Table of Contents

Release Notes

Release Notes 1.2.6

1.1 New features

Identification Service 1.2.6 does not contain new features.

1.2 Handled issues

JIRA CODE	ISSUE CODE	DESCRIPTION
CIS-152	29592	"Keyset does not exist" errors have been resolved.

1.3 Known issues

- The **VirtualRootSslMaBackend** setting is marked as required by the configuration page even when no TLS Mutual Authentication is used. Current workaround is to copy the value of the **VirtualRoot** setting.

Note: A request without parameters doesn't generate an error message. This is intended behavior.

Release Notes 1.2.7

1.1 New features

1.1.1 Certificate delete option

The IDS console now offers a delete option to delete invalid or compromised certificates.

1.1.2 Encrypted JWT tokens

When creating a new client, the data read out from eID cards is now by default encrypted.

Attention: for existing clients that require encryption, the EncryptedTokens setting must be modified manually in the IDS Console.

1.1.3 Certificates are now always loaded into memory and stored in the database

When IDS is started, it now immediately loads the required certificates to memory, which results in a faster readout time for the first user.

In previous versions, IDS only started loading certificates when the first user entered their ID card.

Every 12 hours IDS checks if the certificates loaded into memory are still up to date and downloads their new version if necessary.

1.2 Improvements

1.2.1 Security Improvements

IDS 1.2.7 contains a number of security improvements.

1.3 Handled issues

JIRA CODE	ISSUE CODE	DESCRIPTION
CIS-184	/	Readout issue has been fixed

1.3 Known issues

- The **VirtualRootSslMaBackend** setting is marked as required by the configuration page even when no TLS Mutual Authentication is used. Current workaround is to copy the value of the **VirtualRoot** setting.

Note: A request without parameters doesn't generate an error message. This is intended behavior.

Release Notes 1.2.8

1.1 New features

IDS 1.2.8 is a hotfix version and doesn't contain new features.

1.3 Handled issues

JIRA CODE	ISSUE CODE	DESCRIPTION
CIS-69	/	French translation issues have been fixed.
CIS-196	/	JWT Audience response incorrectly returned array instead of single value.
CIS-185	/	Public key fields can now also be added to the Applications on Client level.

1.3 Known issues

- The **VirtualRootSslMaBackend** setting is marked as required by the configuration page even when no TLS Mutual Authentication is used. Current workaround is to copy the value of the **VirtualRoot** setting.

Note: A request without parameters doesn't generate an error message. This is intended behavior.

Release Notes 1.2.9

Release date: 2019-09-24

1.1 New features

IDS 1.2.9 is a hotfix version and doesn't contain new features.

1.3 Handled issues

JIRA CODE	ISSUE CODE	DESCRIPTION
CIS-69	/	French translation issues have been fixed.
CIS-196	/	JWT Audience response incorrectly returned array instead of single value.
CIS-185	/	Public key fields can now also be added to the Applications on Client level.
CIS-217	/	Parameter name "IsSSlMaEnabled" has been renamed to "IsMTLSAEnabled".

1.3 Known issues

- The **VirtualRootSslMaBackend** setting is marked as required by the configuration page even when no TLS Mutual Authentication is used. Current workaround is to copy the value of the **VirtualRoot** setting.

Note: A request without parameters doesn't generate an error message. This is intended behavior.

Release Notes 1.3.0

Release date: 2019-11-25

1.1 New features

Branding of Identification Service application

An Identification Service application can now be rebranded to suit the clients' look and feel. Tenant administrators can do so by creating new themes and applying them on Application level.

How to rebrand an Identification Service application is explained in full in **Connective - Identification Service 1.3.0 - Branding Documentation – Public**.

1.3 Handled issues

N/A.

1.3 Known issues

JIRA CODE	ISSUE CODE	DESCRIPTION
CIS-232	/	An internal server error occurs when a tenant admin tries to create a theme after the FileRepositoryType setting has been changed from file to Azurefilestorage or Azureblobstorage.
CIS-243	/	A theming issue regarding linked fields occurs when you click cancel in the Color selection window: the linked fields are not restored to their original color.
CIS-228	/	Restore to default buttons for Top bar settings don't work as expected.
/	/	The **VirtualRootSslMaBackend** setting is marked as required by the configuration page even when no TLS Mutual Authentication is used. Current workaround is to copy the value of the **VirtualRoot** setting.

Note: A request without parameters doesn't generate an error message. This is intended behavior.

1. Identification Service

This section describes the technical integration with the Connective Identification Service.

The API for the Identification Service is based on the OpenID Connect protocol, specifically the Authorization Code Flow. The Connective implementation is a basic implementation of this protocol. The "OpenID Connect Basic Client Implementer's Guide" can be consulted for more generic information about the protocol and its implementation as a protocol client. See section 10. References for the URL.

This document will describe the protocol as implemented for the Connective Identification Service.

