multiple formats: PDF, Responsive HTML5, EPUB, Kindle, Microsoft HTML, Basic HTML, Adobe Experience Manager Mobile and as native mobile apps. Optionally, flip the HTML5 layout so that the navigation pane appears on the right for RTL content, and render smoothly on any device - desktop, tablet, or a mobile.

Personalized dynamic content

Empower users to find relevant content faster in the document, TOC, and index in the final Responsive HTML5 or mobile app output. Leverage existing conditional tags and expressions to enable end users to dynamically filter content in the final HTML5 output, thereby delivering personalized help experiences.

Simply assign names for existing conditional tags or expressions, categorize them under any Group you want (such as region, audience, market segment etc.), enable single or multiple selections within a Group, and then display this two-level hierarchy as content filter criteria in the final output.

Customizable HTML5 Layout

Use the customizable Responsive HTML5 layouts to deliver an amazing content consumption experience. With more powerful search options, configure search to appear on the content or topic panes, and show informative context and breadcrumbs in the results. This helps your readers identify the best choice among results. You can also customize the search context message for each topic. The search results can also be configured to appear on the left pane.

Easily show or hide widgets such as glossary, TOC, index, and filters. Convert the layout to right-to-left languages (Arabic and Hebrew) to meet the needs of a global audience, just by changing one property. Also add Facebook and Twitter widgets to your pages to allow users to share the content that they are reading – with a single click!

Provide your users with a different experience for the device (desktop, tablet, and mobile) that they use by customizing the function bar, side bar, and TOC in the output.

Use the powerful, yet easy-to-use Layout Customization tool to customize not only your content, but also the appearance of buttons, labels, fonts, background color, search boxes, and most components of your online content.

Reference page image publishing

Publish reference pages content such as images, headers and footers, and logos, in all output formats supported by FrameMaker Publishing Server.
SVG support in HTML5 output

Let your end users see top quality images regardless of the screen size and resolution of the device on which they are being viewed. This is because FrameMaker Publishing Server embeds the entire SVG code in the final responsive HTML5 output.

Reusable publishing elements

Reuse elements easily with the enhanced multi-device publishing settings file. Create elements, such as responsive HTML5 layouts, CSS, and HTML page templates, just once. Then import or export the setting files for use across different books and documents.

HTML page template support

Improve the usability of your output by adding mini-TOCs, breadcrumbs, and headers and footers. Configure the output settings just once and then easily reuse these across all outputs to give your users a uniform experience.

Integration with leading CMSs

Leverage the out-of-the-box integration with leading Content Management Systems (CMS) like OpenText Documentum, Microsoft SharePoint, DitaExchange and Adobe Experience Manager. You can also use the FMPS APIs to smoothly integrate with other CMSs to enable you to manage publication tasks and enforce version control. Use the Repository view to check-out files, manage dependencies, version management and perform search. Use the advanced metadata-based filter to quickly browse and find relevant topics, DITA maps, or graphics in a large volume of content.
Installation and setup

To work with FrameMaker Publishing Server, do the following:

1) **System requirements**
2) **Plan the deployment architecture**
3) **Install and configure MongoDB**
4) **Install FrameMaker Publishing Server**
5) **Configure the server component**
6) **Configure the client components**
7) **Common customizations for FrameMaker Publishing Server**
8) **Access FrameMaker Publishing Server**

You can also install a free command line tool, such as cURL, to access FrameMaker Publishing Server using the command line.

**NOTE:** For information on issues during installation, see *FMPS troubleshooting*.

**System requirements**

Before installing FrameMaker Publishing Server, make sure that you have the required hardware and software. See *System requirements* for more details.

**Plan the deployment architecture**

The new and much improved scalable architecture of FrameMaker Publishing Server (FMPS) allows you to maximize your system utilization. FMPS is designed to support three types of scaling architecture:

- Vertical scaling
- Horizontal scaling
- Mix scaling or a combination of vertical and horizontal scaling
**Vertical scaling**

The following illustration shows the vertical scaling deployment architecture:

In vertical scaling, FrameMaker Publishing Server component, client component, FrameMaker, and database are installed on the same system. In this architecture, for each publishing task, a new instance of FrameMaker is launched. The number of FrameMaker instances that a system can launch depend on the number of cores. For example, on a 4-core machine, FMPS can launch 1 instance of FrameMaker. On an 8-core machine, there can be a maximum of 3 instances of FrameMaker that can run simultaneously. In case the number of publishing tasks is more than the number of FrameMaker instance, then FMPS keeps such tasks in a queue. Once an existing task completes, then the task in the queue is assigned to the available instance of FrameMaker. This way, FrameMaker Publishing Server also acts as a load balancer.
Horizontal scaling

The following illustration shows the horizontal scaling architecture:

In horizontal scaling, FrameMaker Publishing Server component, client component, FrameMaker, and database are installed on different systems. As shown in the above illustration, FMPS is installed on System 1, and a single instance of FrameMaker along with client component is installed on System 2, System 3, and System 4. You can have as many instances of FrameMaker as you need, but you need only a single instance of FrameMaker Publishing Server. The load balancing happens in a similar way as described in vertical scaling.
**Mixed scaling**

The third possible architecture is a mix of horizontal and vertical scaling:

In a mixed scaling architecture, a single instance of FrameMaker Publishing Server controls multiple instances of FrameMaker spread across multiple systems. As shown in the preceding illustration, System 2 has two instances of FrameMaker on it and System 4 has three instances of FrameMaker, each communicating with a single instance of FrameMaker Publishing Server.

**Install and configure MongoDB**

As FrameMaker Publishing Server uses MongoDB for data storage, you need a working setup of MongoDB on your system before installing FMPS.

To install MongoDB, perform the following steps:

1) Download MongoDB Community Server version 3.6.19 or 4.0.19 from the official website:  
   https://www.mongodb.com/try/download/community

2) Configure the MongoDB Service as follows:
   - **Service Name**: MongoDB (keep the default name)
   - **Data Directory**: Any directory outside the Program Files folder where all programs and users would have the read and write access.
   - **Log Directory**: Specify a directory to store the logs.

3) Configure the IP address of the MongoDB Service in the `mongod.cfg` file. By default, the `mongod.cfg` file is available in the following location:
   `<Drive>:\MongoDB\Server\<version>\bin\`

   *Change the default IP address from 127.0.0.1 to 0.0.0.0. This will ensure that MongoDB can be accessed from any other system by using the host system’s IP address and port.*
4) Create the default user account with administrative privileges to access MongoDB.

To create the basic user account, run the following commands:

a) Open the command prompt.
b) Change the working directory to "%INSTALLDIR%\mongodb\bin".
c) Run "mongod.exe".
d) Run "mongo.exe".
e) In the MongoDB interface, run the following script:

```
use admin
db.createUser(
    { user: "fmadmin", pwd: "fmadmin", roles: [
      { role: "userAdminAnyDatabase", db: "admin" },
      { role: "readWriteAnyDatabase", db: "admin" },
      { role: "dbAdminAnyDatabase", db: "admin" },
      { role: "clusterAdmin", db: "admin" },
      { role: "root", db: "admin" },
      { role: "readWrite", db: "admin" }
    ]}
)
```

If you are using LDAP-based authentication, then you need to create and assign administrative privileges to a LDAP user ID. This is a one-time activity. For the subsequent requests, you can use this user account to grant privileges to other users via API. To assign administrative privileges to a LDAP user ID, perform the following steps:

NOTE: The following instructions are using the CLI. However, you can also use MongoDB Compass, which is free a UI tool to access and work with MongoDB.

1) Open the command prompt.
2) Change the working directory to "%INSTALLDIR%\mongodb\bin".
3) Run "mongod.exe".
4) Run "mongo.exe".
5) In the MongoDB interface, run the following script:

```
use stubFM;db.users.find();
```

Note the user ID that you want to update.
6) Run the following script:

```
db.users.update(
    { _id: ObjectId("5a321b0c602d2e16677760f5") },
    {$set: {
      userPermission: "ADMIN"
    }}
```

If you are using LDAP-based authentication, then you need to create and assign administrative privileges to a LDAP user ID. This is a one-time activity. For the subsequent requests, you can use this user account to grant privileges to other users via API. To assign administrative privileges to a LDAP user ID, perform the following steps:

NOTE: The following instructions are using the CLI. However, you can also use MongoDB Compass, which is free a UI tool to access and work with MongoDB.
This script will update the user with an ID of “5a321b0c602d2e16677760f5” with administrative privileges.

Install FrameMaker Publishing Server

To install FMPS, perform the following steps:

1) Run the FMPS installer. The following Installation Options dialog appears:

2) Based on your deployment strategy, you can choose to:
   - Install only the server component (for horizontal scaling)
   - Install only the client component and FrameMaker (for horizontal scaling)
   - Install all components (for vertical scaling)

If you want to install only the server component, then choose only the second component - Adobe FrameMaker Publishing Server Component.

If you want to install only the client components, then choose the first and the third options - Adobe FrameMaker and Adobe FrameMaker Publishing Client Component.
3) If you want to change the default install location, click the Browse icon next to the Location field and select a new destination.

4) Click **Continue**.

