|  |  |
| --- | --- |
|  | BOSA SIGNATURE SOLUTION  Technical information |

## 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.

## Content

[Summary 2](#_Toc111541803)

[Content 2](#_Toc111541804)

[1. Introduction FTS 3](#_Toc111541805)

[1.1. Signing flow 3](#_Toc111541806)

[1.2. Signing software 3](#_Toc111541807)

[2. Getting started 5](#_Toc111541808)

[2.1. Environments and development process 5](#_Toc111541809)

[2.2. Code repositories 5](#_Toc111541810)

[2.3. Main URLs 7](#_Toc111541811)

[2.4. Setup storage buckets 7](#_Toc111541812)

[2.5. Signing profiles 9](#_Toc111541813)

[2.6. IP Addresses BOSA 9](#_Toc111541814)

[2.7. Example code 10](#_Toc111541815)

[2.8. Details 10](#_Toc111541816)

[3.1. BOSA S3 server 11](#_Toc111541817)

[3.2. The getToken calls 12](#_Toc111541818)

[3.3. The HTTP redirect (FPS to GUI-Sign) 17](#_Toc111541819)

[3.4. The callback (GUI-Sign to FPS) 18](#_Toc111541820)

[3.5. Error codes 20](#_Toc111541821)

[3.6. PDF visible signatures 21](#_Toc111541822)

[3.7. Add signing policies 25](#_Toc111541823)

[3.8. Ask the user to validate the reading of the document before signing 26](#_Toc111541824)

[3.9. Restrict the users who are allowed to sign the document 26](#_Toc111541825)

[3.10. General options 27](#_Toc111541826)

[3.11. Request a specific language when calling the service WEB UI 28](#_Toc111541827)

[3.12. Using XLST file 29](#_Toc111541828)

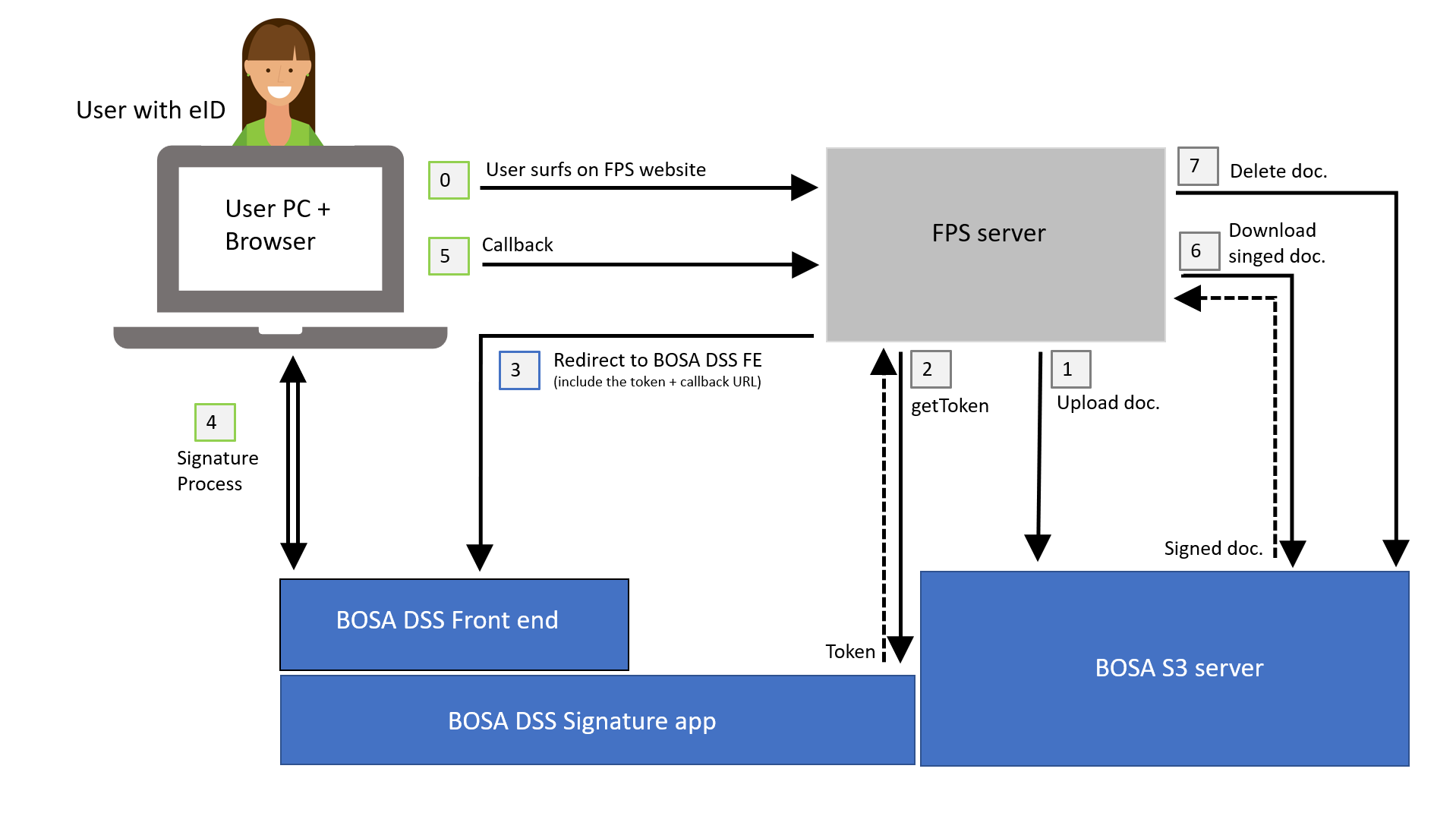
[Sample/test FPS code and service 30](#_Toc111541829)

