Technical documentation for implementation of mojeID

Release 3.0.2

CZ.NIC, z. s. p. o.

07.01.2021

Contents

Chapter 1

Legal Notice

Overview

1.1 Limitation of Liability

Except in cases of damage caused intentionally or due to serious negligence, or damage to person's basic rights, or to maximal extent allowed by the user's legislation system, the CZ.NIC Association holds no responsibility for any direct or indirect damages resulting from usage (including installation) of mojeID, including but not limited to damage to reputation, damage resulting from interrupted work, loss or damage of data, or any other economical damage (e.g. loss of profits, not reaching expected savings, etc.).

Please keep in mind that the information provided in this documentation does not serve as a warranty, explicit nor implicit, especially as a warranty of suitability for a specific purpose or warranty of usability in other legal system than the legal system of the Czech Republic.

1.2 Privacy Protection

mojeID was developed in the Czech Republic and its privacy protection policy is in accordance with the national legislation of the Czech Republic, including opinions of Personal Data Protection Authority. Before using mojeID outside of the Czech Republic make sure that the mojeID data protection policiy is in accordance with the legal requirements of the given country.

1.3 Applicable Legislation and Competent Jurisdiction

The mojeID implementation documentation (and associated documents) are governed and interpreted in all regards in accordance with Czech legislation. All disputes or claims resulting from or associated to using mojeID (or this documentation), including its interpretation, implementation, invalidity, etc. will be definitively settled by Court of Arbitration at the Economic Chamber of the Czech Republic and Agricultural Chamber of the Czech Republic (hereinafter as "court") pursuant to its Rules of Procedure by one arbitrator elected by this court's chairman.

1.4 Conditions for Using mojeid Logo

The CZ.NIC Association is the executor of property copyright rights to figurative mark - mojeID logo and its derived modalities. The CZ.NIC Association hereby allows the usage of the mojeID logo and its associated modalities with regard to implementation, usage and/or promotion of mojeID or promotion of the CZ.NIC Association and its products in any common way logos are used. The right to use mojeID logo and its associated modalities is free of charge, non-exclusive, unlimited in quantity and geography, and limited in time in regard to the usage of mojeID. The user is not required to exercise the right to use the mojeID logo and its associated modalities. The right to use the mojeID logo and its associated modalities cannot be forwarded to a third person without consent from the CZ.NIC Association. The mojeID logo and its associated modalities cannot be abused to damage good reputation of the CZ.NIC Association or used contrary to the interests of the CZ.NIC Association. The mojeID logo and its associated modalities cannot be belittled or used in derogatory manners. The mojeID logo has to be figured as per instructions of the Graphical Manual and used exclusively in such fashion.

Chapter 2

Introduction

This document includes a general introduction to the mojeID service. You can also find here examples and other general information that will help you design the implementation of mojeID support in your web application. It will help you get a basic overview of the steps that will have to be taken to implement mojeID support and you will be able to estimate the complexity of the implementation.

mojeID currently offers three authentication protocols that can be used. They are OpenID Connect, OpenID 2.0, and SAML 2.0.

Tip: If you do not use any of these protocols in your system, we recommend choosing *OpenID Connect*.

It is the newest of the offered protocols and it has some improvements based on the experience from using the other two. Its main advantages are simpler implementation and mobile platforms support.

However, if you already use the OpenID 2.0 or SAML 2.0 protocol in your system, it is logical to use that protocol for integration with mojeID too.

Chapter 3

Terminology

The following terminology is used in the next chapters regarding the implementation of mojeID:

Service provider provider of a web application (or simply an application, because it manages everything automatically without any manual setting) that requires verification of user's *identity* via mojeID

Full access mojeID implementation variant at the service provider, more details at https://www.mojeid.cz/en/provider/options-and-prices/

Limited access mojeID implementation variant at the service provider, more details at https://www.mojeid.cz/en/provider/options-and-prices/

Identity set of data about the user that are linked to an *identifier* and managed by an OpenID provider

Identifier a URL with an `http` or `https`schema that defines and provides certain data in the :term:`identity <Identity>`, e.g. `http://specs.nic.cz/attr/contact/valid`.

Realm the service provider's URL area defining a part of a URL region for which the *identity authentication request* (page 36) is valid

OP

OpenID provider OpenID2 identities provider and maintainer on whose web the authentication is carried out. In case of mojeID, it is the CZ.NIC Association.

OCP

OpenID Connect provider OpenID Connect identities provider and maintainer on whose web the authentication is carried out. In case of mojeID, it is the CZ.NIC Association.

Identity name the name of mojeID *identity* in form of `jmenoidentity.mojeid.cz`, that the user enters in the login form as the identity they want to log in with, e.g. `demo.mojeid.cz`.

Claimed identifier identifier derived from identity name under which the identity is available at OpenID provider and from where it is possible to retrieve matadata of this identifier, e.g. `https://demo.mojeid.cz/#UnIqUe`.

OP endpoint URL where the OpenID2 provider receives messages. In case of mojeID, it is `https://mojeid.cz/endpoint/`.

Registration Endpoint URL where it is possible to register a new service provider according to OpenID Connect Dynamic Client Registration[1] specification.

Client ID unique identifier of a service that uses OpenID Connect. It is assigned on registration and used during all the communication via OpenID Connect.

Client Secret password that certifies the service provider's authenticity in regard to his Client ID. This password can be changed using Registration Access Token.

Registration Access Token token used for authorization of any change of data about the service, e.g. Client Secret

[1] https://openid.net/specs/openid-connect-registration-1_0.html

Authorization Endpoint a URL to which service providers redirect users for login

ID Token contains a confirmation of a successful identity authentication of a user whose data is contained within the ID Token

Access Token a token used to authenticate a UserInfo Endpoint request

UserInfo Endpoint a URL where it is possibe to get detailed data of a user if they are not contained in the ID Token

Token Endpoint a URL where it is possible to get the Access Token, or the Refresh Token, in case they have not been received directly in the response to authentication.

Refresh Token a token that can be used to receive data from the UserInfo Endpoint even without the user's presence.

Chapter 4

Getting started with mojeID

This chapter is an introduction to basic principles of the mojeID service, the forms of mojeID identities and the communication process via supported protocols.

4.1 Basics of mojeID

MojeID is a service that allows its users to create and centrally manage their internet identity (a set of personal data, e.g. first name, surname, e-mail address, phone number, etc. together with login methods). Users can then use this identity to log into various external web applications (applications of different service providers than the identity provider) and they do not have to create individual accounts and repeatedly fill in their basic information and use different usernames and passwords.

The mojeiD service is a specific implementation of the OpenID 2.0 and OpenID Connect 1.0 standard for decentralized management of internet identities which define the ways to verify these centrally managed identities and the forms of their identifiers.

MojeID is specific for the Czech internet environment and offers the service providers additional advantages over the standard OpenID, e.g. extended set of personal data in the identities and their transfering, or more login methods with the possibility to require certain level of authentication.

4.2 MojeID Identity

When creating an identity, users have to choose a name of their identity which uniquely determines each mojeID identity and which is always in the form of `identityname.mojeid.cz` (alphanumeric characters), e.g. `demo.mojeid.cz`.

The users then use this name to log into pages of service providers.

MojeID Identity consists of:

- Information the user includes in their identity (common personal data, such as name, address, phone number, nickname, etc.)
- Information about the user provided by the mojeID service provider, especially information about the physical identity verification (user's personal data verification, or the information about whether the person is older than 18).

Tip: Specific lists of information that can be transferred from the mojeID identity using the individual protocols can be found in *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62), *Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58) and *Appendix 3 – List of Data to be Handed Over (SAML)* (page 68).

4.3 Communication with mojeID

This section generally describes communication processes that take place when a moje ID user logs in to a service that supports a certain protocol.

4.3.1 Communication via OpenID Connect

The process of logging in using mojeID has various variants (based on different schemas) that consist of several steps. As you implement mojeID, you can choose the schema(s) you prefer.

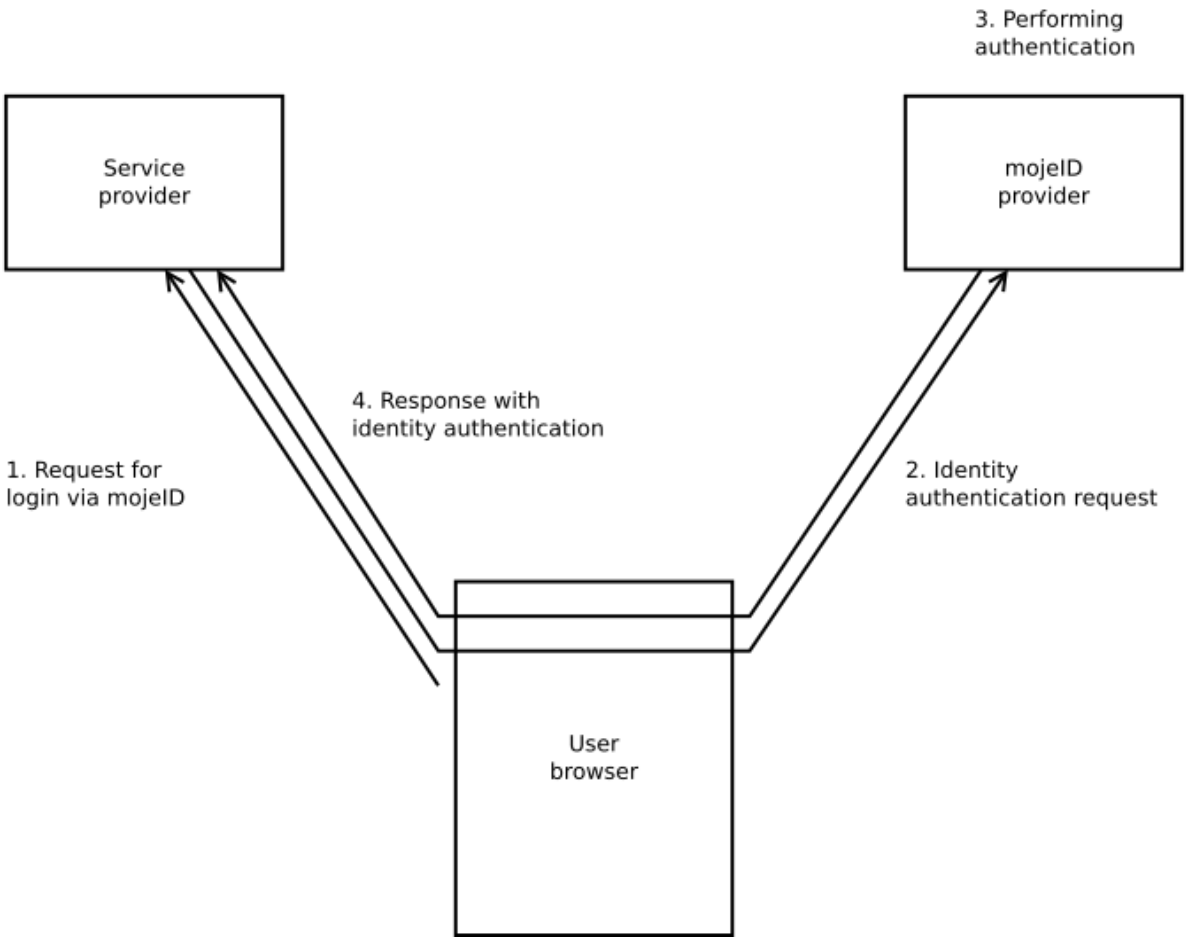
The first steps are the same for all the schemas:

0. **Client's registration** – You have to register your client on mojeID servers before you can use the OpenID Connect protocol.
1. **Requesting login using mojeID** – The user clicks the Log in via mojeID button.
2. **Requesting identity authentication** – The service provider creates an identity authentication request and sends it (indirectly by redirecting the user's browser) to the OpenID Connect provider's endpoint (`Authorization Endpoint`) where the user authenticates.
3. **Performing authentication** – The user logs in at the mojeID login page using one of the login methods to verify their identity. At this moment, we support login with password, digital certificate, one-time password, or security key (FIDO 2).

The next steps depend on the chosen schema.

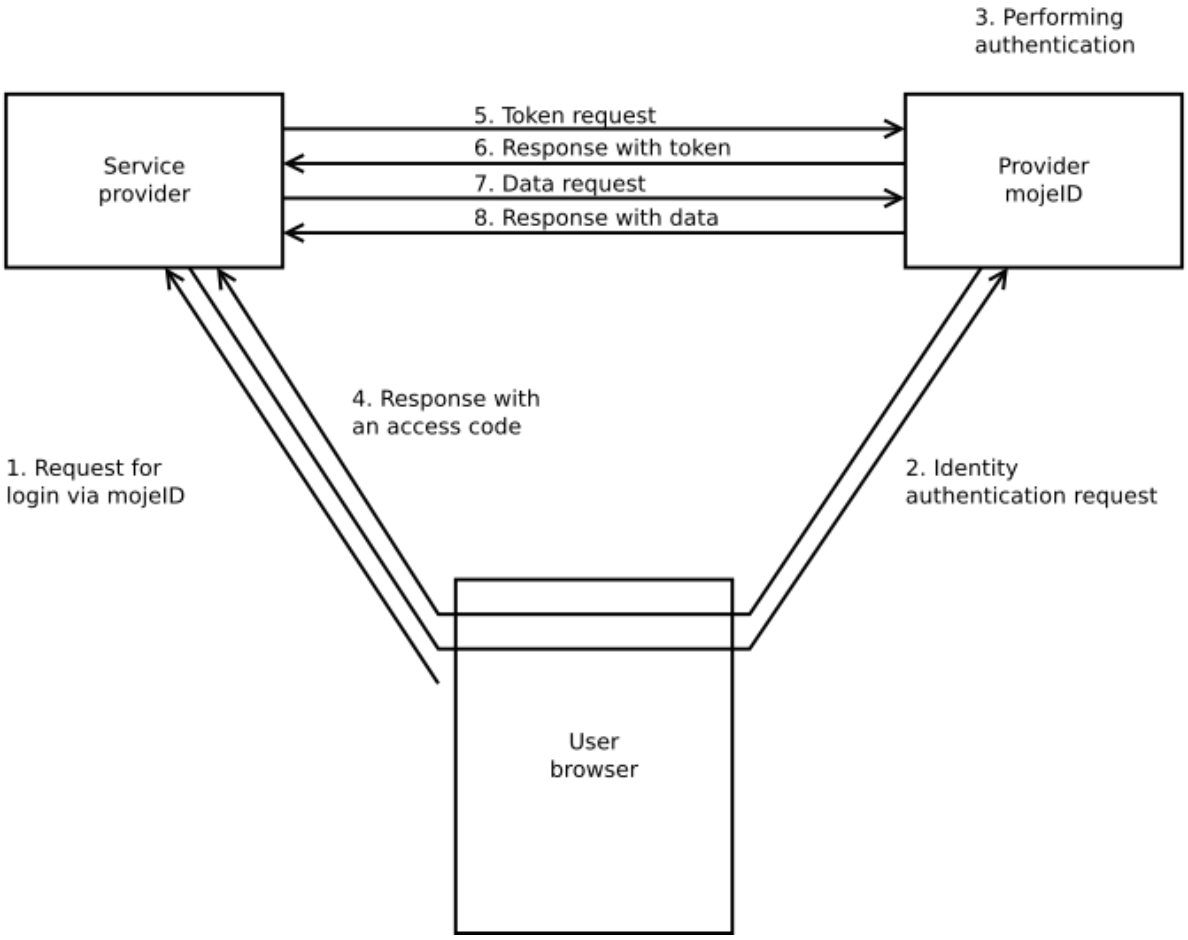
- *Implicit schema* (page 9)
- *Access code* (page 10)
- *Hybrid schema* (page 11)
- *Schema selection* (page 12)