   *The selected FMPS components are installed on the system.*

5) Once the installation is complete, click **Close**.

**IMPORTANT:** By default, on completion of the installation process, FrameMaker Publishing Server and client are launched. It is configured to work in case you have installed all components on a single system (vertical scaling). However, if you have deployed FMPS on multiple systems (horizontal scaling), then you will have to configure FMPS server and client components to work correctly.

Next, you need to customize the server and client components before you can start using FrameMaker Publishing Server. The following sections will guide you through the configuration and customizations that you need to perform.

### Configure the server component

The system where you have deployed the server component contains the files and folders required to configure and launch the server. The typical process of configuring the launching the server component involves the following steps:

1) **Stop the server component**
2) **Configure the server_url.txt file**
3) **Configure the development.json file**
4) **Launch the server using the run.bat file**

The following sections will guide you through the above-mentioned steps.

### Stop the server component

Before you start making configuration changes in the server component, ensure that the server component is not running. If the server component is running on your system, you will see a command prompt window with “FMPS” title as shown below:

![Command Prompt Window](image)

To stop the server component, simply close the command prompt window with “FMPS” title by clicking on the “X” icon.
Configure the server_url.txt file

The server_url.txt file contains the fully qualified domain name (FQDN) and port of the system where your server component is deployed. The default location of server_url.txt file is:

\Program Files\Adobe\AdobeFrameMakerPublishingServer2020\server\server_url.txt

You can open this file in a text editor and provide your server’s host name and port information in the following format:

http[s]://<fully-qualified-domain-name_or_ip-address>:<port>

**NOTE:** By default, when you complete the installation process, the FQDN of the server machine is fetched and entered in the server_url.txt file. However, if the installation process is not able to get the FQDN, then localhost is added in the server_url.txt file, which should be manually updated with FQDN.

Configure the development.json file

The development.json file contains most of the configurable settings for FrameMaker Publishing Server. It allows you to configure the MongoDB settings, user credentials for default users, logging information, and much more.

The following snippet of the development.json file highlights the configuration settings that you can modify as per your deployment setup.

```json
{
    "FMPS": {
        "dbConfig": {
            "MONGO_HOST": "localhost", // The hostname of MongoDB server. Default is localhost.
            "MONGO_PORT": 27017, // The port where MongoDB is listening. Default is 27017.
            "dbName": "stubFM",
            "dbNameLog4j": "logs",
            "dbNameQueue": "stubFM_Queue",
            "username": "fmadmin", // Basic username and password that is created to access MongoDB.
            "password": "fmadmin",
            "authMechanism": "SCRAM-SHA-1", // Authentication mechanism to use to connect with MongoDB. If you are using MongoDB version 3.6.x, then set it to "PLAIN" or "NONE". For version 4.x, you can choose from SCRAM-SHA-1 or none.
        }
    }
}
```
"authSource": "admin",

"SCHEMA_VERSION": 16, //This is used to identify the schema. Please Do Not Change.",

"defaultUser": "fmpsuser", //Default username and password to access FMPS as an Administrator.

"defaultPassword": "fmpsuser",

"defaultEmail": "fmpsuser@enterprise.com", //Default email ID to send notifications. You must replace it with a valid email ID.

"LogQueue": "logqueue",

"LogMonQueue": "logmongoqueue",

"MaxNoOfLogs": "5", //Maximum number of task logs that are returned by the API.

"logLevel": "INFO" // Level of logs to maintain. Supported values are: FATAL, ERROR, WARN, INFO, DEBUG. By default INFO level logs are enabled.

"webConfig": {
    "apiVersion": "v16", // This is used to identify the API versions. Please Do Not Change

    "host": "localhost", // The hostname of the server where FMPS is deployed. If FMPS is started by executing the run.bat file, then the value of this setting is picked from the server_url.txt file. Else, you need to change it manually.

    "port": 7000, // The port where FMPS is listening. If FMPS is started by executing the run.bat file, then the value of this setting is picked from the server_url.txt file. Else, you need to change it manually.

    "sslPort": 6234, // If FMPS is deployed on HTTPS, then specify the port on which it is listening. If FMPS is started by executing the run.bat file, then the value of this setting is picked from the server_url.txt file. Else, you need to change it manually.

    "protocol": "http://", // Specify the protocol “http://” or “https://” where FMPS is deployed. If FMPS is started by executing the run.bat file, then the value of this setting is picked from the server_url.txt file. Else, you need to change it manually.

    "typeOfAuth": "USERLOGIN", // Type of authentication to use with FMPS. Supported values are: USERLOGIN, ANONYMOUS, LDAP. The default
The authentication mechanism is basic username/password (USERLOGIN). If you change this, then the server restart is required.

"daysForTokenExpiryUserLogin": 7, // Days for token to expire used in USERLOGIN and LDAP.

"useSSL": false, // By default SSL is not enabled. In case FMPS is setup on SSL, change this to true.

"taskTimeOut": 60, // If the queued task is not picked up within the specified time (default 60 minutes), then it will retire and is pushed back with +1 number of retries.

"runningTaskTimeOut": 200, // This is the timeout for running task, which means if a publishing task does not complete in 200 minutes, it will retire and is pushed back with +1 number of retries.

"maxRetries": 5, // Max number of retries for a publishing task.

"defaultTimezone": "Atlantic/Reykjavik" // Default time zone for FMPS. Supported values are: Atlantic/Reykjavik for GMT, Asia/Kolkata for India, America/Los_Angeles for PDT, America/New_York for EDT, see https://momentjs.com/timezone/ for time zone information. Ideally this should align with the Server time zone.

"cronConfig": {
  "workscheduler": "*/10 * * * *",
  "TaskPoller": "*/5 * * * *",
  "TaskPollerFailed": "*/40 * * * *"
},

"ldapConfig": { // If you are using LDAP-based authentication, then update the settings in this section.
  "ldapurl": "ldaps://global.enterprise.com:636", // Specify the LDAP URL that is used for authentication.
  "baseDN": "cn=users,dc=enterprise,dc=global,dc=enterprise,dc=com", // LDAP domain name string.
  "tlsOptions": false, // Set as “true” if LDAP uses TLS, else set it to “false”.
  "domainName": "enterprise" // Specify the domain name that is appended with the username. For example, domainname\username.
},

"smtpConfig": { // Settings in this section are used to configure the email server that is used to send email notifications.
  "host": "smtp.enterprise.com", // Specify the SMTP hostname.
  "smtpport": 25, // Specify the SMTP port.
"username": "fmpsuser@enterprise.com", //Specify the SMTP username that is used to send email notifications.

"password": "fmadmin", //Specify the password for the SMTP user.

"requirePassword": false, //Set to true if password is required by the SMTP server and a value for the password settings is provided.

"disableSMTP": true, // By default SMTP is disabled. If you have configured the SMTP server, then set this setting to false.

"triggerAlertEmail": false, //Set to true if you want to enable email notifications for events such as high memory usage or server restart. By default, it is disabled (set to false).

"alertEmail": "fmpsuser@enterprise.com"  //A comma separated email IDs of users to whom an alert email notification is sent.

"appConfig": {
   
   "Company": "Adobe",
   "Product": "FrameMakerPublishingServer",
   "NodeLogs": "NodeLogs",
   "Version": "16",
   "MaxMem": 4096,
   "lengthOfLogMessage": 10000       // Specify the total length of a log message in characters

   "_comment_For_AppConfig": "Settings in this section are used to setup application folders and logs."

   }
}

Launch the server using the run.bat file

To launch the server component, you need to run the run.bat file available at the following location:
\Program Files\Adobe\AdobeFrameMakerPublishingServer 2020\server\run.bat

Typically, you would launch the server only after completing all configurations required for the server as well as the client components. The following section will guide you to configure the client components.

Configure the client components

A client is system where you have installed Adobe FrameMaker and Adobe FrameMaker Publishing Client Component. The client has to establish a connection with the server to be able to receive information about what needs to be published and send a response back.
The typical process of configuring and launching the client component involves the following steps:

1) **Stop the client component**
2) **Configure the server.ini file**
3) **Configure the FrameMaker instance**
4) **Launch the client using the StartWorker.bat file**

The following sections will guide you through the above-mentioned steps.

**Stop the client component**

Before you start making configuration changes in the client component, ensure that the client component is not running. If the client component is running on your system, you will see one or more command prompt windows with “FrameServerEx” title as shown below:

To stop the client component, simply close all command prompt windows with “FrameServerEx” title by clicking on the “X” icon.

**Configure the server.ini file**

The server.ini file contains the parameters to connect to the server component of FMPS. Open the server.ini file in a text editor and configure the following parameters:

**port**

Specify the port on which the server component is listening for incoming requests.

**server**

Specify the fully-qualified domain name or the IP address of the server.

**scheme**

Specify the protocol (http or https) used by the server.

**MaxFMSessions**

Specify the maximum number of concurrent FrameMaker sessions that the can run on a client machine. By default it is set to 4. This setting works in conjunction with the VAR variable setting in the StartWorker.bat file. For more details, see *Launch the client using the StartWorker.bat file.*
doNotDeleteGeneratedFile

Supported values - On and Off. By default, it is set to On, which indicates that the publishing process will not delete any existing file on the given location where the output is saved.