|  |  |
| --- | --- |
|  |  |

## Introduction FTS

### 1.1. Signing 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.

* Citizens installing the software do 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> .

Supported environments:

* Windows/mac/linux
* firefox/chrome (+edge)/safari
* Out of support : 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-beidconnect>
  + Client application (BeidConnect)

### 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://validate.qa.fts.bosa.belgium.be/signandvalidation/signing/getTokenForDocuments>
  + URL for Token request
  + API endpoint
* [https://sign.qa.fts.bosa.belgium.be/sign](https://sign.fts.bosa.belgium.be/sign)
  + 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
* the format of the Xades Multifile output document

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
  + 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:

|  |  |  |  |
| --- | --- | --- | --- |
| **PROFILENAME** | **PROFILE** | **FORMAT** | **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 |  |
| MDOC\_XADES\_LTA | XadES\_BASELINE\_LTA | INTERNALLY DETACHED | **Custom Xades multi document for Justice** |
| 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 Addresses 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 (in PROD it is only accessible from internal network) 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).

### 2.8. Details

### **3.1. BOSA S3 server**

BOSA provides an Amazon S3 compatible server: MinIO. See [here](https://en.wikipedia.org/wiki/MinIO).

The FPS uses this server to:

* Upload unsigned documents and additional files such as XSLT files or ‘PDF signature profiles’ (PSP)
* Download signed documents and JSON signature validation reports
* 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, …) [Here](https://docs.minio.io/) (select ‘MINIO SDKS’ on the left).  
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 calls

There are two getToken calls:

* getTokenForDocument
  + Single file version
* getTokenForDocuments
  + This version supports new options and multi-file signature
  + It also supports single file operations

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’.

The content of the token is opaque to the client and represents the current signing session. It can’t be modified. It only has to be sent in the HTTP redirect call to the BOSA DSS front-end server.

#### 3.2.1 getTokenForDocument

getTokenForDocument example request:

|  |
| --- |
| *{*  *“name”:”minfin”,*  *“pwd”:”a password”,*  *“in”:”test.xml”,*  *“xslt”:”test.xslt”,*  *“out”:”signed\_test.xml”,*  *“prof”:”XADES\_1”,*  *“lang”: “fr”,*  *“noDownload”: true,*  *“policyDescription”: “The Description”,*  *“policyDigestAlgorithm”: “SHA512”,*  *“policyId”: “http://URL\_OF\_THE\_POLICY”,*  *“psfC”: “1,30,20,180,60”,*  *“psfN”: “signature\_1”,*  *“psfP”: true,*  *“psp”: “pspFileName.psp”,*  *“requestDocumentReadConfirm”: true,*  *“signTimeout”: 120,*  *“allowedToSign”: [*  *{*  *“nn”: “BELGIAN\_NATIONAL\_NUMBER”*  *}*  *]*  *}* |

|  |  |  |  |
| --- | --- | --- | --- |
| **JSON Field** | **Mandatory** | **Values** | **Description** |
| *Name* | Yes | aS3buckName | The name of the bucket where the files (pdf/xml/xlst/psp) are stored |
| *Pwd* | Yes | Ag00dPasswOrd! | The password of the bucket where the files are stored |
| *In* | Yes | Test.xml / test.pdf | The name of the input file (stored on the bucket) |
| *Xslt* | *No* | Test.xslt | If the “in” file is of xml format, the “xlst” will be used to pimp the display of the “in” file as HTML. |
| *Out* | Yes | Signed.xml / Signed.pdf | The output file name on the bucket. (The file extension should be the same as the “in” file extension) |
| *Prof* | Yes | XADES\_1 / PADES\_LTA | The signature profile requested. It is linked to the “in” file type. BOSA provides a list of the available values and their meaning. |
| *Lang* | No | de/nl/en/fr | The language that will be used to display the visible part of a signature in a PDF document |
| *noDownload* | No | true/false | If “true”, the user will not be allowed to download the signed file |
| policyId | No | http://...... | Only for XML documents. If present it will be added to the signature. |
| policyDescription | No | Some text | Only for XML documents. If present it will be added to the signature. |
| policyDigestAlgorithm | No | SHA512/SHA256/… | Only for XML documents. If present it will be added to the signature. |
| *psfC* | No | 1,30,20,180,60 / default | The place and size where to display the visible signature in a PDF document (Page#/X/Y/Width/Height) or “default” |
| *psfP* | No | true / false | If “true”, the picture present on the EID card will be added to the visible signature of a PDF |
| *psfN* | No | signField1 | If defined, the visible information of a PDF signature will be place in an existing field (Acroform) already present in the “in” file. |
| *Psp* | No | test.psp | If present, defines the name of a “bucket” file containing a graphical layout used to render the visible signature of a PDF. |
| requestDocumentReadConfirm | No | true/false | If “true”, display a “I have read this document.\*” checkbox to the sign user interface that must be checked before signature is allowed |
| *allowedToSign* | No |  | A list of “National Numbers” if EIDs that are allowed to sign the document |
| *signTimeout* | No. If not defined, value is 120 seconds. | 0 to 10000 seconds | If the time between a user clicks on the “I want to Sign” button and the moment he signs the document is greater than the number of seconds defined by “signTimeout” the signature will be rejected. |

