
The Identity Selector Service Documentation

Release 1.0.0

Leif Johansson

Sep 21, 2020

Contents:

1	Introduction	3
1.1	Architecture	3
1.2	Audience	4
2	Building and Installing	5
3	Configuration	7
4	Deploy to CDN	9
5	Running docker	11
6	Components	13
6.1	OASIS Identity Provider Discovery	13
6.2	Login Button Component	13
6.3	Persistence Service	14
7	Release Notes	15
7.1	Version 1.1.2	15
7.2	Version 1.1.3	15
7.3	Version 1.2.0	15
7.4	Version 1.2.1	15
7.5	Version 1.3.0	15
7.6	Version 1.3.1	16
8	Indices and tables	17

This is the component package for the `thiss.io` suite. This package is used to run `use.thiss.io` and `service.seamlessaccess.org`. The package contains a set of javascript components implementing a shared persistence service for Identity Provider selections and a set of services built on top of that API including a SAML discovery service and an active login button component.

Use together with the `thiss-ds-js` package.

The Identity Selector Software (thiss.io) is an implementation of an identity selector supported by the [Coalition for Seamless Access](#). It implements a discovery service using the [RA21.org recommended practices for discovery UX](#).

The Identity Selector Software suite is a front-channel identity selector for distributed identity ecosystems aka [Federated Identity Management](#). The objective is to simplify the process of choosing an “identity provider” by having the browser remember the users choice in browser local store. Currently the system has been used for large-scale SAML-based identity federations but there are no intrinsic dependencies to SAML as such and the system could be easily adapted to other protocols that follow the common pattern of federation by relying on redirecting the user to an authentication provider of some sort.

The system was designed with privacy as the number one focus. No information is shared with the relying party during the identity provider choice process. This is ensured by relying on the browser security model and judicious use of inter-domain communication using post-message.

This package ([thiss-ds-js](#)) contains the parts needed to write a client that talks to an instance of a [thiss-js](#) service (eg [use.thiss.io](#) or [service.seamlessaccess.org](#)).

1.1 Architecture

The Identity Selector Software (thiss.io) is a set of front-channel (aka browser-based) cross-domain APIs using post-message (built using the [post-robot](#) package):

- A persistence API that allows store & retrieval of information about the last N (3) identity providers used to authenticate a user. Unlike similar project (eg google account chooser) the information stored does not include any PII (eg email-addresses) but only identifies the identity provider used in a way consistent with the authentication protocol used.
- A discovery API that implements [SAML identity provider discovery](#) layered on top of the persistence API

Both of these APIs have a *server* and a *client* component. The client APIs can be found in [thiss-ds-js](#). The server components can be found in this repository. The server component is implemented as javascript running in an iframe fetched from a service URI. This ORIGIN (in the sense of the w3c security model) protects access to the browser local store and ensures that the calling page only has access to the intended API. The calling page (aka the client) is

responsible for initializing the iframe but after this no longer has any control over the code executing inside it. The *server* iframe, while executing in the client browser, is therefore sandboxed from the calling page.

The persistence API is completely protocol agnostic eg has no dependency on SAML, all of which are in the discovery API. Future versions are expected to provide similar APIs for OpenID Connect supporting [OpenID connect federation](#) and possibly other protocols.

1.2 Audience

This documentation is mostly aimed at integrators and developers who want to understand how the components matching the `thiss-ds-js` API are implemented and/or want to deploy their own instances of this software instead of relying on an existing service like `use.thiss.io` or `service.seamlessaccess.org`

Building and Installing

Note that most users won't want to install and deploy their own instance of `this-js` but will most likely want to use one of the existing service instances such as `service.seamlessaccess.org`.

The included Makefile has a number of targets aimed at those who want to build and package their own instance:

- `make setup`: Runs `npm install` to install all node dependencies
- `make start`: Runs a local development instance with a mocked MDQ/Search service (based on `edugain`)
- `make local`: Runs a local development instance for a local `pyFF` instance running on port 8000
- `make build`: Builds the instance running on `this.io` in the `dist` directory
- `make standalone`: Builds a standalone instance used in the docker container (with `envsubst`) in the `dist` directory
- `make sameserver`: Builds a lightweight host agnostic replacement for the deprecated embedded `pyFF DS` when `pyFF` is running on the same server
- `make docker`: Builds a docker container (`this-js:<version>`) based on `standalone` and `nginx`

The thiss-js application is a set of SPAs and web components that are configured via environment variables via calls to `process.env`. Deploying the apps essentially either amounts to building the app with the environment variables set, or substituting the environment variables at runtime. This latter approach is what is done in the docker container `start.sh`.

The thiss-js button component is partially configured by the caller that can pass several parameters into the button change its behaviour. This is documented in detail in the `thiss-ds-js` package.