HTTPTimeout

Specify the time in seconds after which the connection with the server machine is timed out. This happens when no response is received from the server machine during the configured time.

Delay

Specify the time in seconds that the client machine waits for before launching another instance of FrameMaker.

Configure the FrameMaker instance

To optimize the performance of FrameMaker in automated publishing tasks, make the following changes to FrameMaker configuration. These changes are essential to run scheduled tasks on FMPS and modify the behavior in the following ways.

FMPS ignores unresolved cross-references, unresolved text insets, and missing graphics.

• To use MathML, accept the EULA in FrameMaker.
• Check all source documents for errors before initiating a build task.
• FMPS does not report missing fonts. Ensure that the machine that hosts FMPS client component has all the required fonts.

Launch the client using the StartWorker.bat file

The StartWorker.bat file is used to launch FrameMaker instances (or workers) on the client machine that listen to the publishing requests from the server component. Each worker opens and works with a single FrameMaker instance to complete a publishing task. Once the task is complete, the FrameMaker instance is closed by the worker. The MaxFMSessions parameter in the server.ini file controls how many maximum instances of FrameMaker can run parallely. The settings in StartWorker.bat file defines how many instances of FrameMaker will run on a given client machine. For example, say you have set the MaxFMSessions parameter to 4 and in the StartWorker.bat file you have defined 8 workers to run. In this case, the client machine can only run 4 instances of FrameMaker at any given time, even though it has launched 8 workers. If the server sends 5 publishing commands to this client system, then 4 publishing tasks will start immediately, and the 5th task will be pushed to a queue. The moment any task gets complete, then the next available worker will take up the 5th task from the queue and start executing it.

The contents of StartWorker.bat file are:

```
%~d0
 cd %~dp0
SET /a VAR=0
:HOME
SET /a VAR=VAR+1
```
In this file, the line - IF %VAR%==2 goto :End controls how many worker instances will be launched on the client machine. Let’s say, you want to launch 4 worker instances, then you must configure it as:

IF %VAR%==5 goto :End

Note that the value of VAR is always 1+ (or one more) than the number of workers you want to launch on the client machine.

After configuring the maximum number of FrameMaker instances that can run simultaneously and the number of workers, you are ready to launch the client component. To launch the client component, you need to run the StartWorker.bat file available at the following location:

\Program Files\Adobe\AdobeFrameMakerPublishingServer 2020\FrameServerExe\StartWorker.bat

Common customizations for FrameMaker Publishing Server

The following additional customizations are required on the systems where you have deployed the server or client components:

1) Turn off the IE Enhanced Security Configuration.
   See FAQ about Internet Explorer Enhanced Security Configuration (ESC) in Microsoft documentation.

2) Add rules for inbound and outbound ports in Windows firewall. Also ensure that the ports on which you have deployed FrameMaker Publishing Server and MongoDB are opened for listening. By default, FMPS runs on port 7000 and MongoDB runs on port 27017.
   See Create an Inbound Port Rule and Create an Outbound Port Rule in Microsoft documentation.

3) Allow FrameMaker.exe, FrameMakerEx.exe, and node.exe to run through the firewall.

4) Change the following security and privacy settings for Internet Options.
   NOTE: You can access the Internet Options by opening the Control Panel and clicking Internet Options.
   – In the Security tab, set the security to Medium-high.
   – In the Security tab, click Custom level, and ensure that the following settings are Enabled:
     • In the ActiveX controls and plug-ins section: Allow Scriptlets and Binary and scripts behaviors.
     • In the Downloads section: File download and Font download.
     • Enable .NET Framework setup.
In the **Miscellaneous** section: **Access data sources across domains** and **Display mixed content**.

- In the **Scripting** section: **Active scripting**.
  - In the **Privacy** tab, click **Advanced** and configure the following settings:
    - **First-party Cookies**: Accept
    - **Third-party Cookies**: Accept
    - **Always allow session cookies**: Enabled

5) Enable the .NET Framework 3.5 in Control Panel.

### Access FrameMaker Publishing Server

Multiple users can simultaneously access FMPS by logging in to the server that hosts it. Using the web-based user interface of FMPS, you can access it remotely from any machine. To launch and access FMPS, perform the following steps:

1) On the system where you have deployed the server component, start the server by double-clicking the `run.bat` file.

2) On the system where you have deployed the client components, start the client by double-clicking the `StartWorker.bat` file.

3) Access the FMPS web interface by entering the following URL in your browser:

   ```
   http[s]://<FMPS_server>:<port>/index.html
   ```

   **NOTE:** `<FMPS_Server>` is the DNS or IP address of the system hosting FMPS. The default port is 7000.

### FrameMaker Publishing Server dashboard

FMPS provides a simple, web-based user interface using which you can:

- Create, manage, and schedule publication tasks
- Create and use presets to quickly apply and populate settings to the repetitive tasks with similar settings
- Manage and view task logs
- Use in-app API documentation to create custom scripts for scheduling and publishing tasks

Once you have installed FMPS on a machine, you can access it on the network from any machine to use it.
A. Add New <Task/Preset> drop-down - Depending on which tab you are in, this drop-down allows you to create and duplicate Tasks and Presets.

B. The Run button: To run tasks, select one or more tasks and click Run.

**NOTE:** If you have scheduled a task to run, then it cannot be added to the scheduled queue again. If you want to run the task using the Run button, then you must first remove the scheduled task from the task scheduling queue.

C. Incremental search: Search for a Task or Preset name (depending on the tab you are on) - as you type text, FMPS finds matches the text and displays them in the list.

D. Autorefresh settings: You can enable autorefresh for FMPS and specify the number of seconds after which FMPS automatically refreshes.

By clicking on the column names, you can sort the list of tasks by that column. The tasks are color coded: green are successful, red are failed, and orange are queued.

For more information on:
- Using FMPS through a browser on a client machine, see *Create and manage publication tasks*.
- Using FMPS through the APIs on a client machine, see *Work with FMPS using REST APIs*.

**Related topics**
- *Create and manage publication tasks*
- *Tasks*
Autorefresh of the FMPS interface

The FMPS web interface’s autorefresh is disabled by default. You can set it to autorefresh every 10-500 seconds.

1) Click **Settings**

   *The Settings dialog appears.*

2) In the Settings dialog, select **Duration** and enter the number of seconds after which you want FMPS web interface to autorefresh. You can enter a number from 10 to 500.

   Configure Autorefresh in the Settings dialog.

3) Click **Save**.

   **NOTE:** You can also manually refresh FMPS by clicking the refresh button on the FMPS dashboard.
Create and manage publication tasks

You can create publishing task that can be scheduled to run at off-peak times and on regular time intervals to ensure that published content is always current. The FMPS dashboard notifications keep you informed about pending jobs as you turn your attention to other publishing tasks.

Publishing

Tasks

You remotely define the publication tasks in Adobe FrameMaker Publishing Server and schedule them to run at the required interval. A task is a complete collection of all the build information that is required to run a publication task.

For more information, watch this video - FrameMaker Publishing Server: Automated publishing delivered remotely.

Typically, you define a task for each book or source document that you want to create outputs from. For example, if you have a book file from which you want to create Adobe PDF, Basic HTML, Responsive HTML5, or more outputs, specify the following settings for the Task:

Input Source

Location of the source document or book. You can specify sources from the local file system, a network or WebDAV folder, an OpenText Documentum repository, a Microsoft SharePoint repository, DitaExchange or Adobe Experience Manager (AEM).

If you are accessing the server from the same system where you have deployed it, then a local file path will work. However, in other cases, you must specify a network location or the location of your CMS repository. In case the source files are available on a network location of a CMS, you need to specify the User Name and Password of the user who has access rights on the files, else the publishing process will fail.

Output Settings > Format

Output formats required and their settings.

The supported output formats are: PDF, Responsive HTML5, Mobile App, ePub, Kindle, Microsoft HTML Help, AEM Mobile, and Basic HTML.
Presets

Presets for all supported output formats include the settings such as DITAVAL File and Settings File (.STS). These files are saved in MongoDB and attached to the task being created. Preset help you apply these settings to repetitive and similar tasks and save time.

![PDF Settings](image)

NOTE: For more information about configuring output presets using FrameMaker, see the Print and Publish topic in FrameMakerUser Guide.

Output Folder Path

The location where you want the output from the output generation process saved. You can specify the output location as a folder in the local file system, a network or WebDAV folder, or an OpenText Documentum, Microsoft SharePoint repository, DitaExchange, or AEM. Again, the Output Folder Path can be a local system if you are running the server and client components on a single system. For multi-system deployment scenario, you have specify a network location along with user credentials to access the network path.

Pre- and Post-Publish Script

The location of the script file that you want to run before and after the publishing process. You can specify a .jsx or .jsxbin file in the pre-publish script. However, post-publish scripts can contain scripts or batch files that can run from the command prompt only. The script file is uploaded into MongoDB and attached with this task.

Scheduling

Schedule to run the task and the frequency.

Send Log In Email

The Email IDs where you want the task logs to be sent.