Implicit schema



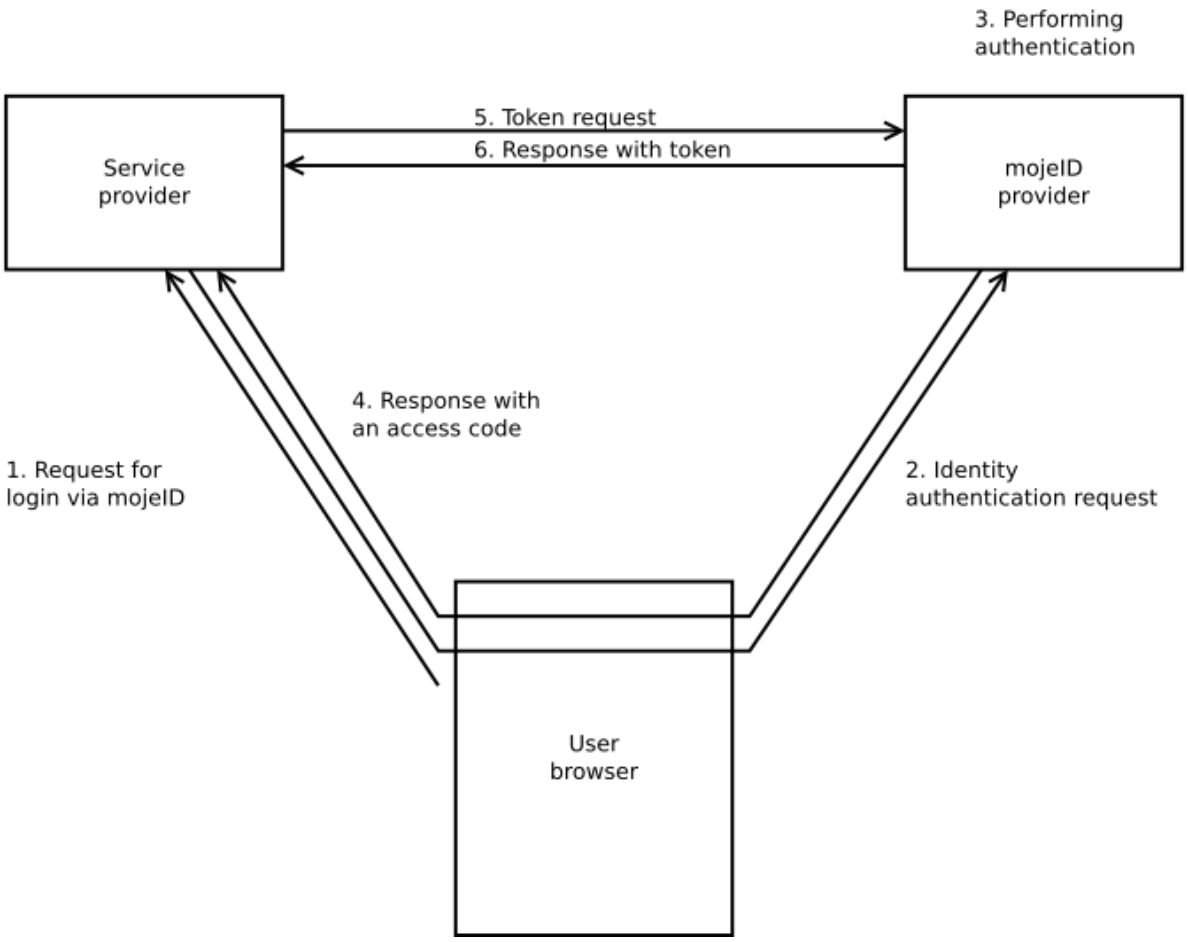
4. **Response with the identity authentication outcome** – After the login and confirmation, the user is redirected back to the service provider's website and via their browser sends the response from mojeID servers with the user identificator and ID token. If the service provider requests it during the identity authentication process, the ID token will include data about the user.

Access code



4. **Response with an access code** – After the login and confirmation, the user is redirected back to the service provider's website and via their browser sends the response from mojeID servers with an access code.
5. **Token request** – The service provider creates a token request using the access code they just received and sends it to the `Token Endpoint`.
6. **Response with a token** – The service provider receives a response with access token and token ID.
7. **Data request** – The service provider creates a user data request using the access token they received and sends it to `UserInfo Endpoint`.
8. **Response with data** – The service provider receives a response with user's data.

Hybrid schema



4. **Response with an access code** – After the login and confirmation, the user is redirected back to the service provider's website and via their browser sends the response from mojeID servers with an access code.
5. **Token request** – The service provider creates a token request using the access code they just received and sends it to the `Token Endpoint`.
6. **Response with a token** – The service provider receives a response with an access token and a token ID that contains the user's data.

Schema selection

For web services that run only in browser (“without server”, e.g. JavaScript), it is best to use *Implicit Schema*.

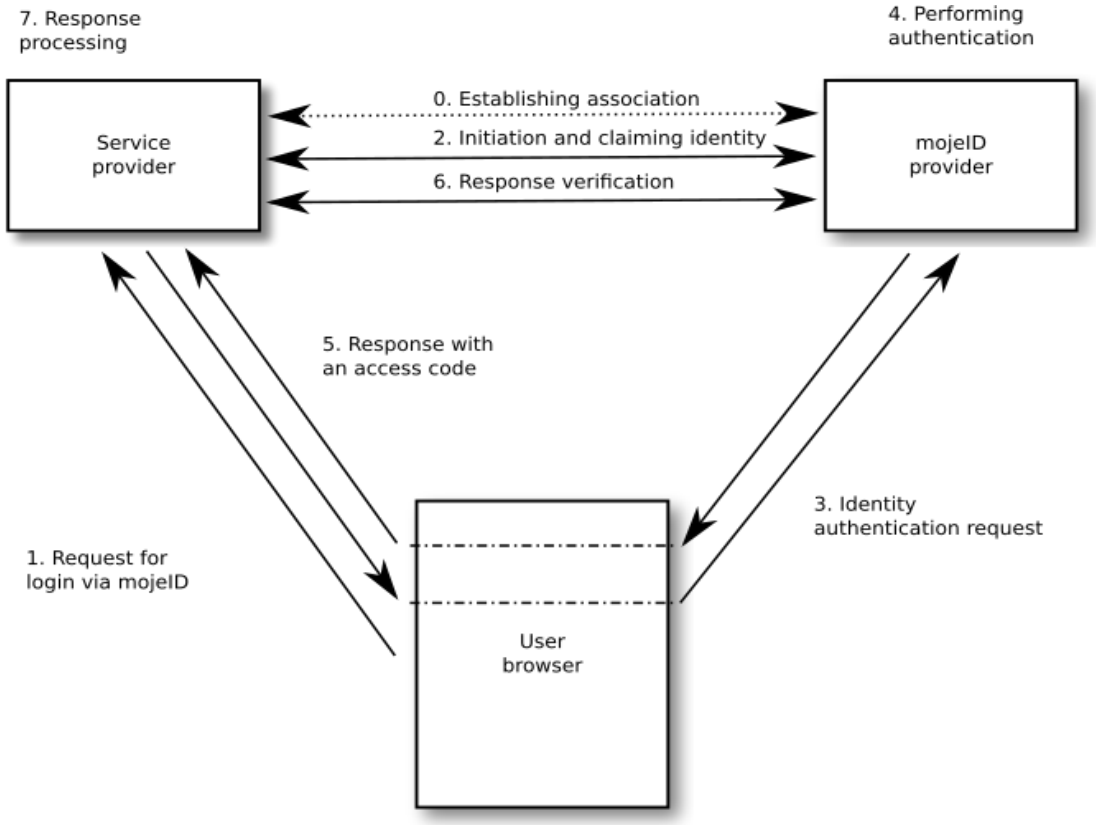
For server services, it is better to use the *Access Code* schema which is more secure.

The following table provides an overview of the basic characteristics of the individual schemas and it helps with the selection of an appropriate login schema.

Characteristic	Implicit Schema	Access Code	Hybrid Schema
All the tokens are returned from the Authorization Endpoint	yes	no	no
All the tokens are returned from the Token Endpoint	no	yes	no
Tokens are not visible in User Agent	no	yes	no
The client can use authentication	no	yes	yes
It is possible to get a Refresh token	no	yes	yes
Communication within a single request	yes	no	no
Most of the communication is server-to-server	no	yes	various

4.3.2 Communication via OpenID 2.0

The process of logging in via mojeID consists of several steps, as shown in the following schema:



0. **Establishing association** – Agreeing on a shared secret to be used to verify messages from the OpenID provider.
1. **Requesting login using mojeID** – The user clicks the **Log in via mojeID** button.
2. **Initiation** – Initiation serves for getting metadata about OpenID providers.
3. **Requesting identity authentication** – The service provider creates an authentication request and sends it (indirectly by redirecting the user's browser) to the OpenID provider's endpoint where the user authenticates.
4. **Performing authentication** – The user logs in at the mojeID login page using one of the login methods to verify their identity. At this moment, we support login with password, digital certificate, one-time password, or a security key (FIDO 2).
5. **Response with the identity authentication outcome** – If the service provider requests it during the identity authentication process, the user is redirected back to the service provider's website and receives a response with the outcome of the identity authentication.
6. **Response verification** – Each message the service provider receives from the OpenID provider indirectly via the user's browser has to be verified to confirm it really comes from the OpenID provider and it has not been changed. This is done either via association in most cases (see item 0), or by explicitly requesting the verification.
7. **Response processing** – Based on whether the login is successful or not, the service provider's application has to react, and if necessary, process more data received from

this response.

4.4 Favicon

A favicon is a graphical element (icon) associated with a certain website or, in case of mojeID, a service. Web browsers can display favicons as a visual symbol of a website's identity in address bar, bookmarks or Favourites.

mojeID displays a favicon in the mojeID login form next to the name of the service the mojeID user is logging into.

Fig. 1: Favicon display example

The use of the favicon differs based on the protocol.

4.4.1 Settings in OpenID Connect

You need to upload the favicon file to your website and set its address as metadata (`logo_uri`) within your client's registration (see *Client Registration* (page 22)).

If the icon is found at the defined URI, it is displayed in the mojeID form, **no matter** the type of access (*full/partial*) služby k mojeID.

4.4.2 Settings for OpenID 2.0 and SAML

You need to explicitly upload the favicon file to our system.

The favicon is downloaded either automatically (once a week), or you can provide it directly to CZ.NIC (e.g. by e-mail to our support) and we will upload it manually. With automatic downloading, the algorithm searches for the favicon on the provider's *realm* based on the W3C favicon standard[2], section *Method 1*.

Favicons cannot be larger than 10 kB. The supported formats are ICO and PNG.

Displaying a favicon for services communicating via this protocol is possible only when the service has *full access*.

[2] http://www.w3.org/2005/10/howto-favicon

Chapter 5

mojeID Support Implementation

This chapter takes you through the details of the individual phases of the communication process that need to be taken into account during the implementation of the support of the protocol. It also describes the prerequisites that need to be met to successfully implement the support.

Important: Due to security reasons, mojeID does not permit displaying of the login page within frames (`<iframe>`).

5.1 Implementation via OpenID Connect (OIDC)

This section will introduce you to the technical aspects of the implementation of mojeID into web applications via the OpenID Connect protocol.

We recommend to study this text in order to properly understand the principles and processes of mojeID / OpenID Connect. Most of the things described here can be solved by using *available libraries* (page 18) for the implementation of OpenID Connect that we recommend to use.

The *Implementation Process Overview* (page 18) section will take you through the implementation process step by step. Other sections describe the individual steps in more details.

The official specification of the OpenID Connect protocol can be found at https://openid.net/specs/openid-connect-core-1_0.html.

MojeID server publishes basic information about OIDC configuration at https://mojeid.cz/.well-known/openid-configuration/.

You can test your implementation using the *mojeID Test Instance* (page 53).

The list of data that can be transferred by the protocol (including their identifiers) is available in the *Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58).

Examples and solutions of error messages can be found in the *Appendix 6 – Examples and Solution of Error Messages* (page 77).

Note: All the examples of source code listed below illustrate implementation in Python using the `pyoidc` library.

5.1.1 Overview of Libraries and Modules

The official OpenID Foundation website offers a list of certified OIDC protocol implementations in several programming languages (see Certified OpenID Connect Implementations[3]. The relevant part for you is implementations for *Relying Party* that corresponds to the service you provide.

For the use in mobile apps, it is best to use libraries for native apps:

- For Android e.g. http://openid.github.io/AppAuth-Android/,
- For iOS e.g. http://openid.github.io/AppAuth-iOS/.

You can also use modules for the most popular platforms:

- WordPress: OpenID Connect Generic Client (daggerheart)[4]
- Drupal: OpenID Connect module[5]
- Magento: OpenID Connect Single Sign-On (SSO) Magento Extension By Gluu[6]
- OpenCart: OpenCart OpenID Connect Single Sign-On (SSO) Extension By Gluu[7]
- Moodle: OpenID Connect Authentication Plugin[8]
- Django: OIDC Django Packages[9]

If you know another one that should be mentioned here, we will be glad to hear from you (techsupport@mojeid.cz).

Caution: We inform you that you can use modules only at your own risk and the CZ.NIC Association is not subject to any liability for any damage.

5.1.2 Implementation Process Overview

This overview includes organizational and technical steps you have to take to implement logging in to your service via mojeID using the OpenID Connect protocol. The individual steps are brief and say what to do, while the link targets provide more details on how to do that, or they contain additional information. The overview can serve as a *checklist*.

Preparing the test environment

1. *Register your service* (page 22) (client) at the *test Registration Endpoint* (page 54) – this way you will get test metadata of your service (*Client ID*, *Client Secret*) and an opportunity to set up certain parameters of the communication.

[3] https://openid.net/developers/certified/
[4] https://wordpress.org/plugins/daggerhart-openid-connect-generic/
[5] https://www.drupal.org/project/openid_connect
[6] https://github.com/GluuFederation/magento-oxd-extension
[7] https://github.com/GluuFederation/opencart-oxd-module
[8] https://moodle.org/plugins/auth_oidc
[9] https://djangopackages.org/grids/g/oidc/

Note: In case of the *Automatic Registration*, the *Client Secret's* validity ends after a certain time period. If you decide to opt for *Automatic Registration*, it is important to set up registration renewal.

2. Send the service's test metadata (*Client ID*) to support (techsupport@mojeid.cz). The support sets up accesses.
3. *Create and set up mojeID test accounts* (page 53).

Implementation and debugging

You will need: text editor, browser, access to hosting, OIDC specifications[10]

You might find *our recommendations for debug tools* (page 46). useful for implementation debugging. During the debugging, you might come accross various error messages. *Appendix 6 – Examples and Solution of Error Messages* (page 77) might help you with them.

1. *Add mojeID button and links* (page 25) to the (template/sites of the) service the user will use to request login. Follow *correct implementation procedure* (page 83)!
2. *Get test OIDC provider configuration* (page 26) (webfinger).
3. Library configuration – enter test *Client ID* and *Client Secret*, or also test endpoints, if the library cannot retreive this information automatically from the OIDC provider's configuration.
4. *Create and send an authentication request* (page 26) to the *Authorization Endpoint* (page 54).

Note: The request should also include the information about the chosen *authentication schema* *</SeznameniSMojeid/ProcesKomunikacePresMojeid/OpenIDConnect/index>*. The following steps correspond to the *Access Code* schema.

5. Process the *authentication response* (page 28) at the return address stated in the request which receives an *access code* (`code`).
6. *Create and send a token request* (page 29) to the *Token Endpoint* (page 54). You will use the received *access code* in the request.
7. Process the response from which you get an *Access Token* (`access_token`) and an *ID Token* (`id_token`, What does ID Token contain?[11]), whose validity has to be verified by the implementation (see ID Token Validation[12]).
8. If the *ID Token* is valid, *create and send a user data request* (page 30) to *UserInfo Endpoint* (page 54). Use the received *access code* in the request.
9. Process the response with the user's data according to the needs of your service.

[10] https://openid.net/specs/openid-connect-core-1_0.html
[11] https://openid.net/specs/openid-connect-core-1_0.html#CodeIDToken
[12] https://openid.net/specs/openid-connect-core-1_0.html#ImplicitIDTValidation

Implementation verification

If you want to operate the service with a full access, we have to perform user test of your implementation before your service transitions to production environment.

1. When you finish debugging your implementation, send a notification to the support team (techsupport@mojeid.cz) that your implementation is ready for user test and attach the address of your service's test instance.
2. When we finish debugging the last details together, your implementation will be ready for the transition to the production environment.

Transition to the production environment

1. To get the full access, you first need to sign a contract.
2. *Register your service* (page 22) (client) at the *production Registration Endpointu* (page 54) – this way you will get production metadata of your service and set up certain parameters of the communication.
3. Send the service's production metadata (Client ID) to the support team (techsupport@mojeid.cz), also in case of a partial access. The support team will add the service into the catalog.
4. *Get a production OIDC provider configuration* (page 26) (webfinger).
5. Reconfigure the implementation with production metadata, or also endpoints.

