Assinare Daemon Quick Guide v2.0

Introduction

Assinare Daemon is a product that allows a web application to communicate with smartcards enabling the extraction of data from a user's Citizen Card and/or the signature of documents.

With the removal of Applet support from most browsers, a different approach was required for allowing access to smartcards from the Web. To tackle this problem, Linkare created Assinare Daemon, a solution comprised of a Java Web Start application running locally as a daemon on the user's machine, to which a web application makes requests via AJAX.

Assinare Daemon currently imposes no restrictions on the technology or architecture used by the web application and is compatible with nearly all browsers currently on the market.

It is highly suitable for scenarios where an organization maintains control of the machines used by the end user, e.g. intranet, PoS, etc. Additionally with a few limitations, it is also applicable in scenarios open to the public, where the end user controls its own machine.

Architecture

The Assinare Daemon architecture is based on three main components:

- Web Application
- Daemon File Store
- User's PC

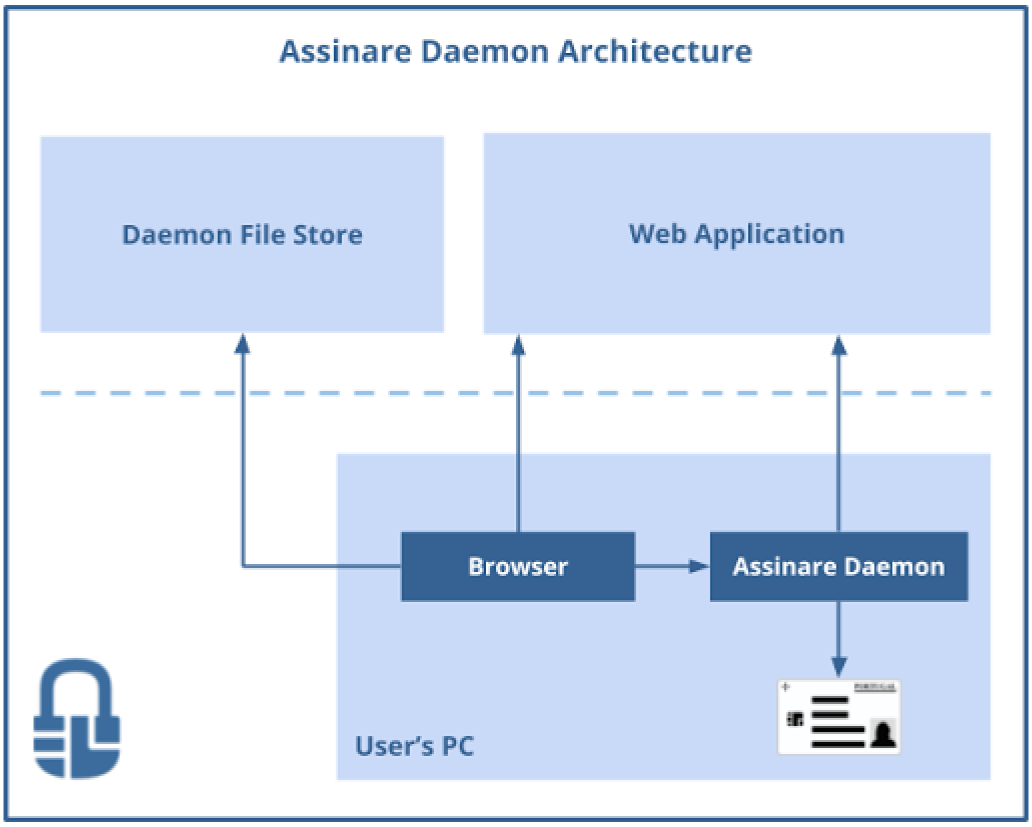
The Web Application shall be the web application provided by the Client to the end users (for instance the Client's Document Management System), suffering some minor modifications like the capability to initiate a Java Web Start application and the capability to make AJAX requests to the Assinare Daemon.
Within the Client's organization, several Web Applications may exist that make use of the same Assinare Daemon installation. This can mean several installations of the same Web Application in distinct environments or completely distinct applications.
Each Web Application involved must offer two HTTP services:

- One that receives a document ID via GET and returns the document's contents.
- Another that receives a document ID and its contents via POST and stores it.