Log Settings

The information you want to capture in the task logs.
Create a task

1) On the FrameMaker Publishing Server dashboard, select the Task tab.
2) In the Add New Task drop-down, select Add New Task.

3) On the new Task screen, specify a Name for the publishing task.

Tasks in the task queue are listed with the task name. Provide intuitive task names that indicate the type of document and the output formats to easily identify your task in the queue.

4) Click the browse button next to the Input Source field.

The Specify Input Source dialog appears. Enter the following details in it:

a) An input source file (.book, .ditamap, .fm, or .mif file). Enter full path and name of the input file.

b) Enter login credentials to access the input file, if required. For example, login information is required to access files over a network or a CMS.

c) If your input file is on a CMS, select the CMS option, choose the CMS type, and specify the CMS details, such as login information, server, and workspace.

5) To edit book settings, click the Modify link in front of Book Settings. In the Book Settings dialog, double-click (or use the move icons) to move the required entries in the Don’t Include and Include lists.
CREATE AND MANAGE PUBLICATION TASKS

PUBLISHING

6) To add an output type, choose an output format, preset, and then click the browse button next to the **Output Folder Path** field. The Specify Output dialog appears. Enter the following details in it:

**NOTE:** By default, FrameMaker Publishing Server does not overwrite existing files with the newly generated output. You can change this setting by turning the `doNotDeleteGeneratedFile` parameter to `Off` in the `server.ini` file. To keep the output files from an earlier publication task, move them to another folder. You can run a post-publish command as part of the publication task to move the output files to another location after the publication task completes.

a) Specify the destination file path where the output file is saved.

b) If required, enter login credentials to save the output file. For example, login information is required save output files on the network or on the CMS.

c) To save output file to a CMS, select **CMS** and enter the CMS details.

7) To add more output types, click **Add Another Output** and repeat step 6.

8) (Optional) Enter the path and name of a **Pre-Publish Script** to specify any automation script that you created.

9) (Optional) Enter the name and path of a **Post-Publish** script to specify any system command that you want to run on the outputs after the publication task is run. You can create a batch script and provide its path so that FrameMaker Publishing Server runs the script after the publication task completes.

10) Click the **Not Scheduled** link to open the **Schedule** dialog. Specify the time and frequency for the task to execute.

11) (Optional) In **Send Log in Email**, specify the email addresses where you want to send the task log. Use semicolons between multiple email IDs.

12) To add or delete information in the task logs, click **Modify** in front of **Log Settings**. In the Log Settings dialog, double-click (or use the move icons) to move the required entries in the **Don’t Include** and **Include** lists.

13) Click **Save**.

**Related Links:**

**Scheduling**

**Run tasks from the list**

On the FrameMaker Publishing Server dashboard, select the tasks from the task list and click **Run**.

**Duplicate a task**

An easy way to create multiple tasks is to make a copy of an existing task, and then make the necessary changes for the duplicated task.

1) On the FrameMaker Publishing Server dashboard, do one of the following:
   - Click a task’s name and in the Task dialog, click the **Duplicate** icon.
   - Select a task and then select **Duplicate Task** from the **Add New Task** drop-down.

   The Task dialog opens with the same task settings and name of the task duplicated with the suffix `-copy`.
2) Edit the name and other details of the task and click **Save**.

**Edit a task**

2) Make the required changes in the task and click **Save**.

**Delete a task**

- On the FrameMaker Publishing Server dashboard, select one or more tasks and click **Delete**.

**NOTE:** If you delete a task when it is currently running, the task is removed, but the current run is allowed to complete.

## Scheduling

Typically, you create publication settings for recurring tasks. With FrameMaker Publishing Server, you can create multiple publication tasks and schedule them to run at specified intervals. In a typical technical publications department, many teams create automated builds that are then run at the required intervals. You can specify the intervals as once, daily, weekly, or monthly. A publication task requires a defined schedule to run or a user can run them manually. For more information on running tasks manually, see **Create a schedule for run**.

After you schedule a task to run, FrameMaker Publishing Server adds these as scheduled tasks to the Windows Task Scheduler. You can view and edit these tasks from Windows Task Scheduler independent of FrameMaker Publishing Server. The scheduled tasks appear with the name of the task and the user name associated with the task.

Keep in mind the following when you determine the build schedules:

- Each source document requires a separate publication task. If you have a large documentation set that heavily uses single-sourcing approach, you will need to create as many publication tasks as the number of book files.
- Builds are queued—if you want to run an immediate publication task when a large number of builds are in the queue, you will have to wait for the builds in the queue to be completed before you can run your publication task.
- If you require daily builds, consult with other teams so that your build schedule doesn’t clash with theirs. If you schedule a large build in the day time, chances are that a once-only build task you want to run urgently will have to wait until the build queue is completed. A good way to ensure that daily build schedules do not affect other build tasks is to plan your daily builds to run in after-office hours.

**Create a schedule for run**

1) In the Task dialog, click the link next to **Scheduling**. The link could be:
   - Not Scheduled: If the task is not scheduled
CREATE AND MANAGE PUBLICATION TASKS
SCHEDULING

- A description of the scheduled task, such as **20:23 every SUN of every week** if the task is scheduled

```plaintext
Schedule

Start Date: 08/25/2020  Time: 20:23
End Date:   MM/DD/YYYY

- Once
- Daily
- Weekly
- Monthly
```

2) In the **Schedule** window, select the required interval, and specify the options.
3) Click **Save**.

**View the schedules for tasks**

FrameMaker Publishing Server lists all the tasks that are defined for the server, and shows their run status. In addition to the list of all tasks, you can see the next run status of each task in the **Next Run** column of the Tasks dashboard.
Work with presets

In FMPS, you can reuse publishing tasks presets and save time.

Presets

In FMPS, you can create presets for output-formats that you want to generate. Presets include settings such as:

- Name for the preset
- DITAVAL file to use
- Settings (.sts) file to use

For repetitive tasks with the same settings, you can create and apply the presets to populate all the settings to save time. You can create private or shared presets. Private presets are visible and accessible to the creator of preset only. Shared presets are visible and accessible to all the users but can be edited only by the users who created them.

Add a preset

1) Select the Preset tab.
2) Using the Add New PDF Preset drop-down, select Add New Preset.
3) Provide a Name for the preset.
4) Specify the location of the DITAVAL file that you want to attach to the preset.
5) Specify the location of the publish settings file that you want to attach to the preset.

**NOTE:** The DITAVAL file and the settings file are uploaded and saved in MongoDB.

6) If you want to make your preset available for others, select Shared. If you do not select Shared, your preset is visible and available to you only.
7) Click Save.

Edit a preset

1) On the Preset tab, click on a preset’s name. *The PDF Preset dialog appears.*
2) Make the required changes and click Save.
Duplicate a preset

Typically, you would need to create a duplicate of a preset when you want to use the same settings file, but a different DITAVAL file. In such a scenario, you can easily create a duplicate preset and use it with a publishing task.

1) On the FrameMaker Publishing Server dashboard, do one of the following:
   – Click a preset’s name and in the PDF Preset dialog, click the Duplicate icon.
   – Select a preset and then select Duplicate Preset from the Add New Preset drop-down.

2) Open the duplicate preset, edit the required files, and click Save.

Delete presets

1) Do one of the following:
   – On the Preset tab, select the presets to be deleted and click Delete.
   – In the PDF Preset dialog, click Delete.
Logs

FrameMaker Publishing Server includes extensive logging features. You can use the FrameMaker Publishing Server logs to troubleshoot any publication task errors, and fix errors in the source documents such as unresolved cross-references. You can define FrameMaker Publishing Server logs to have extensive reports including the number of markers, pages, and so on.

You can specify the log settings for each publication task, or can use the default log settings for all tasks.

Specify default log settings

1) Open the publishing task for which you want to modify the log settings.
2) In the Task dialog, click the Modify link next to Log Settings.
3) In the Log Settings dialog, double-click (or use the move icons) to move the required entries in the Don’t Include and Include lists.
4) Click Save.

View logs

You can view the logs of the completed publication tasks in two ways:

In the Task dialog

In the Task dialog, click on the Log tab. The Log tab displays all the logs relevant to the task in a reverse chronological order.

In the FrameMaker Publishing Server application folder

FMPS store all logs under the base folder:
%appdata%\Adobe\FrameMakerPublishingServer\16\n
You can use log parsers to quickly collect publication-related reports from these logs.

The server component maintains the following logs:

- **Access logs**: The server access logs are stored in the following folder:
  %appdata%\Adobe\FrameMakerPublishingServer\16\Nodelogs\logs

  - **Node logs**: The Node js is another component of the server. The logs for this component are stored in the following folder:
    %appdata%\Adobe\FrameMakerPublishingServer\16\Nodelogs

    *Error logs, warnings, information logs, and server restarts are logged here. The Node js opens a console for reporting logs, which can also be redirected to a temporary file using the command prompt.*
The client component maintains the following logs:

- **FMPSWorker**: The client component (or worker/FrameMakerServerEx.exe) maintains any exception logs of its interaction with the FrameMaker instance in the following folder:
  
  `%appdata%\Adobe\FrameMakerPublishingServer\16\FMPSWorker`
Work with FMPS using REST APIs

FrameMaker Publishing Server comes with out-of-the-box REST APIs that can be used to create publishing tasks to publish content from Microsoft SharePoint, OpenText Documentum, or Adobe Experience Manager. The APIs work on resources such as users, tasks, presets, and more to create, retrieve, update, and delete resources.