That is all.

5.1.3 Client Registration

To communicate with mojeID via OpenID Connect, it is necessary to register a client (service) at the mojeID server. It is possible to use either manual, or automatic registration. *Automatic registration* (page 22) is suitable for dynamacially created clients (JS, mobile devices) and *manual registration* (page 22) is suitable for server clients.

Manual registration

The manual registration can be done at https://mojeid.cz/consumer_admin/. In case of a mojeID test instance, at https://mojeid.regtest.nic.cz/consumer_admin/. You can then edit and delete the managed clients at the same address. The clients created this way have the validity period set to indefinite. Specifications of individual items can be found in the OpenID Connect protocol document (https://openid.net/specs/openid-connect-registration-1_0.html#ClientMetadata).

An example of manual registration of a client in mojeID test instance:

1. In any account that you create in the *mojeID test instance* (page 53), go to https://mojeid.regtest.nic.cz/consumer_admin/ after login.
2. Go to the `New service setup` link. Fill in the required fileds `Client's name`, `List of URIs` and click `Save`.
   - A record with the client's ID is created in the list of managed services.
3. To get `Client secret / Tajemství klienta` go to the `Update` link in the newly created service.
   - A page where you can edit the setup is displayed – `Client secret` is in the last row of the displayed form.
4. To test your setup within the test instance, you can use a test environment at https://mojeid.regtest.nic.cz:8000/consumer/oic/register/ where you register the created client. Then go to https://mojeid.regtest.nic.cz:8000/consumer/oic/start/ where you choose the created client. There you can test different data hand-over scenarios.

Automatic Registration

More details can be found in the OpenID Connect protocol document (https://openid.net/specs/openid-connect-registration-1_0.html). All the necessary settings should be done by the used library. Registration created this way will expire after 24 hours but it can be renewed (see *Registration Change* (page 25)).

An example of registering a client using the library:

```
from oic.oic.consumer import Consumer

client = Consumer(SessionDB(URL), OIC_CONFIG, client_config=OIC_CLIENT_CONFIG)
client.redirect_uris = URL + client.consumer_config['authz_page']
provider_info = client.provider_config(ISSUER)
client.register(provider_info["registration_endpoint"], response_types='code', client_
↪name=MY_CLIENT_NAME)
```

An example of a registration query:

```
POST /oidc/registration HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: mojeid.cz

{
 "application_type": "web",
 "redirect_uris":
     ["https://client.example.org/callback",
      "https://client.example.org/callback2"],
 "client_name": "My Example",
 "logo_uri": "https://client.example.org/logo.png",
 "token_endpoint_auth_method": "client_secret_post"
}
```

An example of the server's response to a registration query:

```
HTTP/1.1 201 Created
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "client_id": "s6BhdRkqt3",
 "client_secret": "ZJYCqe3GGRvdrudKyZS0XhGv_Z45DuKhCUk0gBR1vZk",
 "client_secret_expires_at": 1577858400,
 "registration_access_token": "MY.SECRET.REGISTRATION.ACCESS.TOKEN",
 "registration_client_uri": "https://mojeid.cz/oidc/registration?client_id=s6BhdRkqt3
↪",
 "token_endpoint_auth_method": "client_secret_post",
 "application_type": "web",
 "redirect_uris":
     ["https://client.example.org/callback",
      "https://client.example.org/callback2"],
 "client_name": "My Example",
 "logo_uri": "https://client.example.org/logo.png"
}
```

Note:

Registration can be processed and Client ID and Client Secret can be retrieved also without the library; you only need to send a POST query via curl.

Example:

```
curl --data '{"redirect_uris": "https://navratova-adresa.cz",
  "client_name": "Název služby"}' https://mojeid.cz/oidc/registration/
```

Registration also allows to associate metadata with client registration (see Client Metadata in specification[13]), so the provider can define for example: service name and icon, specifically the attributes `client_name`, `logo_uri`, or `client_uri`.

Information about Registration

A part of the mojeID server's response to a completed registration is a URL where it is possible to get current information about registration (configuration endpoint `registration_client_uri`), and an access code (`registration_access_token`). When sending a GET query to this URL, it is necessary to authenticate using an access code. It needs to be included in the header of the `Authorization` HTTP request.

The server's response has the same format as the response to registration and contains current information about your client on our server.

[13] https://openid.net/specs/openid-connect-registration-1_0.html#ClientMetadata

Registration Change

You can edit certain information about the registered client using the abovementioned configuration endpoint. Configuration has to be done using a POST query with `registration_access_token` added into the `Authorization` header. The request format is the same as with the one for registration and its processing on server is also the same, with the following exceptions:

- It is not possible to change the registered `redirect_uri` and `client_id`.
- The `client_secret` value is ignored. In case the item is included in the request, a new `client_secret` is generated. It is sent in the response to the configuration query.

An example of a configuration query that will ensure generation of a new `client_secret` and a change of `logo_uri` and `policy_uri`.

```
POST /oidc/registration?client_id=MYCLIENTID HTTP/1.1
Accept: application/json
Host: mojeid.cz
Authorization: Bearer MY.SECRET.REGISTRATION.ACCESS.TOKEN

{
 "client_secret": null,
 "logo_uri": "https://client.example.org/another-logo.png",
 "policy_uri": "https://client.example.org/policy-page"
}
```

The server's response to the configuration query is the same as the response to the registration query and contains current information about your client on our server.

5.1.4 Requesting Login via mojeID

The process of identity authentication starts by the user submitting a login request via mojeID at your website. To ensure maximal user friendliness, you can just use a login button (see the following pictures). The username is entered later at the mojeID server.

Fig. 1: *Examples of buttons for login via mojeID*

Logging in to mojeID using a button is the only recommended and correct method.

5.1.5 Initiation

To be able to send an identity authentication request, your library needs to know either the user's identifier, or the OCP endpoint.

Your application will use the identifier and endpoint to send a WebFinger query to retreive details about the OpenID Connect provider. The response to this query includes (among other things):

- **Autorization Endpoint** – this is always `https://mojeid.cz/oidc/authorization/` and this address is used for identity authentication requests.
- **Token Endpoint** – this is always `https://mojeid.cz/oidc/token/` and this address is used for token requests.
- **UserInfo Endpoint** – this is always `https://mojeid.cz/oidc/userinfo/` and this address is used for user data requests.

An example of query for a specific user:

```
GET /oidc/.well-known/webfinger?resource=acct%3Ajoe%40mojeid.cz&rel=http%3A%2F
↪%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fissuer HTTP/1.1
Host: mojeid.cz
```

An example of the server's response:

```
HTTP/1.1 200 OK
Content-Type: application/jrd+json

{
 "subject": "acct:joe@mojeid.cz",
 "links": [
    {"rel": "http://openid.net/specs/connect/1.0/issuer",
     "href": "https://mojeid.cz/oidc/"}
 ]
}
```

5.1.6 Requesting Identity Authentication

Once you know the OCP endpoint, your application sends an identity authentication request using the user's browser redirection. The request includes special parameters for its realization. Correct use of these parameters is done by the OpenID Connect library used for implementation.

Identity authentication request usually includes the following parameters:

- **Return address (URL) of the application** – The address to which the user returns after logging in from the OpenID Connect provider's website and where the outcome of the login is processed.
- **Required groups of data from mojeID** – An identity authentication request has to contain at least *openid* as a required group of data.
- **Required data from mojeID** – An identity authentication request can also include a list of individual data from the mojeID identity which your application requires and which are handed over to your application with the user's consent after a successful login. For each piece of data, its identifier needs to be presented. The data and its identifiers are listed in *Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58). This list has a

JSON format specified in the OpenID Connect documentation[14]. Any item can be marked as required using an expression `"essential": true`.

Examples of items that can be included in the identity authentication request are listed in the following table:

Parameter (key)	Description and value
`scope`	List of required groups of data *openid address*
`response_type`	Determining the required authentication schema *id_token*
`client_id`	Unique service provider's identifier *test_clienti*
`redirect_uri`	Return address from mojeID. *http://www.poskytovatel-example.cz/*
`claims`	More detailed specification of the required data. `{"userinfo": {"name": null, "nickname": {"essential": true}}}`

Example of an authentication request:

```
sid, location = client.begin(path=URL, scope=SCOPE)
HttpResponseRedirect(location)
```

Example of an authentication request query:

Listing 1: Example of requesting data via "scope" (group of data)

```
GET /oidc/authorization/?response_type=code&scope=openid%20profile%20email&client_
↪id=s6BhdRkqt3&state=af0ifjsldkj&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb␣
↪HTTP/1.1
Host: mojeid.cz
```

Listing 2: Example of requesting data via "claims" (individual data)

```
GET /oidc/authorization/?state=950ba54cb302a7c6a814f22a4e5c5445&redirect_uri=https%3A
↪%2F%2Fmojeid.cz%3A8000%2Fconsumer%2Foic%2Ffinish%2F&response_type=code&client_
↪id=8ol68PATaSpA&scope=openid&claims=%7B%22userinfo%22%3A+%7B%22name%22%3A+null%2C+
↪%22nickname%22%3A+%7B%22essential%22%3A+true%7D%7D%7D&ui_locales=off HTTP/1.1
Host: mojeid.cz
```

The response from the server comes only after the authetication is performed. Example of the response can be found in the *Response to Authentication* (page 28) section.

[14] https://openid.net/specs/openid-connect-core-1_0.html#ClaimsParameter

5.1.7 Performing Authentication

When a user comes to the mojeID server with a identity verification request, they see a login page where the login takes place.

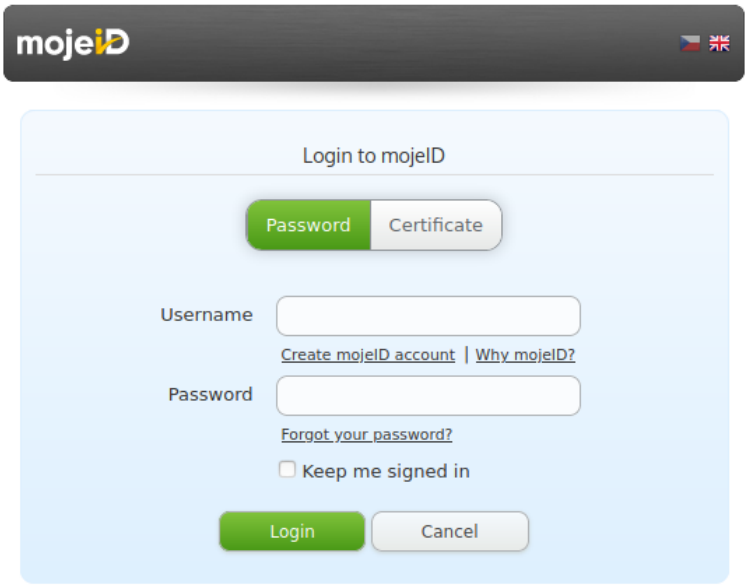


Fig. 2: *mojeID login page*

This authentication is performed by the mojeID servers. Within this authentication, we will try to perform as many tasks specified by the parameters in the identity authentication request as possible. The whole process takes place exclusively within the mojeID systems and requires no activity from your side.

5.1.8 Response to Authentication

When a user completes the authentication process, you will receive a response with its result from the mojeID servers. The structure and contents of this response differs based on the selected communication schema (see *Communication via OpenID Connect* (page 8)).

In case of communication via the *Implicit schema* (page 9), the response includes the user's identifier and ID Token which can contain data about the user.

In case of communication via *Access code* (page 10) or *Hybrid schema* (page 11), the response contains an access code that needs to be used in the next step of the authentication process.

An example of processing the response:

```
aresp, _, _ = client.parse_authz(request.GET.urlencode())
```

An example of the server's response:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=af0ifjsldkj
```

5.1.9 Requesting Token

If you received an access code in the previous step of authentication, you have to replace it with a valid token at the Token Endpoint.

In case of communication via the *Hybrid schema* (page 11), the response includes an access token and ID Token which can contain data about the user. In this case, the authentication and data transfer process is complete.

In case of communication via an *Access code* (page 10), the response again includes a token and ID Token, but it does not contain any data about the user. You have to request them in the next step.

An example of communication:

```
POST /oidc/token/ HTTP/1.1
Host: mojeid.cz
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F
↪%2Fclient.example.org%2Fcb
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "access_token": "SlAV32hkKG",
 "token_type": "Bearer",
 "refresh_token": "8xLOxBtZp8",
 "expires_in": 3600,
 "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
              yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
              NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
              fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
              AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
              Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
              NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
              QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
              K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
              XUVrW0LrLl0nx7RkKU8NXNHq-rvKMzqg"
}
```

5.1.10 Requesting Data

In this step you will use the token received in the previous authentication step to get the user's data. The data needs to be retrieved from the UserInfo Endpoint.

The UserInfo Endpoint always returns an attribute `sub` (*subject*), in the response which uniquely identifies the user and should be used to validate the response using an *ID Token*.

The user's data should be processed only in case the response is found valid.

An example of requesting data:

```
state = aresp.to_dict()['state']
resp = client.complete(state)
uinfo = client.get_user_info(state)
```

An example of communication with server:

```
GET /oidc/userinfo/ HTTP/1.1
Host: mojeid.cz
Authorization: Bearer SlAV32hkKG
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "sub": "248289761001",
 "name": "Jane Doe",
 "given_name": "Jane",
 "family_name": "Doe",
 "preferred_username": "j.doe",
 "email": "janedoe@example.com"
}
```

5.1.11 mojeID LITE Library

Javascript library **mojeID LITE** (or also mojeID Connect) allows to load data from a mojeID identity to a website on the client's side using the OpenID Connect protocol.

This feature can be used, for example, to simply prefill a web form with data of a user with an active mojeID account.

To enable this feature in your web form, you have to perform at least the following steps:

1. *Insert a link to the library.*

   If you want to decrease your dependency on an external website, you can upload this library to your own website. The library can be downloaded here[15]. The library depends on a cryptographic library jsrsasign[16] which is available (in its newest version) on our webiste, so you do not have to insert it directly. The code of the script to insert the library has to be inside `<HEAD>`.

   An example of inserting the library:

```
<script type="text/javascript"
   src="https://www.mojeid.cz/public/media/1542958574/150/"
   data-jsrsasign="https://www.mojeid.cz/public/media/1542956522/149/">
</script>
```

2. *Call a function for creating a MojeidConnect object.*

   This object represents communication with mojeID server. When calling the creating function, you can *set certain parameters* (page 32), that will affect the data transfer process. The code of the script to call the function has to be inside `<HEAD>`.

   An example of creating the object:

```
<script type="text/javascript"> (function() {
   mojeid = createMojeidConnect( {
      clientName: "Sample form",
      claims: ['phone_number', 'family_name', 'given_name', 'nickname',
         'email', 'address', 'birthdate', 'gender', 'website', 'profile']
   } );
})();</script>
```

3. *Attach calling of requestAuthentication() method to the button that activates the prefilling of the form.*

   This method initiates the authentication process and filling the form with the values of the confirmed data.

   An example of a code for the button:

```
<button onclick="mojeid.requestAuthentication()">
Prefill using mojeID
</button>
```

[15] https://www.mojeid.cz/public/media/1542958574/150/
[16] https://kjur.github.io/jsrsasign/

createMojeidConnect(options) function parameters

When calling this function, you can set certain parameters (in dictionary structure) that will affect communication with the mojeID server:

`clientID`

It is possible that the service is already registered in the mojeID server. If yes, this service has a clientID assigned and you can provide it in the parameter. If the clientID parameter is not defined, registration is completed dynamically according to the OpenID Connect specifications[17] using the address from the `regEndpoint` parameter.

`clientName`

In case of dynamic registration, it is possible to define the name of the service that is shown to the user upon data transfer approval. If the name is not defined, the service's URL is used.

`scope`

Required transferred data in form of a group of data. The value is a sublist `['openid', 'profile', 'email', 'phone', 'address']`, while `'openid'` is required. If it is not defined, the value is `['openid']`.

`claims`

Required transferred data in form of individual attributes. The value is a list of attributes. A full list of possible attributes is available in the value of `claims_supported` from server's configuration file[18]. An example of a list: `['phone_number', 'family_name', 'given_name', 'nickname', 'email', 'address', 'birthdate', 'gender', 'website', 'profile']`

`attrDict`

The library assumes the form items have the same `id` as the name of the attribute from the `claims` list. If that is not the case, it is possible to define a map list for the form item id and for the attribute name in this parameter.

`formCallback`

If the map dictionary from `attrDict` is not sufficient, you can define a name of your own JS function that will take care of filling the form.

`display`

The value is either `popup` or `redirect` based on whether the login should be done in a new window or in the existing one. The default value is `popup`.

`regEndpoint`

Registration endpoint's URL according to the OpenID Connect protocol specification[19]. The default value is `https://mojeid.cz/oidc/registration/`.

`authEndpoint`

[17] https://openid.net/specs/openid-connect-registration-1_0.html
[18] https://mojeid.cz/oidc/.well-known/openid-configuration/
[19] https://openid.net/specs/openid-connect-registration-1_0.html

Authentication endpoint's URL according to the OpenID Connect protocol specification[20]. The default value is `https://mojeid.cz/oidc/authorization/`.

Sample form

For easier understanding, you can have a look at and try a full form sample[21].

5.2 Implementation via OpenID 2.0

This section will introduce you to the technical aspects of the implementation of mojeID via the OpenID protocol to web applications.

We recommend to study this text in order to properly understand the principles and processes of mojeID/OpenID. Most of the things described here can be solved by *available libraries* (page 33) for the implementation of OpenID that we recommend to use.

The official specification of the OpenID protocol can be found at https://openid.net/developers/specs.

The list of data that can be transferred by the protocol (including their identifiers) can be found in the *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62).

Examples and solutions of error messages can be found in the *Appendix 6 – Examples and Solution of Error Messages* (page 77).

5.2.1 Overview of Libraries and Modules

The official OpenID Foundation website offers a list of OpenID 2.0 protocol implementations in several programming languages (see Libraries for Obsolete Specifications[22]).

We offer examples of implementation for PHP (ZIP, 140 kB[23]) and Java (ZIP, 3,5 MB[24]).

Note: The examples of implementation cannot be simply installed and run. They will not work with your system as they are. They are meant to serve as an example for creating your own implementation.

Then, for a few most popular open-source platforms, there are modules that we have created to make implementation to these systems easier:

- Drupal (ZIP, 153 kB[25])
- Joomla! (ZIP, 170 kB[26])
- PrestaShop (ZIP, 1,3 MB[27])
- Moodle (ZIP, 184 kB[28])

[20] https://openid.net/specs/openid-connect-registration-1_0.html
[21] https://www.mojeid.cz/public/media/1542960671/153/
[22] https://openid.net/developers/libraries/obsolete/
[23] https://www.mojeid.cz/public/media/1542891506/143/
[24] https://www.mojeid.cz/public/media/1542891505/141/
[25] https://www.mojeid.cz/public/media/1536662546/102/
[26] https://www.mojeid.cz/public/media/1536662546/100/
[27] https://www.mojeid.cz/public/media/1536662546/103/
[28] https://www.mojeid.cz/public/media/1536662546/99/

- Magento (ZIP, 639 kB[29])

We also have a few more in our archives (WordPress, VirtueMart, phpBB, Opencart, osCommerce, Redmine, Zencart) and we are happy to provide them if you ask our support team (techsupport@mojeid.cz).

Caution: We inform you that you can use modules only at your own risk and the CZ.NIC Association is not subject to any liability for any damage.

[29] https://www.mojeid.cz/public/media/1536662546/104/

5.2.2 Establishing Association

The messages you receive indirectly via the user's browser from the OpenID provider are digitally signed. For each such message, the signatures must be verified and it must be confirmed that it really comes from the OpenID provider. Therefore it is possible to choose from two different possibilities – the so-called stateful and stateless communication between your application and the OpenID provider.

In **stateless** communication, you have to verify the message by establishing a communication with the OpenID provider with a request for verification of the given message. That is more performance and time consuming.

Stateful communication begins with establishing a shared secret before the start of the process of the user login, e.g. identity authentication – the so-called establishing association This shared secret is valid for the maximal period of 14 days and after it expires, the association must be established again. Both sides (the OpenID provider and your application) can also declare this shared secret invalid at any time during its period of validity. In such case, it is good to establish the association again, so that the stateless communication needs not to be used.

Tip: The OpenID libraries that can be used to implement mojeID can use both options. In common situations, we recommend using the stateful communication as much as possible. In some cases, it is necessary to use the stateless communication too, e.g. if the shared secret expired or if one side invalidated it, it is necessary to verify messages by stateless communication, until a new association is established.

5.2.3 Requesting Login via mojeID

The process of identity authentication starts by the user requesting to log in via mojeID at your website.

To ensure maximal user friendliness it is implemented using a login button. The username and password is entered later on the mojeID server.

Fig. 3: *Examples of buttons for login via mojeID*

Logging in to mojeID using a button is the only recommended and correct method. The buttons can be downloaded at https://www.mojeid.cz/en/provider/getting-started/#download

5.2.4 Initiation

To be able to send an identity authentication request, you have to state either the user's identifier, or the OCP endpoint for most libraries. If you do not know the user's identifier (e.g. in case of user's login), state the OP endpoint instead.

If you know the user's identifier (e.g. repeated authentication of the user), you can use it to get metadata about the user's identity and about the OpenID provider including the OP endpoint. A HTTP request is sent to the user's identifier and the following can be found in the body of the message received this way:

- **User's claimed identifier** – The final URL from which the page's body with metadata returned.
- **User's internal identifier** – It differs from the identity name by the fact that it is an identifier in form of `https://mojeid.com/id/unique_string`, where the `unique_string` is the user's unique identification in the mojeID system, e.g. `https://mojeid.cz/id/JeDineCny/`. This internal identity must then be checked in the next phases of the login process, because it is an identity recognized by the OpenID provider (see *Response Processing* (page 43)).
- **OP endpoint** – this is always `https://mojeid.cz/endpoint/` and this address is used for identity authentication requests.

5.2.5 Requesting Identity Authentication

Once you know the OP endpoint, or also the claimed identifier and internal identifier, your application sends an identity authentication request using the user's browser redirection. The request includes special parameters for its realization. These parameters are listed in the

message's body using their identifiers. Design of this identity authentication request are again directly carried out by OpenID libraries you will use for the implementation.

Identity authentication request usually includes the following parameters:

- **Return address of the service provider's application** – The address to which the user returns after logging in from the OpenID provider's website and where the outcome of the login is processed in your application.
- **Service provider's URL realm** (hereinafter as *realm*) – defines the part of a URL region for which the identity authentication request is valid. The service provider's return address must lie in this URL region. The *XRDS document* (page 40) or an information about this position must be available at this address.
- **Choice of the required login method** – the choice is done by placing the identifier of the given login method into the identity authentication request. The mojeID service does not only supports the common login by password, but also login by a digital certificate or a one-time password (OTP).
  - **Login using certificate** can be requested via PAPE extension using the following identifier:
    `http://schemas.openid.net/pape/policies/2007/06/phishing-resistant`

    When logging in using a certificate, the following messages are displayed:
    "Poskytovatel služby požaduje přihlášení certifikátem."
    "The service provider wants you to login with your certificate."
  - **Login using one-time password or the MojeID Authenticator application** can be requested using the following identifier:
    `http://schemas.openid.net/pape/policies/2007/06/multi-factor`

    When logging in using a one-time password, the following messages are displayed:
    "Poskytovatel služby požaduje přihlášení jednorázovým heslem nebo MojeID Autentikátorem."
    "The service provider wants you to login with one time password or MojeID Autentikátor."
  - **Login using a security key** can be requested using an identifier:
    `http://schemas.openid.net/pape/policies/2007/06/multi-factor-physical`

    When logging in using a security key, the following messages are displayed:
    "Poskytovatel služby požaduje přihlášení druhým faktorem."
    "The service provider wants you to login with two-factor authentication."
- **Limiting the user's login period** – if the user logs in successfully, the mojeID system keeps this user "logged in". If the user logs in to another service provider during this period, they do not have to authenticate on the mojeID login page. However, you can limit your identity authentication request for any period since the last authentication if you consider it necessary, e.g. because of security. This choice can be requested using the `max_auth_age` field of the PAPE extension. If the user has not logged in to mojeID in the last `max_auth_age` seconds, he needs to log in again.
- **User's claimed identifier to be verified** – The identity name corresponding to this claimed identifier will be displayed to the user on the mojeID login page. If the user chooses the identifier from OP, it contains a special value.

- **Required data from mojeID identity** – identity authentication request can also include a list of individual data from mojeID identity which your application requires and which are handed over to your application with the user's consent after a successful login. For each piece of data, its identifier needs to be presented. MojeID supports requesting the following data (details and formats of the individual items can be found at the address of the identifier of the piece of data, some of them – name, nickname, e-mail. date of birth, postal code, and country – can be retrieved using a simpler identifier of the SReg extension). The list of possible data can be found in the *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62).

Examples of items that can be included in the identity authentication request are listed in the following table:

Parameter (key)	Description (value)
`openid.ns`	Determining the used OpenID protocol. *http://specs.openid.net/auth/2.0*
`openid.claimed_id`	User's claimed identifier. *http://demo.mojeid.cz/*
`openid.identity`	User's internal identifier. *http://mojeid.cz/id/unique_string/*
`openid.assoc_handle`	Identyfing string of a previously established association. *{HMAC-SHA256}{4c486ac3}{Ze6JZA==}*
`openid.return_to`	Return address from mojeID. In older OpenID protocol specifications, this field is called openid.trust_root. *http://www.provider-example.cz/MojeID-Return.html*
`openid.realm`	Service provider's URL region. *http://www.provider-example.cz/*
`openid.ns.ax`	Choosing an extension for attribute exchange. The "ax" string can have any other name your library chooses. Here it is only stated how it will be linked from now on. *http://openid.net/srv/ax/1.0*
`openid.ax.mode`	Attribute exchange mode (receiving, storing). *fetch_request*
`openid.ax.type.firstName`	Requesting an attribute; there can be any attribute instead of `firstName` řetězec. *http://axschema.org/namePerson/first*
`openid.ax.type.validated`	Another attribute – this time information about user's data verification. *http://specs.nic.cz/attr/contact/valid*
`openid.ax.type.jabber`	*http://axschema.org/contact/IM/Jabber*
`openid.ax.required`	A list of attributes about which the service provider claims they are essential for proper account creation/update, or rather for the service provider's application operation itself (required items). *firstName,validated*
`openid.ax.if_available`	A list of extra attributes (optional items). The service provider wants them, but it is ok if he does not get them. *Jabber*
`openid.ns.pape`	Choosing an extension for authentication policies. *http://specs.openid.net/extensions/pape/1.0*
`openid.pape.max_auth_age`	Number of seconds since the last authentication. If the user did not authenticate during this period, they have to authenticate again. *3600*
`openid.pape .preferred_auth_policies`	A space-separated list of identifiers of the required policies. *http://schemas.openid.net/pape/policies/2007/06/phishing-resistant*

5.2.6 Performing Authentication (XRDS and *realm*)

When a user comes to the mojeID endpoint with an identity authentication request from your application, they see a login page where the login takes place.

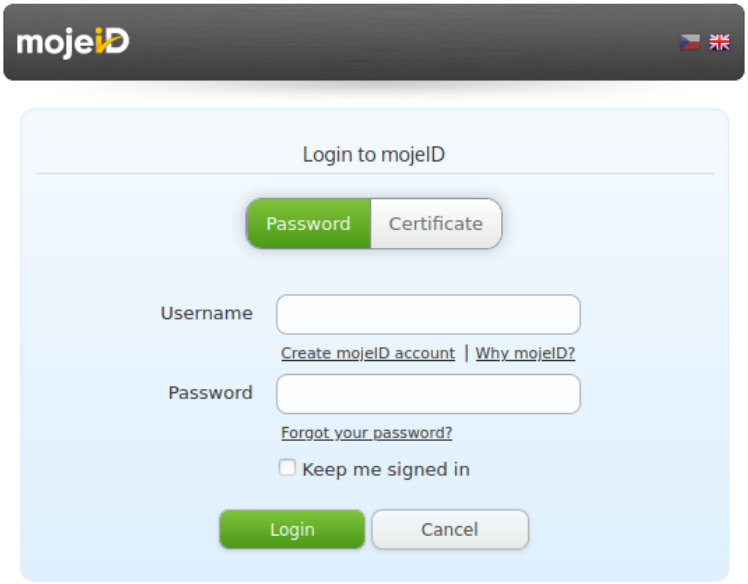


Fig. 4: *A dialog for inputting mojeID identifier on mojeID website.*

This authentication is performed by the mojeID servers. Within this authentication, we will try to perform as many tasks specified by the parameters in the identity authentication request as possible. The whole process takes place within the mojeID systems and requires no activity from your side. A part of this process is verification of your return address; the user is informed of its outcome. Another part of this auhentication is the YADIS protocol getting more data about you and verifying them against your data in the message. The correct response to a query from the YADIS protocol returns either an XRDS, or an HTML document in which the XRDS document's position is disclosed.

XRDS Document and its Format

The XRDS document's position is disclosed by the following META tag in site's header that you place at the address of your *realm*at OpenID:

```
<meta http-equiv="x-xrds-location" content="http://www.example.com/xrds.xml"/>
```

An example of a custom XRDS document for login into mojeID:

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://$xrds" xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service>
      <Type>http://specs.openid.net/auth/2.0/return_to</Type>
      <URI>http://www.provider-example.com/MojeID-Return.html</URI>
    </Service>
  </XRD>
</xrds:XRDS>
```

An example of an XRDS document for registration into mojeID:

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://$xrds" xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service>
      <Type>http://specs.openid.net/auth/2.0/assert_url</Type>
      <URI>URL rozhraní</URI>
```

(continues on next page)

(continued from previous page)