Basic parameters:

- `MDQ_URL`: URL to the MDQ (metadata query protocol) endpoint
- `SEARCH_URL`: URL to the MDQ search service
- `BASE_URL`: the URL where the applications are published
- `STORAGE_DOMAIN`: the `ORIGIN` used for the storage/persistence layer
- `DEFAULT_CONTEXT`: the context where storage objects are persisted
- `LOGLEVEL`: controls logging to the browser console

Configuration related to access control

- `WHITELIST`: a comma-separated list of `ORIGIN`s allowed to access the persistence layer directly

Configuration related to notice and consent/privacy policy notice

- `LEARN_MORE_URL`: a URL where the user can learn more about privacy of the service
- `SERVICE_NAME`: the name of the service
- `SERVICE_URL`: the information URL/landing page of the service

CHAPTER 4

Deploy to CDN

If you are deploying to a CDN origin server (eg netlify or github pages) simply copy the files from the build directory over to the CDN origin server.

CHAPTER 5

Running docker

In order to run your own instance of `thiss-js` you need a search-capable MDQ server (eg `pyFF` or `thiss-mdq`) with some metadata in it. Assuming your MDQ is running on port 8080 in a container called `mdq` the following should work:

```
# docker run -ti -p 9000:80 \  
-e MDQ_URL=http://mdq:8080/entities \  
-e SEARCH_URL=http://mdq:8080/entities \  
-e BASE_URL=http://localhost:9000/ \  
-e STORAGE_DOMAIN="example.com" \  
thiss-js:1.0.0
```

- Replace `example.com` with the domain of your DS instance - eg `localhost` if you are just experimenting.
- Some MDQ implementations have multiple search endpoints - you only need one that is capable of returning JSON-formatted metadata for this to work.
- Running your own instance of `thiss-js` means having your own ORIGIN for browser local storage. If you want to share storage domain with another instance of `thiss-js` then you're better off implementing your own discovery frontend (eg to `thiss.io`). This is documented in github.com/TheIdentitySelector/thiss-ds-js.
- The docker container does not currently support overriding all configuration parameters. Consult the `start.sh` script in the `docker` dir for details.

The `thiss-js` includes the following components:

- A persistence service accessible via the `thiss-ds-js` PersistenceService API in the `/ps/` URI context.
- A SAML discovery service in the `/ds/` URI context
- A login button component available via `/thiss.js`

Each service requires some form of integration to be used by a relying party. A good introduction to the various forms of integration is the [integration guide](#) over at [thiss.io](#).

By order of complexity the alternatives are:

6.1 OASIS Identity Provider Discovery

Using the SAML discovery service requires a SAML SP implementation supporting the [SAML identity provider discovery protocol](#), eg [Shibboleth](#), [SimpleSAMLphp](#) or [pysaml2](#). In this case you simply configure the SP to use `https://your.thiss-js.instance/ds/` as the discovery service URL eg `https://use.thiss.io/ds` or `https://service.seamlessaccess.org/ds`.

6.2 Login Button Component

If your SP supports SAML discovery *or something similar* you can deploy the login button component to your SP to simplify the discovery process for your users if they typically only use a single IdP. In this case the user only has to find their identity provider once for each device/browser. On each following visit to your relying party the user can simply click on the login button component and be taken directly to their preferred identity provider.

An example of how to do this is described in <https://thiss.io/use/>. Essentially you load a javascript from the service called `thiss.js` which contains a function to initialize and render the login button component on a DOM selector of your choice.

6.3 Persistence Service

In order to directly interact with the persistence service and low-level discovery components you need to implement your own components using the low-level APIs in [thiss-ds-js](#).

The persistence service supports ACLs based on whitelisting (currently). Turn on by providing a comma-separated list of domains in the env variable WHITELIST. Only ORIGINS that end with any of the items in the list (remember that port-numbers are part of the ORIGIN if present!) are allowed to call the API when this feature is turned on. This is only meant for small scale deployments.

7.1 Version 1.1.2

- Support for whitelisting domains.

Whitelisting is a mechanism for simple ORIGIN-based ACLs in the persistence API. This approach is meant for small scale deployments and is not expected to scale. To turn on provide the WHITELIST environment variable eg via the env plugin in webpack as illustrated by the standard Makefile

7.2 Version 1.1.3

- compatibility fixes for IE11

7.3 Version 1.2.0

- correctly implement hide-from-discovery
- fix footer

7.4 Version 1.2.1

- support for building in a docker container - no need to have node installed to deploy

7.5 Version 1.3.0

- UX for overriding limits on search results

- Notice and consent information

7.6 Version 1.3.1

- Configurable links and service name in privacy notice
- Updated privacy notice text

CHAPTER 8

Indices and tables

- `genindex`
- `search`