The end-point or the base URL of FMPS APIs is:

http[s]://<FMPS_server>[:<port>/]

**NOTE:** `<FMPS_Server>` is the DNS or IP address of the system hosting FMPS. The default port is 7000.

REST API protocol

From a client machine, you can query FMPS using the following methods to create, retrieve, update, or delete (CRUD) resources. According to the REST protocol, everything on the server, such as tasks, presets, users, is treated as a resource. For more information on the REST protocol, see [http://en.wikipedia.org/wiki/Representational_state_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer)

In the REST API URLs and method types table, the various GET, POST, DELETE, and PUT requests map to CRUD framework as:

<table>
<thead>
<tr>
<th>REST API method types</th>
<th>CRUD aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post</td>
<td>Create</td>
</tr>
<tr>
<td>Get</td>
<td>Retrieve</td>
</tr>
<tr>
<td>Put</td>
<td>Update</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete</td>
</tr>
</tbody>
</table>

The following topics describe the REST APIs available to you for building your own FMPS custom client. Each API has its a specific URL that you can query. The methods, such as Get and Post, are the type of operations you are performing on the REST API. The URLs include parameters that you can pass to an API.

Create and update (Post and Put) requests have parameters passed in the body of the HTTP request as a JSON file as well as in the URL. Get and Delete have parameters in the URLs only. The various task type links in the Help URL include the schema required for the various tasks.

REST API for user management

The following REST APIs are available for managing user and their permissions in FMPS.
Register a new user

A POST method that registers a new user, and can only be used when `typeOfAuth` is set as `USERLOGIN` in `development.json`. Only a user with administrative privileges can register other users.

**Request URL**

`http://<FMPS_server>:<port>/v16/auth/register`

**Header**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request. This must be of another user with administrative privileges.</td>
</tr>
</tbody>
</table>

**Header example**

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfadfas"
}
```

The authorization header must also contain the basic authentication type authorization containing credentials (email and password) of the user who is to be registered. For example:

`Authorization: Basic YWJjZEBhZG9iZS5jb206YWJjZA==`

**Request body**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>JSON</td>
<td>An email ID with which the user account is created.</td>
</tr>
</tbody>
</table>

**JSON body example**

```json
{"email":"fmadmin@adobe.com"}
```

**Successful response**

Returns a HTTP 200 (Successful) response with the user object.

**Error response**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
<tr>
<td>406</td>
<td>String</td>
<td>The request is not acceptable</td>
</tr>
<tr>
<td>409</td>
<td>String</td>
<td>User with the same credentials is already present.</td>
</tr>
</tbody>
</table>
Change user permission

A PUT method that changes the user permissions. There are two types of user permissions: USER and ADMIN. Only a user with ADMIN permissions can invoke this API.

To assign administrative permissions to your default (first) LDAP user, you will have to use the MongoDB’s command-line tool. See Install and configure MongoDB for more details.

Request URL

http://<FMPS_server>:<port>/v16/user/changePermission/:id

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request. This must be of another user with administrative permissions.</td>
</tr>
</tbody>
</table>

Header example

```
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The user ID of a user whose has to be granted administrative permissions.</td>
</tr>
</tbody>
</table>

Parameter example

```
5e4302fafdfdfadsfdfsasgafsafsag
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userPermission</td>
<td>String</td>
<td>The type of permission that you want to assign to the user. Use ADMIN for administrative permissions, and USER for non-administrative permissions.</td>
</tr>
</tbody>
</table>

JSON body example

```
{
    "userPermission": "ADMIN"
}
```
Successful response

Returns a HTTP 200 (Successful) response with the user object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
<tr>
<td>422</td>
<td>String</td>
<td>The request is sent with incorrect data.</td>
</tr>
</tbody>
</table>

User login with USERLOGIN authentication

A POST method that logs the user into the system when USERLOGIN authentication mechanism is used.

Request URL

http://<FMPS_server>:<port>/v16/auth/login

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
<tr>
<td>Authorization</td>
<td>String</td>
<td>A USERLOGIN (basic) authentication type authorization header containing credentials (email and password) of the user that wants to login.</td>
</tr>
</tbody>
</table>

Header example

```json
{
   "content-type": "application/json",
}
```

The authorization header must also contain the basic authentication type authorization containing credentials (email and password) of the user who wants to login. For example:

```
Authorization: Basic dXNlc3N1cm5hbWVAYWRvYmUyY29tOnBhc3N3b3Jk
```

Successful response

Returns a HTTP 200 (Successful) response with the user’s access token.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>
User login with LDAP authentication

A POST method that logs the user into the system when LDAP authentication mechanism is used.

Request URL

http://<FMPS_server>:<port>/v16/auth/ldap

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>Authorization</td>
<td>String</td>
<td>A USERLOGIN (basic) authentication type authorization header containing credentials (email and password) of the user that wants to login.</td>
</tr>
</tbody>
</table>

Header example

```
{   "content-type": "application/json",}
```

The authorization header must also contain the basic authentication type authorization header containing credentials (email and password) of the user who wants to login. For example:

```
Authorization: Basic dXNlcm5hbWVAYWRvYmUuY29tOnBhc3N3b3Jk
```

Successful response

Returns a HTTP 200 (Successful) response with the user’s access token.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
<tr>
<td>401</td>
<td>String</td>
<td>Use is not authorized.</td>
</tr>
<tr>
<td>406</td>
<td>String</td>
<td>The request is not acceptable.</td>
</tr>
</tbody>
</table>

REST APIs for working with presets

The following REST APIs are available for working with publishing presets.
WORK WITH FMPS USING REST APIs

REST APIs for Working with Presets

Upload an STS file

A POST method that uploads a settings (.sts) file in a preset.

Request URL

http://<FMPS_server>:<port>/v16/presets/sts/upload

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```
{
  "content-type": "multipart/form-data",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>form-data</td>
<td>Path of the settings file that you want to upload.</td>
</tr>
</tbody>
</table>

JSON body example

```
multipart/form-data
```

Successful response

Returns a HTTP 200 (Successful) response with the settings file object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Upload a DITAVAL file

A POST method that uploads a DITAVAL file in a preset.

Request URL

http://<FMPS_server>:<port>/v16/presets/ditaval/upload
WORK WITH FMPS USING REST APIS
REST APIs for Working with Presets

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```
{
    "content-type": "multipart/form-data",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>form-data</td>
<td>Path of the DITAVAL file that you want to upload.</td>
</tr>
</tbody>
</table>

JSON body example

```
multipart/form-data
```

Successful response

Returns a HTTP 200 (Successful) response with the DITAVAL file’s object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Create a preset

A POST method that creates a preset, which contains one settings (.sts) file and an optional DITAVAL file.

Request URL

```
http://<FMPS_server>:<port>/v16/presets
```

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
Header example

```
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>A unique name of the preset that you want to create.</td>
</tr>
<tr>
<td>Shared</td>
<td>String</td>
<td>Specify whether the preset is shared or not. Possible values are “NO” and “YES”.</td>
</tr>
<tr>
<td>ditavalid</td>
<td>String</td>
<td>Unique ID of the DITAVAL file to associate with the preset. This is an optional parameter.</td>
</tr>
<tr>
<td>ststypeid</td>
<td>String</td>
<td>Unique ID of the settings (.sts) file to associate with the preset. This is a mandatory parameter.</td>
</tr>
</tbody>
</table>

JSON body example

```
{
    "Name": "test1234",
    "Shared": "NO",
    "ditavalid": "23525252542gsgsgxgb",
    "ststypeid": "23525252542sagRWEWGGSG"
}
```

Successful response

Returns a HTTP 200 (Successful) response with the newly created preset object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Update a preset

A PUT method that updates a preset, which contains one settings (.sts) file and an optional DITAVAL file.

Request URL

http://<FMPS_server>:<port>/v16/presets/:id
### Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

**Header example**

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

### Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>A unique ID of the preset that you want to update.</td>
</tr>
</tbody>
</table>

**Parameter example**

```bash
5e4302fafdfdfadsfdfsafgsafgsag
```

### Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Name         | String  | A unique name of the preset that you want to update.  
**NOTE:** The name of the preset cannot be updated. |
| Shared       | String  | Specify whether the preset is shared or not. Possible values are "NO" and "YES". |
| ditavalid    | String  | Unique ID of the DITAVAL file to associate with the preset. This is an optional parameter. |
| ditadirty    | String  | Specified only if the DITAVAL file is updated. Possible values are YES or NO. |
| stsfileid    | String  | Unique ID of the settings (.sts) file to associate with the preset. This is a mandatory parameter. |
| stsdirty     | String  | Specified only if the settings file is updated. Possible values are YES or NO. |

**JSON body example**

```json
{
    "Name": "test1234",
    "Shared": "NO",
```
"stsfileid": "{{UStsId}}",
"stsdirty": "YES",
"ditavalid": "23522522542gsxgsxgxbc",
"ditadirty": "YES"
}

Successful response
Returns a HTTP 200 (Successful) response.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Delete a preset
A DELETE method that deletes a preset using the preset’s ID.