```
    </Service>
  </XRD>
</xrds:XRDS>
```

where the URI tag must include your application's return address from the identity authentication request. During the whole process of getting the document, your server must not return any redirection (HTTP code 3xx), otherwise the document is considered invalid/forged.

Choosing a Suitable *Realm*

Realm is a unique identifier of the service you provide within the mojeID system, so its correct selection makes orientation for users easier. According to the OpenID specification, the *realm* should correspond to the part of the URL-space for which the request is valid. In case of login, the *realm* should not be smaller than the part of URL-space covered by consequently created session.

We therefore recommend to use exactly one realm for one second level-domain. Because two URLs that differ even only by schema are different according to specifications, we strongly recommend using exclusively an HTTPS schema if it is available. This also prevents users' data interception as they are sent to your application.

If you use only one second-level domain, we recommend choosing a *realm* in form of `https://example.com/` or `https://www.example.com/`.

Important: The return address must have the same domain as the *realm*, otherwise the OpenID query is invalid.

If you use third or lower level subdomains, we recommend using substitute symbol `*` and choose the *realm* in form of `https://*.example.com/`. Such *realm* allows using return addresses with any subdomain, e.g.: `https://www.example.com/`, `https://sub.example.com/return/address/`, `https://sub.do.main.example.com/`, but not with the domain itself (in this case `https://example.com/`).

The XRDS document will be searched for at the URL where the * symbol is replaced by "www", i.e. at `https://www.example.cz/`. All the used forms must be in HTTPS protocol.

Important: The *realm* must not include an IP address, always use a domain name.

5.2.7 Response with the Identity Authentication Outcome

In case your application requested it, it is indirectly (via redirection of user's browser) sent back a message with the response, or more precisely the outcome of the identity authentication and other data it requested. This response is again in form of an HTTP message, while the body of this message includes the individual data representing the individual pieces of information of the identity authentication process outcome.

These are examples of the items comprising the response to the identity authentication request:

Parameter (key)	Description (value)
`openid.claimed_id`	Returns the user's claimed identifier, it can differ from the default by a fragment. You can use this string to match user specific data. When comparing, it is important to take into account all the parts of the string, including schema and fragment. *https://demo.mojeid.cz/#unikatni_retezec*
`openid.op_endpoint`	MojeID endpoint URL. *https://mojeid.cz/endpoint/*
`openid.response_nonce`	Unique response tag. No two responses have the same tag – it prevents from the response being sent repeatedly (the so-called replay attack). *2010-07-22T16:13:08ZiEnTtR*
`openid.signed`	A list of fields that are signed with a signature, see the following key. *assoc_handle, claimed_id, ns, op_endpoint, pape.auth_policies, response_nonce, signed*
`openid.sig`	Signature of the listed fields to verify authenticity. *hdtOpg3jCup1n6+elCXn+yLZAYc=*
`openid.ax.type.firstName`	Mapping the official URL identifier to a string used in the message. *http://axschema.org/namePerson/first*
`openid.ax.value.firstName`	The value of identity attribute for the given string. *Andrew*
`openid.pape.auth_policies`	A space-separated list of login policies that were actually applied. *http://schemas.openid.net/pape/policies/2007/06/phishing-resistant*
`openid.pape.auth_time`	The time of the user's identity verification on server (always in UTC). *2005-05-15T17:11:51Z*

5.2.8 Response Verification

Each message with a response is digitally signed and must be verified. The following parts of the message are verified:

- **return URL** – The value `openid.return_to` must correspond with the URL to which the query was delivered. All the parameters of this URL must be included in the HTTP message your application received.
- **claimed identifier** – Metadata associated with the claimed identifier obtained during the initiation or by repeating a part of this process must match the data included in this message – claimed identifier, internal identifier, OP endpoint and protocol version.

- **response nonce** – A message with the same nonce from this OpenID provider was not yet received.
- **signature** – All the fields that must be signed are signed and the signature is valid. The signature can be either verified by the application itself within the status communication, or the OpenID provider requests the verification of the signature.

If all these conditions are fulfilled, then the message is **valid** and it was verified that the claimed identifier belongs to the user. However, all the parts should be processed by the library that implements the protocol.

5.2.9 Response Processing

If the message with the response to the identity authentication request is successfully verified, your application can process the data included in the response and finish the process of login via mojeID. The web application must perform this processing at the return address included in the identity authentication request.

Login Outcome

When processing the login outcome, it is necessary to take care of the following special situations regarding a successfull login:

- **The user's first login** – if the user who has successfully logged in visits your application for the first time, it is usually necessary that you create an account for them where the data retrieved from the mojeID identity will be stored, plus, of course, also all the other application-specific data. When creating such account, it is recommended to:
  - use the data retrieved from the mojeID identity instead of filling in the registration form, or show the user only such fields that could not be filled with the data retrieved from mojeID
  - and inform the user of the data from mojeID identity that the given application needs and recommend them to allow its transfer upon each login.
- **Repeated login vs. new user login** – each time the response is processed the claimed identity of the user must be checked, because two different users might have the same identity name in case one person cancels their mojeID identity (and thus releases their identity name) and another person creates an identity with the same identity name. Such users are then distinguished using a unique string at the end of the claimed identity's URL.
- **Login of a user who did not requested it directly** – your application might receive a response with a successful login even if the user did not request the login directly in your application. It is a situation that should not be considered an error – the login request came from a different website than the one to which the data returns (the protocol does not stores the information about the application that generated the message; if you require such information, you have to add it yourself). However, thanks to the information presented at the mojeID login, the users are always aware to which service they are logging in and to whom they are providing their data.

When processing the login outcome, it is necessary to take care of the following situations regarding an unsuccessfull login:

- **Login request denial** – After the user comes to the login page, they can deny the login request, for example because they did not initiated it themselves. Your application then has to take care of this state.
- **Impossibility of immediate authentication** – Your application can force identity authentication without a contact with the user. If the OpenID provider is not able to provide such authentication, this type of response is returned which means the classic authentication needs to be performed. Some libraries do not distinguish this state from the previous state.
- **Protocol error** – The OpenID provider returns this type of message if it is able to determine your application's return address but it is not able to distinguish another field of the message because it contains data that contradict the protocol. The OpenID provider returns this type of messages, for example, if he receives a message with an internal identifier he is not able to verify.

MojeID Identity Data

If you enquire about the mojeID identity data, it is necessary to take care of the following special situations:

- **User's repeated login** – upon each repeated login of the user using mojeID, it is necessary to check whether the data stored in your application's internal account match the data retrieved from the user's mojeID identity upon login. In case they differ, the data in the internal account need to be updated with the data from the mojeID identity, as those are probably up to date.
- **Required data not received** – the user always has a chance to choose which data will and will not be handed over to your application upon login. Therefore, it is possible that the application requests certain data, yet it does not receive them due to the user's choice. It is recommended to take care of this situation and divide the data requested by the application into required data that are essential for the application to work, and optional data that the application does not need. You can then design the application's operation based on this division.

A special case is the possibility of allowing only the login of fully identified or validated (physically verified) mojeID users. The `http://specs.nic.cz/attr/contact/valid` and `http://specs.nic.cz/attr/contact/status` items are handed over every time an application of a provider with **full access** requests them. The data which the user does not allow to be handed over are returned in the body of the message without a value.

5.3 Implementation via SAML

SAML is a protocol that historically precedes the newer OpenID protocols. If your system already supports SAML (for example an installation of Shibboleth system or similar), it is also possible to use this protocol to enable mojeID.

SAML 2.0 implementation is based on specifications available at https://wiki.oasis-open.org/security/FrontPage

To enable mojeID, you need to send the service's metadata to techsupport@mojeid.cz, and you might also need to register mojeID metadata listed at https://mojeid.cz/saml/idp.xml. The certificate listed in metadata can change, so the metadata need to be updated from time to time. Metadata signature can be verified using the certificate at https://mojeid.cz/saml/cert.

Because SAML messages are *base64-encoded* and *deflated*, you can convert them to a readable XML for the debugging purposes (you can use for example https://www.samltool.com/decode.php).

The list of data that can be transferred by the protocol (including their identifiers) is available in the *Appendix 3 – List of Data to be Handed Over (SAML)* (page 68) and *Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)* (page 70).

Examples and solutions of error messages can be found in the *Appendix 6 – Examples and Solution of Error Messages* (page 77).

5.4 Problems with Implementation

This section informs about some possible problems with implementation and offers ways to solve them or avoid them.

5.4.1 Differences Between the Protocols

A major difference between the protocols is that each protocol can hand over only certain data from the mojeID identity and the set of data is different for each protocol.

We work on their unification, but at this moment, **it is not possible** to hand over all the identity data using each supported protocol.

The data that are handed over for each protocol are listed in the following appendices:

- *Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58),
- *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62),
- *Appendix 3 – List of Data to be Handed Over (SAML)* (page 68),
- *Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)* (page 70).

5.4.2 Transition to a Different Protocol

In general, the transition to a different protocol is performed by the user logging in to a service using one of the current login methods and then they login again using the new protocol. This way, the service provider can assign the existing user a new protocol identifier.

Transition from OpenID 2.0 protocol to the new OpenId Connect protocol

If you want to transition from the original OpenID 2.0 protocol to the latest OpenID Connect, send an identity authentication request via the OpenID Connect protocol with the scope parameter extended with an openid2 value, and you will receive an OpenID 2.0 identity together with an OpenID Connect identity.

More details about the migration process can be found in these specifications[30].

5.4.3 Adjusting Communication with mojeID Server

To debug communication issues, we recommend to use developer tools in the internet browser. They enable checking network activities: queries and responses exchanged between the client (your implementation) and the mojeID server. This can help you detect a possible error in the data that are handed over.

Note: For more complicated issues, when you have to contact out technical support, it is useful to attach a recorded communication log to the description of the issue.

In `Firefox`, you can use built-in tools or extensions (e.g. FireBug):

1. The developers tools can be enabled in *Main Menu* → *Web Developer* or by a shortcut `Ctrl+Shift+I`.
2. Then you switch to the *Network* tab (or call this tab directly using the shortcut `Ctrl+Shift+Q`).

In `Chrome`, you can use built-in tools:

1. The developers tools can be enabled in *Main menu* → *More Tools* → *Developer tools* or by a shortcut `Ctrl+Shift+I`.
2. Then switch to the *Network* tab.

Debugging in a Pop-up Window

If you implement the user authentication via mojeID using a new pop-up window, you need to do the following to record the communication:

1. Let the pop-up window generate for the first time.
2. Before sending the request to the mojeID server, right click inside it and open the debug tool by choosing the following item in the menu:
   - Chromium: *Inspect*
   - Firefox: *Inspect Element*
   - FireBug plugin: *Inspect Element in Firebug*
3. Call a pop-up window refresh (e.g. `F5` or `Ctrl+R`).
4. Continue reporting network communication in the debug tool in a normal way.

[30] https://openid.net/specs/openid-connect-migration-1_0.html

Chapter 6

Interface for Creating mojeID Accounts

This chapter describes the process of registration mojeID accounts via your application.

6.1 Requesting Creation of a mojeID Account

The user selects an option of creating a mojeID account within your application. That will generate a HTTPS POST request to a registration server in user's browser at `https://mojeid.cz/registration/endpoint/`. The parameters of the request contain a username and all the other available data about the user (the list of data for registration include *Appendix 5 – List of Data for Registration* (page 73)), plus:

- **the service provider identifier** (*realm*) – a selectable URI with a value based on the type of the communication protocol:
  - in case of OpenID 2.0, it should be the same value as the one used for logging into mojeID,
  - in case of OpenID Connect, it must be the assigned `client_id`,
- **unique transaction identifier** (`registration_nonce`) – used to match the response to this request.

You also have the possibility to offer the user a transfer of an existing account in the central register by choosing the address `https://mojeid.cz/transfer/endpoint/`. In such case, the data about the user are ignored and the username (= contact identifier) that cannot be changed is used. If the identifier is invalid, it cannot be transferred to mojeID and the user has to contact the current registrator to ask for change.

Then the user is displayed a list of data to be entered into mojeID once the registration is completed. The basic data also shows its value and it can be changed. The user will then consent to the service usage policy within the registration form and it will be verified with CAPTCHA.

6.2 Checking Data Validity

After a registration form is submitted, the registration server checks data validity and asks the user to correct any errors. In case the data is valid, the process of registration of a new account is initiated. The registration server saves all the necessary data in this account and adds your identification (service provider identifier, realm). Then, the identification of the user starts with sending PIN1 and PIN2.

The next step is informing your application of a successful registration.