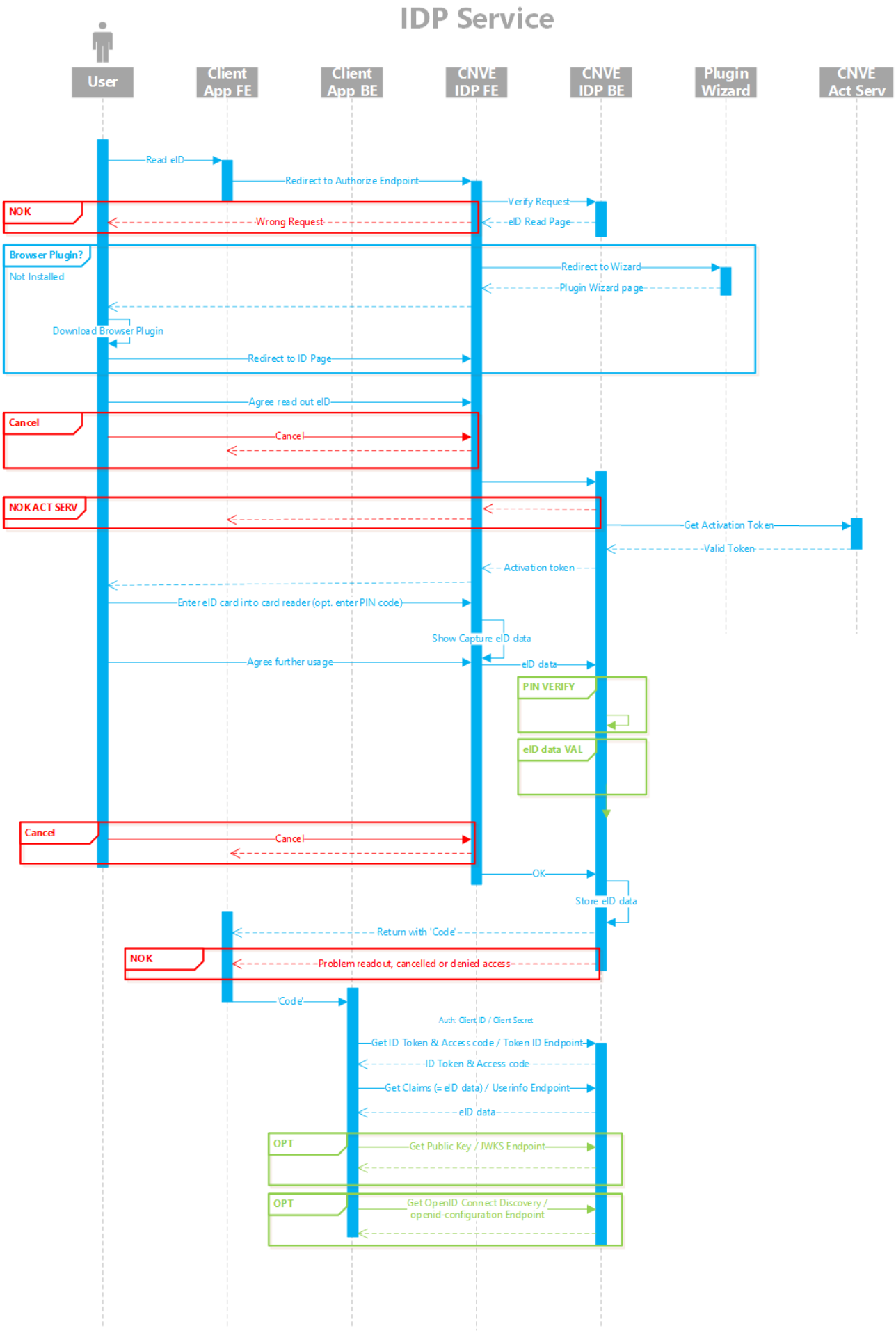
The OpenID Connect protocol goes as follows:

- The customer application redirects the end user's browser to the Identification Service's Authorization Endpoint, indicating what user information will be requested.
    - Browser scenario
        - If not installed, the user is asked to install the Connective Browser Packages using the Connective Plugin Wizard.
        - The requested data is fetched from the eID card and presented to the user in the eID readout page.
        - The user either accepts or refuses to share his/her data with the customer application.
    - Mobile scenario
        - If not installed, the user is asked to install the Connective app.
        - The Connective app opens and shows the eID readout page.
        - The requested data is fetched from the eID card and presented to the user in the eID readout page.
        - The user either accepts or refuses to share his/her data with the customer application.
        - The user goes to the mobile browser to continue.
- The Identification Service redirects the user's browser to customer application with a "code".
- The customer application's backend calls the Token Endpoint and authorizes itself, in exchange the customer application receives an "access token" (to get access to the user's data) and an "id token" which mainly serves as an identifier for the user.
- The customer application's backend calls the UserInfo endpoint to retrieve the requested user information formatted as claims.

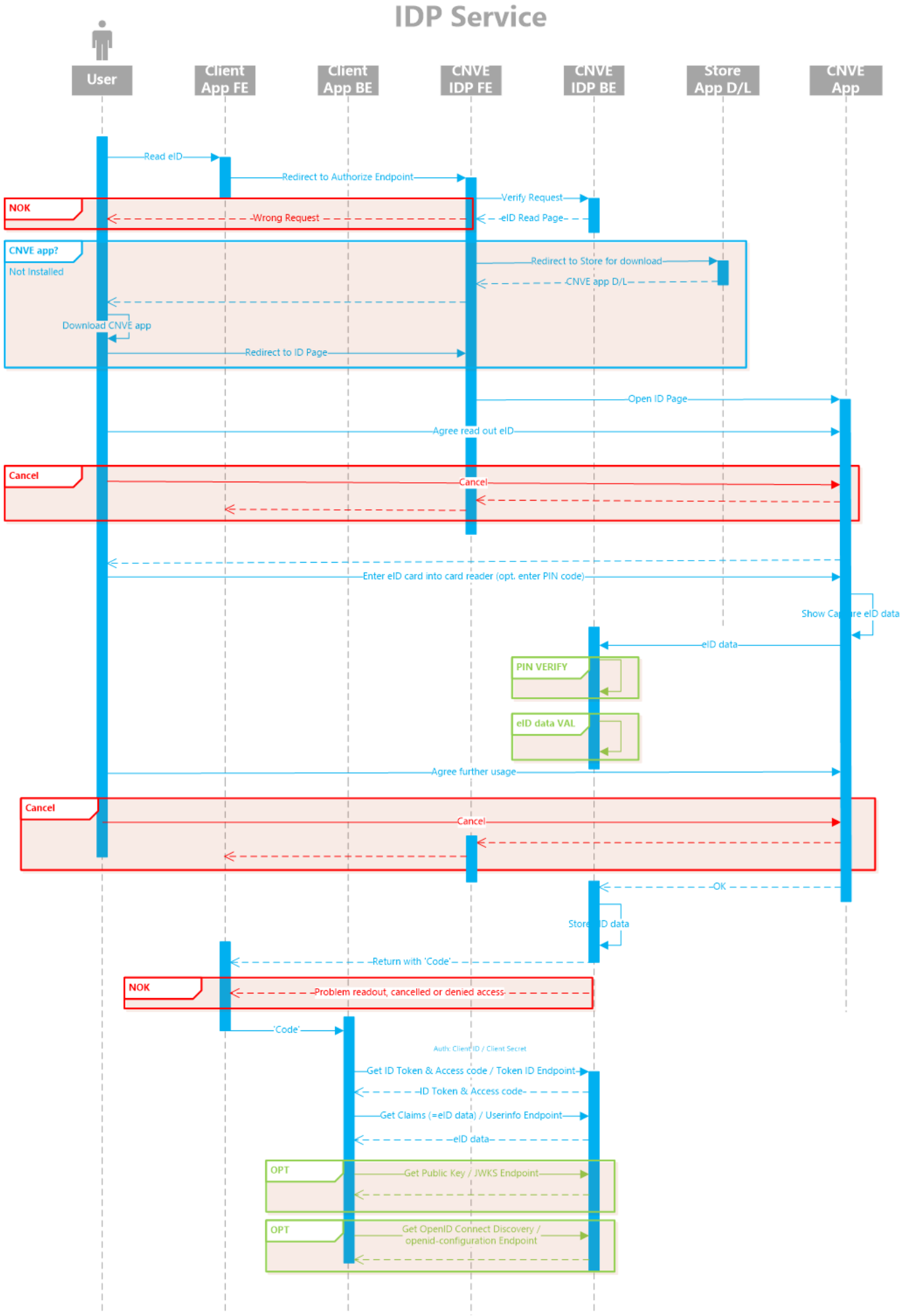
The Identification Service also implements the Jwks Endpoint which lets the customer application retrieve all public keys necessary to verify JWT tokens which were issued by the service.