#### 3.2.2 getTokenForDocuments

To allow multifile signature requirements a new “getTokenForDocuments” was added.

The first use case is “Xades Multifile”, which allows:

* Creating a single XML file containing the files encoded in Base64 as XML elements
* Signing the XML elements as Xades “internally detached”
* Support custom XML output formats through an XSLT file
* Allow signing file without preview screen

In order to support future uses cases the “signType” parameter was added with value “XadesMultiFile”. Note that only “MDOC\_XADES\_LTA” signProfile can be used with XadesMultifile

The native XML file created will follow this format:

<root>

<file id=”ID1” name=”File1” size=”123”>BASE64\_File\_content</file>

<file id=”ID2” name=”File2” size=”456”>BASE64\_File\_content</file>

</root>

An “outXsltPath” parameter allows the client transform this native format to fix their custom XML formats.

Example XSLT:

<?xml version="1.0"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<xsl:comment> Created By BosaSign 1.0 </xsl:comment>

<namesp:SignedDoc xmlns: namesp ="http:// fgov.be">

<xsl:for-each select="root/file">

< namesp:DataFile ContentType="EMBEDDED\_BASE64"

id="{@id}"

fileName="{tokenize(@name, '/')[last()]}"

mime="{tokenize(@name, '\.')[last()]}"

size="{@size}" />

</xsl:for-each>

</ namesp:SignedDoc>

</xsl:template>

</xsl:stylesheet>

A second use case (signType : “Standard”) is available in order to sign a single or multiple documents.

getTokenForDocuments example request:

|  |
| --- |
| *{*  *"bucket":"0qtp70go8dsfsdfsfdsfs88z",*  *"password":"a password",*  *"signType": "Standard",*  *"outDownload": false,*  *"selectDocuments": true,*  *"previewDocuments": true,*  *"requestDocumentReadConfirm": false,*  *"outPathPrefix": "OUT\_",*  *"signProfile": "PADES\_LTA",*  *"altSignProfile": "XADES\_LTA",*  *"nnAllowedToSign": "3423423423,45345345354",*  *"policy": {*  *"id": "http://signinfo.....",*  *"digestAlgorithm": "SHA512",*  *"description": "The policy"*  *},*  *"signTimeout": 200,*  *"inputs": [*  *{*  *"filePath": "test.pdf"*  *“pspFilePath”: “test.psp”,*  *“psfC”: “01,10,10,200,100”,*  *“psfP”: true*  *},*  *{*  *"filePath": "quotes.xml",*  *"displayXsltPath": "quotes.xslt"*  *},*  *{*  *"filePath": "nd\_test.pdf”,*  *“psfN”: “signatureID”,*  *“signLanguage”: “en”*  *},*  *{*  *"filePath": "test.xml"*  *}*  *]*  *}* |