In case of communication via OpenID 2.0, the server uses the URI that identifies your realm and tries to find an *XRDS document* (page 40) with at least one `<xrd:Service>` element containing the following elements:

- `<xrd:Type>` with a value of `http://specs.nic.cz/registration/assert_url` and

- `<xrd:URI>` with a URL interface where the information about registration is sent.

During this process, there must not be any redirection and the URL interface must be within the service provider's URI (realm) (see https://openid.net/specs/openid-authentication-2_0.html#realms).

In case of communication via OpenID Connect, the URL for sending information must be entered during the *client registration* (page 22) process using the `assertion_uris` key that contains a list of addresses (encrypted in a JSON) to send the messages to.

Your application directly sends a HTTPS POST message to the interface determined by the URL. The message contains three parameters:

- `registration_nonce` — a unique transaction identifier for matching with the original request,
- mojeID user's identifier based on the used protocol:
  - `claimed_id` – in case of the OpenID 2.0 protocol,
  - `sub` – in case of the OpenID Connect protocol,
  - `status` – status with the value `REGISTERED`.

Your application first has to verify this message:

- it has to check if the message was delivered to one of the addresses listed in the *Requesting Creation of a mojeID Account* (page 47) of a mojeID Account section,
- it has to check if the `registration_nonce` transaction was really created,
- it has to check if the client certificate used to create an SSL tunnel is valid and signed by the certificate authority, the CZ.NIC Association. This certificate is available at https://www.mojeid.cz/en/provider/getting-started/#download for both production and test environment. The certificate is needed for notification in both the production and test environment.

If you do not use HTTPS and you want to try logging in and creating accounts in the test environment, you do not need this certificate.

If you use HTTPS and you are in the test environment, you need this certificate to send notifications from registration. It is not needed for logging in (only general public data is transferred between mojeID and your server, so it is not necessary to check the "identity" of the requester).

The notifications are sent after registration, partial identification (PIN1 and PIN2) and identification (PIN3) to `assert_url` listed in the XRDS document in the realm. This works also in the test environment. If you want your application to be able to receive notifications, you need a realm with HTTPS. When the notification is received, it is necessary to response with a `'mode:accept\n'` string, where the new lines are marked with `\n`.

Tip: The client certificate verification can be done by an HTTP server, e.g. Apache with the `SSLVerifyClient` configuration option.

If all the requirements are met, your application can match the mojeID identifier with its record of the user during the processing of this message for the purpose of authentication via mojeID.

Note: If this message cannot be sent securely using HTTPS, the registration continues without sending this message.

6.3 Completing Registration

Your application sends a response to the message from the *Checking Data Validity* (page 47) section in the body of a HTTP response in a form of key-value of the OpenID protocol

- **outcome** (`mode`) – value `accept` or `reject` indicating, whether the user's account was successfully paired,
- **reason for denial** (`reason`) – an optional parameter that includes the reason the pairing was not performed.

If a response in the correct format is not received, the message with the result of the registration will be sent to another address from the *Checking Data Validity* (page 47) section, until a response is obtained or until all the addresses are used.

The registration then continues either with a direct request to enter PIN1 and PIN2, and going to the user's profile where they choose a password, or the user is shown an information about the completion of the registration.

If you have activated the *full access*, your application will also receive information about a change of the status of the user's account. These messages are sent in a similar way as described in the *Checking Data Validity* (page 47) section, with two parameters in each message:

- mojeID user's identifier based on the used protocol:
  - `claimed_id` – in case of OpenID 2.0,
  - `sub` – in case of OpenID Connect,
- `status` – account status, one of the following values:
  - `CONDITIONALLY_IDENTIFIED` – partially identified (PIN1 and PIN2 entered),
  - `IDENTIFIED` – identified (PIN1, PIN2 and PIN3 entered),
  - `VALIDATED` – validated (PIN1, PIN2, PIN3[31], and the validation flag entered).

If this message cannot be sent or no response is received, the information of the change of state will be sent repeatedly each five minutes for the period of six hours, until your application accepts or refuses it. On the other hand, the message about the completion of the registration is synchronous – it is sent only once.

[31] The PIN3 for validation of the mojeID account is optional. Therefore, there can be a situation when the user has a validated account with only PIN1 and PIN2 entered.

Chapter 7

Logging out of mojeID

It is logical based on the way mojeID works that your service cannot automatically log a user out because it would log them out of other services they are logged into via mojeID too. However, in rare cases, a user might need to be logged out of mojeID too. For example, when they logged in from someone else's device.

Then it is desirable that upon or after logging out of your service, the user is asked if they want to be logged out of mojeID too.

If the user chooses this option, redirect them or provide a link to `https://mojeid.cz/logout/` where they can confirm the logout.

We recommend implementing this option if users often access your service from public devices (e.g. in a library or internet café) and if it is not securely solved, e.g. by deleting data after finishing working with the browser.

However, its implementation is not mandatory.

Chapter 8

mojeID Test Instance

It is possible to test your implementation using our mojeID test instance where you can test logging of mojeID users, registering of new accounts and transferring of accounts from the central register.

Before you start testing, send the metadata you are going to use for testing to techsupport@mojeid.cz. This metadata differs for each protocol (see infomation about the individual protocols below).

Important: Use different metadata than for the production instance!

We will grant you access to the test instance and set up a so-called *full access*, for the purpose of testing, so that you can receive all the mojeID account data, including `status`, `valid` and more that are transferred only to the providers with *full access*.

8.1 Test Accounts

To test mojeID, we recommend creating three test users with different levels of validation[32]:

- partially identified, with only PIN1 and PIN2 entered,
- identified, with PIN1, PIN2 and PIN3 entered,
- validated, with PIN1, PIN2, PIN3, and the validation flag entered.

This allows you to test returned values in the status parameter for both variants of identification and for validation.

Create all three test accounts at https://mojeid.regtest.nic.cz/registration/. You can enter any contact information. PIN1, PIN2 and the verification letter with PIN3 are not sent; enter the following universal PINs instead:

- PIN1: `11111111` (8 times “1”),
- PIN2: `22222222` (8 times “2”),
- PIN3: `33333333` (8 times “3”).

To validate an account, it is necessary to generate the *Validation Request* document (PDF) from the corresponding user’s profile.[33]. To generate the document, it is necessary to enter a date of birth. Send the generated PDF document to techsupport@mojeid.cz. Then, we will set the validation flag on the corresponding profile.

[32] https://www.mojeid.cz/en/jak-na-to/identifikace-aneb-piny-validace/
[33] https://mojeid.regtest.nic.cz/editor/

8.2 Mutual Endpoints

Part of the interface addresses does not depend on the selected protocol. Those addresses are listed here. However, you will also need addresses of endpoints specific for individual protocols that are listed below.

A test instance with more detailed outputs in case of errors is available at the following addresses:

- Registering a new mojeID account: `https://mojeid.regtest.nic.cz/registration/endpoint/`
- Transferring a contact to mojeID from the domain registry: `https://mojeid.regtest.nic.cz/transfer/endpoint/`

The following addresses will be available to implement mojeID to production environment:

- Registering a new mojeID account: `https://mojeid.cz/registration/endpoint/`
- Transferring a contact to mojeID from the domain registry: `https://mojeid.cz/transfer/endpoint/`

8.3 OpenID Connect

Metadata that need to be sent to support

- `Client_ID` you will use for testing – a combination of 12 characters (lower- and uppercase letters and digits) generated automatically upon the registration of the service

Specific endpoints for the protocol

- **Addresses of the test endpoints:**
  - Registration Endpoint: `https://mojeid.regtest.nic.cz/oidc/registration/`
  - Authorization Endpoint: `https://mojeid.regtest.nic.cz/oidc/authorization/`
  - Token Endpoint: `https://mojeid.regtest.nic.cz/oidc/token/`
  - UserInfo Endpoint: `https://mojeid.regtest.nic.cz/oidc/userinfo/`

  A full description of OIDC configuration in JSON: https://mojeid.regtest.nic.cz/.well-known/openid-configuration/
- **Addresses of the production endpoints:**
  - Registration Endpoint: `https://mojeid.cz/oidc/registration/`
  - Authorization Endpoint: `https://mojeid.cz/oidc/authorization/`
  - Token Endpoint: `https://mojeid.cz/oidc/token/`
  - UserInfo Endpoint: `https://mojeid.cz/oidc/userinfo/`

  A full description of OIDC configuration in JSON: https://mojeid.cz/.well-known/openid-configuration/

8.4 OpenID 2.0

Metadata that need to be sent to support

- the *realm* you will use for testing (URL), see *Choosing a Suitable Realm* (page 41)

Specific endpoints for the protocol

- test endpoint: `https://mojeid.regtest.nic.cz/endpoint/`
- production endpoint: `https://mojeid.cz/endpoint/`

8.5 SAML

The metadata of the test instance are available at: https://mojeid.regtest.nic.cz/saml/idp.xml

Metadata that need to be sent to support

- string `entityID` you will use for testing – maximal length 1024 characters, specifications recommend the string to be in a form of URL[34] and to include a domain name of the provider or the provided service

  Example: `https://sluzba.example.cz`

- an XML file with the service metadata (`EntityDescriptor`), that contains the same `entityID`

  You can find more details on how to get the file with metadata in this article about metadata preparation[35].

Endpoints specific for the protocol

- test endpoint: `https://mojeid.regtest.nic.cz/saml/`
- production endpoint: `https://mojeid.cz/saml/`

[34] https://en.wikipedia.org/wiki/URL#Syntax
[35] https://www.eduid.cz/en/tech/metadata-preparation

Chapter 9

Appendices

List od Appendices

9.1 Appendix 1 – List of Data to be Handed Over (OpenID Connect)

Data	*Claim* identifier	Data type
OpenID2 identifier for migration from an older protocol	openid2_id	*SINGLE_OPTIONAL_STRING*
Name		
Whole name	name	*SINGLE_OPTIONAL_STRING*
First name	given_name	*SINGLE_OPTIONAL_STRING*
Surname	family_name	*SINGLE_OPTIONAL_STRING*
Nickname	nickname	*SINGLE_OPTIONAL_STRING*
Email		
Main	email	*SINGLE_OPTIONAL_STRING*
Flag – email verified	email_verified	*SINGLE_OPTIONAL_BOOLEAN*
Notify	mojeid_email_notify	*SINGLE_OPTIONAL_STRING*
Other	mojeid_email_next	*SINGLE_OPTIONAL_STRING*
Home address		
Full address	mojeid_address_def	*OPTIONAL_ADDRESS_STRING*
Street	mojeid_address_def_street	*SINGLE_OPTIONAL_STRING*
Street 2	mojeid_address_def_street2	*SINGLE_OPTIONAL_STRING*
Street 3	mojeid_address_def_street3	*SINGLE_OPTIONAL_STRING*
City	mojeid_address_def_city	*SINGLE_OPTIONAL_STRING*
State	mojeid_address_def_state	*SINGLE_OPTIONAL_STRING*
ZIP code	mojeid_address_def_postal_code	*SINGLE_OPTIONAL_STRING*
Country	mojeid_address_def_country	*SINGLE_OPTIONAL_STRING*
Mailing address		
Full address	address	*OPTIONAL_ADDRESS*
Street	mojeid_address_mail_street	*SINGLE_OPTIONAL_STRING*
Street 2	mojeid_address_mail_street2	*SINGLE_OPTIONAL_STRING*
Street 3	mojeid_address_mail_street3	*SINGLE_OPTIONAL_STRING*
City	mojeid_address_mail_city	*SINGLE_OPTIONAL_STRING*
State	mojeid_address_mail_state	*SINGLE_OPTIONAL_STRING*
ZIP code	mojeid_address_mail_postal_code	*SINGLE_OPTIONAL_STRING*
Country	mojeid_address_mail_country	*SINGLE_OPTIONAL_STRING*
Flag – address verified	mojeid_address_mail_verified	*SINGLE_OPTIONAL_BOOLEAN*
Billing address		
Full address	mojeid_address_bill	*OPTIONAL_ADDRESS_STRING*
Street	mojeid_address_bill_street	*SINGLE_OPTIONAL_STRING*
Street 2	mojeid_address_bill_street2	*SINGLE_OPTIONAL_STRING*
Street 3	mojeid_address_bill_street3	*SINGLE_OPTIONAL_STRING*
City	mojeid_address_bill_city	*SINGLE_OPTIONAL_STRING*

Continued on next page

Table 1 – continued from previous page

Data	*Claim* identifier	Data type
State	mojeid_address_bill_state	*SINGLE_OPTIONAL_STRING*
ZIP code	mojeid_address_bill_postal_code	*SINGLE_OPTIONAL_STRING*
Country	mojeid_address_bill_country	*SINGLE_OPTIONAL_STRING*
Shipping address		
Full address	mojeid_address_ship	*OPTIONAL_ADDRESS_STRING*
Company name	mojeid_address_ship_company_name	*SINGLE_OPTIONAL_STRING*
Street	mojeid_address_ship_street	*SINGLE_OPTIONAL_STRING*
Street 2	mojeid_address_ship_street2	*SINGLE_OPTIONAL_STRING*
Street 3	mojeid_address_ship_street3	*SINGLE_OPTIONAL_STRING*
City	mojeid_address_ship_city	*SINGLE_OPTIONAL_STRING*
State	mojeid_address_ship_state	*SINGLE_OPTIONAL_STRING*
ZIP code	mojeid_address_ship_postal_code	*SINGLE_OPTIONAL_STRING*
Country	mojeid_address_ship_country	*SINGLE_OPTIONAL_STRING*
Phone		
Mobile	phone_number	*SINGLE_OPTIONAL_STRING*
Flag – mobile verified	phone_number_verified	*SINGLE_OPTIONAL_BOOLEAN*
Other	mojeid_phone_mobile	*SINGLE_OPTIONAL_STRING*
Home	mojeid_phone_home	*SINGLE_OPTIONAL_STRING*
Work	mojeid_phone_office	*SINGLE_OPTIONAL_STRING*
Fax	mojeid_phone_fax	*SINGLE_OPTIONAL_STRING*
Other data		
Date of birth	birthdate	*SINGLE_OPTIONAL_STRING*
Gender	gender	*SINGLE_OPTIONAL_STRING*
Age	mojeid_age	*SINGLE_OPTIONAL_INT*
ID number	mojeid_ident_card	*SINGLE_OPTIONAL_STRING*
Passport number	mojeid_ident_pass	*SINGLE_OPTIONAL_STRING*
MPSV identifier	mojeid_ident_ssn	*SINGLE_OPTIONAL_STRING*
ISIC card number *Only for full access*	mojeid_isic	*SINGLE_OPTIONAL_STRING*
Flag – older than 18	mojeid_is_adult	*SINGLE_OPTIONAL_BOOLEAN*
Flag – student *Only for full access*	mojeid_student	*SINGLE_OPTIONAL_BOOLEAN*
Flag – validation *Only for full access*	mojeid_valid	*SINGLE_OPTIONAL_BOOLEAN*
VAT (IČO)	mojeid_vat	*SINGLE_OPTIONAL_STRING*
VAT (DIČ)	mojeid_ident_vat	*SINGLE_OPTIONAL_STRING*
Public PGP key	mojeid_public_pgp	*SINGLE_OPTIONAL_STRING*
Bank account	mojeid_bank_account	*SINGLE_OPTIONAL_STRING*
Bank account (IBAN)	mojeid_bank_account_iban	*SINGLE_OPTIONAL_STRING*
Data box	mojeid_isds	*SINGLE_OPTIONAL_STRING*

Continued on next page

Table 1 – continued from previous page

Data	*Claim* identifier	Data type
URL		
Main	profile	*SINGLE_OPTIONAL_STRING*
Personal	website	*SINGLE_OPTIONAL_STRING*
Blog	mojeid_url_blog	*SINGLE_OPTIONAL_STRING*
Work	mojeid_url_office	*SINGLE_OPTIONAL_STRING*
RSS	mojeid_url_rss	*SINGLE_OPTIONAL_STRING*
Facebook	mojeid_url_facebook	*SINGLE_OPTIONAL_STRING*
Twitter	mojeid_url_twitter	*SINGLE_OPTIONAL_STRING*
LinkedIN	mojeid_url_linkedin	*SINGLE_OPTIONAL_STRING*
instagram	mojeid_url_instagram	*SINGLE_OPTIONAL_STRING*
pinterest	mojeid_url_pinterest	*SINGLE_OPTIONAL_STRING*
tumblr	mojeid_url_tumblr	*SINGLE_OPTIONAL_STRING*
wordpress	mojeid_url_wordpress	*SINGLE_OPTIONAL_STRING*
foursquare	mojeid_url_foursquare	*SINGLE_OPTIONAL_STRING*
youtube	mojeid_url_youtube	*SINGLE_OPTIONAL_STRING*
blogger	mojeid_url_blogger	*SINGLE_OPTIONAL_STRING*
gravatar	mojeid_url_gravatar	*SINGLE_OPTIONAL_STRING*
about_me	mojeid_url_about_me	*SINGLE_OPTIONAL_STRING*
Flickr	mojeid_url_flickr	*SINGLE_OPTIONAL_STRING*
Vimeo	mojeid_url_vimeo	*SINGLE_OPTIONAL_STRING*
IM		
ICQ	mojeid_im_icq	*SINGLE_OPTIONAL_STRING*
Skype	mojeid_im_skype	*SINGLE_OPTIONAL_STRING*
Jabber	mojeid_im_jabber	*SINGLE_OPTIONAL_STRING*
Hangouts	mojeid_im_google_talk	*SINGLE_OPTIONAL_STRING*
Windows Live	mojeid_im_windows_live	*SINGLE_OPTIONAL_STRING*

SINGLE_OPTIONAL_BOOLEAN Boolean or *null*

SINGLE_OPTIONAL_INT Whole number or *null*

SINGLE_OPTIONAL_STRING String or *null*

OPTIONAL_ADDRESS Object or *null*

Listing 1: OPTIONAL_ADDRESS object schema