The OpenID Connect Discovery endpoint can be called to retrieve information of where all the other endpoints are located and what modes are supported.

1.1 Sequence diagram for browser scenario



1.2 Sequence diagram for mobile scenario



1.3 TLS Mutual Authentication

For extra security, the customer application may wish to retrieve user information only over a TLS-connection with Mutual Authentication. While Identification Service connections always use HTTP/S, the Mutual Authentication assures that the server only allows clients to connect when they present a valid client certificate to the server. This **client certificate** needs to be provided up-front to Connective so that it can be trusted by the Identification Service.

Which kind of certificates are allowed depends on the environment the Identification Service is hosted in: our cloud solution may allow self-signed certificates, but an on-premise installation may do further verification through the CA which provided the client certificate, in which case the CA needs to be trusted by the server. Either way the Identification Service uses certificate pinning so that only the current set of trusted certificates is allowed.

The steps in the sequence diagrams shown in the previous sections remain mostly the same when using Mutual Authentication. However, for this setup there might be two servers hosting the Identification Service: one hosting the application for end users and an MTLS-secured server hosting the REST APIs with which the client application should integrate.

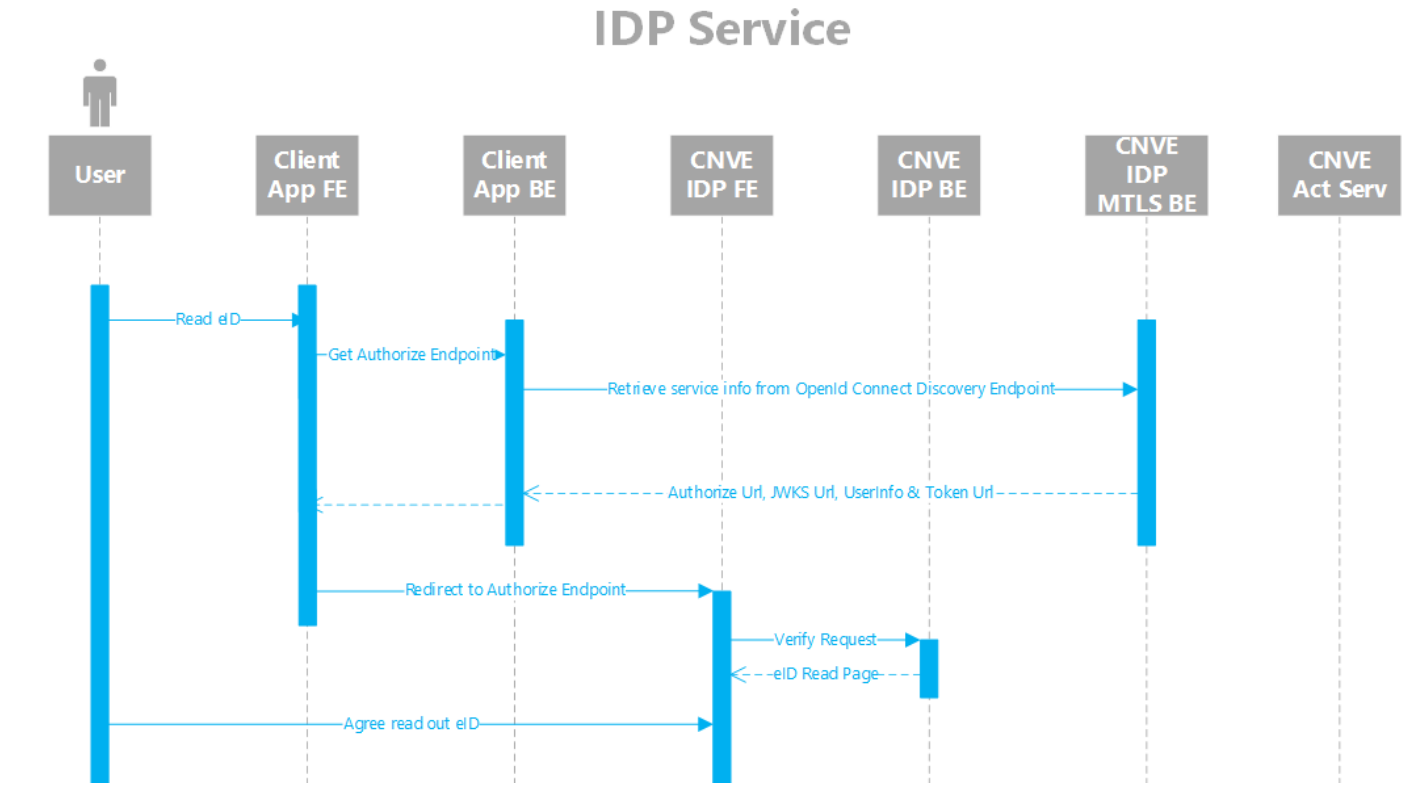
The client application backend should talk to the MTLS-secured server in all cases instead of to the Identification Service backend to which the end user might connect during the readout of their eID. This way the client application will retrieve the proper URLs from the OpenId Connect Discovery endpoint to which the end user needs to be redirected. The MTLS-secured server will check the client certificate whereas the other backend server will always reject any calls from the client application to the Token and Userinfo endpoints.

