Java Relying Party API v1.0 – Programmer's Guide

4 June 2018

Authors: Peter Höbel – peter.hoebel@open-xchange.com
Vittorio Bertola – vittorio.bertola@open-xchange.com

INDEX

SCOPE OF THIS DOCUMENT

This document describes how to implement an ID4me Relying Party service using the official ID4me Java Relying Party API to authenticate users and retrieve their personal information against ID4me compatible identity servers (an authority and an agent).

Please note that at this point in time the ID4me service is still experimental. No guarantee is given that the present API, or even the underlying specification and architecture, will not change before final release.

Please refer to the more general *"ID4me technical overview"* document for a full description of the ID4me architecture and of the authentication flow that this API implements.

API OVERVIEW

The Relying Party API is a JAR file, developed in pure Java, which provides the functionality needed to fetch the ID4me DNS record for a given ID4me identifier, discover the ID4me identity servers (identity authority and identity agent) from that record, authenticate the user against the identity authority, receive the identity handle and data from the identity authority and fetch the user information from the identity agent. It also provides the functionality to register the relying party automatically to any ID4me identity authority.

The client side of the authentication process consists of two phases:

1. In a first phase, which should be triggered by the code that receives the ID4me login request and username from your website's login form, the library will fetch the DNS record, discover the authority, register your client to the authority if necessary, and submit the authentication request, including the list of claims (user information fields) that you would like to know about the user;

2. In a second phase, which must respond to an HTTPS call by the identity authority at an endpoint that you have to set up on your website, the library will parse the result of the authentication process supplied by the authority, and, if necessary, connect to the identity agent to retrieve the claims that the user has agreed to share with you.

At the end of the second phase, if the process is successful, the library will return the user information to you; at that point in time, you should match that information with your own account database, for example to create a new local account for the user if he is unknown to you, or to record or update the user information in his local account, or to perform any local initializations that are necessary after login.

Please remember that the unique identifier of a user in the ID4me system is the *"identity handle"*, which is derived by the *"iss"* and *"sub"* claim values defined by the authority for the specific identity and returned by the library after the authentication process. This value is guaranteed to be stable for any given identity, though user may still be able in the future to move to a different authority and thus change their handle. You should not use any other key to identify the user and match it with your local accounts, including the ID4me identifier itself (which may change for a number of reasons) or the *"sub"* value alone (which may be used for different identities by different authorities).

INSTALLATION

To install the library, download the JAR file and place it in the appropriate directory in your project, so that it can be included in your build path. The JAR file is designed to work with Java 1.8 and above.

For the ID4me library to work, you will also need to download and add the JAR files for the following other libraries:

Library	**License**	**Version tested**	**Link**
DNSJava	BSD-3-clause	2.1.7	http://www.dnsjava.org/
Servlet API	GPL-2 | CDDL-1.0	4.0.0-b07	https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
JSON	JSON "No Evil"	20170516	https://github.com/stleary/JSON-java
Nimbus JOSE+JWT	Apache-2.0	4.33	https://connect2id.com/products/nimbus-jose-jwt
DNSSECJava	EPL-1.0	1.1.3	https://mvnrepository.com/artifact/org.jitsi/dnssecjava
SLF4J	MIT	1.7.25	https://mvnrepository.com/artifact/org.slf4j/slf4j-api

Please note that the Java Relying Party API is released under the MIT license, while all these libraries are distributed under their own terms as stated. All of them fall under permissive free software licenses, with the possible exception of the "No Evil" JSON license, depending on interpretation. It is up to you to verify their compatibility with the licensing of your project.

Also, in case you prefer, you can access the complete source of the library both as a downloadable archive from the ID4me website, and as a Git repository from ID4me's Gitlab at *https://gitlab.com/ID4me* , where you can find the latest development version. The license allows you to do whatever you want with the code, but in case you make enhancements and modifications that can be useful to others, we encourage you to upstream them by submitting them as pull requests to the Git repository.

CONFIGURATION

To perform the authentication, you have to configure the library and provide the values for a number of local settings through property files. You can find examples of these property files in the source tree, in the *"examples"* directory. These files can be located on your system in any directory you like, as long as you provide the correct path when initializing the library.