|  |  |  |  |
| --- | --- | --- | --- |
| **JSON Field** | **Mandatory** | **Values** | **Description** |
| *signType* | No | Standard / XadesMultiFile | Default : XadesMultiFile  **Standard***: 1 to n input files producing as many signed files (See also outPathPrefix)*  **XadesMultiFile***: Sign 1 to N files and place their content, as Base64 in and XML elements, then sign this document* |
| *bucket* | Yes | aS3buckName | The name of the bucket where the files (pdf/xml/xlst/psp) are stored |
| *password* | Yes | Ag00dPasswOrd! | The password of the bucket where the files are stored |
| *inputs* | Yes | See below | The input values for each file to be signed |
| *input.filePath* | Yes | “folder/infile.pdf” / “file.xml” | The file path of the file to sign |
| *input.displayXsltPath* | No | “folder/quote.xslt” / “test.xslt” | When signing an XML file, the xslt will be applied to the XML file to render it in a more user friendly way |
| *input.xmlEltId* | No | “ID1”, “ABCDE”, ... | Only for ‘XadesMultiSign’. The input file will be stored as Base64 in the “file” XML element with XML attribute ‘Id’ set to the value of “xmlEltId” |
| *inputs.psfC* | No | 1,30,20,180,60 / default | The place and size where to display the visible signature in a PDF document (Page#/X/Y/Width/Height) or “default” |
| *inputs.psfP* | No | true / false | If “true”, the picture present on the EID card will be added to the visible signature of a PDF |
| *inputs.psfN* | No | signField1 | If defined, the visible information of a PDF signature will be place in an existing field (Acroform) already present in the “in” file. |
| *inputs.pspFilePath* | No | test.psp | If present, defines the name of a “bucket” file containing a graphical layout used to render the visible signature of a PDF. |
| *inputs.signLanguage* | No | de/nl/en/fr | The language that will be used to display the visible part of a signature in a PDF document |
| *nnAllowedToSign* | No | 3423423423,45345345354 | A list of comma separated “National Numbers” of EIDs that are allowed to sign the document |
| *signTimeout* | No. If not defined, value is 120 seconds. | 0 to 10000 seconds | If the time between a user clicks on the “I want to Sign” button and the moment he signs the document is greater than the number of seconds defined by “signTimeout” the signature will be rejected. A value of 10 second is considered a minimum for signing a document. |
| *signProfile* | No \* | XADES\_1 / PADES\_LTA | The signature profile requested. It is linked to the “input” files types. BOSA provides a list of the available values and their meaning.  \*For XadesMultiFile please use MDOC\_XDADES\_LTA |
| *altSignProfile* | No \* | Same as signProfile | \* If XML and PDF files are to be signed “signProfile” and “altSignProfile” must be filled, with a profile for XML (XADES\_) files and one for PDF files (PADES\_) |
| *selectDocuments* | No | true/false | If true, when signing muliple files with “Standard” sign type, allow the user to sign a subset of the proposed files |
| *previewDocuments* | Yes | true/false | If “false”, in combination with “XadesMultiFile” sign type, disables file preview. Otherwise must be set to “true” |
| requestDocumentReadConfirm | No | true/false | If “true”, display a “I have read this document.\*” checkbox to the sign user interface that must be checked before signature is allowed |
| *outPathPrefix* | Depends on signType | “Folder/signed\_” / “outSigned”/... | For “Standard” sign type, will be used to prefix the filename of each input file to create the signed output file |
| *outXsltPath* | No | Justice.xslt | For ‘XadeMultiFile’ only, the xslt permits to have user defined format output XML |
| *outFilePath* | No | “Folder/Signed.xml” / “Signed.pdf” / ... | The output file name on the bucket. (The file extension should be the same as the “in” file extension)  Valid only for single file output |
| *outDownload* | No | true/false | If “true”, the user will be allowed to download the signed file |
| *policy* | No | The enclosing object | The parameters of the policy to apply for an XML signature |
| policy.Id | No | http://...... | The URL where the policy is hosted. Only for XML documents. If present it will be added to the signature. |
| Policy.description | No | Some text | A textual description. Only for XML documents. If present it will be added to the signature. |
| policy.digestAlgorithm | No | SHA512/SHA256/… | Only for XML documents. If present it will be added to the signature. |

### 3.3. The HTTP redirect (FPS to GUI-Sign)

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 mandatory token obtained in the getToken call: this is at the end of the URL path
* And 4 HTTP GET parameters:
  + A mandatory RedirectUrl to the FPS
    - It must be URL encoded
  + An optional language parameter
    - Values: en, nl, fr, de
  + A list of optional names parameter
    - “name” : default value for all languages
    - “name\_fr”, “name\_nl”, “name\_de”, “name\_en” : value for a specific language
    - Any combination is name\_ is possible. If a value is not defined for the browser language “name” value is used
    - They are used to show the FPS name in the User’s browser before the redirect
    - It must be URL encoded
  + An optional HookURL
    - A URL that will be called at specific moment during the sign progress

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* |

HookURL

This URL will receive HTTP “POST” JSON requests from the browser at some steps in the signing flow.

The service handling the HTTP “POST” request MUST allow CORS from the calling domain.

See <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> for more information.

The step of the workflow will be identified by the “id” field:

{

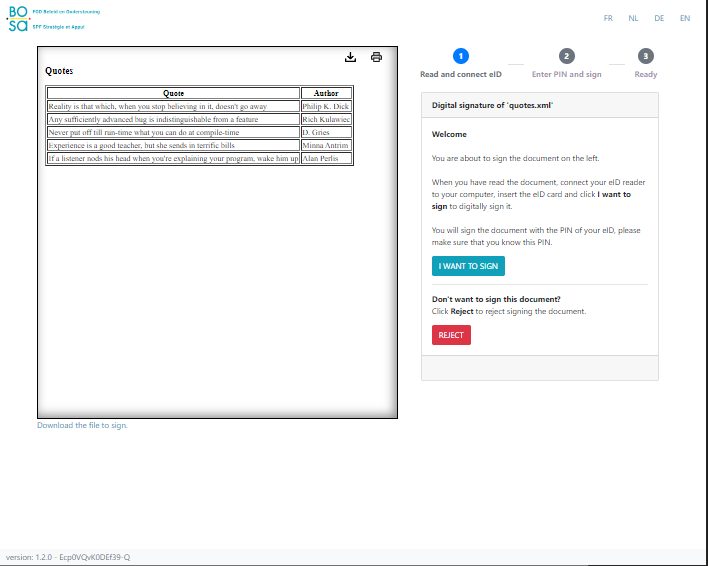
"id": "FILE\_SIGNED",

"fileName":"nd\_test.pdf"