```
{
   "formatted": SINGLE_OPTIONAL_STRING,
   "street_address": SINGLE_OPTIONAL_STRING,
   "locality": SINGLE_OPTIONAL_STRING,
   "region": SINGLE_OPTIONAL_STRING,
   "postal_code": SINGLE_OPTIONAL_STRING,
   "country": SINGLE_OPTIONAL_STRING,
}
```

OPTIONAL_ADDRESS_STRING String or *null*; string contains a serialized object *OPTIONAL_ADDRESS*, e.g. `"{\"formatted\": \"Sunny 5, Prague\"}"`.

9.2 Appendix 2 – List of Data to be Handed Over (OpenID 2.0)

Note: All the values are handed over as strings.

For **flags** (see table) you can expect values `"true"` or `"false"` **with different capitalization**.

If a piece of data is not entered, no value is handed over and only the data's key is returned.

Data	AX identifier	SReg identifier
Name		
Whole name	http://axschema.org/namePerson	fullname
	http://specs.nic.cz/attr/contact/name	
First name	http://axschema.org/namePerson/first	
	http://specs.nic.cz/attr/contact/name/first	
Surname	http://axschema.org/namePerson/last	
	http://specs.nic.cz/attr/contact/name/last	
Nickname	http://axschema.org/namePerson/friendly	nickname
	http://specs.nic.cz/attr/contact/nickname	
Email		
Main	http://axschema.org/contact/email	e-mail
	http://specs.nic.cz/attr/email/main	
Notify	http://specs.nic.cz/attr/email/notify	
Other	http://specs.nic.cz/attr/email/next	
Home address		
Street	http://axschema.org/contact/postalAddress/home	
	http://specs.nic.cz/attr/addr/main/street	
Street2	http://axschema.org/contact/postalAddressAdditional/home	
	http://specs.nic.cz/attr/addr/main/street2	

Continued on next page

Table 2 – continued from previous page

Data	AX identifier	SReg identifier
Street3	http://specs.nic.cz/attr/addr/main/street3	
City	http://axschema.org/contact/city/home	
	http://specs.nic.cz/attr/addr/main/city	
State	http://axschema.org/contact/state/home	
	http://specs.nic.cz/attr/addr/main/sp	
Country	http://axschema.org/contact/country/home	
	http://specs.nic.cz/attr/addr/main/cc	
ZIP code	http://axschema.org/contact/postalCode/home	
	http://specs.nic.cz/attr/addr/main/pc	
Mailing address		
Street	http://specs.nic.cz/attr/addr/mail/street	
Street2	http://specs.nic.cz/attr/addr/mail/street2	
Street3	http://specs.nic.cz/attr/addr/mail/street3	
City	http://specs.nic.cz/attr/addr/mail/city	
State	http://specs.nic.cz/attr/addr/mail/sp	
Country	http://specs.nic.cz/attr/addr/mail/cc	
ZIP code	http://specs.nic.cz/attr/addr/mail/pc	
Flag – address verified (`"true"` / `"false"`)	http://specs.nic.cz/attr/addr/mail/verified	
Billing address		
Street	http://axschema.org/x/contact/postalAddress/billing	
	http://specs.nic.cz/attr/addr/bill/street	
Street2	http://axschema.org/x/contact/postalAddressAdditional/billing	
	http://specs.nic.cz/attr/addr/bill/street2	
Street3	http://specs.nic.cz/attr/addr/bill/street3	
City	http://axschema.org/x/contact/city/billing	
	http://specs.nic.cz/attr/addr/bill/city	

Continued on next page

Table 2 – continued from previous page

Data	AX identifier	SReg identifier
State	http://axschema.org/x/contact/state/billing	
	http://specs.nic.cz/attr/addr/bill/sp	
Country	http://axschema.org/x/contact/country/billing	
	http://specs.nic.cz/attr/addr/bill/cc	
ZIP code	http://axschema.org/x/contact/postalCode/billing	
	http://specs.nic.cz/attr/addr/bill/pc	
Shipping address		
Company	http://specs.nic.cz/attr/addr/ship/company_name	
Street	http://axschema.org/x/contact/postalAddress/shipping	
	http://specs.nic.cz/attr/addr/ship/street	
Street2	http://axschema.org/x/contact/postalAddressAdditional/shipping	
	http://specs.nic.cz/attr/addr/ship/street2	
Street3	http://specs.nic.cz/attr/addr/ship/street3	
City	http://axschema.org/x/contact/city/shipping	
	http://specs.nic.cz/attr/addr/ship/city	
State	http://axschema.org/x/contact/state/shipping	
	http://specs.nic.cz/attr/addr/ship/sp	
Country	http://axschema.org/x/contact/country/shipping	
	http://specs.nic.cz/attr/addr/ship/cc	
ZIP code	http://axschema.org/x/contact/postalCode/shipping	
	http://specs.nic.cz/attr/addr/ship/pc	
Phone		
Mobile	http://axschema.org/contact/phone/default	
	http://specs.nic.cz/attr/phone/main	
Other	http://axschema.org/contact/phone/cell	
	http://specs.nic.cz/attr/phone/mobile	
Home	http://axschema.org/contact/phone/home	

Continued on next page

Table 2 – continued from previous page

Data	AX identifier	SReg identifier
	http://specs.nic.cz/attr/phone/home	
Work	http://axschema.org/contact/phone/business	
	http://specs.nic.cz/attr/phone/work	
Fax	http://axschema.org/contact/phone/fax	
	http://specs.nic.cz/attr/phone/fax	
Other data		
Date of birth	http://axschema.org/birthDate	dob
	http://specs.nic.cz/attr/contact/ident/dob	
Age	http://specs.nic.cz/attr/contact/age	
Gender	http://axschema.org/person/gender	gender
	http://specs.nic.cz/attr/contact/gender	
ID number	http://specs.nic.cz/attr/contact/ident/card	
Passport number	http://specs.nic.cz/attr/contact/ident/pass	
MPSV identifier	http://specs.nic.cz/attr/contact/ident/ssn	
ISIC card number *Only for full access*	http://specs.nic.cz/attr/contact/isic	
Flag – older than 18 (`"true"` / `"false"`)	http://specs.nic.cz/attr/contact/adult	
Flag – student *Only for full access* (`"true"` / `"false"`)	http://specs.nic.cz/attr/contact/student	
Flag – validation *Only for full access* (`"true"` / `"false"`)	http://specs.nic.cz/attr/contact/valid	
Account status *Only for full access*	http://specs.nic.cz/attr/contact/status	
Image (base64)	http://specs.nic.cz/attr/contact/image	
Company name	http://axschema.org/company/name	

Continued on next page

Table 2 – continued from previous page

Data	AX identifier	SReg identifier
	http://specs.nic.cz/attr/contact/org	
VAT (IČO)	http://specs.nic.cz/attr/contact/ident/vat_id	
VAT (DIČ)	http://specs.nic.cz/attr/contact/vat	
Public PGP key	http://specs.nic.cz/attr/public_pgp	
Bank account	http://specs.nic.cz/attr/bank/national	
Bankovní account (IBAN)	http://specs.nic.cz/attr/bank/iban	
Data box	http://specs.nic.cz/attr/contact/isds	
Internet addresses		
Main	http://axschema.org/contact/web/default	
	http://specs.nic.cz/attr/url/main	
Blog	http://axschema.org/contact/web/blog	
	http://specs.nic.cz/attr/url/blog	
Personal	http://specs.nic.cz/attr/url/personal	
Work	http://specs.nic.cz/attr/url/work	
RSS	http://specs.nic.cz/attr/url/rss	
Facebook	http://specs.nic.cz/attr/url/facebook	
Twitter	http://specs.nic.cz/attr/url/twitter	
LinkedIN	http://specs.nic.cz/attr/url/linkedin	
Instagram	http://specs.nic.cz/attr/url/instagram	
pinterest	http://specs.nic.cz/attr/url/pinterest	
tumblr	http://specs.nic.cz/attr/url/tumblr	
wordpress	http://specs.nic.cz/attr/url/wordpress	
foursquare	http://specs.nic.cz/attr/url/foursquare	
youtube	http://specs.nic.cz/attr/url/youtube	
blogger	http://specs.nic.cz/attr/url/blogger	
gravatar	http://specs.nic.cz/attr/url/gravatar	
about_me	http://specs.nic.cz/attr/url/about_me	
Flickr	http://specs.nic.cz/attr/url/flickr	

Continued on next page

Table 2 – continued from previous page

Data	AX identifier	SReg identifier
Vimeo	http://specs.nic.cz/attr/url/vimeo	
Instant Messaging		
ICQ	http://axschema.org/contact/IM/ICQ	
	http://specs.nic.cz/attr/im/icq	
Skype	http://axschema.org/contact/IM/Skype	
	http://specs.nic.cz/attr/im/skype	
Jabber	http://axschema.org/contact/IM/Jabber	
	http://specs.nic.cz/attr/im/jabber	
Hangouts	http://specs.nic.cz/attr/im/google_talk	
Windows Live	http://specs.nic.cz/attr/im/windows_live	

9.3 Appendix 3 – List of Data to be Handed Over (SAML)

Table 3: General identifiers

Data	Identifier (URI format)	Identifier (BASIC format)
Name		
Whole name	urn:oid:2.5.4.3	urn:mace:dir:attribute-def:cn
First name	urn:oid:2.5.4.42	urn:mace:dir:attribute-def:givenName
Surname	urn:oid:2.5.4.4	urn:mace:dir:attribute-def:sn
Nickname	urn:oid:2.5.4.65	urn:mace:dir:attribute-def:pseudonym
Email		
Main	urn:oid:0.9.2342.19200300.100.1.3	urn:mace:dir:attribute-def:mail
Home address		
Full address	urn:oid:2.5.4.16	urn:mace:dir:attribute-def:postalAddress
Street	urn:oid:2.5.4.9	urn:mace:dir:attribute-def:street
City	urn:oid:2.5.4.7	urn:mace:dir:attribute-def:l
State	urn:oid:2.5.4.8	urn:mace:dir:attribute-def:st
Country	urn:oid:2.5.4.6	urn:mace:dir:attribute-def:c
ZIP code	urn:oid:2.5.4.17	urn:mace:dir:attribute-def:postalCode
Phone		
Mobile	urn:oid:2.5.4.20	urn:mace:dir:attribute-def:telephoneNumber
Fax	urn:oid:2.5.4.23	urn:mace:dir:attribute-def:facsimileTelephoneNumber
Other data		
Date of birth	urn:oid:1.3.6.1.4.1.2428.90.1.3	urn:mace:dir:attribute-def:norEduPersonBirthDate
Age	http://www.stork.gov.eu/1.0/age	
Gender	urn:oid:1.3.6.1.4.1.25178.1.2.2	
Image (base64)		urn:mace:dir:attribute-def:photo

Continued on next page

Table 3 – continued from previous page

Data	Identifier (URI format)	Identifier (BASIC format)
Company name	urn:oid:2.5.4.10	urn:mace:dir:attribute-def:o
URL		
Main	urn:oid:1.3.6.1.4.1.27630.2.1.1.17	
Work	urn:oid:1.3.6.1.4.1.27630.2.1.1.120	

Table 4: eduID identifiers

Data	Identifier (URI formát)
eduID	
eduPersonPrincipalName	urn:oid:1.3.6.1.4.1.5923.1.1.1.6
eduPersonScopedAffiliation	urn:oid:1.3.6.1.4.1.5923.1.1.1.9
eduPersonTargetedID	urn:oid:1.3.6.1.4.1.5923.1.1.1.10
eduPersonUniqueId	urn:oid:1.3.6.1.4.1.5923.1.1.1.13

9.4 Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)

Table 5: specs.nic.cz identifiers

Date	Identifier
Name	
Whole name	http://specs.nic.cz/attr/contact/name
First name	http://specs.nic.cz/attr/contact/name/first
Surname	http://specs.nic.cz/attr/contact/name/last
Nickname	http://specs.nic.cz/attr/contact/nickname
Email	
Main	http://specs.nic.cz/attr/email/main
Notify	http://specs.nic.cz/attr/email/notify
Other	http://specs.nic.cz/attr/email/next
Home address	
Street	http://specs.nic.cz/attr/addr/main/street
Street2	http://specs.nic.cz/attr/addr/main/street2
Street3	http://specs.nic.cz/attr/addr/main/street3
City	http://specs.nic.cz/attr/addr/main/city
State	http://specs.nic.cz/attr/addr/main/sp
Country	http://specs.nic.cz/attr/addr/main/cc
ZIP code	http://specs.nic.cz/attr/addr/main/pc
Mailing address	
Street	http://specs.nic.cz/attr/addr/mail/street
Street2	http://specs.nic.cz/attr/addr/mail/street2
Street3	http://specs.nic.cz/attr/addr/mail/street3
City	http://specs.nic.cz/attr/addr/mail/city
State	http://specs.nic.cz/attr/addr/mail/sp
Country	http://specs.nic.cz/attr/addr/mail/cc
ZIP code	http://specs.nic.cz/attr/addr/mail/pc
Flag – address verified	http://specs.nic.cz/attr/addr/mail/verified
Billing address	
Street	http://specs.nic.cz/attr/addr/bill/street
Street2	http://specs.nic.cz/attr/addr/bill/street2
Street3	http://specs.nic.cz/attr/addr/bill/street3
City	http://specs.nic.cz/attr/addr/bill/city
State	http://specs.nic.cz/attr/addr/bill/sp
Country	http://specs.nic.cz/attr/addr/bill/cc
ZIP code	http://specs.nic.cz/attr/addr/bill/pc
Shipping address	
Company	http://specs.nic.cz/attr/addr/ship/company_name
Street	http://specs.nic.cz/attr/addr/ship/street
Street2	http://specs.nic.cz/attr/addr/ship/street2
Street3	http://specs.nic.cz/attr/addr/ship/street3

Continued on next page

Table 5 – continued from previous page

Date	Identifier
City	http://specs.nic.cz/attr/addr/ship/city
State	http://specs.nic.cz/attr/addr/ship/sp
Country	http://specs.nic.cz/attr/addr/ship/cc
ZIP code	http://specs.nic.cz/attr/addr/ship/pc
Phone	
Mobile	http://specs.nic.cz/attr/phone/main
Other	http://specs.nic.cz/attr/phone/mobile
Home	http://specs.nic.cz/attr/phone/home
Work	http://specs.nic.cz/attr/phone/work
Fax	http://specs.nic.cz/attr/phone/fax
Other data	
Date of birth	http://specs.nic.cz/attr/contact/ident/dob
Age	http://specs.nic.cz/attr/contact/age
Gender	http://specs.nic.cz/attr/contact/gender
ID number	http://specs.nic.cz/attr/contact/ident/card
Passport number	http://specs.nic.cz/attr/contact/ident/pass
MPSV identifier	http://specs.nic.cz/attr/contact/ident/ssn
ISIC card number *Only for full access*	http://specs.nic.cz/attr/contact/isic
Flag – older than 18	http://specs.nic.cz/attr/contact/adult
Flag – student *Only for full access*	http://specs.nic.cz/attr/contact/student
Flag – validation *Only for full access*	http://specs.nic.cz/attr/contact/valid
Account status *Only for full access*	http://specs.nic.cz/attr/contact/status
Image (base64)	http://specs.nic.cz/attr/contact/image
Company name	http://specs.nic.cz/attr/contact/org
VAT (IČO)	http://specs.nic.cz/attr/contact/ident/vat_id
VAT (DIČ)	http://specs.nic.cz/attr/contact/vat
Public PGP key	http://specs.nic.cz/attr/public_pgp
Bank account	http://specs.nic.cz/attr/bank/national
Bank account (IBAN)	http://specs.nic.cz/attr/bank/iban
Data box	http://specs.nic.cz/attr/contact/isds
Internet addresses	
Main	http://specs.nic.cz/attr/url/main
Blog	http://specs.nic.cz/attr/url/blog
Personal	http://specs.nic.cz/attr/url/personal
Work	http://specs.nic.cz/attr/url/work
RSS	http://specs.nic.cz/attr/url/rss
Facebook	http://specs.nic.cz/attr/url/facebook
Twitter	http://specs.nic.cz/attr/url/twitter
LinkedIN	http://specs.nic.cz/attr/url/linkedin
instagram	http://specs.nic.cz/attr/url/instagram

Continued on next page

Table 5 – continued from previous page

Date	Identifier
pinterest	http://specs.nic.cz/attr/url/pinterest
tumblr	http://specs.nic.cz/attr/url/tumblr
wordpress	http://specs.nic.cz/attr/url/wordpress
foursquare	http://specs.nic.cz/attr/url/foursquare
youtube	http://specs.nic.cz/attr/url/youtube
blogger	http://specs.nic.cz/attr/url/blogger
gravatar	http://specs.nic.cz/attr/url/gravatar
about_me	http://specs.nic.cz/attr/url/about_me
Flickr	http://specs.nic.cz/attr/url/flickr
Vimeo	http://specs.nic.cz/attr/url/vimeo
Instant Messaging	
ICQ	http://specs.nic.cz/attr/im/icq
Skype	http://specs.nic.cz/attr/im/skype
Jabber	http://specs.nic.cz/attr/im/jabber
Hangouts	http://specs.nic.cz/attr/im/google_talk
Windows Live	http://specs.nic.cz/attr/im/windows_live

9.5 Appendix 5 – List of Data for Registration

Data	Format	Registration
Name		
First name	string (max 50 characters)	first_name
Surname	string (max 50 characters)	last_name
Email		
Main	email address (max 200 characters) *STD-EMAIL*	email__default__email
Notification	email address (max 200 characters) *STD-EMAIL*	email__notify__email
Other	email address (max 200 characters) *STD-EMAIL*	email__next__email
Home address		
Street	string (max 200 characters)	address__default__street1
Street2	string (max 200 characters)	address__default__street2
Street3	string (max 200 characters)	address__default__street3
City	string (max 200 characters)	address__default__city
State	string (max 200 characters)	address__default__state
ZIP code	string (max 50 characters)	address__default__postal_code
Country	kód Country podle ISO3166 *STD-COUNTRY*	address__default__country
Billing address		
Street	string (max 200 characters)	address__billing__street1
Street2	string (max 200 characters)	address__billing__street2
Street3	string (max 200 characters)	address__billing__street3
City	string (max 200 characters)	address__billing__city
State	string (max 200 characters)	address__billing__state
ZIP code	string (max 50 characters)	address__billing__postal_code
Country	kód Country podle ISO3166 *STD-COUNTRY*	address__billing__country

Continued on next page

Table 6 – continued from previous page

Data	Format	Registration
Shipping address		
Company	string (max 200 characters)	address__shipping__company_name
Street	string (max 200 characters)	address__shipping__street1
Street2	string (max 200 characters)	address__shipping__street2
Street3	string (max 200 characters)	address__shipping__street3
City	string (max 200 characters)	address__shipping__city
State	string (max 200 characters)	address__shipping__state
ZIP code	string (max 50 characters)	address__shipping__postal_code
Country	kód Country podle ISO3166 *STD-COUNTRY*	address__shipping__country
Mailing address		
Street	string (max 200 characters)	address__mailing__street1
Street2	string (max 200 characters)	address__mailing__street2
Street3	string (max 200 characters)	address__mailing__street3
City	string (max 200 characters)	address__mailing__city
State	string (max 200 characters)	address__mailing__state
ZIP code	string (max 50 characters)	address__mailing__postal_code
Country	kód Country podle ISO3166 *STD-COUNTRY*	address__mailing__country
Phone		
Mobile	string that follows regular expression: ^+[0-9]{1,3}.[0-9]{1,14}$	phone__default__number
Work	string that follows regular expression: ^+[0-9]{1,3}.[0-9]{1,14}$	phone__office__number
Other	string that follows regular expression: ^+[0-9]{1,3}.[0-9]{1,14}$	phone__mobile__number
Home	string that follows regular expression: ^+[0-9]{1,3}.[0-9]{1,14}$	phone__home__number

Continued on next page

Table 6 – continued from previous page

Data	Format	Registration
Phone - Fax	string that follows regular expression: ^+[0-9]{1,3}.[0-9]{1,14}$	phone__fax__number
Other data		
Date of birth	date in the RFC3339 format (YYYY-MM-DD) *STD-DATE*	birth_date
Gender	Value “M” or “F”	gender
ID number	string (max 50 characters)	id_card_num
Passport number	string (max 50 characters)	passport_num
MPSV identifier	string (max 50 characters)	ssn_id_num
ISIC card number	string (max 50 characters)	card_isic
Company name	string (max 200 characters)	organization
VAT (IČO)	string (max 50 characters)	vat_id_num
VAT (DIČ)	string (max 50 characters)	vat_reg_num
Internet addresses		
Main	string (max 255 characters)	urladdress__main__url
Blog	string (max 255 characters)	urladdress__blog__url
Personal	string (max 255 characters)	urladdress__personal__url
Work	string (max 255 characters)	urladdress__office__url
RSS	string (max 255 characters)	urladdress__rss__url
Facebook	string (max 255 characters)	urladdress__facebook__url
Twitter	string (max 255 characters)	urladdress__twitter__url
LinkedIN	string (max 255 characters)	urladdress__linkedin__url
instagram	string (max 255 characters)	urladdress__instagram__url
pinterest	string (max 255 characters)	urladdress__pinterest__url
tumblr	string (max 255 characters)	urladdress__tumblr__url
wordpress	string (max 255 characters)	urladdress__wordpress__url
foursquare	string (max 255 characters)	urladdress__foursquare__url

Continued on next page

Table 6 – continued from previous page

Data	Format	Registration
youtube	string (max 255 characters)	urladdress__youtube__url
blogger	string (max 255 characters)	urladdress__blogger__url
gravatar	string (max 255 characters)	urladdress__gravatar__url
about_me	string (max 255 characters)	urladdress__about_me__url
Instant Messaging		
ICQ	string (max 255 characters)	imaccount__icq__username
Skype	string (max 255 characters)	imaccount__skype__username
Windows Live	string (max 255 characters)	imaccount__windows_live__username
Jabber	string (max 255 characters)	imaccount__jabber__username
Hangouts	string (max 255 characters)	imaccount__google_talk__username

