|  |
| --- |
| Document Conversion |
| Backend Documentation |

**docsbox**

# **Overview**

This document is full documentation of the method of the docsbox – document conversion service. The service needs docker to start, exemple:

/$ cd docsbox

/docsbox/$ docker-compose build

/docsbox/$ docker-compose up

Any configuration can be done in the file config.yml in the config directory, also can be added new volumes in the docker-compose.yml, for exemple:

services:

web:

restart: always

build: ./docsbox

image: oikeusministerio/common-conversion

depends\_on:

- redis

ports:

- "8000:8000"

volumes:

- ./docsbox:/home/docsbox

- /media:/home/docsbox/media

- /**’directory to the certificate’**:/home/docsbox/certificate.pem

extra\_hosts:

- it1.integraatiopalvelu.fi:46.30.128.27

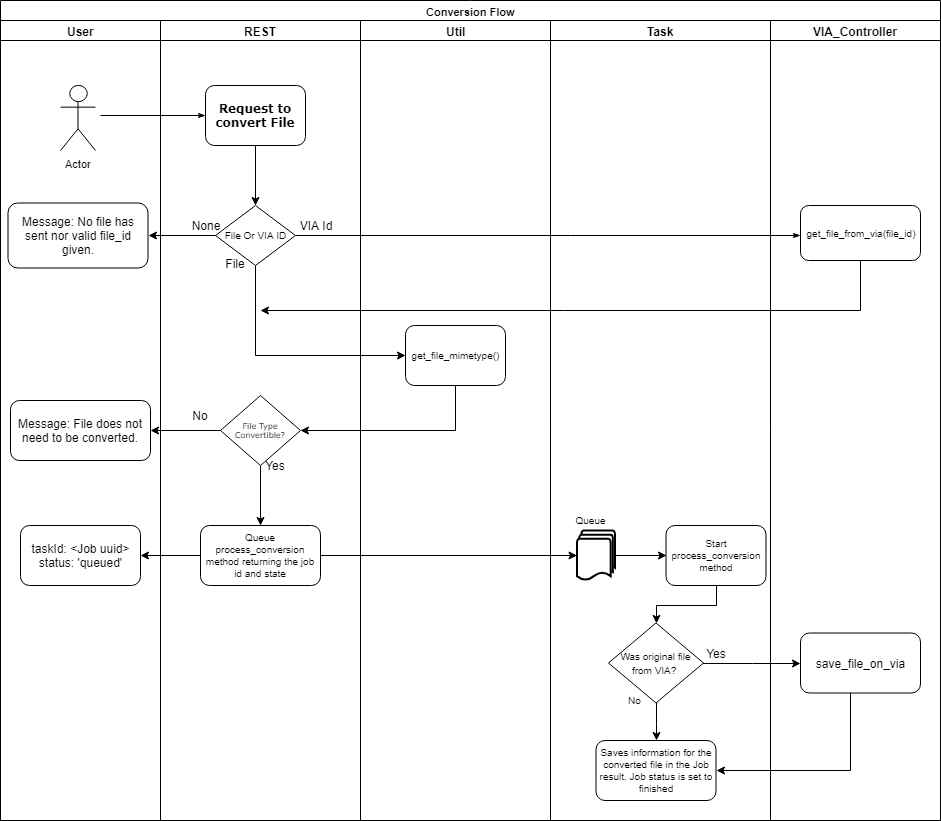
command: gunicorn -b :8000 docsbox:app

## Conversion workflow

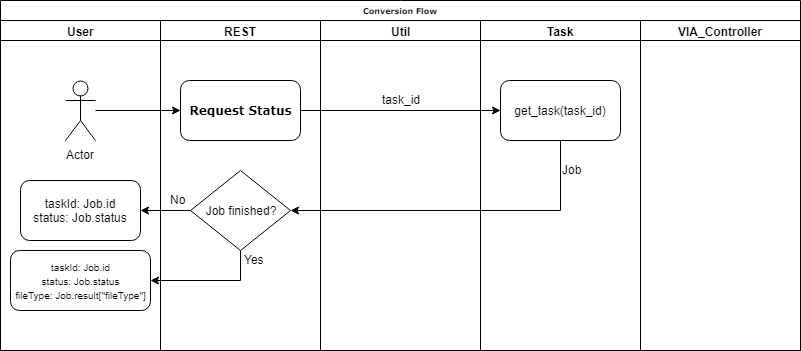
### Get File

## C:\Users\nuno.franco\Downloads\Untitled Diagram.png

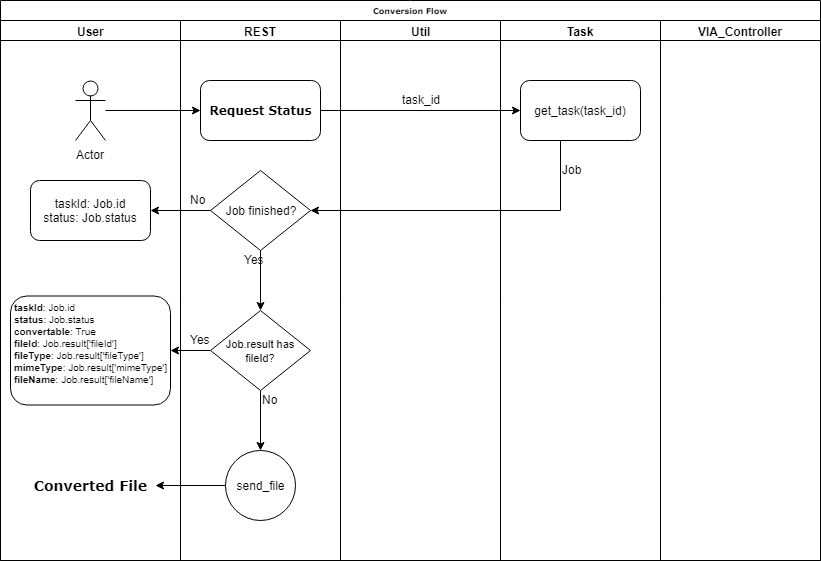
### Convert



### Get Status



### Get Converted File/File Id



# **REST API - /conversion-service/**

## POST

### get-file-type/<file\_id>

Request can be done with a VIA file id as a path variable or with a file (in the later case the path still needs an id but will not read it).

Returns the file type and if the file is convertible

Sending File annexed to the request (file\_id must be add or else it will not recognise path, although it will not need it, exemple:

**$** curl -X POST -F "file=@test.doc" http://localhost/conversion-service/get-file-type/01234ABCD

{

"fileType": "Microsoft Word",

"convertable": true

}

If there is not a file annexed to the request it will verify the file\_id and request the file to VIA service, example:

**$** curl -X POST http://localhost/conversion-service/get-file-type/02127a06-d078-4935-a6f9-b7cbdbff4959

{

"fileType": "Microsoft Word",

"convertable": true

}

If no file is annexed and the file\_id is not a valid uuid it will give the following message:

**$** curl -X POST http://localhost/conversion-service/get-file-type/01234ABCD

{

"message": "No file has sent nor valid file\_id given."

}

### convert/<file\_id>

Request can be done with a VIA file id as a path variable or with a file (in the later case the path still needs an id but will not read it).

Returns the file type and if the file is convertible

Sending File annexed to the request (file\_id must be add or else it will not recognise path, although it will not need it, exemple:

**$** curl -X POST -F "file=@test.doc" http://localhost/conversion-service/convert/0

{

"taskId": "bbf78afd-011c-4815-95da-17b810fa4f5f",

"status": "queued"

}

If there is not a file annexed to the request it will verify the file\_id and request the file to VIA service, also Content-Disposition header can be added with the file name, example:

**$** curl -X POST --header "Content-Disposition: filename.doc" http://localhost/conversion-service/convert/02127a06-d078-4935-a6f9-b7cbdbff4959

{

"taskId": "bbf78afd-011c-4815-95da-17b810fa4f5f",

"status": "queued"

}

If file type is not in the list for conversion, it will return the following message:

**$** curl -X POST -F "file=@test6.odt" http://localhost/conversion-service/convert/0

{

"message": "File does not need to be converted."

}

If no file is annexed and the file\_id is not a valid uuid it will give the following message:

**$** curl -X POST http://localhost/conversion-service/convert/01234ABCD

{

"message": "No file has sent nor valid file\_id given."

}

## GET

### status/<task\_id>

Returns the Status of the task with the given id.