The Daemon File Store shall be a simple HTTP server, hosting the JNLP and JAR files required to execute the Assinare Daemon as a Java Web Start application.
NOTE: The Daemon File Store might be hosted in the same machine, or even the same application server, as the Web Application. However, keeping them conceptually separate allows for some interesting scenarios, like the usage of Linkare's own demo servers as the Daemon File Store during evaluation/testing/development of the solution.

The User's PC shall be, as the name indicates, the PC used by the user. This does not mean that the user might necessarily have ownership of that machine as is the case for scenarios like intranets or PoS machines. This machine must have the Java JRE installed and a working browser, which must be configured to trust on the CA which issued the SSL certificate used by the daemon.

The relationship between these components is highlighted on the diagram below:



Typically, a signature process shall follow these steps:

1. Using the browser on his PC, the User accesses the Web Application and identifies a document that requires a digital signature.
2. On the Web Application, the User activates the button (or link) to launch the Assinare Daemon.

3. The Web Application redirects the User’s browser to the JNLP file in the Daemon File Store.
4. The User’s browser downloads the JNLP file and launches a Java Web Start process, passing the JNLP file as an argument.
5. The JWS process retrieves the JNLP and JAR files required from the Daemon File Store and initiates the Assinare Daemon.
6. On the Web Application, the User initiates the signing of a document.
7. The Web Application makes an AJAX request to the Assinare Daemon, passing information about the document, including its location, and various parameters of the signature. The Web Application also registers a JS callback to be executed when the signing process finishes (successfully or otherwise).
8. The Assinare Daemon uses the parameters received to connect to the Web Application and retrieve the document.
9. The Assinare Daemon raises an application window that presents a document preview to the User, collects required information, and generally guides the User throughout the signing process.
10. The User completes the signing process.
11. The Assinare Daemon connects to the Web Application to upload the signed document.
12. The Assinare Daemon responds to the Web Application with the relevant status information.
13. The callback previously registered by the Web Application is executed allowing it to update its interface in any business-relevant fashion.

Obtaining artefacts

Upon request, Linkare shall provide the Client with a customised build of the Assinare Daemon.

To do so, Linkare requires the following informations:

1. The URL on which the Daemon File Store will serve the Assinare Daemon files (JNLP and JARs).
2. The domains on which the Web Application(s) will be made available to the User.
3. Optionally, a certificate (and respective certification chain) may be provided to secure the connection between the User’s Browser and the Assinare Daemon. If this is not provided, Linkare will use its own certificates to secure the connection.

With this information, Linkare shall provide the Client with three artefacts:

- A ZIP file containing the JNLP and JAR files to be placed on the Daemon File Store.
- A ZIP file containing a set of JS files, to be included in the Web Application, that provide an API for communicating with the Assinare Daemon.
- The certificate for Linkare's CA, used to issue the certificate used by the Assinare Daemon's HTTPS endpoints. This will not be needed if the Client provides its own certificate when requesting the customised build (see step 3, above).

Installation

To install Assinare Daemon, perform the following steps:

1. Extract the ZIP file containing the Assinare Daemon files into the correct location in the Daemon File Store.
   This step is mandatory for production or pre-production environments. For development or evaluation environments, however, an evaluation version of Assinare Daemon, hosted by Linkare at test.assinare.eu may be used.
2. Optionally, include polyfills for ECMAScript up to version 2017 and for the Fetch API. These are required by the Assinare JS API.
   Which polyfills are actually necessary, if any, depend on the target browsers defined by each Client.
   A suggested polyfill implementation is polyfill.io, which is known to work with Assinare.
3. Include one or multiple files of the Assinare JS API in the Web Application and load them as described in section Loading the JS API.
4. To simplify some tasks, Oracle's Deployment Toolkit may optionally also be included in the Web Application. See more information here.

To prepare the User's PC for communicating with the Assinare Daemon, perform the following steps:

1. Install the applicable CA certificate on the User's Browser. If the Client provided a certificate to Linkare then that is the one to be installed. Otherwise, the certificate provided by Linkare should be installed.
2. If the User's system is Linux based then the package **PCSC lite** must be installed and the respective daemon must be running when attempting to use the Assinare Daemon.

