**Oauth2 Cookbook**

**DOCUMENT HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description of changes / remarks |
| 0.01 | 09/03/2023 | Florent Marteau | First version. |

**TABLE OF CONTENT**

*Introduisez vos titres / sous-titres / textes dans le corps du document (pas dans la table des matières).*

*Faites une mise à jour de la table des matières (clic droit sur la table des matières puis <Update Field>) avant de publier une nouvelle version*

[1. Introduction 4](#_Toc64466024)

[2. Definitions 5](#_Toc64466025)

[3. OAuth grant flows 6](#_Toc64466026)

[3.1. Authorization code grant flow 6](#_Toc64466027)

[3.2. Implicit grant flow (deprecated) 7](#_Toc64466028)

[3.3. Client credentials grant flow 8](#_Toc64466029)

[4. Obtaining Authorization 9](#_Toc64466030)

[4.1. Technical requirements 9](#_Toc64466031)

[4.2. Authorization code grant flow 9](#_Toc64466032)

[4.3. Implicit grant flow 13](#_Toc64466033)

[4.4. Client Credentials grant flow 16](#_Toc64466034)

[4.5. Client Authentification 18](#_Toc64466035)

[5. Obtaining user informations using OpenID 19](#_Toc64466036)

[5.1. OpenId Connect 19](#_Toc64466037)

[5.2. Authorization code grant flow with openid 21](#_Toc64466038)

[5.3. Implicit grant flow with opened (deprecated) 25](#_Toc64466039)

[5.4. Accessing userinfo endpoint 28](#_Toc64466040)

[6. OAuth for desktop and mobile applications 29](#_Toc64466041)

[6.1. Initiating the Authorization Request from a Native App 29](#_Toc64466042)

[6.2. Receiving the Authorization Response in a Native App 30](#_Toc64466043)

[6.3. Fat Client userguide 31](#_Toc64466044)

[6.4. Using the fatClient 33](#_Toc64466045)

[7. Token introspection 34](#_Toc64466046)

[7.1. Introspection request 35](#_Toc64466047)

[7.2. Introspection response 35](#_Toc64466048)

[8. JWKS Uri 37](#_Toc64466049)

[9. PKCE (Proof Key for Code Exchange by OAuth Public Clients) 38](#_Toc64466050)

[9.1. Protocol 38](#_Toc64466051)

[10. Environments 40](#_Toc64466052)

[10.1. Integration 40](#_Toc64466053)

[10.2. Acceptation 40](#_Toc64466054)

[10.3. Production 40](#_Toc64466055)

[11. External document references 40](#_Toc64466056)

# Introduction

In the traditional client-server authentication model, the client requests an access-restricted resource (protected resource) on the server by authenticating with the server.

In order to provide third-party applications access to restricted resources, the resource owner shares its credentials with third party.

This creates several problems and limitations:

* Third-party applications are required to store the resource owner's credentials for future use, typically a password in clear-text.
* Servers are required to support password authentication, despite the security weaknesses inherent in passwords.
* Third-party applications gain overly broad access to the resource owner's protected resources, leaving resource owners without any ability to restrict duration or access to a limited subset of resources.
* Resource owners cannot revoke access to an individual third party revoking access to all third parties, and must do so by changing the third party's password.
* Compromise of any third-party application results in compromise of the end-user's password and all of the data protected by that password.

OAuth addresses these issues by introducing an authorization layer and separating the role of the client from that of the resource owner.

In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner.

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token, a string denoting a specific scope, lifetime, and other access attributes.

Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner.

The client uses the access token to access the protected resources hosted by the resource server.

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol.

It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

# Definitions

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

OAuth defines several terms used throughout this document:

***resource owner***

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

***resource server***

The server hosting the protected resources, accepting and responding to protected resource requests using access tokens.

***client***

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other ).

***client identifier***

The authorization server issues the registered client a client identifier -- a unique string representing the registration information provided by the client.

The client identifier is not a secret; it is exposed to the resource owner and MUST NOT be used alone for client authentication. The client identifier is unique to the authorization server.

***authorization server***

The server issuing access tokens to the client after successfully the resource owner and obtaining authorization.

***redirection endpoint***

After completing its interaction with the resource owner, the authorization server directs the resource owner's user-agent back to the client. The authorization server redirects the user-agent to the client's redirection endpoint previously established with the authorization server during the client registration process or when making the authorization request.