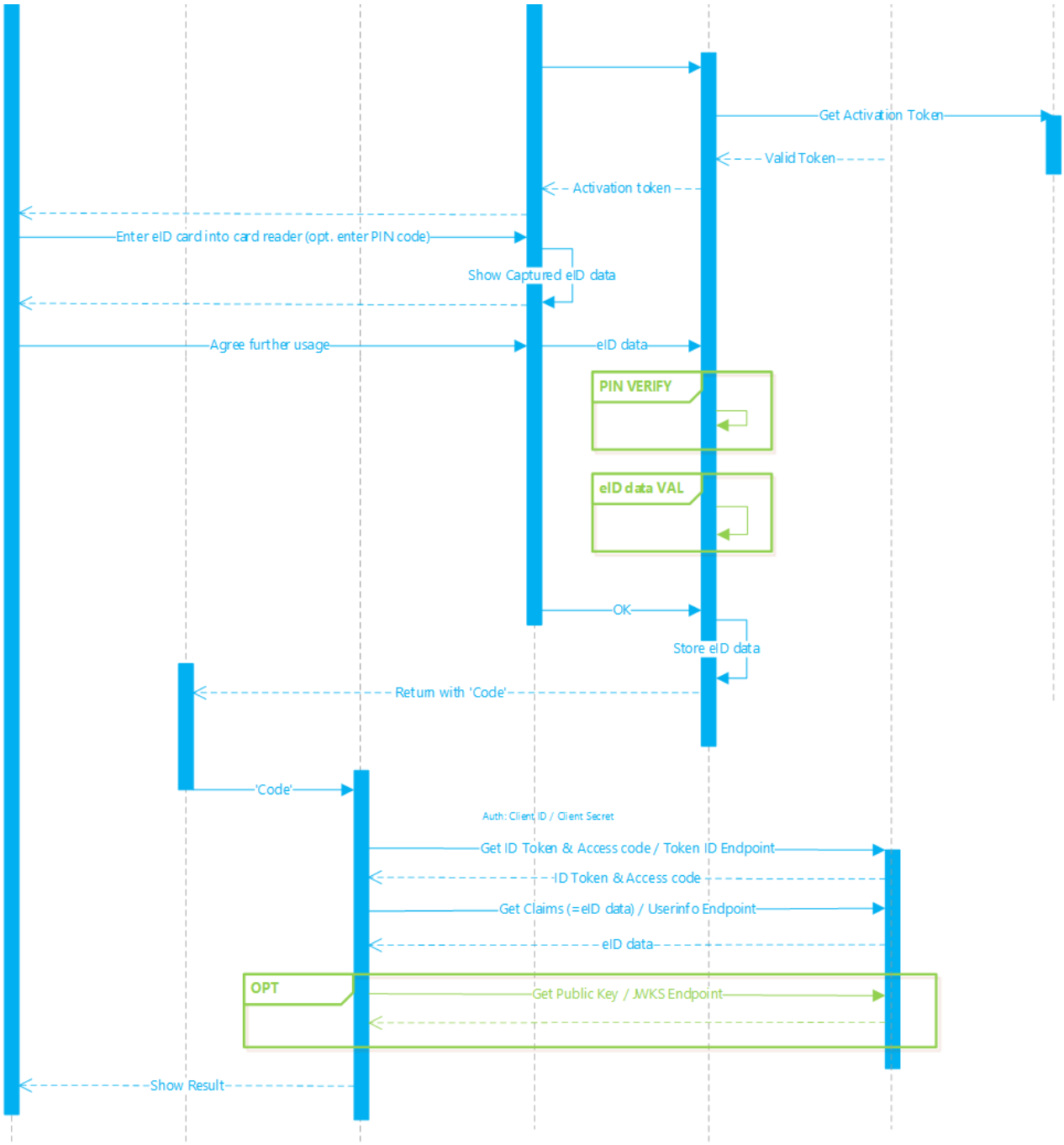
Below you'll find another sequence diagram with the following changes:

1. The client application connects to the OpenId Connect Discovery endpoint to learn the location of the Authorize endpoint, amongst other things.
2. Once the client application knows the Authorize URL it can redirect the end user to it. The eID readout happens as usual.
3. When the end user has given his consent to read identification details the client application receives a code which it should then send to the MTLS-secured server to receive a token.
4. This token can then be exchanged for user information through the UserInfo endpoint, again by connecting to the MTLS-secured server.

Note that the negative flows for the end user have been left out for clarity reasons.

TLS MA Sequence Diagram




Get Activation Token
Valid Token
Activation token
Enter eID card into card reader (opt. enter PIN code)
Show Captured eID data
Agree further usage
eID data
PIN VERIFY
eID data VAL
OK
Store eID data
Return with 'Code'
'Code'
Auth: Client ID / Client Secret
Get ID Token & Access code / Token ID Endpoint
ID Token & Access code
Get Claims (=eID data) / Userinfo Endpoint
eID data
OPT
Get Public Key / JWKS Endpoint
Show Result

1.4 Identity card edge cases

The primary audience of the Connective Identification Service is an adult user who wants to prove one's identity by inserting the identity card into a card reader and providing the card's secret pin. The service itself supports other use cases but this may bring attention to a number of cases:

- The identity card has recently expired
- The identity card may have been stolen
- The identity card may need to be replaced and the old card can no longer be used for signing
- The card may no longer accept a Pin due to more than 3 failed attempts of entering it
- The customer process requires a child or adolescent to prove their identity

1.4.1 Expired, stolen or replaced identity card

The Identification Service does not check the expiry dates printed on the card, simply because a person's identity does not expire. It also does not check against the **checkdoc** database of the government to see whether it is stolen or in the process of being replaced. The Connective Identification Service only checks whether the user information has not been tampered with and that the holder of the card knows the card's Pin.