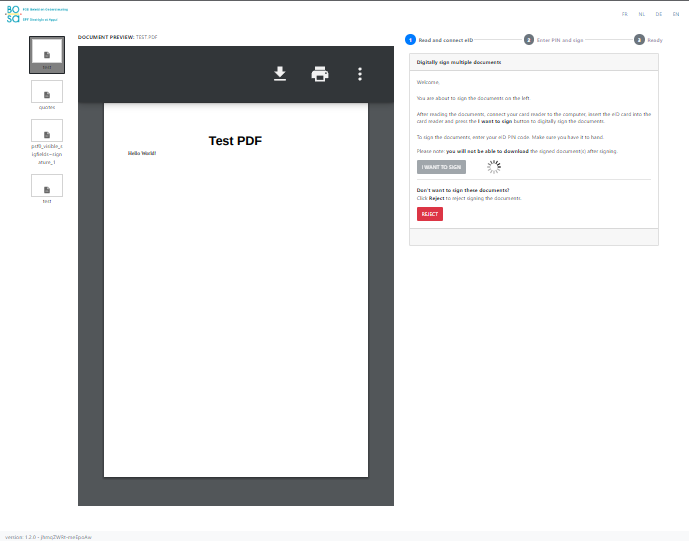
}

The above message means the file “nd\_test.pdf” was fully signed

With the token, the user’s browser will download the unsigned document(s) and, if present, the XSLT file(s) from the BOSA DSS server. It will then display the document(s) (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.



**Single XML file (with XSLT preview) signature**



**Multifile signature**

### 3.4. The callback (GUI-Sign to FPS)

After the document is signed and available on the BOSA S3 server, the User’s browser will be redirected to the callback URL specified in the redirect (see above). The callback URL is completely specified by the FPS and can contain extra query parameters (i.e. a session ID) besides “RedirectUrl", "language" and ”name”.

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 error during the signature session, the following parameters are added to the RedirectUrl:

* 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](http://eservices.minfin.fgov.be/callback?session=762087326&ref=20210423092112378&err=CERT_REVOKED)****[&ref=20210423092112378&err=CERT\_REVOKED](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.

### 3.5. 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 | Origin |
| BEID\_CONNECT\_ERROR | Couldn't connect to the eID card | Browser |
| NO\_READER | No reader found on the user's PC | Browser |
| UNSUPPORTED\_READER | unsupported smart card reader | Browser |
| SIGNATURE\_FAILED | eID card returned an invalid signature | Browser |
| CARD\_BLOCKED | eID card blocked | Browser |
| CARD\_ERROR | eID card error | Browser |
| USER\_CANCELLED | User cancelled the signing operation | Browser |
| SIGN\_CERT\_EXPIRED | Signing certificate expired | Backend |
| CERT\_CHAIN\_INCOMPLETE | No or incomplete certificate chain | Backend |
| NO\_SIGN\_CERT | No signing certificate provided | Backend |
| INVALID\_S3\_LOGIN | Invalid user name or password | Backend |
| NO\_DOC\_TO\_VALIDATE | DSSDocument is null | Backend |
| NO\_CERT\_TO\_VALIDATE | The certificate is missing | Backend |
| CERT\_REVOKED | Certificate (probably the signing cert) revoked | Backend |
| INTERNAL\_ERR | Unexpected error occured | Backend |
| INVALID\_DOC | Document validation (after signing) failed | Backend |
| UNKNOWN\_PROFILE | Unknown signature profile | Backend |
| EMPTY\_PARAM | Empty (null) parameter in request | Backend |
| INVALID\_TOKEN | Invalid token in request | Backend |
| INVALID\_SIGNATURE\_LEVEL | Invalid signature level for document | Backend |
| NOT\_ALLOWED\_TO\_SIGN | Selected certificate is not allowed to sign (See NN allowed to sign) | Backend |
| ERR\_PDF\_SIG\_FIELD | PDF signature field problem: not specified when needed or invalid value | Backend |
| SIGN\_PERIOD\_EXPIRED | The “SignDocumentForToken” call was executed too late (See “signTimeout”) | Backend |
| STORAGE\_ERROR | Error with the S3 storage | Backend |
| \_ERROR | Suffix added to generic backend errors | Backend |
| INVALID\_PARAM | Returned by the GetToken calls when input is invalid | Backend |
| BLOCKED\_DOWNLOAD | User not allowed downloading signed files | Backend |

### 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.

**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 parameters 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 visible PDF signature is created.

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:**

|  |
| --- |
| {  "version” : 2,  "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: the psp json file is optional, if the file is not provided or if any value is missing from a provided PSP file, default values will be used.