**$** curl -X GET http://localhost/conversion-service/status/bbf78afd-011c-4815-95da-17b810fa4f5f

{

"fileType": "PDF/A",

"taskId": "bbf78afd-011c-4815-95da-17b810fa4f5f",

"status": "finished"

}

### get-converted-file/<task\_id>

Returns the converted file if the conversion request was not through VIA, else it returns the new filename, filetype and VIA file ID (fileId).

File from VIA, service returns:

**$** curl -X GET http://localhost/conversion-service/get-converted-file/bbf78afd-011c-4815-95da-17b810fa4f5f

{

"convertable": true,

"fileId": "92180232-5d1a-456f-80d7-2cbc596afb57",

"fileName": "filename.pdf",

"mimeType": "application/pdf",

"fileType": "PDF/A",

"status": "finished",

"taskId": "bbf78afd-011c-4815-95da-17b810fa4f5f"

}

File given through REST API, service returns converted file directly, example:

**$** curl -X GET -O http://localhost/conversion-service/get-converted-file/bbf78afd-011c-4815-95da-17b810fa4f5f

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 212 100 212 0 0 15142 0 --:--:-- --:--:-- --:--:-- 15142

## DELETE

### delete-tmp-file/<task\_id>

Given a task id it will run the task for the deletion of the converted file

**$** curl -X DELETE http://localhost/conversion-service/delete-tmp-file/bbf78afd-011c-4815-95da-17b810fa4f5f

"finished"

# Utils Module Methods

Module serves as a support for other modules, for analysing and setup.

## make\_zip\_archive (string, string): string, string

Method receives name to be given to the .zip file and the directory path to be compressed and returns the path of the zip file and the full filename.

## set\_option (dictionary, string): dictionary

Method receives a dictionary of options for the conversion and the file mimetype, the method will check if the options gived are valid and if the option param is empty it will return a default dictionary (ex: options= {format:”pdf”}).

## make\_thumbnails (Image, string, string): string

Method receives an Image, a directory path (preference a TemporaryDirectory) and the size of the thumbnails (width, height). Saves the images in the given directory with the given image size.

## get\_pdfa\_version (Nodes): string

Method receives a XMPMetadata Nodes and returns a string with the PDF version.

## get\_file\_mimetype (File): string

Method receives a File and returns the file mimetype, if the file is a pdf/a returns its mimetype as “application/pdfa” to diferenciate of a normal PDF.

## remove\_extension (string): string

Method receives a string with a filename and returns the file name without the extension.

## is\_valid\_uuid (string): boolean

Method returns True if given string corresponds to a uuid.

# Tasks Module Methods

Module in charge of methods that can communicate, utilize the python RQ Job class library.

## get\_task (string): Job

Given a task Id return the Job associated.

## do\_task (string): Job

Given a task Id runs and returns the Job associated.

## remove\_file (string): void

**Method can be scheduled**. Removes file in given path.

import datetime

[…]

**remove\_file**.schedule(datetime.timedelta(seconds=app.config["ORIGINAL\_FILE\_TTL"]), tmp\_file.name)

## create\_tmp\_file\_and\_get\_mimetype (File, string, bool, bool): dictionary

Method receives a File, a string with the file name (without extencion), a bool if the file is a stream and a bool if the NamedTemporaryFile **should** be deleted at the end of the method. The method returns a dictionary with the file mimetype and the NamedTemporaryFile if the last boolen flag is True.

if request.files and "file" in request.files:

mimetype = **create\_tmp\_file\_and\_get\_mimetype**(request.files["file"], None, schedule\_file\_del=False)['mimetype']

elif file\_id and is\_valid\_uuid(file\_id):

try:

r = get\_file\_from\_via(file\_id)

if r.status\_code == 200:

mimetype = **create\_tmp\_file\_and\_get\_mimetype**(r, None, stream=True, schedule\_file\_del=False)['mimetype']

## process\_convertion (string, dictionary, dictionary): dictionary

**Method can be scheduled.** Method receives the path to the file, a dictionary with the options for the conversions and another dictionary with meta information (file name, file mimetype via\_allowed\_users). Depending of the filetype the it will call the correct method to convert the given file, if the meta dictionary has value in the via\_allowed\_users key then it will send the converted file to VIA. Method will return a dictionary with filename, filetype and fileId if file was saved in VIA.