Please note though that when this edge case occurs, verification of the user information may or may *not* fail. The exact failure mode depends on which checks are requested and on whether the government marked the card as unworthy of trust. Again, it is recommended for the customer's business process to do extra validation.

The customer application *can* opt to read cards like this by disabling verification of the user information, but the Identification Service will be unable to detect forgeries or impersonation. It is therefore recommended to only disable verification if somebody can also verify the identity document in person, e.g. at a branch office.

1.4.2 Locked identity card

The holder of the card may have attempted to enter an incorrect pin until the card refuses any further attempts of checking the pin. The Connective Identification Service will stop the process if a Pin Check was requested and the card refuses.

The customer application can opt to read the user information by disabling the pin check, though this also means that the Identification Service will be unable to detect if the card holder is the actual subject of the information on the card. It is therefore recommended to only disable pin checks if somebody can verify the identity document in person, e.g. at a branch office.

1.4.3 Belgian kids-ID

The Belgian kids-ID needs to be treated as a permanently locked card. The child or legal guardian is never informed of the pin on the card, making any pin check impossible.

The customer application can opt to read the user information by disabling the pin check, but as stated above, the card holder can no longer be guaranteed to be the subject of the card. It is recommended to have a parent or legal guardian confirm the identity and do further checks in the process of the customer application.

2. Getting started

Some customer information needs to be collected before the Connective Identification Service can be used:

Each customer needs to define a "**display name**" **for their company**. This string will show up on the summary screen to indicate clearly which company requested identification.

Each application from the customer needs to define an "**application display name**" as well to better clarify what the user information will be used for when showing the summary screen.

There is a need to agree on which **authentication method** to use for exchanges between the customer application's backend and the Identification Service backend:

- Basic authentication:

  Encodes the password and transmits it through the Authorization HTTP header

- "POST" authentication:

  Sends the password together with other parameters in a form-url-encoded POST body.

The method to use depends largely on the OpenID Connect library used by the customer application. Either method will have to communicate over a HTTP/S connection, so the password is never sent "in the clear". Basic authentication is preferred because it allows the HTTP server to quickly filter bad requests.

Each application also needs to decide whether they will receive the user information as a simple **JSON** object or as a **signed JWT** (RS256 algorithm). Once more this depends on the customer application's choice of OpenID Connect library. It is recommended to use JSON for its simplicity. Using a JWT does have the effect that the data is signed, allowing the data to be validated at a later point. **Please note** though that the user information JWT contains personal information on top of authentication information – it is thus not advisable to pass it back to the user as one would with a plain authentication token.

Another necessary bit of information is the **redirect URL**. As explained in section 1, OpenID Connect works by redirecting the user between the customer application and the Identification Service. While the user information cannot be retrieved without a password, an attacker may still try to initiate a readout flow and redirect the end user to a phishing website. Each application should therefore use a single redirect URL per application and give this URL **up-front** to Connective so that it can be stored in the configuration. An error will be raised if the redirect URL in a given parameter does not match the one registered in the configuration. A redirect URL should use the HTTP/S scheme, cannot contain any query parameters and must always be passed to the Connective Identification Service in the exact same way it was registered.

If necessary, there can be a custom limit on the time between redirecting the user's browser back to the customer application and requesting the user's information; by default this limit is 10 seconds. Waiting any longer will cause an error.

For extra security, the customer application may wish to retrieve user information over a TLS-connection with Mutual Authentication. If this is the case, the **client certificate** needs to be provided up-front so that it can be trusted by the Identification Service.

Connective can then proceed to initialize the configuration. Once done, each application will receive a unique OpenID Connect **client_id** associated with the application's redirect URL and a **unique password**.

3. Authorization Endpoint

The authorization endpoint is the entry point for identity data readout. The customer application redirects the user's browser towards this entry point:

```
https://<servername>/Authorize?<readout parameters>
```

The parameters are the following:

```
https://<authorization endpoint url>?response_type=code&client_id=<client_id>&redirect_uri=
<redirect_uri>&scope=<scope>&state=<state>&nonce=<nonce>&ui_locales=<ui_locales>
```

All parameters are mandatory by default unless indicated as optional.

- **client_id**: Identifier for the customer application, supplied by Connective
- **redirect_url**: URL where the Identification Service will redirect the user's browser to with the intent to deliver a **code** to the customer application when successful, or to inform the customer application about errors. Must match the application's configuration as seen in section 2. Getting Started.
- **scope**: space separated list of requested scopes. This variable's value should **always** start with **openid** followed by the scopes documented below in section 3.5. Supported Scopes and Returned Claims.
- **state**: value which must store some information about the calling application's state so that the calling application can confirm that a readout happened through its own doing when the user returns to it. This value may be encrypted but must be secured against modification (e.g. by verifying it with a HMAC).
- **nonce**: (optional) When the client supplies a nonce (a unique number) then it will be included in the output of the backend. This way the value can be checked later on in the process.
- **ui_locales**: (optional) Language code to indicate in which language to present the UI (e.g. **nl be**). Available languages are Dutch, French, English and German (**nl-be, fr-fr, en-us, de-de**). English is the default.

3.1 Example

Redirect the user to a URL like the following to retrieve validated personal info from the backend:

```
https://ids.example.org/Authorize?
response_type=code&client_id=exampleclient&redirect_uri=https%3A%2F%2Fmyapp.example.com%2Fopenidauth&scope=openid%20connective:beid_personalinfo%20connective:beid_verifieddata%20connective:beid_verifiedpin&state=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXUyJ9.eyJjdXN0b21lcmlkIjoiNTgyOTYiLCJ0cmFuc2FjdGlvbiI6Ijc4MjI0MiJ9.S7qcY6nEbwa5Ds9qwQtBhyLLlXRobz9duKtgTOecYZk&nonce=848866&ui_locales=en-us
```

3.2 Incorrect Request

When either the **redirect_uri** or the **client_id** is wrong or missing, an error message will be presented to the user without redirecting back to the **redirect_uri**.