**3. PSP Values and defaults**

|  |  |  |
| --- | --- | --- |
| Field name | Default value | Explanation |
| bgColor | #D0D0D0 | V1 : Background color of both the text and the signature  V2 : Background color of the text  Format : |
| Texts |  | A list of ‘language’/’text’ pair  See below “Some PSP parameters in detail ” |
| textSize | 14 | The size of the font to use |
| font |  | The name of the font to use  V1 Format : <FontName>/<b><i> ( Sample : “Serif/bi” )  For V2 the “/<b><i>” are removed from the name |
| textPadding | 20 | # pixels between the text and the image |
| textAlignH | LEFT | The horizontal text alignment and can be “LEFT”, “CENTER”, “RIGHT” see [here](https://ec.europa.eu/cefdigital/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureAlignmentHorizontal.html) |
| textAlignV | TOP | [here](https://ec.europa.eu/cefdigital/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureAlignmentVertical.html) |
| textPos | BOTTOM | The position of the text relative to the image: “TOP”, “BOTTOM”, “LEFT”, “RIGHT” |
| textColor | #0000FF |  |
| defaultCoordinates |  | Same format as psfC. Not used |
| imageDpi | 400 | Image resolution |
| Image |  | An image to used in the signature |
| version | 1 | If version is missing or ‘1’ redering will be made by the legacy renderer.  Version ‘2’ or above will be rendered using DSS library.  The V1 fields thererefore will behave slightly differently when used as V2 inputs |
| textWrapping | FONT\_BASED | Values: FILL\_BOX, FILL\_BOX\_AND\_LINEBREAK, FONT\_BASED  V2 only – see [here](https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/TextWrapping.html) |
| imageScaling | ZOOM\_AND\_CENTER | Values: CENTER, STRETCH, ZOOM\_AND\_CENTER  V2 only – see [here](https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/ImageScaling.html) |
| horizAlignment | NONE | Values: CENTER, LEFT, NONE, RIGHT  V2 only – see [here](https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureAlignmentHorizontal.html) |
| vertAlignment | NONE | Values: BOTTOM, MIDDLE, NONE, TOP  V2 only – see [here](https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureAlignmentVertical.html) |
| bodyBgColor | #D0D0D0 | V2 only |
| Rotation | AUTOMATIC | Values : AUTOMATIC, NONE, ROTATE\_180, ROTATE\_270, ROTATE\_90  V2 only – see [here](https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/apidocs/eu/europa/esig/dss/enumerations/VisualSignatureRotation.html) |
| zoom | 100 | V2 only |

**4. Some PSP parameters in detail**

**4.1 TEXTS**

The “texts” is a “per-language” list of text to be used in the signature.

It can contain the following macros that will be replaced by information from the Subject DN of the signing certificate:

* **%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
* **%nn%**: this is replaced by the national numbers (subject's "SERIALNUMBER") in the user’s certificate
* **%d(XXX)%**: this is replaced by the signing date, XXX is according to Java’s [SimpleDateFormat](https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html)  
  e.g. %d(“d MMMM yyyy)% is replaced by e.g. “4 March 2021” in English

Example of a signing certificate with Subject DN:

commonName=Tom Test

serialNumber=73040102749

givenName=Tom

surname=Test,countryName=BE

"texts" : {  
 "en": "Signed by %gn% %sn%\nOn %d(MMM d YYYY)%",  
 "nl": "Getekend door %gn% %sn%\nOp %d(MMM d YYYY)%",  
 "fr": "Signé par %gn% %sn%\nLe %d(MMM d YYYY)%",  
 "de": "Unterzeichnet von %gn% %sn%\nAm %d(MMM d YYYY)%"  
},

Example "texts” field from a PSP

**4.1 FONT**

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.

**4.2 IMAGE**

The “image” is a base64 image, the type (png, jpg, …) is automatically detected.  
If value is "ZGVmYXVsdA==" (“default” in Base64) then the image  will be used

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

If no “image” is specified in the PSP then no image is used.

**5. 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 (Acroform) with this value
* If (a) visible signature field(s) is/are present, then either psfN or psfC must be specified

**6. Current restrictions of V1 PSP:**

* 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

Request user to validate lecture of the document before signing

### 3.7. Add signing policies

Signing policies are only when signing an XML document . A set of parameters can be added to the "getTokenForDocument(s)" call.