The redirection endpoint URI MUST be an absolute URI as defined by [RFC3986 section 4.3](https://tools.ietf.org/html/rfc3986#section-4.3). The endpoint URI MAY include an application/x-www-form-urlencoded" formatted query component which MUST be retained when adding additional query parameters. The endpoint URI MUST NOT include a fragment component.

***access token***

Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

The token is a string representing the authorization granted to the client by the resource owner. The string is opaque to the client.

The access token provides an abstraction layer, replacing different authorization constructs (e.g., username and password) with a single token understood by the resource server. This abstraction enables issuing access tokens more restrictive than the authorization grant used to obtain them, as well as removing the resource server's need to understand a wide range of authentication methods.

***access token scope***

The authorization and token endpoints allow the client to specify the scope of the access request using the "scope" request parameter. In turn, the authorization server uses the "scope" response parameter to inform the client of the scope of the access token issued.

The value of the scope parameter is expressed as a list of space-delimited, case-sensitive strings. The strings are defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.

If the client omits the scope parameter when requesting authorization, the authorization server processes the request using the pre-defined default values associated with the client.

***refresh token***

Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner). Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token.

A refresh token is a string representing the authorization granted to the client by the resource owner. The string is opaque to the client.

The token denotes an identifier used to retrieve the authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers.

***client assertion***

In this implementation of the OAuth2 protocol, the client assertion is a JSON Web Token or compact, URL-safe means of representing client authentication information.

# OAuth grant flows

## Authorization code grant flow

The authorization code is obtained by using an authorization server as an intermediary between the client and resource owner.

Instead of authorization directly from the resource owner, the client the resource owner to an authorization server (via its user-agent) which in turn directs the resource owner back to the client with the authorization code.

Before directing the resource owner back to the client with the authorization code, the authorization server authenticates the resource owner and obtains authorization.

Because the resource owner authenticates with the authorization server, the resource credentials are never shared with the client.

The authorization code provides a few important security benefits, such as the ability to authenticate the client, as well as the transmission of the access token directly to the client without passing it through the resource owner's user-agent and potentially exposing it to others, including the resource owner.



## Implicit grant flow (deprecated)

The implicit grant is a simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. In the implicit flow, instead of issuing the client authorization code, the client is issued an access token directly (as the result of the resource owner authorization).

The grant type is implicit, as no intermediate credentials (such as an authorization code) are issued (and later used to obtain an access token).

When issuing an access token during the implicit grant flow, the authorization server does not authenticate the client. In some cases, the client identity can be verified via the redirection URI used to deliver the access token to the client. The access token may be exposed to the resource owner or other applications with access to the resource owner's user-agent.

Implicit grants improve the responsiveness and efficiency of some clients (such as a client implemented as an in-browser application), since it reduces the number of round trips required to obtain an access token.

The latest security recommendations recommends not using this flow (<https://tools.ietf.org/html/draft-ietf-oauth-security-topics-16#section-2.1.2>) The recommended flow is the [authorization code grant flow with a public client with PKCE](#_PKCE_(Proof_Key) (no security needed while contacting the backend to exchange the authorization code and a code challenge to link the frontend and backend requests together)



## Client credentials grant flow

The client credentials (or other forms of client authentication) can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously arranged with the authorization server. Client credentials are used as an authorization grant typically when the client is acting on its own behalf (the client is also the resource owner) or is requesting access to protected resources based on an authorization previously arranged with the authorization server.



# Obtaining Authorization

## Technical requirements

### HTTP

The endpoints and methods described hereafter all use HTTP as application-level protocol.

### Transport Layer Security

Client applications MUST use TLS (i.e. HTTPS) communication with the authorization and resource endpoints to ensure protection of credentials and access tokens.

## Authorization code grant flow

The [authorization code grant type](https://tools.ietf.org/html/rfc6749#section-4.1) is used to obtain both access tokens and refresh tokens and is optimized for confidential clients.

Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

### Authorization code request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

***response\_type***

REQUIRED. Value MUST be set to "code".

***client\_id***

REQUIRED. The client identifier as described in « Definitions ».

**redirect\_uri**

OPTIONAL. As described in « Definitions » under « redirection endpoint ».

***scope***

OPTIONAL. The scope of the access request as described « Definitions » under « access token scope».