In all other cases, the user's browser will be redirected to the **redirect_uri** with a URL parameter "**error**" containing the error code, as well as a parameter "**error_description**" indicating the problem in more detail. The **state** parameter will contain the state value originally sent to the authorization endpoint.

```
https://<redirecturi>?error=<error>&error_description=<erro_description>&state=<state>
```

The value of the error and error_description parameters can be one of the following:

ERROR	ERROR_DESCRIPTION
unsupported_response_type	**Response_type** other than 'code' is not supported
invalid_scope	**Scope** parameter contains unrecognized scopes
server_error	Server encountered unexpected state
invalid_request	Parameter **response_type** is missing
invalid_request	Parameter **scope** is missing
Invalid_request	Parameter **state** is missing

3.3 Readout canceled or failed

The readout request may have been correct but there might have been some issues: the user might not have given consent, the user may have canceled before the readout could complete, the user may refuse to install the Connective browser package or the eID data might no longer be trusted. In such case the identification service instructs the user's browser to redirect to the given redirect uri with a few extra parameters describing the condition:

```
https://<redirecturi>?error=access_denied&error_description=<subcode of error>&state=<state>
```

The error_description parameter will be one of following values, used as sub-error-code:

ERROR	ERROR_DESCRIPTION
access_denied	user_cancelled_or_denied_access
access_denied	not_able_to_readout_eid_card

3.4 Successful readout

A screen will be presented to the user to start the readout of the eID. If the Connective browser package is not yet installed, the user will be redirected to the wizard. Next, the activation service will be called to activate the Connective browser package. Depending on the requested scope, any necessary files will be read from the eID.

For mobile users, the Connective app will open to read the eID using a Bluetooth card reader. After the readout, the user continues in the mobile browser.

When the eID readout was successful and the user approved the use and processing of their data, the user's browser will be redirected to the specified redirect URL. This URL will be appended with a number of URL parameters:

- The **code** parameter for accessing the Token endpoint described in section 4.1.
- The **state** parameter which contains the value of the state parameter from the initial URL to the Authorize endpoint. The customer application needs to verify that it has not been modified and that the readout session is from a legitimate recently started process.

```
https://<redirecturi>?code=<code>&state=<state>
```

For the example in section 3.1:

```
https://myapp.example.com/openidauth?
code=YF5sl24DYQVtVhDJlirYFeDH1675Ety5uKL4gaAglRB_zWJnWFrmkdWyKgObQAYn&state=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXUy
J9.eyJjdXN0b21lcmlkIjoiNTgyOTYiLCJ0cmFuc2FjdGlvbiI6Ijc4MjI0MiJ9.S7qcY6nEbwa5Ds9qwQtBhyLLlXRobz9duKtgTOecYZk
```

3.5 Supported Scopes and Returned Claims

The set of returned information depends on the scope parameter. This section and the following sub-sections list the possibilities.

It is **strongly recommended** to pass the **beid_verifieddata** scope (see section 3.5.7 below) and the **beid_verifiedpin** scope (see section 3.5.6 below) in all cases. Failing to pass these scopes does not guarantee that the user information is authentic or that the card actually belongs to the person holding that card.

Please note that all claims discussed in the following sections are **optionally** returned. If a claim's data is not present on the identity card or is otherwise undecipherable then it will be left out. Text claims which are present are returned the way they are stored on the eID. In some cases this results in an empty string value or weirdly-formatted data. For future-proofing the customer application should treat most returned claims as free-form text without any delimiters.

3.5.1 Default Behavior

When no Identification Service-specific scopes are defined (that is, other than the required **openid** scope), the user information returned from the final step will contain just a sub claim containing the user's card number or national number (see section 3.5.3 below).

Again, this mode of operation is **not recommended**. Any card may be inserted without checking if the holder knows that card's secret pin or the card may have been forged. Verifying the data and checking the pin makes sure that the **sub** claim is related to the card holder.

3.5.2 Personal Info

Use the **connective:beid_personalinfo** scope to retrieve the following claims:

CLAIM NAME	DESCRIPTION
http://ids.connective.eu/beid/card_number	String containing the number of the card. Each card has a unique number, replacing the card will give a new number to one person.
.../card_validity_date_from	String containing a date formatted as yyyy-mm-dd
.../card_validity_date_to	String containing a date formatted as yyyy-mm-dd
.../card_delivered_in_town	String of the municipality which delivered the card
.../name	String with the name
.../two_given_first_names	String containing one or more given names
.../first_letter_of_3rd_given_name	String containing the first letter of 3rd given name
.../nationality	String with the nationality
.../birth_location	String containing birth location
.../birth_date	String containing a date formatted as yyyy-mm-dd
.../gender	String containing "M" or "F"
.../noble_condition	String stating the person's nobility

CLAIM NAME	DESCRIPTION
.../document_type	String declaring the type of document by means of a number: 1: Belgian citizen 6: Kids ID (<12 years old) 7: bootstrap card 8: "habilitation / machtigings" card 11: card A 12: card B 13: card C 14: card D 15: card E 16: card E+ 17: card F 18: card F+ 19: card H
.../special_status	String reserved for special states. Only "0" is used at this moment

3.5.3 Include National Number

By default, the **sub** claim is the card number and no national number is returned when requesting the personal information listed in section 3.5.2 above.

Adding the **connective:beid_nationalnumber** scope will change the contents of the **sub** claim to the national number and add an extra claim containing the national number. This means that the card number is not returned unless the personal information is actually requested.

CLAIM NAME	DESCRIPTION
http://ids.connective.eu/beid/national_number	String containing the national number

3.5.4 Address Info

Adding the **connective:beid_addressinfo** scope will read out the address information.

CLAIM NAME	DESCRIPTION
http://ids.connective.eu/beid/street_and_number	String containing the street and number. No guaranteed delimiter.
.../location	String containing the current residence location
.../zip_code	String containing a postal code

3.5.5 Photo

Including the **connective:beid_photo** scope adds an extra claim containing a small picture.

CLAIM NAME	DESCRIPTION
http://ids.connective.eu/beid/photo	String containing the identity card's picture as a base64 encoded JPEG image

3.5.6 Pin Check

Specify the connective:beid_verifiedpin scope to request a pin check. When a user fails to enter the correct pin then the identity reading process stops with an error as seen in section 3.3. Readout canceled or failed.

Without this scope and claim no guarantees can be made about the holder of the card as the presented card may belong to someone else.

