|  |  |
| --- | --- |
| **BOSA signature solution**FPS information  | bosa.png |

# Summary

This document is intended for an FPS (Federal Public Service) that wishes to integrate the BOSA signature solution ‘Federal Trust Services’ (FTS). It allows users (Belgian citizens or residents) who are surfing on the FPS website to sign a document with their eID or foreigner card. This document, for example a PDF or XML, has been prepared or created by the FPS.

The whole signing process is handled by BOSA. The FPS only has to upload the unsigned document to BOSA, redirect the user to BOSA and wait for a callback; then the signed document can be downloaded.

Contents

[1. Introduction FTS 2](#_Toc78555550)

[1.1. Singing flow 2](#_Toc78555551)

[1.2. Signing software 3](#_Toc78555552)

[2. Getting started 3](#_Toc78555553)

[2.1. Environments and development process 3](#_Toc78555554)

[2.2. Code repositories 4](#_Toc78555558)

[2.3. Main URLs 5](#_Toc78555559)

[2.4. Setup storage buckets 6](#_Toc78555561)

[2.5. Signing profiles 7](#_Toc78555562)

[2.6. IP adresses BOSA 8](#_Toc78555563)

[2.7. Example code 8](#_Toc78555564)

[3. Details 8](#_Toc78555565)

[3.1. BOSA S3 server 9](#_Toc78555566)

[3.2. The getToken call 9](#_Toc78555567)

[3.3. The HTTP redirect 10](#_Toc78555568)

[3.4. The callback to the FPS 10](#_Toc78555569)

[3.5. Error codes 12](#_Toc78555570)

[3.6. PDF visible signatures 12](#_Toc78555571)

[3.7. Disabling downloads 15](#_Toc78555572)

[3.8. Request a specific language when calling the service WEB UI 15](#_Toc78555573)

[3.9. Using XLST file 15](#_Toc78555574)

[4. Sample/test FPS code and service 16](#_Toc78555575)

[5. References 16](#_Toc78555576)

# Introduction FTS

## **Singing** flow

The goal is to make eSignature integration available using 4 generic steps for customers of the service. From a customer perspective, the flow is as follows (customer is “FPS”):



**Steps needed from a FPS perspective**

1. **Upload** the document to be signed to the private storage bucket
2. **Request** a Token to enable signing of the document
3. **Redirect** to the end user – based on the acquired Token

--- *(User Signs the document using BOSA webapp, followed by a callback to the FPS) ---*

1. **Download** the signed document
2. *(Delete documents)*

How the document is signed is defined by the second step (request Token). This step includes the name for the signed document, configuration profiles that should be used, the type of signature, and whether the user can download the signed document or not. The call also allows to force the user interface language to be used if desired.

The maximum time to redirect end-users to the FTS is 5 minutes and the maximum time to sign is 5 hours. If the browser session exceeds 5 hours, the end user will be notified that the token has become invalid.

## 1.2. Signing software

In order to sign a document using eID in a browser, the end user needs software that facilitates the communication between the card and the browser. The signing service provides 2 new components to sign with eID: a web browser extension and a local component.

|  |  |  |
| --- | --- | --- |
| **TA** | TEST | continuously deployed from commits to “developer” code branch. |
| **QA** | QUALITY | continuously deployed from commits to “master” code branch. |
| **INT** | INTEGRATION | deployed from stable release. |
| **PROD** | PRODUCTION | deployed from stable release. IP filtering will be required. |

* For citizens installing the software does not require admin rights. The signing procedure will take care of the installation if the components are missing, and will do so in a non-intrusive manner (the sign flow is not broken).
* For organizations an msi installer is available for deployment by IT at <https://eid.belgium.be/en/download/beidconnect> .