***state***

RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter is used for preventing cross-site request forgery attacks.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using TLS:

GET /authorize?response\_type=code&client\_id=s6BhdRkqt3&state=xyz HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5

The authorization server validates the request to ensure that all required parameters are present and valid.

If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

### Authorization code response

If the resource owner grants the access request, the authorization server issues an authorization code and delivers it to the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format:

***code***

The authorization code generated by the authorization server. The authorization code expires shortly after it is issued to mitigate the risk of leaks. The client MUST NOT use the authorization code more than once. If an authorization code is used more than once, the authorization server denies the request and revokes (when possible) all tokens previously issued based on that authorization code. The authorization code is bound to the client identifier and redirection URI.

***state***

The exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=xyz

The client MUST ignore unrecognized response parameters. The authorization code string size is left undefined by this specification. The client should avoid making assumptions about code value sizes.

### Error response

If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server SHOULD inform the resource owner of the error and MUST NOT automatically redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format:

***error***

A single ASCII error code from the following:

*invalid*\_*request*

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.

*unauthorized\_client*

The client is not authorized to request an authorization code using this method.

*access\_denied*

The resource owner or authorization server denied the request.

*unsupported\_response\_type*

The authorization server does not support obtaining an authorization code using this method.

*invalid\_scope*

The requested scope is invalid, unknown, or malformed.

*server\_error*

The authorization server encountered an unexpected condition that prevented it from fulfilling the request (needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect).

*temporarily\_unavailable*

The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server (needed because a 503 Service Unavailable HTTP status code cannot be returned the client via an HTTP redirect).

Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

***error\_description***

Human-readable ASCII text providing additional information, used to assist the client developer in understanding the error that occurred.

***error\_uri***

A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.

***state***

The exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: https://client.example.com/cb?error=access\_denied&state=xyz

### Access token request

The client makes a request to the token endpoint by sending the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

***grant\_type***

REQUIRED. Value MUST be set to "authorization\_code".

***code***

REQUIRED. The authorization code received from the server.

***redirect\_uri***

REQUIRED, if and only if the "redirect\_uri" parameter was included in the authorization request and their values MUST be identical.

***client\_assertion\_type***

REQUIRED for a confidential client. Value must be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”

***client\_assertion***

REQUIRED for a confidential client. A signed JWT that authenticates the client with the authorization server as described in [RFC 7519](https://tools.ietf.org/html/rfc7519). The content of the JWT is described in section “Client authentification”.

***client\_id***

REQUIRED for public clients. The client identifier as described in “Definitions”.

For example, the client makes the following HTTP request using TLS:

POST /token HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5/token

Content-Type: application/x-www-form-urlencoded

grant\_type=authorization\_code&code=SplxlOBeZQQYbYS6WxSbIA&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&client\_assertion\_type=…&client\_assertion=

### Access token response

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token.

If the request client authentication failed or is invalid, the authorization server returns an error response.

**access\_token**

The access token issued by the authorization server.

**token\_type**

The type of the token issued. The token type is “Bearer”

**expires\_in**

The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.

**refresh\_token**

OPTIONAL. The refresh token, which can be used to obtain new access tokens using the same authorization grant as described in “Refreshing access token”

**scope**

OPTIONAL, if identical to the scope requested by the client; otherwise, REQUIRED.

An example successful response:

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"2YotnFZFEjr1zCsicMWpAA",

"token\_type":"Bearer",

"expires\_in":3600,

"refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA"

}

An example unsuccessful response:

HTTP/1.1 400 Bad Request

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"error":"invalid\_request",

"error\_description":"Request was missing the client\_id parameter.",

"error\_uri":”https://tools.ietf.org/html/rfc6749”,

}

### Refreshing access token

If the authorization server issued a refresh token to the client, the client makes a refresh request to the token endpoint by adding the following parameters using the "application/x-www-form-urlencoded" with a character encoding of UTF-8 in the HTTP request entity-body:

***grant\_type***

REQUIRED. Value MUST be set to "refresh\_token".

***refresh\_token***

REQUIRED. The refresh token issued to the client.

***scope***

OPTIONAL. The scope of the access request.The requested scope MUST NOT include any scope not originally granted by the resource owner, and if omitted is treated as equal to the scope originally granted by the resource owner.