CLAIM NAME	DESCRIPTION
http://ids.connective.eu/beid/pinverified	A string containing “true” if the pin is correct.

3.5.7 Verified Data

The **connective:beid_verifieddata** scope will check the trustworthiness of all the data requested using the other scopes. It does this by checking the cryptographic integrity of a number of signatures and requesting online if some or all of them have been blacklisted (see section 1.4.1).

If the card cannot be trusted to be genuine then the identity reading process stops with an error as seen in section 3.3. Readout canceled or failed.

This scope is currently not forced to be on because some scenarios on mobile devices may want to skip verification as a workaround for the long readout time of mobile card readers. However, this scope can only be left out if there is another sufficiently strong authentication method to hold the right user accountable for any read data. It is then up to the customer application to verify that the newly read personal info is valid and matches older personal info.

It is therefore recommended to specify this scope in all cases when in doubt.

CLAIM NAME	DESCRIPTION
http://ids.connective.eu/beid/dataverified	A string containing “true” if the data is verified.

3.5.8 Skip Summary

The customer application may already have a screen to ask the user’s consent about reading personal information and a second screen to show the retrieved information. In this case the Connective Identification Service can skip the final summary screen and return immediately to the customer application by specifying the **connective:beid_nosummary** scope.

4. Token Endpoint

The token endpoint should be called from the customer application's backend. Requests need the code received from the Authorization endpoint (see section 3.4 Successful readout) to exchange it for an access code and an IdToken JWT.

The access code is needed to get the readout information from the UserInfo endpoint.

The IdToken is an identification token (combination issuer-subject).

4.1 Request

When calling the Token endpoint, the customer application needs to authenticate by using its unique **client_id** and **unique password**. As mentioned in section 2. Getting started there are 2 methods of providing these credentials, and the exact method used is decided during registration. If the method needs to change, then Connective needs to alter the registration.

In both cases the HTTP Method is POST. The content type should be set to **application/x-www-form-urlencoded**.

The Token endpoint can be found at the following URL:

```
https://<server name>/identityapi/v1/openidconnect/tokens
```

The body should contain at least the following parameters:

- **grant_type** should be a fixed value set to "authorization_code" (without quotes)
- **code** as received from the Authorization endpoint
- **redirect_uri** as provided at registration time and to the Authorization endpoint (see section 2. Getting Started).

4.1.1 Basic authentication scheme

This authentication scheme uses the HTTP Authorization header as specified for the Basic authentication scheme in RFC 1945.

Please note that there is one extra *complication*: OpenID Connect expects both the **client_id** and password to be converted to an UTF-8 string and to be URL encoded **before** concatenation and base64 encoding. In short, the header's value is built like this:

```
Basic <base64( url-encode(utf8('clientid')) + ':' url-encode(utf8('password')) )>
```

4.1.2 POST" authentication scheme

This authentication scheme expects a few extra parameters in the request's body.

- **client_id** as given by Connective after registration (see section 2. Getting Started)
- **client_secret** the unique password for this customer application

These values are all URL-encoded like the rest of the parameters in the POST body.

4.2 Example

Here is how a request might look, assuming that the application is registered using Basic authentication:

```
POST /identityapi/v1/openidconnect/tokens HTTP/1.1
Host: ids.example.org
Content-Type: application/x-www-form-urlencoded
Authorization: Basic ZXhhbXBsZWNsaWVudDpodW50ZXIy

grant_type=authorization_code&code=YF5sl24DYQVtVhDJlirYFeDH1675Ety5uKL4gaAglRB_zWJnWFrmkdWyKgObQAYn&redirect_
uri=https%3A%2F%2Fmyapp.example.com%2Fopenidauth
```

4.3 Incorrect request

- Http status code 400 if the request is incorrect (missing parameters or incorrect value)
- Http status code 401 if credentials are not correct or the token is expired
- Http status code 404 if the code does not exist (for that client)

4.4 Correct request

A correct request to the Token endpoint will give following response:


```
{
 "access_token":"EDmobWFEMuP8JxZEy2k7wWtb9NUEWHuSphs1Qj9d3Rm5lBG0oCXX3Rm",
 "token_type":"Bearer",
 "expires_in":10,
 "id_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2lkcy5leGFtcGxlLm9yZy8iLCJpYXQiOjE0ODg1NDUxMTUsImV4cCI6MTQ4ODU0NTEyNSwiYXVkIjoiZXhhbXBsZWNsaWVudCIsInN1YiI6Ijc3ODg5OTEyMyIsIm5vbmNlIjoiODQ4ODY2In0.1sVHkE0MQyrepWNK_yQ7NbQ4IxyyMD3L9ewFA4yUaS8"
}
```


Where the parameters mean the following:

- **access_token**: this parameter is used as credential to access the UserInfo endpoint (see section 5. UserInfo Endpoint).
- **token_type**: specifies the access token's type, a fixed value.
- **expires_in**: this number tells the caller for how many seconds the **access_token** will remain valid.
- **id_token**: IdToken in JWT format, see below.

4.4.1 IdToken details

The id token is a JWT. Information on using JWT is not in scope of this document, please see the references in section 10. References for more information.

The payload of the IdToken JWT consists of the following:

```
{
   "iss":"<issuer identifier, url of the server>",
   "iat":<token creation time in seconds since 01/01/1970>,
   "exp":<token expiry time in seconds since 01/01/1970>,
   "aud":"<clientid of customer application>",
   "sub":"<user's card number or national number>",
   "nonce":"<optional nonce value from initial request on Authorization endpoint>"
}
```

5. UserInfo Endpoint

The customer application's backend calls the UserInfo endpoint to retrieve the eID data in claims form.

To do so, the customer application needs to provide the **access_token** as received from the Token endpoint in the HTTP Authorization header. The access token is only valid for a limited amount of time (see section 4.4 Correct request) so this call must be done as soon as possible after the Token endpoint call.

The UserInfo endpoint can be found at the following URL:

```
https://<server name>/identityapi/v1/openidconnect/userinfo
```

5.1 Request

The HTTP method is GET.

The **access_token**'s value needs to be prepended with "Bearer " which can then be used as value for the HTTP Authorization header.