Queueing the method returns a RQ Job class with id and status (‘queued’, ‘started’, ‘finished’, ‘canceled’).

[. . .]

task = **process\_convertion**.queue(tmp\_file.name, options, {"filename": filename, "mimetype": mimetype, "via\_allowed\_users": via\_allowed\_users})

return {"taskId": task.id, "status": task.status}

When job is finished the result of the scheduled method is saved in the Job.result.

task = get\_task(task\_id)

if task:

if task.status == "finished":

if task.result:

if "fileId" in task.result:

return {

"taskId": task.id,

"status": task.status,

"convertable": True,

"fileId": task.result["fileId"],

"fileType": task.result["fileType"],

"mimeType": task.result["mimeType"],

"fileName": task.result["fileName"]

}

else:

try:

return send\_from\_directory(app.config["MEDIA\_PATH"], task.id, as\_attachment=True, attachment\_filename=task.result["fileName"])

## process\_document\_conversion (string, dictionary, dictionary, string): dictionary

Method that converts “Document” type files (Text, Presentations, Spreadsheets), receives the path to the file, a dictionary with the options for the conversions, another dictionary with meta information (file name, file mimetype) and a Job id. The file in converted according the options and is saved with the given name with the new file extension. Its created a task to delete the new file and returns a dictionary with the new file name and the new file type for the converted file.

## process\_image\_conversion (string, dictionary, dictionary, string): dictionary

**Method not yet** implemented. Method that converts Image type files, receives the path to the file, a dictionary with the options for the conversions, another dictionary with meta information (file name, file mimetype) and a Job id.

## process\_audio\_conversion (string, dictionary, dictionary, string): dictionary

**Method not yet implemented**. Method that converts Audio type files, receives the path to the file, a dictionary with the options for the conversions, another dictionary with meta information (file name, file mimetype) and a Job id.

## process\_video\_conversion (string, dictionary, dictionary, string): dictionary

**Method not yet implemented**. Method that converts Video type files, receives the path to the file, a dictionary with the options for the conversions, another dictionary with meta information (file name, file mimetype) and a Job id.

## thumbnail\_generator (string, dictionary, dictionary, string, string): string, string

Method receives the path to the converted file, a dictionary with the options for the conversions, another dictionary with meta information (file mimetype), a Job id and the path to the original document. This method creates a TemporaryDirectory to create the thumbnails and returns directory in a zip as the job id as its name.

# VIA Controller Module Methods

Module serves to communication with the VIA fileservice to request or upload files.

## get\_file\_from\_via (string): File

Method receives a file id, requests the associated file from VIA and returns the response.

VIA url and cert path is set in the config.yml file so that:

def **get\_file\_from\_via**(file\_id):

url=app.config["VIA\_URL"] + "/" + file\_id

cert=app.config["VIA\_CERT\_PATH"]

return get(url=url, cert=cert, stream=True)

## save\_file\_on\_via (string, string, string): Response

Method receives a file path, the file type and the via\_allowed\_users, posts the file to VIA and returns the response.

VIA url, cert path is set in the config.yml file, the VIA allowed users may be sent when its made the conversion request, there can be set a default VIA allowed users as well in the config.yml file so that:

def **save\_file\_on\_via**(file\_path, mime\_type, via\_allowed\_users):

data = open(file\_path, "rb")

cert=app.config["VIA\_CERT\_PATH"]

headers = {'VIA\_ALLOWED\_USERS': via\_allowed\_users, 'Content-type': mime\_type}

return post(url=app.config["VIA\_URL"], cert=cert, data=data, headers=headers)

# Logs

## GraylogLogger (string, Flask, string): LoggerAdapter

* name: string
* app: Flask class
* logtype: string

Creates a logger with a GelfHTTPHandler to connect with graylog with the given *name* and with the Flask *app* config of the given *logtype*.

### Log (string, string, dict)

* level: string
* msg: string
* extra: dict

Extends the log method of python LoggerAdapter adding the extra dict to the log as extra information.

## GelfHTTPHandler ()

Extends python logging Handler and graypy BaseGELFHandler to connect to a graylog server.

### Emit (LogRecord)

Formats de LogRecord to GELF, stablished the connection with the graylog server and sends the message.