* policyId: string containing the URL of the signature policies. This URL must be public and immutable since it is part of the signature, and must be accessible during signature validation.
* policyDescription: string containing a "short" description.
  + Optional parameter
* policyDigestAlgorithm: string containing the hashing algorithm to hash the signature policies file referenced in the “policyId” parameter.
  + Default value is “SHA256”

{

*"name": "minfin",*

*"pwd":"U84SnLEQvp",*

*"in":"test.pdf",*

*"out":"signed\_test.pdf",*

***"policyId": "http://[url to the signing policies file].pdf",***

***"policyDescription": "Policy Description",***

***"policyDigestAlgorithm": "SHA512"***

}

### 3.8. Ask the user to validate the reading of the document before signing

By default, the user signing the document will not be prompted to confirm reading the document before signing. If this is not desired and the user needs to confirm the read, a "requestDocumentReadConfirm" parameter can be added to the "getTokenForDocument" call, with its value set to "true":

{

*“name”: “minfin”,*

*“pwd”:”U84SnLEQvp”,*

*“in”:”test.pdf”,*

*“out”:”signed\_test.pdf”,*

***“*** ***requestDocumentReadConfirm”: true,***

}

### 

### 3.9. Restrict the users who are allowed to sign the document

By default, any user (presenting a valid certificate) is authorized to sign the document. If document signing should be restricted to a list of authorized users, an "allowedToSign" parameter can be added to the "getTokenForDocument" call. This parameter will define a list of national numbers (nn) authorized to sign the document. The "nn" (national number) must be formatted as defined in the subject's "SERIALNUMBER" certificate. Up to 32 "nn" entries are supported.

{

*"name": "minfin",*

*"pwd":"U84SnLEQvp",*

*"in":"test.pdf",*

*"out":"signed\_test.pdf",*

***"allowedToSign": [***

***{***

***"nn": "12345678900"***

***},***

***{***

***"nn": "01050399864"***

***}***

***]***

}

### 

### 3.10. General options

#### 3.10.1 Download option

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 similar parameter exists for “getTokenForDocuments” : “outDownload”. The different name reflects the inverted logic.

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.

ZIP file downloads

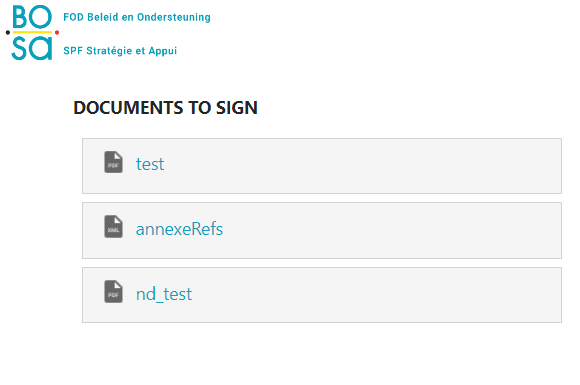
For ‘Standard’ signing type, when signing multiple documents, the files are stored individually on the S3 server but the download proposed is a ZIP file containing the signed files.

#### 

#### 3.10.1 Preview

The ‘XadesMultiFile’ signing type has the option to display only the file names of the files to sign instead of the file preview. Since WYSIWYS (What you sign is what you see) is mandatory with EIDAS the file names are also download links.

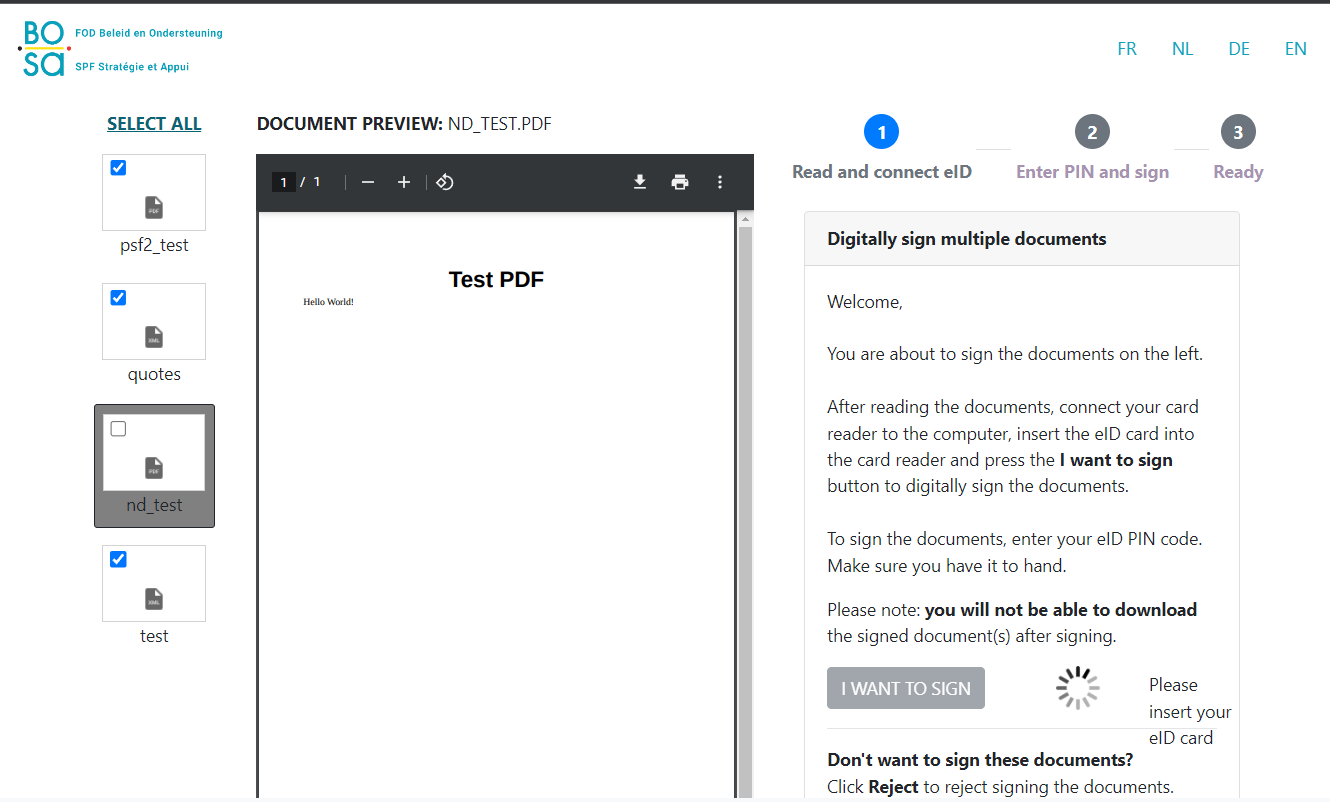
If “*previewDocuments”* parameter is set to “false” the documents will not be previewed