***client\_assertion\_type***

REQUIRED for confidential clients. Value must be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”

***client\_assertion***

REQUIRED for confidential clients. A signed JWT that authenticates the client with the authorization server as described in [RFC 7519](https://tools.ietf.org/html/rfc7519). The content of the JWT is described in section “Client authentification”.

For example, the client makes the following HTTP request using transport-layer security:

POST /token HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5

Content-Type: application/x-www-form-urlencoded

grant\_type=refresh\_token&refresh\_token=tGzv3JOkF0XG5Qx2TlKWIA&client\_assertion\_type=…&client\_assertion=…

If valid and authorized, the authorization server issues an access token. If the request failed verification or is invalid, the authorization server returns an error response (See “Access token response”).

The authorization server MAY issue a new refresh token, in which case the client MUST discard the old refresh token and replace it with the new refresh token.

The authorization server MAY revoke the old refresh token after issuing a new refresh token to the client.

If a new refresh token is issued, the refresh token scope is identical to that of the refresh token included by the client in the request.

## Implicit grant flow

The [implicit grant type](https://tools.ietf.org/html/rfc6749#section-4.2) is used to obtain access tokens (it does not support the issuance of refresh tokens) and is optimized for public clients known to operate a particular redirection URI.

These clients are typically implemented in a browser using a scripting language such as JavaScript.

Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

Unlike the authorization code grant type, in which the client makes separate requests for authorization and for an access token, the client receives the access token as the result of the authorization request.

The implicit grant type does not include client authentication, and relies on the presence of the resource owner and the registration of the redirection URI.

Because the access token is encoded into the redirection URI, it may be exposed to the resource owner and other applications residing on the same device.

### Authorization request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

***response\_type***

REQUIRED. Value MUST be set to "token".

***client\_id***

REQUIRED. The client identifier as described in « Definitions ».

***redirect\_uri***

OPTIONAL. As described in « Definitions » under « redirection endpoint ».

***scope***

OPTIONAL. The scope of the access request as described « Definitions » under « access token scope».

***state***

RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter is used for preventing cross-site request forgery attacks.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using TLS:

GET /authorize?response\_type=token&client\_id=example:client&state=xyz HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5

The authorization server validates the request to ensure that all required parameters are present and valid. The authorization server also verifies that the optional redirection URI (if present in request) to which it will redirect the access token matches a redirection URI registered by the client.

If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

### Access token response

If the resource owner grants the access request, the authorization server issues an access token and delivers it to the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format:

***access\_token***

The access token issued by the authorization server.

***token\_type***

The case insensitive type of the token issued as described in [RFC6749 section 7.1](https://tools.ietf.org/html/rfc6749#section-7.1).

***expires\_in***

The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.

***scope***

The scope of the access token which is omitted if identical to the scope requested by the client.

***state***

The exact value received from the client.

The authorization server will NOT issue a refresh token.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: http://example.com/cb#access\_token=2YotnFZFEjr1zCsicMWpAA

 &state=xyz&token\_type=example&expires\_in=3600

The client MUST ignore unrecognized response parameters and avoid making assumptions about value sizes. Wali response parameters are passed as query param instead of in the fragment.

### Error response

If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server informs the resource owner of the error and does not automatically redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format:

***error***

 A single ASCII error code from the following:

*invalid\_request*

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.

*unauthorized\_client*

The client is not authorized to request an access token using this method.

*access\_denied*

The resource owner or authorization server denied the request.

*unsupported\_response\_type*

The authorization server does not support obtaining an access token using this method.

*invalid\_scope*

The requested scope is invalid, unknown, or malformed.

*server\_error*

The authorization server encountered an unexpected condition that prevented it from fulfilling the request (this error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect).

*temporarily\_unavailable*

The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server (this error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect).

Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

***error\_description***

Human-readable ASCII text providing additional information, used to assist the client developer in understanding the error that occurred.

***error\_uri***

A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.

***state***

The exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: https://client.example.com/cb#error=access\_denied&state=xyz

## Client Credentials grant flow

The client can request an access token using only its client credentials (or other supported means of authentication) when the client is requesting access to the protected resources under its control, or those of another resource owner that have been previously arranged with the authorization server (the method of which is beyond the scope of this specification).

The [client credentials grant type](https://tools.ietf.org/html/rfc6749#section-4.4) must only be used by confidential clients.