The first group of settings must be specified in the *id4me.properties* text file. Here is an example:

```
registration.data.path=/opt/registrationdata/
logo.uri=https://my-domain.org/my-logo.png
redirect.uri=https://my-domain.org/id4me/logon
dnsssec_root_key=. IN DS 19036 8 2 49AAC11D7B6F6446702E54A1607371607A1A41855200FD2CE1CDDE32F24E8FB5
client.name = My-ID4me-Client
dns.resolver=8.8.8.8
```

This is the meaning of the various properties:

Property	Meaning
registration.data.path	Path where the client registration information for each identity authority will be stored; this should be a stable writeable directory which gets preserved over time (see the "Dynamic Client Registration" section below).
logo.uri	URI of a logo of your service that the identity authority can display in the login form, or null if no logo is provided.
redirect.uri	URI of the callback endpoint that you are going to create to receive the redirection of the identity authority after the login.
dnssec_root_key	DS record for the signing key of the root zone of the DNS; you can keep the one provided in the example until 11 October 2018[1]; after that date, the new value will be (all on a single line): `.  IN DS 20326 8 2 E06D44B80B8F1D39A95C0B0D7C65D084 58E880409BBC683457104237C7F8EC8D`
client.name	A string of your choice identifying your service; we recommend that it also contains your URL or domain name to make it as univocal as possible.
dns.resolver	IP address for your local DNSSEC-enabled DNS resolver server; if you don't have one, you can use one of the public ones (e.g. 8.8.8.8 or 9.9.9.9).

The second configuration file, *claims.parameters.json*, allows you to define which pieces of information you want to acquire about each user that logs into your platform using ID4me. It contains a JSON array of objects like the following one:

```
[
        {
                "name": "email",
                "essential": true,
                "reason": "Needed to create the profile"
        },
        {
                "name": "name",
                "reason": "Displayname in the user data"
        },
        {
                "name": "given_name"
        }
]
```

Each element of the array adds one claim to the request for information, specified in the *"name"* key; please refer to the ID4me technical overview document for a list of the claims defined in ID4me at this point in time (they include at least the basic ones specified in the OpenID Connect Core standard). Two more optional keys can be added; the *"essential"* key (defaulting to *false* if missing) defines whether the claim is mandatory for your service, and in that case the login will be denied if the user does not agree to share that piece of

[1] Please note that this date depends on ICANN and can still be postponed. Keep an eye over announcements for the "Root zone KSK rollover".

information with you; the *“reason”* key allows you to specify the purpose for which you are asking to acquire that piece of information, and its value will be shown to the user “as is” in the consent request form.

Please note that it is up to you – not to ID4me or to the identity authority – to make sure that all your requests for data, especially the mandatory ones, comply with applicable privacy and data protection regulations.

DYNAMIC CLIENT REGISTRATION

Whenever the relying party connects for the first time to a specific identity authority, before being able to perform the authentication flow, it has to register itself; the API implements this through a simple call, which will be triggered automatically when necessary.

As a result of the client registration, a text file, containing the registration data as a JSON object, is saved in the file system. The file name consists of the identity authority hostname plus *“.json”* (e.g. *auth.freedom-id.de.json*). You can set the path for storing these files in the *id4me.properties* file; while losing these files will just trigger a new registration whenever necessary, unnecessary client registrations slow down performances and make it harder for authorities to track the usage of their systems – so please ensure that these files are written in a reasonably safe location.

Example registration data:

```
{
  "grant_types": ["authorization_code"],
  "subject_type": "public",
  "mutual_tls_sender_constrained_access_tokens": false,
  "application_type": "web",
  "registration_client_uri": "https://auth.freedom-id.de/clients/abc53omf3ute",
  "redirect_uris": ["https://domainid.example.com/domainid/logon"],
  "registration_access_token": "abcE8LxhtfnHwDMk6qNZpT9JO965YVvQoDolU.YsxpLHI",
  "token_endpoint_auth_method": "client_secret_basic",
  "client_id": "abc53omf3ute",
  "client_secret_expires_at": 0,
  "client_id_issued_at": 1526024600,
  "client_secret": "ABC2342rsV8cVOA_Rl9MZcoIicZHfRVRH8veQYeurc",
  "client_name": "My-Relying-Party",
  "response_types": ["code"],
  "id_token_signed_response_alg": "RS256"
}
```

THE AUTHENTICATION PROCESS

The process to log in a user and fetch his information with the ID4me API looks as follows.

Phase 1, called by your own login form:

1. Get the ID4me identifier for a user (e.g. username.id4me.org), for example by reading it from the submission of a web form.
2. Create an instance of *Id4meLogon*.
3. Create an instance of *Id4meSessionData*. At this stage, if necessary, the library will perform the dynamic client registration and save the resulting information in the filesystem.
4. Call *Id4meLogon.authorize(Id4meSessionData)* to get an authorization URI.
5. Redirect the browser to this authorization URI.