STD-EMAIL Email address in a format that follows **RFC 2822**[36]

STD-COUNTRY Country code as per ISO 3166[37]

STD-DATE Date in a format that follows **RFC 3339**[38]

[36] https://tools.ietf.org/html/rfc2822.html
[37] https://www.iso.org/iso-3166-country-codes.html
[38] https://tools.ietf.org/html/rfc3339.html

9.6 Appendix 6 – Examples and Solution of Error Messages

The following article describes the most common error messages that can occur during the implementation of mojeID. The text also provides recommendations on how to solve the issues or what to focus on.

9.6.1 Error Messages on Test Instance

The errors are rendered directly form the used libraries. The most important ones are described here:

- *"Error parsing document as XML"* and *"Not a XRDS document"* – Both mean there is an invalid XRDS document. This message usually describes a problem in an XRDS document with an invalid XML code (usually because it contains nonstandard unicode characters). It is possible to check the source code at http://www.xmlvalidation.com and find out where the error is.
- *"No XRD present in tree"* – the XRDS document has no XRD element. Check the contents of the XRDS document (see the *XRDS Document and its Format* (page 40)). Mind also the case of the letters inside tags!
- *"HTTP Response status from identity URL host is not 200. Got status XXX"* – a query for *realm* or an XRDS document returned a different HTTP status code than 200.
- Errors from cURL are in form of „(XX, . . .)", where XX is the number of the error from the list of libcURL errors (see https://curl.haxx.se/libcurl/c/libcurl-errors.html)

9.6.2 Problems Verifying the Return Address

When the verification of the service's return address fails, the user is shown one of the following messages based on the phase in which the negative outcome occured:

a. **If the connection with a service failed**

   "Nelze ověřit důvěryhodnost služby, kam se přihlašujete přes mojeID. Buďte zvláště obezřetní při předávání údajů z mojeID této službě."
   "We can not validate authenticity of the service where you want to login with mojeID. Use extra caution when handing over the data from mojeID."

   This message is displayed when a query for realm or an XRDS document returns a HTTP status code 4xx or 5xx. If that is not the case, the message can describe a certificate issue when using HTTPS.

   For HTTPS to work correctly, it is necessary to have a valid certificate that you can get from a certification authority (see also *Problems with Unencrypted Connection* (page 80)). You also need a so-called *intermediate* certificates, so that the XRDS document is searched for. The server certificate has to be set up correctly, e.g. on an Apache server, *intermediate* certificates are set up using the SSLCertificateChainFile directive or SSLCertificateFile (see documentation for setting up SSL in Apache[39].

   A list of certification authorities supported by mojeID can be found at https://wiki.mozilla.org/CA/Included_Certificates

[39] https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#sslcertificatechainfile

When troubleshooting issues with SSL and certificates, you can use direct tools, such as `wget` or `curl` programs, or a mechanism of a used library, that can detect issues better than common browsers.

b. **If the connection with a service was successful, but the validation of the return address failed**

"Tento požadavek na přihlášení přes mojeID o sobě tvrdí, že přichází z jiné stránky, než tomu ve skutečnosti je. Zvažte, zda vůbec chcete pokračovat s předáváním údajů z vašeho mojeID."
"This mojeID login request claims to be from other site than it really is. Consider carefully whether you want to continue with handing over the data from your mojeID."

Return address verification can fail due to the following reasons:

- *Realm* did not return HTTP status 200.
- There is no XRDS document on the *realm*, therefore the service cannot be verified. The XRDS document has to be placed on the *realm* in one of the three following ways:
  - The XRDS document can be placed directly in the HTTP header,
  - the XRDS document can be saved directly at the address of the *realm* (sent directly in the response),
  - the location can be described in the HTML header in a META tag.
- A redirection occurred during the downloading of the XRDS document.
- When the address `return_to` in an OpenID request does not match the address `return_to` in the XRDS document. The `return_to` address from an OpenID request can contain only several additional parameters, the so-called query string, not a subdirectory in a path.
- When the address `return_to` in an OpenID request "is not an extension" of the address of the *realm*.

The term address A "is an extension" of address B means that:

- the protocol is the same.
- the domain is the same or also contains a subdomain if the domain B starts with *.,
- the port is the same,
- the path is the same or contains a subdirectory, and
- query string (?key=value&key2=value2) is the same or with additional parameters.

Table 7: Examples: the address A "is an extension" of the address B

Claim validity	Address A	Address B
yes	https://example.com/hello/	https://example.com/hello/
no	http://example.com/hello/	https://example.com/hello/
no	https://example.com:8080/hello/	https://example.com/hello/
yes	https://example.com/hello/hi/	https://example.com/hello/
no	https://example.com/hello/	https://example.com/hi/
no	https://example.com/hello/	https://example.com/hello/hi/
yes	https://example.com/hello/ ?key=value	https://example.com/hello/ ?key=value
yes	https://example.com/hello/ ?key=value&key2=value2	https://example.com/hello/ ?key=value
no	https://example.com/hello/ ?key=value	https://example.com/hello/ ?key=value&key2=value2
yes	https://subdomain.example.com/hello/ ?key=value	https://*.example.com/

c. **If it is not possible to manage the URL service's area in mojeID**

"Tento realm není dobře definovaný a nelze k němu nastavit důvěru."
"This realm is not sane and thus you can not set trust for it."

Check that your realm (described in the identity verificatin request) does not contain an IP address, characters not supported in URL, or a URI fragment[40]. See also *Choosing a Suitable Realm* (page 41).

9.6.3 Problems with Unencrypted Connection

Your browser might display the following message when redirecting back to your website:

"Informace, které jste zadali, budou odeslány přes nezašifrované spojení a mohly by jednoduše být přečteny třetí stranou. Určitě chcete pokračovat v odesílání?"
"The information you have entered will be sent over an unencrypted connection and could easily be read by a third party. Are you sure you want to continue sending it?"

Note: This message comes from Firefox and it will probably look a little different in other browsers.

This message can appear at all *realms* without HTTPS. The data that are handed over (i.e. user's personal information too) travel through the internet unencrypted and the browser says it leaves encrypted mojeID website towards a service that does not use encryption. We do not recommend the unencrypted protocol (HTTP), but it is possible to use it.

This issue can be solved easily by using a basic SSL certificate that can be downloaded here: http://www.startssl.com/, the *StartSSL Free* version for noncommercial services and the *StartSSL Verified* version for commercial services. This certificate secures your data transfer and at the same time, you see the level of authentication of the user.

[40] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Generic_syntax

9.6.4 Selecting Required Logging Method

The required login method is selected by placing the identifier of the given login method into the identity verification request. The mojeID service supports not only the common login by password, but also login by a digital certificate or a one-time password (OTP).

- When loging in **using a certificate**, the following message is displayed:

  "Poskytovatel služby požaduje přihlášení certifikátem."
  "The service provider wants you to login with your certificate."

- When logging in using a **one-time password** or an **authenticator**, the following message is displayed:

  "Poskytovatel služby požaduje přihlášení jednorázovým heslem nebo MojeID Autentikátorem."
  "The service provider wants you to login with one time password or MojeID Autentikátor."

- When logging in using a **security key**, the following message is displayed:

  "Poskytovatel služby požaduje přihlášení druhým faktorem."
  "The service provider wants you to login with two-factor authentication."

Method identifiers and an example of a request with requesting a login method can be found in the *Requesting Identity Authentication* (page 36) section.

9.6.5 Problems with Library for PHP

One of the most common error messages is *"FAILED TO CREATE AUTH REQUEST: not a valid OpenID"* and *"Ověření OpenID selhalo: No OpenID information"*.

Some errors might be caused by a wrong configuration of your server. You can try to fix them in the following way:

- You need to make sure that the cURL for the given PHP version is installed, active (phpinfo should say so) and that the cURL is not disabled in `php.ini`.
- It might also be necessary to check the `/etc/php5/conf.d/curl.ini` for a line `extension=curl.so` and add it if it is not there.
- Download and install the newest version of cURL, see also https://curl.haxx.se/download.html.

We also recommend downloading and getting familiar with *sample implementation in PHP* (page 33).

9.6.6 Error Messages in JSON (OIDC)

Error messages contain the error code in form of an ASCII string under the `error` key. A human readable error description should be found in a JSON response under the `error_description` key.

mojeID can return the following error codes:

Error Code	Possible Causes
`unauthorized_client`	Wromg `client_id`, wrong `client_secret`, incorrectly used authentication.
`invalid_request`	Missing required parameters, one or more parameters unreadable/unparseable.

9.7 Appendix 7 – Correct Implementation Procedure

When implementing the mojeID service, follow these best practices:

1. Logging into the mojeID service should be initiated only by a **Log in using mojeID** button, as described in the *Requesting Login via mojeID* (page 36) section.
2. There should be text links **Why mojeID** and **Create a mojeID account** next to or under the **Log in using mojeID** button.
    1. Direct the "Why mojeID" link to a local page explaining the benefits of using mojeID on your page (local benefits) or the information page https://www.mojeid.cz/en/why-mojeid/.
    2. The **Create a mojeID account** text link can be replaced by a **Create a mojeID account** button as per the example. Direct the button to a local mojeID registration page or to a universal mojeID registration form.
    3. If it is not possible to add links to the button as per the previous points 2.1 and 2.2, we recommend to add them to an administration page of the user's local account.
3. In case the **Log in using mojeID** button is not placed on your main page, place a **Supports mojeID** logo there as per the example, with a link to the place in your system where mojeID is used, or to the local page in your system that contains information on the mojeID service.
4. The data that are required to be handed over have to be in line with your system:
    1. Only the items that are required for the registration process in your system can be marked as required.
    2. The other items have to be marked as optional.
    3. You must not require the disclosure of items that you do not use in your system.
5. If you require the disclosure of the user's personal data during the login using mojeID, it is recommended (in case this data differs from the data stored in the local account of your service) to let the user decide whether they want to keep the existing data in the service's local account, or whether they should be updated by the data retreived from mojeID.
6. The implementation of the mojeID service needs to be designed in such way that the mojeID user can choose from the following two options when they first access your service using mojeID:
    1. link mojeID with an existing local account, or
    2. create a new local account using data retrieved from mojeID and link this newly created local account with mojeID.
7. In the user's local account administration:
    1. we recommend to display the user's mojeID identifier upon linking with the mojeID account.
    2. we recommend to show a link or a button "Create a mojeID account" as per the point n. 2. In case the user does not have their local account linked with mojeID, and therefore probably does not have a mojeID account, we recommend to prefill the mojeID registration form with the data from the user's local account.

3. the user needs to have an option to link mojeID with an existing local account, if it is not already linked.
4. the user needs to have an option to unlink the local account from mojeID.

8. Changes of the appearance of buttons and other graphical elements are possible only with an explicit consent from the CZ.NIC Association.

Chapter 10

Record of Changes

Version	Segment	Change description
3.0.2	Whole documentation	Fix minor typos and untranslated text
3.0.1	Whole documentation	Change the term “disclose” to “hand over” in the context of the data to be handed over Fix minor typos
3.0	Whole documentation	Added English version of the documentation
2.18	*Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58)	Added a missing piece of data that is handed over — `country` (OIDC)
	Requesting Identity Authentication (page 36) *Appendix 6 – Examples and Solution of Error Messages* (page 77)	Added a security key login method, including corresponding messages
	Communication via OpenID Connect (page 8) *Communication via OpenID 2.0* (page 13)	Added a login method — security key
2.17	*Client Registration* (page 22)	Added a part about manual registration of a client in mojeID test environment for OIDC
2.16	*Transition to a Different Protocol* (page 45)	Text of the Transition to a Different Protocol section changed, specific information about transition from OID2 to OIDC added
	Differences Between the Protocols (page 45)	Added spaces around a slash in the Implementation via OpenID Connect (OIDC) section
	Implementation via OpenID Connect (OIDC) (page 17)	Changed two capital letters for more consistency in the Identity Authentication Request
2.15	*Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58) *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62) *Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)* (page 70)	Fixed attribute For full access in the list of data to be handed over

Continued on next page

Table 1 – continued from previous page

Version	Segment	Change description
2.14	*Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58) *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62)	Described data types of the data to be handed over
2.13	*Legal Notice* (page 1)	Added legal notice regarding documentation
	Record of Changes	Reworked with the newest changes on top
2.12	*Client Registration* (page 22), *Requesting Data* (page 30)	Fixed invalid JSON examples
	Appendix 1 – List of Data to be Handed Over (OpenID Connect) (page 58) *Appendix 2 – List of Data to be Handed Over (OpenID 2.0)* (page 62) *Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)* (page 70) *Appendix 5 – List of Data for Registration* (page 73)	Deleted "Opencard number" and "google_plus" data to be handed over in all protocols
2.11	Everywhere	Fixed links to the new mojeID website
2.10	*Implementation Process Overview* (page 18)	Added implementation via OIDC process overview
	Overview of Libraries and Modules (page 18)	Added Overview of libraries and modules for OIDC
	Overview of Libraries and Modules (page 33)	Added Overview of libraries and modules for OID2
2.9	*Favicon* (page 14)	Favicon's purpose explanation and instruction for its setup
	Logging out of mojeID (page 51)	Instructions for unsubscription
	Appendix 6 – Examples and Solution of Error Messages (page 77)	Changed recommendation for SSL tuning
2.8	*mojeID Support Implementation* (page 17)	Important note on restricted usage of frameworks
2.7	*mojeID Test Instance* (page 53)	Updated addresses according to the new test server and explicitly listed all OIDC endpoints
	Everywhere	New technical support email address techsupport@mojeid.cz
2.6	Everywhere	Changed the order of protocols – OIDC is now the first one
	Performing Authentication (XRDS and realm) (page 39)	Changed chapter name
2.5	*Client Registration* (page 22)	Added the possibility of manual registration via OenID Connect via the new mojeID server interface

Continued on next page

Table 1 – continued from previous page

Version	Segment	Change description
2.4	*Interface for Creating mojeID Accounts* (page 47)	Added support of direct registration via OpenID Connect
2.3	*mojeID Test Instance* (page 53)	Added information for testing communication via OIDC and SAML
	Implementation via OpenID Connect (OIDC) (page 17)	Added examples of code and communication for implementation via OIDC
	Adjusting Communication with mojeID Server (page 46)	Added recommendation for adjusting communication
	Appendix 6 – Examples and Solution of Error Messages (page 77)	Added note on error responses in JSON for OIDC, replaced obsolete link
	Appendix 2 – List of Data to be Handed Over (OpenID 2.0) (page 62) *Appendix 1 – List of Data to be Handed Over (OpenID Connect)* (page 58) *Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)* (page 70)	Added a piece of data for disclosure – data box (ISDS)
2.2	*Appendix 4 – List of Data to be Handed Over (SAML specs.nic.cz)* (page 70)	Added list of other identificators for disclosure of data via SAML
2.1	*Basics of mojeID* (page 7)	Moved links to protocol specifications to *Implementation via OpenID 2.0* (page 33) and *Implementation via OpenID Connect (OIDC)* (page 17)
	Implementation via OpenID Connect (OIDC) (page 17)	Added link to configuration of OIDC on mojeID server
	Client Registration (page 22)	Added mention of client's metadata and extra information on manual registration
	mojeID LITE Library (page 31)	Added a whole segment
	Implementation via SAML (page 45)	Added a link to a certificate for verifying metadata and to a tool for dcoding SAML messages
	Problems with Implementation (page 45)	Added a whole segment
	Appendix 6 – Examples and Solution of Error Messages (page 77)	Added a link to a tool for testing SAML settings
	Record of Changes	Added a whole segment

Index