### Authorization request and response

Since the client authentication is used as the authorization grant, no additional authorization request is needed.

### Access token request

The client makes a request to the token endpoint by adding the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

**grant\_type**

REQUIRED. Value MUST be set to "client\_credentials".

**scope**

OPTIONAL. The scope of the access request.

***client\_assertion\_type***

REQUIRED. Value must be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”

***client\_assertion***

REQUIRED. A signed JWT that authenticates the client with the authorization server as described in [RFC 7519](https://tools.ietf.org/html/rfc7519). The content of the JWT is described in section “Client authentification”.

 For example, the client makes the following HTTP request using

 transport-layer security (with extra line breaks for display purposes

 only):

 POST /token HTTP/1.1

 Host: https://services.socialsecurity.be/REST/oauth/v5

 Content-Type: application/x-www-form-urlencoded

 grant\_type=client\_credentials&client\_assertion\_type=…&client\_assertion=…

### Access token response

If the access token request is valid and authorized, the authorization server issues an access token. A refresh token is never included. If the request failed client authentication or is invalid, the authorization server returns an error response.

The response contains the following:

***access\_token***

The access token issued by the authorization server.

***token\_type***

The case insensitive type of the token issued as described in [RFC6749 section 7.1](https://tools.ietf.org/html/rfc6749#section-7.1).

***expires\_in***

The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.

An example successful response:

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"2YotnFZFEjr1zCsicMWpAA",

"token\_type":"Bearer",

"expires\_in":3600

}

### Error response

The authorization server responds with an HTTP 400 (Bad Request)

status code (unless specified otherwise) and includes the following

parameters with the response:

**error**

REQUIRED. A single ASCII [USASCII] error code from the following:

*invalid\_request*

The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.

*invalid\_client*

Client authentication failed (e.g., unknown client, no client authentication included, or unsupported

authentication method). The authorization server MAY return an HTTP 401 (Unauthorized) status code to indicate which HTTP authentication schemes are supported. If the client attempted to authenticate via the "Authorization" request header field, the authorization server MUST respond with an HTTP 401 (Unauthorized) status code and include the "WWW-Authenticate" response header field matching the authentication scheme used by the client.

*invalid\_grant*

The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client.

*unauthorized\_client*

The authenticated client is not authorized to use this authorization grant type.

*unsupported\_grant\_type*

The authorization grant type is not supported by the authorization server.

*invalid\_scope*

The requested scope is invalid, unknown, malformed, or exceeds the scope granted by the resource owner.

Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

**error\_description**

OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error\_description" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

**error\_uri**

OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the "error\_uri" parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.

The parameters are included in the entity-body of the HTTP response using the "application/json" media type as defined by [RFC4627]. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can

vary.

An example unsuccessful response:

HTTP/1.1 400 Bad Request

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"error":"invalid\_request",

"error\_description":"Request was missing the client\_id parameter.",

"error\_uri":”https://tools.ietf.org/html/rfc6749”,

}

## Client Authentification

A signed JWT that authenticates the client with the authorization server as described in [RFC 7519](https://tools.ietf.org/html/rfc7519). The content of the JWT is described in section “Client authentification”

**jti**

REQUIRED. The "jti" (JWT ID) claim provides a unique identifier for the JWT. The "jti" claim can be used to prevent the JWT from being replayed. The "jti" value is a case- sensitive string.

**iss**

REQUIRED. The "iss" (issuer) claim identifies the principal that issued the JWT. The processing of this claim is generally application specific. The "iss" value is a case-sensitive string containing a StringOrURI value.

**sub**

REQUIRED. The "sub" (subject) claim identifies the principal that is the subject of the JWT. The claims in a JWT are normally statements about the subject. The subject value MUST either be scoped to be locally unique in the context of the issuer or be globally unique. The processing of this claim is generally application specific. The "sub" value is a case-sensitive string containing a StringOrURI value.

**aud**

REQUIRED. The "aud" (audience) claim identifies the recipients that the JWT is intended for. Each principal intended to process the JWT MUST identify itself with a value in the audience claim.

**exp**

REQUIRED. The "exp" (expiration time) claim identifies the expiration time on or after which the JWT MUST NOT be accepted for processing. The processing of the "exp" claim requires that the current date/time MUST be before the expiration date/time listed in the "exp" claim. Some small leeway is provided, no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value.