Request URL
http://<FMPS_server>:<port>/v16/presets/:id

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the preset that you want to delete.</td>
</tr>
</tbody>
</table>

Parameter example

5e4302fadfdasdfsdfsadgasdfsadgasdfas

Successful response
Returns a HTTP 200 (Successful) response with the deleted preset object.
Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get all presets

A GET method that retrieves all presets available in the system.

Request URL

http://<FMPS_server>:<port>/v16/presets

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Successful response

Returns a HTTP 200 (Successful) response with an array of all preset objects.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get a specific output preset

A GET method that retrieves an existing output preset using its unique ID.

Request URL

http://<FMPS_server>:<port>/v16/presets/:id
WORK WITH FMPS USING REST APIS

REST APIs for working with tasks

Create a task

A POST method that creates a new publishing task. A task can be created using the preconfigured output presets, such as PDF, EPUB, Responsive HTML5, and more, or a custom preset. To create a task with custom preset, you must have the settings file and DITAVAL (optional) file IDs before creating a task. Use the Upload an STS file and Upload a DITAVAL file APIs to upload the settings and DITAVAL files and get the file IDs.
Request URL

http://<FMPS_server>:<port>/v16/tasks

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>A unique name of the task that you want to create.</td>
</tr>
<tr>
<td>Input</td>
<td>JSON</td>
<td>Contains the path of the file to publish and parameters to configure the book components. In case the input files are hosted on a CMS, then the connector settings are also included in the Input parameter.</td>
</tr>
<tr>
<td>PrePublish</td>
<td>JSON</td>
<td>Path of the script file to execute during the prepublish process.</td>
</tr>
<tr>
<td>PostPublish</td>
<td>JSON</td>
<td>Path of the script file or commands to execute after the publishing process completes.</td>
</tr>
<tr>
<td>LogInfo</td>
<td>JSON</td>
<td>Specify the information to capture in the log file for the current task.</td>
</tr>
<tr>
<td>Outputs</td>
<td>JSON</td>
<td>Contains the path to publish the output, the output format, and the preset ID to use. In case the output is to be stored on a CMS, then the login credentials for the CMS are also included in the Outputs parameter.</td>
</tr>
<tr>
<td>Notification</td>
<td>JSON</td>
<td>Specify an email ID on which the notification email is sent.</td>
</tr>
</tbody>
</table>

JSON body example

The following JSON request is for creating a task with the preconfigured output presets.

```json
{
    "Name": "SamePresetMultiOut4322",
    "FMServerTask": {
        "Version": 16,
```
"Jobs": [
  {
    "Input": {
      "InputFile": "C:\Sample\Books\Special\Tables.fm",
      "UpdateBook": {
        "UpdateOLELinks": "NO",
        "UpdateTextInsets": "NO",
        "UpdateNumbering": "NO",
        "UpdateXrefs": "NO",
        "ApplyMasterPages": "NO",
        "UpdateGeneratableComponents": "NO"
      }
    },
    "PrePublish": {
      "ExtendScript": {
        "ScriptFile": ""
      },
      "prePublishid": ""
    },
    "PostPublish": {
      "SystemCommand": {
        "ScriptFile": ""
      },
      "postPublishid": ""
    },
    "LogInfo": {
      "SupportedEncodings": "NO",
      "AvailableFontFamilies": "NO",
      "NumberOfChaptersInBook": "NO",
      "EmptyBookError": "NO",
      "NumberOfIndexMarkersPresent": "NO",
      "ListofReferencedGraphics": "NO",
      "ListofMissingGraphics": "NO",
      "UnresolvedXrefInBook": "NO",
      "UnresolvedTextRefInBook": "NO",
      "ListofFilesInBook": "NO",
      "ListofChaptersInBook": "NO",
      "ListofGeneratableFilesInBook": "NO",
      "NumberOfAuthorMarkers": "NO",
      "NumberOfHF1Markers": "NO",
      "NumberOfHF2Markers": "NO",
      "NumberOfCommentMarkers": "NO",
      "NumberOfSubjectMarkers": "NO",
      "NumberOfGlossaryMarkers": "NO",
      "NumberOfEquationMarkers": "NO",
      "NumberOfHyperTextMarkers": "NO",
      "NumberOfCrossRefMarkers": "NO",
      "NumberOfConditionalTextMarkers": "NO",
      "NumberOfHTMLMacroMarkers": "NO"
    }
  }
]
"Outputs": [
    {
        "OutputFile": "Z:\SamePresetMultiOut3232\Responsive",
        "OutputFormat": "RH_RESPONSIVEHELP",
        "Configuration": {
            "presetid": "23465376sfgfdhgfhdghfgh"
        }
    },
    {
        "OutputFile": "Z:\SamePresetMultiOut65646464\html",
        "OutputFormat": "RH_BASICHTML",
        "Configuration": {
            "presetid": "gfsdsdhgfhghdhgd"
        }
    },
    {
        "OutputFile": "Z:\SamePresetMultiOut54353543542\pdf",
        "OutputFormat": "FM_PDF",
        "Configuration": {
            "presetid": "23554235235234sgsdg"
        }
    }
],
"Notification": {
    "Emails": "test@adobe.com"
}
]

The following JSON request is for creating a task with the custom output presets.

{
    "Name": "testcustompreset",
    "_id": null,
    "runtaskid": "",
    "FMServerTask": {
        "Version": 16,
        "Jobs": [{
            "Input": {
                "InputFile": "/content/dam/testfm/Samples/UserGuide/Legal.fm",
                "Connector": {
                    "ConnectorType": "Adobe Experience Manager",
                    "Server": "http://yyyy.corp.adobe.com:4502/",
                    "UserField1": "crx.default",
                    "UserName": "",
                    "Password": "",
                    "id": ""
                }
            }
        }
    }
}
{"UpdateBook": {
    "UpdateOLELinks": "NO",
    "UpdateTextInsets": "NO",
    "UpdateNumbering": "NO",
    "UpdateXrefs": "NO",
    "ApplyMasterPages": "NO",
    "UpdateGeneratableComponents": "NO"
}}
},
"PrePublish": {
    "ExtendScript": {
      "ScriptFile": ""
    },
    "prePublishid": ""
},
"PostPublish": {
    "SystemCommand": {
      "ScriptFile": ""
    },
    "postPublishid": ""
},
"LogInfo": {
    "SupportedEncodings": "NO",
    "AvailableFontFamilies": "NO",
    "NumberOfChaptersInBook": "NO",
    "EmptyBookError": "NO",
    "NumberOfIndexMarkersPresent": "NO",
    "ListofReferencedGraphics": "NO",
    "ListofMissingGraphics": "NO",
    "UnresolvedXrefInBook": "NO",
    "UnresolvedTextRefInBook": "NO",
    "ListofFilesInBook": "NO",
    "ListofChaptersInBook": "NO",
    "ListofGeneratableFilesInBook": "NO",
    "NumberOfAuthorMarkers": "NO",
    "NumberOfHF1Markers": "NO",
    "NumberOfHF2Markers": "NO",
    "NumberOfCommentMarkers": "NO",
    "NumberOfSubjectMarkers": "NO",
    "NumberOfGlossaryMarkers": "NO",
    "NumberOfEquationMarkers": "NO",
    "NumberOfHyperTextMarkers": "NO",
    "NumberOfCrossRefMarkers": "NO",
    "NumberOfConditionalTextMarkers": "NO",
    "NumberOfHTMLMacroMarkers": "NO"
},
"Outputs": [{
    "OutputFile": "/content/dam/fmdita-outputs",
    "Connector": {
      "ScriptFile": ""
    }
}]}
The following JSON request is for creating a task using the input files available on a CMS.

```json
{
    "Name": "AEM Task1",
    "_id": null,
    "runtaskid": "",
    "FMServerTask": {
        "Version": 16,
        "Jobs": [
            {
                "Input": {
                    "InputFile": "/content/dam/testfm/Samples/PartsCatalog/PartsCatalog.book",
                    "Connector": {
                        "ConnectorType": "Adobe Experience Manager",
                        "Server": "AEMHOST",
                        "UserField1": "crx.default",
                    }
                }
            }
        ]
    }
}
```
"UserName": "AEMUSER",
"Password": "AEMPASS",
"id": ""
},
"UpdateBook": {
 "UpdateOLELinks": "NO",
 "UpdateTextInsets": "NO",
 "UpdateNumbering": "NO",
 "UpdateXrefs": "NO",
 "ApplyMasterPages": "NO",
 "UpdateGeneratableComponents": "NO"
 }
},
"PrePublish": {
 "ExtendScript": {
 "ScriptFile": ""
 },
 "prePublishid": ""
 },
"PostPublish": {
 "SystemCommand": {
 "ScriptFile": ""
 },
 "postPublishid": ""
 },
"LogInfo": {
 "SupportedEncodings": "NO",
 "AvailableFontFamilies": "NO",
 "NumberofChaptersInBook": "NO",
 "EmptyBookError": "NO",
 "NumberofIndexMarkersPresent": "NO",
 "ListofReferencedGraphics": "NO",
 "ListofMissingGraphics": "NO",
 "UnresolvedXrefInBook": "NO",
 "UnresolvedTextRefInBook": "NO",
 "ListofFilesInBook": "NO",
 "ListofChaptersInBook": "NO",
 "ListofGeneratableFilesInBook": "NO",
 "NumberofAuthorMarkers": "NO",
 "NumberofHF1Markers": "NO",
 "NumberofHF2Markers": "NO",
 "NumberofCommentMarkers": "NO",
 "NumberofSubjectMarkers": "NO",
 "NumberofGlossaryMarkers": "NO",
 "NumberofEquationMarkers": "NO",
 "NumberofHyperTextMarkers": "NO",
 "NumberofCrossRefMarkers": "NO",
 "NumberofConditionalTextMarkers": "NO",
 "NumberofHTMLMacroMarkers": "NO"
 },
"Outputs": [
  {
    "OutputFile": "/content/dam/fmdita-outputs",
    "Connector": {
      "ConnectorType": "Adobe Experience Manager",
      "Server": "AEMHOST",
      "UserField1": "crx.default",
      "UserName": "AEMUSER",
      "Password": "AEMPASS",
      "id": ""
    },
    "OutputFormat": "FM_PDF",
    "Configuration": {
      "presetid": "PresetId"
    }
  }
],
"Notification": {
  "Emails": "user@enterprise.com"
},
"Schedule": {
  "_id": null
}
]