The environments that are the supported:

* windows/mac/linux
* firefox/chrome (+edge)/safari/internet explorer

# 2. Getting started

## 2.1. Environments and development process

|  |  |  |
| --- | --- | --- |
| TA | TEST | continuously deployed from commits to “developer” code branch. |
| QA | QUALITY | continuously deployed from commits to “master” code branch. |
| INT | INTEGRATION | deployed from stable release. |
| PROD | PRODUCTION | deployed from stable release. IP filtering will be required. |

### Day-to-Day development

* Development is done on “developer” code branch.
* Changes are deployed automatically to TA.
* New features are developed here.
* Bug fixing is done here.

### Feature complete development

* Completed features are pushed to the “master” code branch.
* Changes are deployed automatically to QA.
* Development is done on TA, QA receives the changes when feature complete.
* Stable releases are created from QA: a consistent set of features is tagged on QA, it can

then be deployed to INT for testing and to PROD when integration testing proves to be successful.

### Deliverables and artifacts

* Development is done in an open source development model.
* The service is based on Docker containers.
* The BOSA service is based on a stable version that passed integration testing.

## 2.2. Code repositories

Code repositories

BOSA uses 2 code repositories.

**GitLab: internal**

* BOSA hosted and private
* Gitlab is the primary repository where BOSA development is done
* It has 2 code branches: “develop” and “master”
* The repository is integrated with the build & deploy pipeline that is used to host the service for all environments, which is why it is private
* The TA environment tracks the “develop” branch at any time
* The QA environment tracks the “master” branch at any time
* The INT and PROD environments are deployed using static/stable versions of the service that are tagged on the master branch

**GitHub: public**

* Public code repository
* Automated mirror of the BOSA GitLab “master” branch
* Stable release tags are mirrored as well: the version that is available on PROD can be checked out from the code
* Repositories

<https://github.com/Fedict/fts-test-environment>

Script for local deployment of the actual signing service for testing

<https://github.com/Fedict/fts-mintest>

Code for mintest code example, a basic implementation how to call the service including several base scenario’s

<https://github.com/Fedict/fts-documentation>

Repository for documentation and architecture documents

<https://github.com/Fedict/fts-sign-validation>

Signing and validation engine

<https://github.com/Fedict/fts-gui-sign>

Signing frontend application (webflows)

<https://github.com/Fedict/fts-esealing>

Sealing application

<https://github.com/Fedict/fts-eidlink>

Client application (BeidConnect – repository to be renamed)

## 2.3. Main URLs

URLs are based on QA.

* Replace QA with TA or INT for those environments
* Remove QA in the URL for PROD

Example:

*PROD: https://dp.fts.bosa.belgium.be*

*TA: https://dp.ta.fts.bosa.belgium.be
QA: https://dp.qa.fts.bosa.belgium.be
INT: https://dp.int.fts.bosa.belgium.be*

### URLs

* [https://dp.qa.fts.bosa.belgium.be](https://dp.fts.bosa.belgium.be/)
	+ URL for the storage bucket
	+ API endpoint
	+ WEBUI to manage the bucket (including password reset)
	+ MINIO webUI available
* [https://sign.qa.fts.bosa.belgium.be](https://sign.fts.bosa.belgium.be/)
	+ URL for the signing application
	+ API endpoint
* [https://validate.qa.fts.bosa.belgium.be/signandvalidation/signing/getTokenForDocument](https://validate.fts.bosa.belgium.be/signandvalidation/signing/getTokenForDocument)
	+ URL for Token request
	+ API endpoint
* [https://sign.qa.fts.bosa.belgium.be/sign](https://sign.fts.bosa.belgium.be/sign) (yet to be launched)
	+ WEBUI with manual procedure (upload and sign)
* [https://mintest.qa.fts.bosa.belgium.be](https://mintest.fts.bosa.belgium.be/)
	+ Hosted version of fts-test-environment application
	+ Implements full flow using the storage bucket and redirection
* [https://validate.qa.fts.bosa.belgium.be/signandvalidation/swagger-ui.html](https://validate.fts.bosa.belgium.be/signandvalidation/swagger-ui.html)
	+ Swagger specification REST API

## 2.4. Setup storage buckets

Your storage bucket is a private storage location that holds the files to be signed when using eSignature, and configuration data that consists of customer-defined profiles that allow you to define:

* how XML files are visualized when signing (XSLT)
* how visible signature fields should be added to PDF documents

BOSA will clean out old documents that were not deleted by the users of the service, except the configuration profiles. For this reason, every bucket has a mandatory “config” directory that is exempt from cleaning and should only be used to hold the XML and PDF profiles mentioned above.