**nbf**

REQUIRED. The "nbf" (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. Some small leeway is provided, no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value.

**iat**

REQUIRED. The "iat" (issued at) claim identifies the time at which the JWT was issued. This claim can be used to determine the age of the JWT. Its value MUST be a number containing a NumericDate value.

# Obtaining user informations using OpenID

## OpenId Connect

The primary extension that OpenID Connect makes to OAuth 2.0 to enable End-Users to be Authenticated is the ID Token data structure. The ID Token is a security token that contains Claims about the Authentication of an End-User by an Authorization Server when using a Client, and potentially other requested Claims. The ID Token is represented as a [JSON Web Token (JWT)](http://openid.net/specs/openid-connect-core-1_0.html#JWT) [JWT].

The following Claims are used within the ID Token for all OAuth 2.0 flows used by OpenID Connect:

***iss***

Issuer Identifier for the Issuer of the response. The iss value is a case sensitive URL using the https scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.

***sub***

Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client, e.g., 24400320 or AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4. It MUST NOT exceed 255 ASCII characters in length. The sub value is a case sensitive string.

***aud***

Audience(s) that this ID Token is intended for. It MUST contain the OAuth 2.0 client\_id of the Relying Party as an audience value. It MAY also contain identifiers for other audiences. In the general case, the aud value is an array of case sensitive strings. In the common special case when there is one audience, the aud value MAY be a single case sensitive string.

***exp***

Expiration time on or after which the ID Token MUST NOT be accepted for processing. The processing of this parameter requires that the current date/time MUST be before the expiration date/time listed in the value. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. See [RFC 3339](http://openid.net/specs/openid-connect-core-1_0.html#RFC3339) [RFC3339] for details regarding date/times in general and UTC in particular.

***iat***

Time at which the JWT was issued. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

***auth\_time***

Time when the End-User authentication occurred. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. When a max\_age request is made or when auth\_time is requested as an Essential Claim, then this Claim is REQUIRED; otherwise, its inclusion is OPTIONAL. (The auth\_time Claim semantically corresponds to the OpenID 2.0 [PAPE](http://openid.net/specs/openid-connect-core-1_0.html#OpenID.PAPE) [OpenID.PAPE] auth\_time response parameter.)

***nonce***

String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token. If present in the ID Token, Clients MUST verify that the nonce Claim Value is equal to the value of the nonce parameter sent in the Authentication Request. If present in the Authentication Request, Authorization Servers MUST include a nonce Claim in the ID Token with the Claim Value being the nonce value sent in the Authentication Request. Authorization Servers SHOULD perform no other processing on nonce values used. The nonce value is a case sensitive string.

The ID Token also contains the UserInfo claims alongside the previously described parameters. OpenID Connect Clients use scope values to specify what access privileges are being requested for Access Tokens. The scopes associated with Access Tokens determine what resources will be available in the ID Token.

For OpenID Connect, scopes can be used to request that specific sets of information be made available as Claim Values.

OpenID Connect defines the following scope values that are used to request Claims:

**profile**

This scope value requests access to the End-User's default profile Claims, which are: name, family\_name, given\_name, middle\_name, nickname, preferred\_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated\_at.

***email***

This scope value requests access to the email and email\_verified Claims.

***address***

This scope value requests access to the address Claim.

***phone***

This scope value requests access to the phone\_number and phone\_number\_verified Claims.

The detail of these claims is described in section 5.1 of the [OpenID specification](http://openid.net/specs/openid-connect-core-1_0.html#StandardClaims).

Multiple scope values MAY be used by creating a space delimited, case sensitive list of ASCII scope values.

ID Tokens MAY contain other Claims. Any Claims used that are not understood MUST be ignored.

## Authorization code grant flow with openid

The [authorization code grant type](https://tools.ietf.org/html/rfc6749#section-4.1) is used to obtain both access tokens and refresh tokens and is optimized for confidential clients.

Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

### Authorization code request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

***response\_type***

REQUIRED. Value MUST be set to "code".

***client\_id***

REQUIRED. The client identifier as described in « Definitions ».

**redirect\_uri**

REQUIRED. As described in « Definitions » under « redirection endpoint ».