5.2 Example

```
GET /identityapi/v1/openidconnect/userinfo HTTP/1.1
Host: ids.example.org
Authorization: Bearer EDmobWFEMuP8JxZEy2k7wWtb9NUEWHuSphs1Qj9d3Rm5lBG0oCXX3Rm
```

5.3 Incorrect request

Http status code 401 Unauthorized if the authorization header is missing or the used **access_code** is unknown or expired.

5.4 Successful request

As mentioned in section 2. Getting started the format of the response depends on a preference set when the customer application is registered by Connective.

This preference decides whether the result is plain JSON or a signed JWT.

The actual contents of the response depend on the scopes given to the Authorization endpoint. For more info see section 3.5 Supported Scopes and Returned Claims.

5.4.1 Plain json

The content type of the HTTP response will be **application/json**.

The content will be in following form:

```
{
    "sub" : "<identifier of the user/card>",
    "http://ids.connective.eu/beid/name" : "<value>"
    ....
}
```

5.4.2 Signed JWT

The content type of the HTTP response will be **application/jwt**.

The payload of the JWT (without header and signature) will be in following form:

```
{
    "iss" : "<issuer identifier>",
    "aud" : "<clientid of the customer application>",
    "sub" : "<identifier of the user/card>",
    "http://ids.connective.eu/beid/name" : "<value>"
    ....
}
```

6. OpenID Connect Discovery

The Connective identification service implements the OpenID Connect Discovery endpoint to find out all the URIs of the endpoints in the Identification service.

The Discovery endpoint can be found at the following URL:

```
https://<server name>/.well-known/openid-configuration
```

The HTTP method is GET, no authentication is necessary.

6.1 Example

Example output of the endpoint:

```
{
"issuer":"https://ids.exampleorg/OpenIdConnect",
"authorization_endpoint":"https://ids.example.org/OpenIdConnect/Authorize",
"token_endpoint":"https://ids.example.org/OpenIdConnect/identityapi/v1/openidconnect/tokens",
"jwks_uri":"https://ids.example.org/OpenIdConnect/identityapi/v1/openidconnect/keys
"userInfo_endpoint":"https://ids.example.org/OpenIdConnect/identityapi/v1/openidconnect/userinfo",
"response_types_supported":["code"],
"subject_types_supported":["public"],
"id_token_signing_alg_values_supported":["RS256"],
"token_endpoint_auth_methods_supported":["client_secret_post","client_secret_basic"]
}
```

7. Jwks Endpoint

In order to validate the signatures of JWTs returned by the Connective identification service, the Jwks endpoint can be called to retrieve the public keys used to validate signatures. The identifier of the particular key which was used in a signature will be present in a header of the JWT.

The Jwks endpoint can be found at the following URL:

```
https://<server name>/identityapi/v1/openidconnect/keys
```

The HTTP method is GET, no authentication is necessary.

7.1 Example

Example output from the Jwks endpoint:


```
{
    "keys": [
                {
                    "kty": "RSA",
                    "alg": "RS256",
                    "use": "sig",
                    "kid": "02B1174234C29F8EFB69911438F597FF3FFEE6B7",
                    "n":
"uzMHUsTLuKcrftWFPAwDPkg2xY7i6r9Q6_9mygvvXNMQFV8jbxvsnfMvWE0XF_OyH9JZ0w9XCOkzvQc31wt5VKXJw9n-
Qt1mfOAx4COhyTcI0ptdeOHvg7KociG65xP2HFoM4S6QP_T102e9rQcon6BehCNAqZPA1RYFWFMHO__jBf0Lml7vDHLmYTih8ewcJ1CUJuZfDp
MwNQHFa7wVfZrA39lSt0ZOZPdDtui2fpgItyM6K7Ug-
B8p47krJ8EUNSYeWAlWSuLYM1iLpWwSQ6lnQXZwvaOu3mM_jkTstIfd9Mxj7XIXSTuNXwx_E5PO5eiKyUpUkbduNXhOMuK8EQ",
                    "e": "AQAB"
                }
    ]
}
```

8. Version Endpoint

Connective's Implementations team may request the exact version of the Connective Identification Service to reproduce a given behavior.

This version can be queried by calling the following endpoint:

```
https://<server name>/identityapi/v1/openidconnect/version
```

The HTTP method is GET, no authentication is necessary.

8.1 Example

Example output of the version call:

```
{
    "fileVersion": "1.0.17094.01",
    "productVersion": "1.0.0"
}
```

9. API Errors

The backend endpoints like the Token Endpoint or UserInfo Endpoint may return error responses which contain only an error code and a generic message. The response does not contain full details as a security measure.

9.1 Token Endpoint

ERROR	ERROR_DESCRIPTION
CATQ0001	The authorization code in the request did not match a readout session. Either this code never existed, or the original session got deleted after some time.
CATQ0002	The authorization code can no longer be used because the session is older than the timespan defined by the **CodeDuration** configuration parameter.
CATQ0003	The readout session was requested with a different **client_id** than the one used in this Token Endpoint request.
GTKQ0001	The access token in the request did not match a readout session. Either the given token never existed, or the original session got deleted after some time.

9.2 UserInfo Endpoint

ERROR	ERROR_DESCRIPTION
GUIQ0001	The access token in the request did not match a readout session. Either the given token never existed, or the original session got deleted after some time.
GUIQ0002	The access token can no longer be used because the session is older than the timespan defined by the **TokenDuration** configuration parameter.

10. References

- OpenID Connect Core (http://openid.net/specs/openid-connect-core-1_0.html)
- OpenID Discovery (http://openid.net/specs/openid-connect-discovery-1_0.html)
- OpenID Basic Client Implementer's Guide (http://openid.net/specs/openid-connect-basic-1_0.html)
- RFC 6749 The OAuth 2.0 Authorization Framework (https://tools.ietf.org/html/rfc6749)
- RFC 6750 The OAuth 2.0 Authorization Framework: Bearer Token Usage (https://tools.ietf.org/html/rfc6750)
- RFC 7519 JSON Web Token (JWT) (https://tools.ietf.org/html/rfc7519)
- RFC 7515 JSON Web Signature (JWS) (https://tools.ietf.org/html/rfc7515)