The following JSON request is for creating a task and scheduling the execution time.

{
  "Name": "ScheduledTask",
  "_id": null,
  "runtaskid": "",
  "FMServerTask": {
    "Version": 16,
    "Jobs": [{
      "Input": {
        "InputFile": "E:\HTMLBUGS\basic.fm",
        "UpdateBook": {
          "UpdateOLELinks": "NO",
          "UpdateTextInsets": "NO",
          "UpdateNumbering": "NO",
          "UpdateXrefs": "NO",
          "ApplyMasterPages": "NO",
          "UpdateGeneratableComponents": "NO"
        }
      },
      "PrePublish": {

```
"ExtendScript": {
  "ScriptFile": ""
},
"prePublishId": ""
},
"PostPublish": {
  "SystemCommand": {
    "ScriptFile": ""
  },
  "postPublishId": ""
},
"LogInfo": {
  "SupportedEncodings": "NO",
  "AvailableFontFamilies": "NO",
  "NumberOfChaptersInBook": "NO",
  "EmptyBookError": "NO",
  "NumberOfIndexMarkersPresent": "NO",
  "ListofReferencedGraphics": "NO",
  "ListofMissingGraphics": "NO",
  "UnresolvedXrefInBook": "NO",
  "UnresolvedTextRefInBook": "NO",
  "ListofFilesInBook": "NO",
  "ListofChaptersInBook": "NO",
  "ListofGeneratableFilesInBook": "NO",
  "NumberOfAuthorMarkers": "NO",
  "NumberOfHF1Markers": "NO",
  "NumberOfHF2Markers": "NO",
  "NumberOfCommentMarkers": "NO",
  "NumberOfSubjectMarkers": "NO",
  "NumberOfGlossaryMarkers": "NO",
  "NumberOfEquationMarkers": "NO",
  "NumberOfHyperTextMarkers": "NO",
  "NumberOfCrossRefMarkers": "NO",
  "NumberOfConditionalTextMarkers": "NO",
  "NumberOfHTMLMacroMarkers": "NO"
},
"Outputs": [{
  "OutputFile": "E:\HTMLBUGS\OUT",
  "OutputFormat": "FM_PDF",
  "Configuration": {
    "PresetName": "test1597120516",
    "PresetType": "PDF",
    "presetId": "5f322004900d1e21a89e92de"
  }
}]
},
"Notification": {
  "Emails": ""
}
"MetaInfo": {
"Creator": {
"username":"fmpsuser","email":"fmpsuser@adobe.com","userType":"ADMIN"}
},
"Schedule": {
"_id": null,
"Daily": {
"StartDate": {
"Year": 2020,
"Month": "SEP",
"DayofMonth": 12
},
"Time": {
"Hour": 9,
"Minute": 15,
"AMPM": "AM"
},
"Interval": 1
}
}

Successful response
Returns a HTTP 200 (Successful) response with the newly created task’s object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Delete a task
A DELETE method that deletes a task using the task’s ID.

Request URL
http://<FMPS_server>:<port>/v16/tasks/:id

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task that you want to delete.</td>
</tr>
</tbody>
</table>

Parameter example

5e4302fafdfdfadsfdfsafgsafgsag

Successful response

Returns a HTTP 200 (Successful) response with the deleted task’s object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get all tasks

A GET method that retrieves all tasks available in the system.

Request URL

http://<FMPS_server>:<port>/v16/tasks

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```
Successful response

Returns a HTTP 200 (Successful) response with an array of all task objects.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get a specific task

A GET method that retrieves an existing task using its unique ID.

Request URL

`http://<FMPS_server>:<port>/v16/task/:id`

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task that you want to retrieve.</td>
</tr>
</tbody>
</table>

Parameter example

`5e4302fafdfadfsdfasgafa5gsag`

Successful response

Returns a HTTP 200 (Successful) response with the task’s object.
Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get the status of a task of a specific user

A GET method that retrieves the status of an existing task created by a specific user.

Request URL

http://<FMPS_server>:<port>/v16/tasks/status/:taskid/:userid

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskid</td>
<td>String</td>
<td>The unique ID of the task that you want to retrieve.</td>
</tr>
<tr>
<td>userid</td>
<td>String</td>
<td>The unique ID of the user who has created the task.</td>
</tr>
</tbody>
</table>

Parameter example

5e4302fadfddadsfadsagafsafgasag
5e4302fadfddadsfadsagafsafgasag

Successful response

Returns a HTTP 200 (Successful) response with the status of the retrieved task.
### Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

### Update a task

A PUT method that updates a task using its task ID.

**Request URL**

http://<FMPS_server>:<port>/v16/tasks/:id

**Header**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

**Header example**

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task that you want to update.</td>
</tr>
</tbody>
</table>

**Parameter example**

5e4302fafdfdfadsfdasgafsafgsag

**Request body**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>A unique name of the task that you want to update.</td>
</tr>
</tbody>
</table>

**NOTE:** The name of the task cannot be updated.
WORK WITH FMPS USING REST APIs
REST APIs for Working with Tasks

JSON body example
Refer to the example given under Create a task.

Successful response
Returns a HTTP 200 (Successful) response with the updated task’s object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Upload a post-publish script file

A POST method that uploads a post-publish script file to use in a task. Once the file is uploaded, you can use the file’s ID in a task to associate with the corresponding task.

Request URL

http://<FMPS_server>:<port>/v16/tasks/post/upload

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
</tbody>
</table>
WORK WITH FMPS USING REST APIS

REST APIS FOR WORKING WITH TASKS

Header example

```json
{
    "content-type": "multipart/form-data",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>form-data</td>
<td>Path of the file that you want to upload and execute in the post-publish process.</td>
</tr>
</tbody>
</table>

JSON body example

```
multipart/form-data
```

Successful response

Returns a HTTP 200 (Successful) response with the post-publish script file’s object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Upload a prepublish script file

A POST method that uploads a prepublish script file use in a task. Once the file is uploaded, you can use the file’s ID in a task to associate with the corresponding task.

Request URL

```
http://<FMPS_server>:{port}/v16/tasks /pre/upload
```

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
WORK WITH FMPS USING REST APIS

REST APIS TO WORK WITH SCHEDULING TASKS

Header example

```json
{
  "content-type": "multipart/form-data",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>form-data</td>
<td>Path of the file that you want to upload and execute in the prepublish process.</td>
</tr>
</tbody>
</table>

JSON body example

`multipart/form-data`

Successful response

Returns a HTTP 200 (Successful) response with the prepublish script file’s object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

REST APIs to work with scheduling tasks

The following REST APIs are available for working with scheduling tasks.

Schedule a task

A POST method that schedules a task using the task ID. At any time, there can be only one task in the schedule queue. Also, you must keep a delay of ~1 second in between two consecutive tasks.

Request URL

```
http://<FMPS_server>:<port>/v16/queue/scheduleTask/:id
```

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task that you want to schedule.</td>
</tr>
</tbody>
</table>

Parameter example

5e4302fafdfdfadsfdfsafsaafgsag

Successful response

Returns a HTTP 200 (Successful) response with scheduled event.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get ID of a running task

A GET method that retrieves the running ID of a publishing task.

Request URL

```
http://<FMPS_server>:<port>/v16/queue/task/running/:id
```

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```
Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task that you want to retrieve the running ID for.</td>
</tr>
</tbody>
</table>

Parameter example

5e4302fadfadfadfasdfasgafasfag

Successful response

Returns a HTTP 200 (Successful) response with the queue object that contains the running ID of the publishing task.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Stop a running task

A POST method that stops a running publishing task.

Request URL

http://<FMPS_server>:<port>/v16/queue/stopTask

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Request body

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runid</td>
<td>String</td>
<td>ID of the running task that you want to stop.</td>
</tr>
</tbody>
</table>
JSON body example

{"runid":"454654456464"}

Successful response

Returns a HTTP 200 (Successful) response with an object to terminate the task.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get running object details for a GUID

A GET method that retrieves the running object for a task using its GUID.

Request URL

http://<FMPS_server>:<port>/v16/queue/task/lastrunguid/:guid

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>guid</td>
<td>String</td>
<td>GUID of the task that is scheduled for running in the publishing queue.</td>
</tr>
</tbody>
</table>

Parameter example

342342553423425

Successful response

Returns a HTTP 200 (Successful) response with the queue object that contains the running details of the task.
**Error response**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
<tr>
<td>404</td>
<td>String</td>
<td>Task not found for the given GUID.</td>
</tr>
</tbody>
</table>

**Get all objects present in the running task queue**

A GET method that retrieves all objects present in the running task queue. This includes completed, queued, terminated, or currently running tasks.

**Request URL**

http://<FMPS_server>:<port>/v16/queue

**Header**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

**Header example**

```
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

**Successful response**

Returns a HTTP 200 (Successful) response with the list of queued objects.