Launching the Assinare Daemon

To launch the Assinare Daemon it is necessary to redirect the User's Browser to the JNLP URL. This will cause the Browser to download said file and launch Java Web Start, using the file as input.
In the simplest case, we may create a launch link or button, like so:

```
<a href="<jnlp url>">Launch</a>
```

```
<button onclick="window.location='<jnlp url>'">Launch</button>
```

If using Oracle's Deployment Toolkit, we may instead do:

```
<a href="javascript:deployJava.launch('<jnlp url>');">Launch</a>
```

```
<button onclick="deployJava.launch('<jnlp url>')">Launch</button>
```

The JNLP URL to use typically depends on the address and configuration of the Daemon File Store, however, when using the evaluation version of Assinare it assumes the value `http://test.assinare.eu/assinare-web/assinareDaemon.jnlp`.

Loading the JS API

Assinare's JS API is provided as a set of two JavaScript files which implement the same functionality, with the following differences:

- `assinareAPI.es6.min.js` provides the JS API as an ES6 module, uses modern features of the JS language which result in smaller file size and better performance.
- `assinareAPI.es5.min.js` provides the JS API as an ES5 UMD module, offers compatibility down to IE11 at the cost of file size and performance.

Both files can be used together or independently depending on the use case. The recommended way, however, is to use them together in a pattern that allows for "progressive enhancement", as so:

```
<script type="module" src="_path_/assinareAPI.es6.min.js"></script>
<script nomodule src="_path_/assinareAPI.es5.min.js"></script>
```

This pattern will result in only one of the modules being loaded at any one time, the ES6 module being loaded in modern browsers and the ES5 module being loaded in legacy browsers.

Using the JS API

Global objects

The JS API provides the object `assinare` in the page's global scope. This object contains two sub-objects:

- `assinare.id` for extracting data from the smartcard
- `assinare.sign` for creating digital signatures

Obtaining an instance

Both of these objects offer a method `getInstance()`. Both take an object with instantiation parameters and a callback function. The callback function may be omitted, in which case a Promise will be returned. Some examples:

```
let idParams = {};

window.assinare.id.getInstance(
    idParams,
    function (err, newInstance) {
        if (err) {
            // handle error
            console.error(err);
        } else {
            // use newInstance or store it for later
        }
    }
);
```

```
let signParams = {
    getFileUrlPrefix: "...",
    getSignedFileUrlPrefix: "...",
    putFileUrl: "...",
    authCookies: "...",
    language: "...",
    country: "..."
};

window.assinare.sign.getInstance(
    signParams,
    function (err, newInstance) {
        if (err) {
            // handle error
            console.error(err);
        } else {
            // use newInstance or store it for later
        }
    }
);
```

Data extraction instance parameters

None as of yet

Signature instance parameters

`getFileUrlPrefix`
The first part of the URL used by the Assinare Daemon to retrieve documents, via a GET request. The document ID will be appended to it before a request.
For example, to obtain the document with ID `FILE001`, when `getFileUrlPrefix` is configured as `http://test.assinare.eu/assinare-web/GetFile?name=`, the Assinare Daemon will make a request to `http://test.assinare.eu/assinare-web/GetFile?name=FILE001`.

`putFileUrl`
The URL used by the Assinare Daemon to upload signed files to the Web Application. The Assinare Daemon will perform a multipart POST request on this URL. The body of the request will contain the signed documents contents. The document ID will be sent in the `filename` directive of the `Content-Disposition` HTTP header.

`authCookies`
A string in the format of the `Set-Cookie` HTTP header containing the cookies required to identify the User's session on the Web Application, e.g.:
`JSESSIONID=NGr9gQFfKp4WHMZTU3MOpY0mwhgIHyAhxJ8QqIYu;`.
Must be supplied but may be an empty string.
It is used by the Assinare Daemon to make requests to the Web Application on behalf of the User.

`language`
Optional
A two or three letter code identifying the language to use on the Assinare Daemon's interface.
Currently, `en` and `pt` are well supported.

`country`
Optional
A two or three letter code identifying a country to further distinguish the language selected above. It allows, for example, to differentiate between `pt-PT` and `pt-BR`.##### Signature instance parameters

~~`getSignedFileUrlPrefix`~~
~~*Deprecated, Optional*~~

Extracting card data

Any instance acquired from `assinare.id` offers the following three methods, which may be invoked any number of times in any order:

- `getCitizenData(callbackFn)`
- `getCitizenAddress(callbackFn)`
- `getCitizenPicture(callbackFn)`

All receive a single parameter `callbackFn` which shall be a callback function. This parameter may be omitted in which case a Promise will be returned. An example:

```
function processData(err, data) {
    if (err) {
        // handle error
        console.error(err);
    } else {
        // use the data
        console.log(data);
    }
};

assinareIdInstance.getCitizenData(processData);
```

Performing signatures

An instance obtained from `assinare.sign` allows for the signature of:

- A single PDF
- A set of PDFs in batch
- A set of files of any type, which Assinare will collect within a container know as ASiC ([Associated Signature Containers](#)).