To access the bucket an account is needed. To access the config directory a specific, second account is used. The reason for the split is that visible signature fields actually add content to the document when signing. Using a second account allows for more scrutiny who can define and alter this. It is possible to use the same password for both accounts if the split is not a requirement.

As BOSA offers 4 environments which accounts each, a fixed naming scheme is used based on the chosen name for the storage bucket:

* based on the basename buckets will be created all environments;
* per bucket 2 accounts are configured o one for managing the files that require signing as well as the signed versions;
* another one to manage the configuration files for XLST visualization and to create visual signature fields in PDF documents.

The account passwords will be set to a default and must be changed before usage. The naming scheme used is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **environment** | **bucket** | account | Account-config |
| TA | bucket  | Bucket-ta | bucket-config-ta |
| QA | bucket  | Bucket-qa | bucket-config-qa |
| INT | bucket  | Bucket-int | bucket-config-int |
| PROD | bucket  | bucket | bucket-config |

It is possible to request multiple storage buckets should business requirements demand this.

## 2.5. Signing profiles

To instruct the service what level of signature should be generated on a document, base profiles have been defined for common use cases. The profile definitions are available on <https://github.com/Fedict/fts-sign-validation/tree/master/parameters/signature>.

The current profiles that are readily available are:

|  |  |  |  |
| --- | --- | --- | --- |
| **PROFIELNAAM** | **PROFIEL** | **FORMAAT** | **REMARK** |
| CADES\_1 | CAdES\_BASELINE\_B | ASiC\_S |   |
| CADES\_2 | CAdES\_BASELINE\_B | ASiC\_E |   |
| CADES\_LTA | CAdES\_BASELINE\_LTA | ASiC\_E DETACHED |   |
| CADES\_LTA\_ENVELOPING | CAdES\_BASELINE\_LTA | ENVELOPING |   |
| PADES\_1 | PAdES\_BASELINE\_B | ENVELOPED |   |
| PADES\_LTA | PAdES\_BASELINE\_LTA | ENVELOPED |   |
| PADES\_LTA\_EXP\_ALLOW | PAdES\_BASELINE\_LTA | ENVELOPED | **ALLOWS SIGNING WITH EXPIRED CERTIFICATES. NOT AVAILABLE IN INT/PROD** |
| XADES\_1 | XAdES\_BASELINE\_B | ENVELOPED |   |
| XADES\_2 | XAdES\_BASELINE\_B | ENVELOPING |   |
| XADES\_LTA | XAdES\_BASELINE\_LTA | ENVELOPED |   |
| XADES\_LTA\_EXP\_ALLOW | XAdES\_BASELINE\_LTA | ENVELOPED | **ALLOWS SIGNING WITH EXPIRED CERTIFICATES. NOT AVAILABLE IN INT/PROD** |

For reference the resulting signatures include the following elements per international standards:

|  |
| --- |
| **B: SIGNATURE WITH CERTIFICATE CHAIN** |
| **T: B + TIME STAMP** |
| **LT: T + REVOCATION DATA** |
| **LTA: LT + 2ND TIME STAMP** |

While the B and LTA options should cover customer needs, T and/or LT profiles can easily be added.

## 2.6. IP adresses BOSA

IP addresses behind CallBackUrl.

|  |  |  |
| --- | --- | --- |
| **TA** | ECOSYS-NONPUB-TA | 193.191.245.151 |
| **QA** | ECOSYS-NONPUB-QA   | 193.191.245.150 |
| **INT** | ECOSYS-NONPUB-QA   | 193.191.245.150 |
| **PROD** | ECO-HIDE-PR-PUB  | 193.191.245.204 |

## 2.7. Example code

BOSA DT maintains a code example how to integrate with the service for eSignatures that covers the common use cases. The source code is publicly available on <https://github.com/Fedict/fts-test-environment/tree/master/mintest> The example itself is deployed and hosted on all environments except PROD and can be accessed and tested using the following URL (example URL is for QA): <https://mintest.qa.fts.bosa.belgium.be>.