The authority, after completing the authentication, will redirect the user's browser to the callback endpoint URL that you have supplied in the configuration.

Phase 2, called by the callback endpoint:

6. Extract the *"code"* parameter from the query arguments of the HTTPS request.
7. Call *Id4meLogon.authenticate(Id4meSessionData, code)* to fetch and validate an access token from the ID4me identity authority.
8. Optionally, if user information is to be retrieved, call *Id4meLogon.userinfo(Id4meSessionData)* to fetch the information from the ID4me identity agent.
9. Call *getIdentityHandle()* and *getUserinfo()* to acquire the user's information for further processing.

EXAMPLE USE OF THE API

The API contains two public classes which supply the methods needed to perform a client registration at the ID4me identity authority, authenticate a user on the ID4me identity authority and receive the information from the ID4me identity agent.

To initiate a ID4me logon process, an instance of the classes *org.id4me.Id4meLogon* and *org.id4me.Id4meSessionData* is needed. The *org.id4me.Id4meSessionData* instance holds the user's current ID4me session data which is used by the methods of *org.id4me.Id4meLogon*.

Before you can create these objects, you need to define some parameters which are specific for the relying party and are needed by the *Id4meLogon* class; see the "Configuration" section. Next, you can create an instance of Id4meLogon, providing as arguments the path and filename of the two configuration files:

```
Id4meLogon logon_handler = new Id4meLogon("/path/to/id4me.properties",
"/path/to/claims.parameters.json");
```

Then you can create an instance of *Id4meSessionData*; to do this, you also need to receive from the calling code the ID4me username that the user entered in the login form, which we will call *userid* in the example. The second parameter in the call enables automatic dynamic client registration, so it should always be set to *true* unless for specific reasons.

```
Id4meSessionData session_data = logon_handler.createSessionData(userid, true);
```

Now you can get the authorization URI from the *logon_handler* and redirect the browser to this URI; *response* is the *HttpServletResponse* of the original web request deriving from the login form.

```
String authorizationUri = logon_handler.authorize(session_data);
response.sendRedirect(authorizationUri);
```

This concludes phase 1 of the process.

As for phase 2, it will have to be triggered by the code that responds to HTTPS requests directed to your callback endpoint URI. Assuming that you have access to the *Id4meLogon* and *Id4meSessionData* objects created for this web session, your endpoint now can authenticate the user. Thus you have to get the value of the parameter *code* from the *HttpServletRequest* request, and pass it on to the authenticate method of the *logon_handler*:

```
String code = request.getParameter("code");
boolean auth_ok = logon_handler.authenticate(session_data, code);
```

The value of *auth_ok* will confirm whether the login was successful or not, though the code will also throw exceptions in case of error, so in practice you will never get a *false* return value from *authenticate*.

If you also need pieces of user information, you now can get them from the *logon_handler*:

```
boolean access_ok = logon_handler.userinfo(session_data);
```

The session will now contain all the information you know about the user identity, received either from the authority or from the agent. You can access this information from *session_data* through the *getUserInfo* method; you also have a specific method to get the identity handle, which is the only globally unique value that you can use to identify the identity. While you can calculate the user's identity handle from the data in the *userinfo* object, for future compatibility it is recommended to use the *getIdentityHandle()* function.

```
String identity_handle = session_data.getIdentityHandle();
JSONObject userinfo = session_data.getUserinfo();
```

The *userinfo* object returned by *getUserinfo()* is a JSON object containing all the technical claims supplied by the identity authority and necessary to verify a user, as defined in the OpenID Connect standard, plus all the personal claims which the user has approved at the identity authority's consent form. Example userinfo object:

```
{
"aud": "dnp66omfc7ute",
"sub": "abc0rOabc3fYIYdykhnv/ff0+ABCiwHYEZ99E2NL23urExB+1Sr+eE6NnO2P32",
"id4me.identity": "user.mydomain.org",
"nbf": 1527148074,
"updated_at": 1527143874,
"iss": "https://identityagent.de",
"exp": 1527148374,
"iat": 1527148074,
"email": "user@mydomain.org"
}
```

At this point, the process is complete and you can proceed with your own initializations and with redirecting the user to a proper welcome page or to a terms & conditions acceptance page if necessary.

FURTHER REFERENCE

A detailed Javadoc documentation for the library is available from the ID4me website.

LICENSE

The license for the present library is contained in the LICENSE file that you will find in the root of the source distribution. For your convenience, the text of the license (which is a standard MIT "Expat" license) is added hereafter.