To access these operations two methods are available, one for PDFs, the other for containers, to know:

- `signDocuments(docs, callbackFn, signatureParams)`
- `signContainer(docs, callbackFn, signatureParams)`

Either one, receives the same parameters:

- `docs` the list of ids of the documents to sign as a comma-separated string OR as an array
- `callbackFn` a callback function. May be *null* in which case a Promise will be returned
- `signatureParams` a JS object containing several properties to customise the resulting signature. Differs between PDF and container signature

An example:

```
function signingDoneMsg(error, data) {
    if (!error) {
        // signature success
        // data.signatureStatus contains the signature success status
        // data.docName contains the document name
        console.log("Signature success", data.signatureStatus, data.docName);
    } else {
        // handle error
        // error.signatureStatus contains the signature error status
        // error.message contains error message
        console.error("Signature error", error.signatureStatus, error.message);
    }
};

assinareSignInstance.signDocuments(
    "FILE001,FILE002,FILE003",
    signingDoneMsg,
    {}
);
```

PDF signature parameters

`contact`
Default value for the “Contact Details”/“Detalhes do Contacto” field.

`location`
Default value for the “Place”/“Localização” field.

`reason`
Default value for the “Reason”/“Razão” field.

`sigRenderingMode`
Default value for the “Signature Options”/“Opções de Assinatura” field. Controls the appearance of the signature when placed in the document. Must be one of the following strings:

- PRE_DEFINED_LOGO
  Create signature with text and a default logo.
- LOGO_CHOOSED_BY_USER
  Create signature with text and a logo selected by the user. Enables usage of option “logoFileURL” detailed below.
- TEXT_ONLY
  Create signature with text only and no logo.
- INVISIBLE
  Create a signature without a visual representation. Signature will still be viewable and verifiable from detail panels on Acrobat Reader.

`logoFileURL`
Default value for the logo to be used when creating the signature.

`tsaUrl`
URL of the TSA to use when timestamping the signature. May take the special value “none” to disable secure timestamps altogether. If ommited, the TSA related to the Portuguese “Cartão de Cidadão” will be used.

Experimental parameters

`percentX`
Default horizontal position (float) of the signature on the page, as a percentage (0-1) of the page width.

`percentY`
Default vertical position (float) of the signature on the page, as a percentage (0-1) of the page height.

`pageNumber`
Default page (integer) on which to place the signature.

`width`
Default width (integer) of the signature.

`height`
Default height (integer) of the signature.

`fieldName`
Default name to use for the PDF signature field.

`doLTV`
Default value (boolean) for the “Long term validation”/“Validação de Longo Prazo” field.

`uiMode`
Defines the user interface to use for PDF signing. Must be one of the following strings:

- FULL (default)
- SIMPLE
- MINIMAL

Container signature parameters

`location`
Default value for the “Place”/“Localização” field.

`claimedRole`
Default value for the “Signer’s Role”/“Papel do Assinante” field.

`commitmentType`
Default value for the “Type of Commitment”/“Tipo de Compromisso” field. Must be one of the following strings:

- PROOF_OF_ORIGIN
  Corresponds to http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin
- PROOF_OF_RECEIPT
  Corresponds to http://uri.etsi.org/01903/v1.2.2#ProofOfReceipt
- PROOF_OF_DELIVERY
  Corresponds to http://uri.etsi.org/01903/v1.2.2#ProofOfDelivery
- PROOF_OF_SENDER
  Corresponds to http://uri.etsi.org/01903/v1.2.2#ProofOfSender
- PROOF_OF_APPROVAL
  Corresponds to http://uri.etsi.org/01903/v1.2.2#ProofOfApproval
- PROOF_OF_CREATION
  Corresponds to http://uri.etsi.org/01903/v1.2.2#ProofOfCreation

`tsaUrl`
URL of the TSA to use when timestamping the signature. May take the special value “none” to disable secure timestamps altogether. If ommited, the TSA related to the Portuguese “Cartão de Cidadão” will be used.

`withLTA`
Default value (boolean) for the “Long-term Validation”/“Validação de Longo Prazo” field.

`containerName`
Default value for the “Filename”/“Nome do Ficheiro” field.