A developer-centric guide explain this into more detail is available at <https://github.com/Fedict/fts-documentation/tree/master/FPS_info> (file: BOSA signature solution.docx).

# 3. Details



## **3.1. BOSA S3 server**

BOSA provides an Amazon S3 compatible server: MinIO [1].

The FPS uses this server to:

* Upload unsigned documents and additional files such as XSLT files or ‘PDF signature profiles’
* Download signed documents
* Delete the unsigned and signed documents

When the FPS registers for the signature solution, it receives a username and password for this BOSA S3 server. With this, the FPS can then upload, download and delete documents.

*It is important to note that this BOSA S3 server is not intended for long time storage: the space allocated to an FPS is limited, and the FPS itself is responsible for timely deleting documents that are no longer needed.*

There are free client SDKs available for several languages (Java, python, .NET, …) on [2].
An FPS can use these to implement the upload/download/delete calls.

Implementation note: to connect to an S3 server, typically 3 things are needed: a ‘bucket name’ (this can be seen as a directory on the S3 server), a username and a password.
For the BOSA S3 server however, the ‘bucket name’ and username are the same, therefore only a username and password are provided during the registration.

## 3.2. The getToken call

This is a REST call (a HTTP POST containing a json string) from the FPS to the BOSA DSS server; the BOSA DSS server will return a ‘token’.

This token is a string that contains a.o. the names of the unsigned (‘in’) and signed (‘out’) documents. It is cryptographically protected so they can’t be modified. Note: that it has nothing to do with OAuth or OpenID.

For the FPS, this token is transparent and does not have to be parsed/interpreted/.. It only has to be sent in the HTTP redirect call to the BOSA DSS front-end server.

Here is an example of the contents of a getToken request:

|  |
| --- |
| *{* *"name":"minfin",* *"pwd":"U84SnLEQvp",* *"in":"test.xml",* *"xslt":"test.xslt",* *"out":"signed\_test.xml",* *"prof":"XADES\_1"**}* |

The “xslt” parameter is optional. It can be used when the input document is an XML file and it use by the user’s browser to convert the XML file into HTML code for viewingin the browser.

The “prof” parameter specifies which type of signature is requested. BOSA provides a list of the available values and their meaning.

## 3.3. The HTTP redirect

After the FPS has uploaded the document and obtained the token, it will do a HTTP redirect of the User’s browser to the BOSA DSS front-end server; this will handle the actual signing process.

This redirect contains the following parameters:

* The token obtained in the getToken call: this is at the end of the URL path
* A RedirectUrl to the FPS: this is a query parameter
* A language parameter: this is a query parameter. Values: en, nl, fr, de
* A name parameter: this is used to show the FPS name in the User’s browser

Example of a redirect URL:

|  |
| --- |
| *https://sign.ta.fts.bosa.belgium.be/sign/SDZEgF935d6e9.DKEi8D7nFDZz8EEZLazemJFeLIE.2FIDsdf987dILEJFd?redirectUrl=http%3A%2F%2F%2Feservices.minfin.fgov.be%2Fcallback%3Ffilename%3Dsigned\_test.pdf&language=en&name=FPS%20XXX* |

With the token, the user’s browser will download the unsigned document and, if present, the XSLT file from the BOSA DSS server. It will then display the document (the PDF, the XML or the XLST-transformed XML) and read out the certificates from the User’s eID/foreigner card. It will then ask to enter the PIN and proceed with the actual signing.

## 3.4. The callback to the FPS

After the document is signed and available on the BOSA S3 server, the User’s browser will do a callback to the callback URL specified in the redirect (see above). The callback URL is completely specified by the FPS and can contain query parameters like a session ID.