**Error response**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

**Get the details of a specific queued object**

A GET method that retrieves the status of a particular queued object. The queued object’s ID is used to retrieve all details.
Request URL

http://<FMPS_server>:<port>/v16/queue/:id

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```json
{
  "content-type": "application/json",
  "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the queued object that you want to retrieve.</td>
</tr>
</tbody>
</table>

Parameter example

5e4302fafdfdasfdfsafgsafgsag

Successful response

Returns a HTTP 200 (Successful) response with the queued object.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Other REST APIs to work with the server and logs

The following REST APIs are available for checking server status, connection parameters, and working with FMPS logs.

Get the server status

A GET method that returns the status of the server component.
Request URL

http://<FMPS_server>:<port>/ping

Successful response

Returns a HTTP 200 (Successful) response with JSON object containing information about the server’s status.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 500  | String | A error occurred while retrieving the server’s status.

Get server connection parameters

A GET method that retrieves the server’s connection parameters.

Request URL

http://<FMPS_server>:<port>/v16/connectionParameter

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
</tbody>
</table>

Header example

```
{
  "content-type": "application/json",
}
```

Successful response

Returns a HTTP 200 (Successful) response with JSON object containing server’s connection parameters.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 400  | String | A bad request is sent.

Enable or disable logging and change log level

A POST method that enables or disables logging. By default, logs are turned off. You can also configure logs from the application’s user interface, for more details see *Specify default log settings*. 

*Specify default log settings*
In addition to enabling logs, you can also configure the log level. The possible values for log level are: FATAL, ERROR, WARN, INFO, and DEBUG. By default, log level is set to INFO.

You can either enable the logs or set the log level, or configure both by providing the required parameters in query string of the API.

**Request URL**

http://<FMPS_server>:<port>/v16/enableLogging?enable=[]&level=[]

**Header**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

**Header example**

```json
{
  "X-Access-Token": "eyJ0eXAiOiJIYj0KKhjshjafakjfkadfas"
}
```

**Request body**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Integer</td>
<td>Query parameter that enables (1) or disables (0) logs.</td>
</tr>
<tr>
<td>level</td>
<td>String</td>
<td>Query parameter that sets the level of logging information to capture. Possible values are: FATAL, ERROR, WARN, INFO, and DEBUG.</td>
</tr>
</tbody>
</table>

**JSON body example**

```json
?enable=0
?level=ERROR
```

**Successful response**

Returns a HTTP 200 (Successful) response.

**Error response**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
<tr>
<td>403</td>
<td>String</td>
<td>Access not granted.</td>
</tr>
</tbody>
</table>
Get maximum number of configured logs

A GET method that retrieves the maximum number of logs configured for a task. By default, this number is configured in the `development.json file`. The parameter `MaxNoOfLogs` controls this number, which is set to 5 by default.

**Request URL**

http://<FMPS_server>:<port>/v16/logs

**Header**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>

**Header example**

```
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

**Successful response**

Returns a HTTP 200 (Successful) response with the number of maximum configured logs for tasks.

**Error response**

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get last $n$ number of logs for a specific task

A GET method that retrieves the last $n$ number of logs for a specific task. It uses the task ID and the number specified in the parameters to retrieve the logs.

**Request URL**

http://<FMPS_server>:<port>/v16/logs/task/list/:id/:lastnrun

**Header**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
Header example

```json
{
    "content-type": "application/json",
    "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task for which you want to retrieve the logs.</td>
</tr>
<tr>
<td>lastnrun</td>
<td>Integer</td>
<td>The number of last ( n ) logs to retrieve for the given task.</td>
</tr>
</tbody>
</table>

Parameter example

```text
5e4302fadfdfadsfasegfsafgsag
5
```

Successful response

Returns a HTTP 200 (Successful) response with the last \( n \) logs for the task.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get logs for a specific task by its run ID

A GET method that retrieves the logs for a specific task using the run ID. It uses the task ID and the run ID in the parameters to retrieve the logs. In addition, you can also specify the type of logs to retrieve in the query string.

Request URL

```
http://<FMPS_server>:<port>/v16/logs/task/run/:id/:runid
```

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
WORK WITH FMPS USING REST APIs
OTHER REST APIs TO WORK WITH THE SERVER AND LOGS

Header example

```json
{
   "content-type": "application/json",
   "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>The unique ID of the task for which you want to retrieve the logs.</td>
</tr>
<tr>
<td>runid</td>
<td>Integer</td>
<td>ID of the run task for which you want to retrieve the logs.</td>
</tr>
</tbody>
</table>

Parameter example

```
5e4302fafdfdfadsfdasgfsafgsag
5e4302fafdfdfadsfdasgfsafgsag
```

Successful response

Returns a HTTP 200 (Successful) response with the logs for the given task and run ID.

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Get all clients (or workers)

A GET method that retrieves all clients that are available and ready to take publishing tasks from the server.

Request URL

```
http://<FMPS_server>:<port>/v16/worker/getallworkers
```

Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>String</td>
<td>Type of content that is sent in the request.</td>
</tr>
<tr>
<td>X-Access-Token</td>
<td>String</td>
<td>The authentication token that is sent in the request.</td>
</tr>
</tbody>
</table>
WORK WITH FMPS USING REST APIs

ACCESS API DOCUMENTATION

Header example

{  "content-type": "application/json",
   "X-Access-Token": "eyJ0eXAiOiJKhjshjafakjfkadfas"
}

Successful response

Returns a HTTP 200 (Successful) response with an array of all clients (or workers).

Error response

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>String</td>
<td>A bad request is sent.</td>
</tr>
</tbody>
</table>

Access API documentation

You can access the complete API documentation from FMPS user interface. To access the documentation, perform the following steps:

1) Access the FMPS web interface by entering the following URL in your browser:

   `http[s]://<FMPS_server>:<port>/index.html`

   **NOTE:** `<FMPS_Server>` is the DNS or IP address of the system hosting FMPS. The default port is 7000.

2) Click Settings

   The Settings dialog appears.

3) In the Settings dialog, click on the (highlighted) API documentation link.

   **NOTE:** You can also access the API documentation by entering `http[s]://<FMPS_server>:<port>/documentation/index.html` URL in your browser.

   Access the API documentation

   ![Settings dialog]

4) The FMPS API documentation is launched in a new tab.
From the left navigation panel, click on the API that you want to view. The right (body) area displays the API details.

**FMPS troubleshooting**

1) What products are installed via FMPS?
   a) FrameMaker (summer release of 2020)
   b) FrameMaker Publishing Server Component
   c) FrameMaker Publishing Client Component

2) If a task/log/file is not getting deleted via the APIs, what could be the issue?
   a) There could be a problem with the permissions. You should give full permissions to the domain\user who runs the task.

3) I get a message "Object reference not set to an instance of an object". What can be the problem?
   a) The website type you are accessing is not correct. Check if the http port is open or https is correctly configured.

4) How do I enable http ports for the FMPS website to be accessed from other computers?
   a) Say you want to allow Win Server allow for port access.
   b) To access the firewall, select Start -> Type in "Firewall" -> Click on "Windows Firewall With Advanced Security"
   c) To configure an inbound traffic rule:
      i. Right click "Inbound Rules" on the left pane
      ii. Choose "New Rule"
      iii. Choose "Port"
      iv. Under "Specific ports" enter your port number (default port is 7000)
      v. Continue with "Next" until the end of the wizard, naming the rule when prompted
d) Check if your port is now accessible - you might need to restart the Windows Firewall service
(under "Services")

5) Why do I not get an email after a task is complete?
   a) Your virus scan may be blocking the emails. Try disabling that setting. For example, in McAfee
      -> Access protection properties -> (uncheck) Prevent mass mailing worms from sending mails

6) Why am I not able to see Math ML equations in the published documents?
   a) Check whether you have accepted EULA for Math ML. Open FrameMaker and go to Special >
      MathML, if the EULA pops up, accept it.
   b) Check whether JRE is installed on the machine. If not, install it from:
Legal notices

For legal notices, visit the Legal Notices page.