**Multiple documents without preview**

#### 3.10.1 File selection

For multiple documents use cases, if “*selectDocuments”* parameter is set to “true”, the documents in the token provided list are individually selectable.



**Multiple documents with preview and File selection**

### 3.11. 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.***

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

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

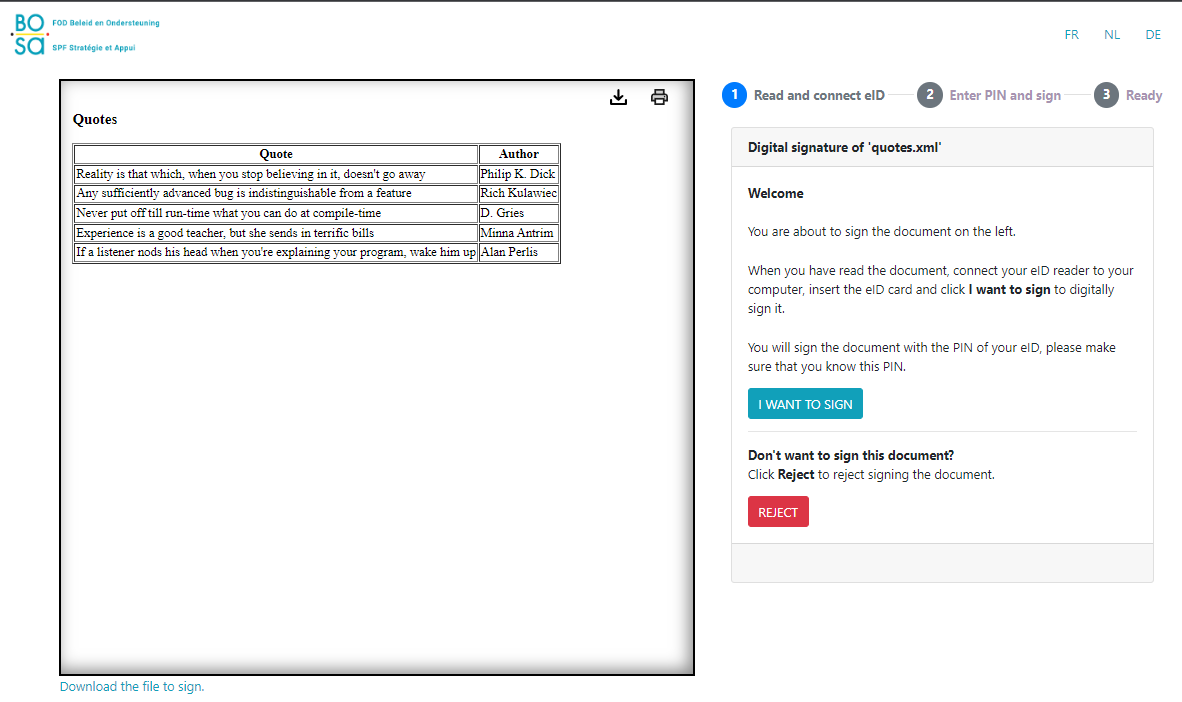
*System.out.println("  Callback: " + callbackURL);*

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

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

### **3.12. Using XLST file**

XML documents are not user friendly, if at all readable. In order to meet the “WYSIWYS” requirement an XSLT file can associated to each XML file and display a more readable version of the information.



**An XML displayed with an XSLT**

As the XSLT modifes the view of the documents it also poses a challenge to future non-repudiation uses.

Therefore it is advised to implement the below measures:

* display ALL the information in the XML in some form in the XSLT transformed view.
* Include the XLST or a HASH of it in the data to sign
* Maintain historical copies of the XSLTs
* Any measure to maintain non-repudiation

## 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:

[https://mintest.qa.fts.bosa.belgium.be](https://mintest.qa.fts.bosa.belgium.be/)