Next to the signed document, a json file is added to the BOSA S3 server that contains the signing certificate and the simple and the detailed validation report. The filename of this json file is the output filename of the signed document, appended with ".validationreport.json" (e.g. if the filename of the signed document = "*8uSOFNM03ns4F8N.xml*" then filename of the json file = "*8uSOFNM03ns4F8N.xml.validationreport.json*") Just like the other files, it is up to the FPS to delete this json file.

If the validation report will be used/saved by the FPS, it should be downloaded and then deleted from the S3 at the same time as the signed document. This is to avoid the case that a document with the same name should (accidentally) be used while the validation report of the previous signed document is still present. In case of doubt, it is possible to link both together by means of the reference-id as shown below:

|  |  |
| --- | --- |
| Signed document:signed_doc.png | *Validation report:*validation_report.png |

In case of an error, the following parameters are added:

* err: an error string, e.g. *CERT\_REVOKED*
* details: a string with error details, this parameter is optional
* ref: a unique reference

Examples of a full callback URL:

|  |
| --- |
| ***http://eservices.minfin.fgov.be/callback?session=349317359*** |

|  |
| --- |
| ***http://eservices.minfin.fgov.be/callback?session=762087326****&ref=20210423092112378&err=CERT\_REVOKED* |

|  |
| --- |
| ***http://eservices.minfin.fgov.be/callback?session=650807315****&ref=20210423091821774&details=* *exp.%20date%20=%202017.05.29%2022:00:00&err=SIGN\_CERT\_EXPIRED* |

The bold part is the original callback URL specified by the FPS in the redirect, the rest has been added by the browser’s javascript.

Below is a list of the error strings that have been defined so far.

## Error codes

URL: <https://github.com/Fedict/fts-documentation/tree/master/FPS_info> (file: error\_constants.txt).

Error codes that the service might include on callback to the relying party.

|  |  |
| --- | --- |
| **Error constant** | **Explanation** |
| BEID\_CONNECT\_ERROR | Couldn't connect to the eID card |
| NO\_READER | No reader found on the user's PC |
| UNSUPPORTED\_READER | unsupported smart card reader |
| SIGNATURE\_FAILED | eID card returned an invalid signature |
| CARD\_BLOCKED | eID card blocked |
| CARD\_ERROR | eID card error |
| USER\_CANCELLED | User cancelled the signing operation |
| SIGN\_CERT\_EXPIRED | Signing certificate expired |
| CERT\_CHAIN\_INCOMPLETE | No or incomplete certificate chain |
| NO\_SIGN\_CERT | No signing certificate provided |
| INVALID\_SIG\_DATE | Signing date out of bounds |
| INVALID\_S3\_LOGIN | Invalid user name or password |
| NO\_CERT\_TO\_VALIDATE | The certificate is missing |
| NO\_DOC\_TO\_VALIDATE | DSSDocument is null |
| NO\_TOKEN | Required parameter token not provided |
| CERT\_REVOKED | Certificate (probably the signing cert) revoked |
| INTERNAL\_ERR | Unexpected error occured |
| INVALID\_DOC | Document validation (after signing) failed |
| UNKNOWN\_PROFILE | Unknown signature profile |
| EMPTY\_PARAM | Empty (null) parameter in request |
| INVALID\_TOKEN | Invalid token in request |
| PARSE\_ERROR | Couldn't parse request |

## 3.6. PDF visible signatures

For PDF signing, it is possible to create visible a signature field or to use an existing one.

There are quite a number of options (image, text, colors, ..) that can be specified. Because the token is limited in size it has been decided to use a ‘**PDF signature profile**’ file (see below) that is to be uploaded to the S3 server to hold most of the parameters and only provide a minimum of parameters in the **getTokenForDocument** call.

**1. So the getTokenForDocument call (and therefore the token) can only contain these parameters:**

* **psp**: the name of the ‘PDF signature profile’ file
* **psfN**: the name of an existing PDF visible signature field
* **psfC**: the ‘coordinates’ for a new PDF visible signature field, this is either
	+ the string “default” to use the default coordinates that are specified in the ‘PSP’ file
	+ a comma-separated value of 5 integers: *page*, the *x* and *y* coordinates, *width* and *height*
* **psfP**: a boolean, if true then the photo from the eID card is read and used as image
* **lang**: the language to be used for the text in the signature field, this language must be specified in the “texts” field of the PSP file

All parameter are optional, if none are specified then an invisible PDF signature is made.
If psfN or psfC (they are mutually exclusive) are specified then a visble PDF signature is made/used.

 This is an example of the contents of a getTokenForDocument request:

|  |
| --- |
| *{* *"name":"minfin",* *"pwd":"U84SnLEQvp",* *"in":"test.pdf",* *"out":"signed\_test.pdf",* *"****psp****":"minfin1.psp",* *"****psfC****":"1,20,30,180,60",* *"****psfP****":true,* *"****lang****":"en"**}* |

So the *psfC* value means: put a signature field on the **1**st page, at coordinates (**20**, **30**) from the left top corner of the page, with width **180** and height **60**

**2. The PSP (PDF signature profile) file is a json file as in the example below:**

|  |
| --- |
| { "bgColor" : "#D0D0D0", "texts" : { "en" : "Signed by %gn% %sn%", "de" : "Unterzeichnet von %gn% %sn%", "nl" : "Getekend door %gn% %sn%", "fr" : "Signé par %gn% %sn%" }, "font": "freescpt", "textSize" : 14, "textPadding" : 20, "textAlignH" : "CENTER", "textAlignV" : "MIDDLE", "textPos" : "BOTTOM", "textColor" : "#0000FF", "defaultCoordinates" : "1,30,20,120,60", "imageDpi" : 400, "image" : "ZGVmYXVsdA=="} |

Remark: this profile json file is optional, if not present then default values are taken

The “bgColor” and “textColor “values for the RGB color code scheme, e.g. “#0000FF” = blue.

The “texts” can contain the following macros:

* **%gn%** : this is replaced by the Given Name in the user’s certificate
* **%sn%**: this is replaced by the Surname in the user’s certificate
* **%rrn%**: this is replaced by the RRN in the user’s certificate
* **%d(XXX)%**: this is replaced by the signing date, XXX is according to Java’s SimpleDateFormat [4]
e.g. %d(“d MMMM yyyy)% is replaced by e.g. “4 March 2021” in English

For the “font”, a number of fonts will be installed and their names documented. Currently there is the DSS default font and a “freescpt” font that resembles handwriting.

The “textAlignH” is the horizontal text alignment and can be “LEFT”, “CENTER”, “RIGHT”
Default value is “NONE”, see [5]

The “textAlignV” is the vertical text alignment and can be “TOP”, “MIDDLE”, “BOTTOM”
Default value is “NONE”, see [6]

The “textPos” defined the position of the text relative to the image: “TOP”, “BOTTOM”, “LEFT”, “RIGHT”

The “image” is a base64 image, the type (png, jpg, …) is automatically detected;
or it can be the base64 encoding of the word “default”: "ZGVmYXVsdA==".

In case the “psfP” parameter is set to *true* in the getTokenForDocument call, then the photo of the eID card is used and the “image” value in the PSP is ignored.

All values are optional; in case of absence a default is used. If no “image” is specified then no image is used.

**3. Example query parameters in the redirect URL to BOSA**

* psfN=signature\_1 # PDF signature field name
* psfC=1,200,50,300,50 # PDF signature field coordinates: page,x,y,width,height
* psfP=true # if the photo should be read from the eID and used in the signature field

 (this is because the token can't be parsed by the front-end)

Remarks:

* psfN and psfC cannot be used together
* In case of psfN, the PDF should contain a visible signature field with this value
* If (a) visible signature field(s) is/are present, then either psfN or psfC must be specified

**4. Current restrictions:**

* With the default font, the “textSize” parameter seems to be ignored
* The “freescpt” font is not by default installed on Linux
* When the image is on top or below the text, it expands or contracts horizontally in order to take the full width

## 3.7. Disabling downloads

Normally, the user signing the document is will be offered the signed version of the document at the end of the signing process. In case this is not desired, a “noDownload” parameter may be added to the “getTokenForDocument” call, with its value set to “true”:

|  |
| --- |
| { *"name": "minfin",* *"pwd":"U84SnLEQvp",* *"in":"test.pdf",* *"out":"signed\_test.pdf",****"noDownload": true,***} |

A document signed in this way will still be uploaded to the BOSA S3 service, but its signed version will not be made available to the user through the BOSA signing service.

Please note that only the download of the *signed* version will be blocked. The *unsigned* version *will* still be made available, because the user needs to know what they are signing.

## 3.8. Request a specific language when calling the service WEB UI

***private static String LANGUAGE = "en";*** *// options: en, nl, fr, de*

*// 3. Do a redirect to the BOSA DSS front-end*

*// Format:* ***[Fout! De hyperlinkverwijzing is ongeldig.](https://%7Bhost:port%7D/sign/%7Btoken%7D?redirectURL=%7BcallbackURL%7D&language=%7Blanguage%7D)***

*System.out.println("\n3. Redirect to the BOSA DSS front-end");*

*String callbackURL = localUrl + "/callback?filename=" + outFileName;*

*System.out.println("  Callback: " + callbackURL);*

***String redirectUrl = bosaDssFrontend + "/sign/" + URLEncoder.encode(token) +***

***"?redirectUrl=" + URLEncoder.encode(callbackURL) + "&language=" + LANGUAGE;***

## **3.9. Using XLST file**

*// 2. Do a 'getToken' request to the BOSA DSS*

*String json = "{\n" +*

 *" \"name\":\"" + s3UserName + "\",\n" +*

 *" \"pwd\":\"" + s3Passwd + "\",\n" +*

 *" \"in\":\"" + inFileName + "\",\n";*

*if (null != xsltFile)*

***json += ( " \"xslt\":\"" + xsltFile.getName() + "\",\n" );***

 *json += " \"out\":\"" + outFileName + "\",\n" +*

 *" \"prof\":\"" + profileFor(inFileName) + "\"\n" +*

*"}";*

# 4. Sample/test FPS code and service

A simple Java service has been made that can serve as an example for the FPS.

It provides a simple ‘home page’ where the use can select a document to sign.
Then the sample service uploads this document, requests a token and does a redirect.
Finally a callback page is foreseen where the signed document is downloaded and the unsigned and signed document are deleted from the S3 server.

To build the sample: install maven and type the following on a command prompt:
 *mvn package*

To run the sample service on Linux/MacOSX:
 *java -cp "target/lib/\*":target/test\_fps-0.0.1-SNAPSHOT.jar com.zetes.projects.bosa.testfps.Main*Or on Windows:
 *java -cp "target/lib/\*";target/test\_fps-0.0.1-SNAPSHOT.jar com.zetes.projects.bosa.testfps.Main*

When started successfully, the service will show the URL to which to surf to:

|  |
| --- |
| *Service started - press Ctrl-C to stop**Surf with your browser to http://localhost:8080* |

When the service starts, it reads configuration parameters from the ***config.txt*** file.
*To actually be able to connect to the BOSA S3 service, an account should be requested from BOSA; and this account information (username and password) must be filled in this config file (i.e. replace the USERNAME\_HERE and PASSWORD\_HERE strings).*

**Demo service**

The above code has been deployed and can be found at the following URL: [3].

# 5. References

[1] MinIO: <https://en.wikipedia.org/wiki/MinIO>

[2] MinIO SDKs: <https://docs.minio.io/> - select ‘MINIO SDKS’ on the left

[3] Sample/test FPS service: [https://mintest.qa.fts.bosa.belgium.be](https://mintest.qa.fts.bosa.belgium.be/)

[4] <https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>

[5] <https://ec.europa.eu/cefdigital/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureAlignmentHorizontal.html>

[6] <https://ec.europa.eu/cefdigital/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureAlignmentVertical.html>