***scope***

REQUIRED. The scope of the access request as described « Definitions » under « access token scope». Must at least contain “openid”

***state***

RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter is used for preventing cross-site request forgery attacks.

***nonce***

RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter is used for preventing cross-site request forgery attacks.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using TLS:

GET /authorize?response\_type=code&client\_id=s6BhdRkqt3&state=xyz HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5

The authorization server validates the request to ensure that all required parameters are present and valid.

If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

### Authorization code response

If the resource owner grants the access request, the authorization server issues an authorization code and delivers it to the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format:

***code***

The authorization code generated by the authorization server. The authorization code expires shortly after it is issued to mitigate the risk of leaks. The client MUST NOT use the authorization code more than once. If an authorization code is used more than once, the authorization server denies the request and revokes (when possible) all tokens previously issued based on that authorization code. The authorization code is bound to the client identifier and redirection URI.

***state***

The exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=xyz

The client MUST ignore unrecognized response parameters. The authorization code string size is left undefined by this specification. The client should avoid making assumptions about code value sizes.

### Error response

If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server SHOULD inform the resource owner of the error and MUST NOT automatically redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format:

***error***

A single ASCII error code from the following:

*invalid*\_*request*

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.

*interaction\_required*

The Authorization Server requires End-User interaction of some form to proceed. This error MAY be returned when the prompt parameter value in the Authentication Request is none, but the Authentication Request cannot be completed without displaying a user interface for End-User interaction.

*login\_required*

The Authorization Server requires End-User authentication. This error MAY be returned when the promptparameter value in the Authentication Request is none, but the Authentication Request cannot be completed without displaying a user interface for End-User authentication.

*consent\_required*

The Authorization Server requires End-User consent. This error MAY be returned when the prompt parameter value in the Authentication Request is none, but the Authentication Request cannot be completed without displaying a user interface for End-User consent.

*unauthorized\_client*

The client is not authorized to request an authorization code using this method.

*access\_denied*

The resource owner or authorization server denied the request.

*unsupported\_response\_type*

The authorization server does not support obtaining an authorization code using this method.

*invalid\_scope*

The requested scope is invalid, unknown, or malformed.

*server\_error*

The authorization server encountered an unexpected condition that prevented it from fulfilling the request (needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect).

*temporarily\_unavailable*

The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server (needed because a 503 Service Unavailable HTTP status code cannot be returned the client via an HTTP redirect).

Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

***error\_description***

Human-readable ASCII text providing additional information, used to assist the client developer in understanding the error that occurred.

***error\_uri***

A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.

***state***

The exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: https://client.example.com/cb?error=access\_denied&state=xyz

### Access token request

The client makes a request to the token endpoint by sending the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

***grant\_type***

REQUIRED. Value MUST be set to "authorization\_code".

***code***

REQUIRED. The authorization code received from the server.

***redirect\_uri***

REQUIRED, the "redirect\_uri" parameter included in the authorization request and the two values MUST be identical.

***client\_assertion\_type***

REQUIRED. Value must be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”

***client\_assertion***

REQUIRED. A signed JWT that authenticates the client with the authorization server as described in [RFC 7519](https://tools.ietf.org/html/rfc7519). The content of the JWT is described in section “Client authentification”.

For example, the client makes the following HTTP request using TLS:

POST /token HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5/token

Content-Type: application/x-www-form-urlencoded

grant\_type=authorization\_code&code=SplxlOBeZQQYbYS6WxSbIA&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&client\_assertion\_type=…&client\_assertion=

### Access token response

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token.

If the request client authentication failed or is invalid, the authorization server returns an error response.

**access\_token**

The access token issued by the authorization server.

**id\_token**

ID Token value associated with the authenticated session. Its content is described in section “OpenId Connect”

**token\_type**

The type of the token issued. The token type is “Bearer”

**expires\_in**

The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.

**refresh\_token**

OPTIONAL. The refresh token, which can be used to obtain new access tokens using the same authorization grant as described in “Refreshing access token”

**scope**

OPTIONAL, if identical to the scope requested by the client; otherwise, REQUIRED.

An example successful response:

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"2YotnFZFEjr1zCsicMWpAA",

“id\_token”:” eyJhbGciOiJSUzI1NiJ9.eyJhdWQiOiJvYXV0aDI6dGVzdDppb… …”,

"token\_type":"Bearer",

"expires\_in":3600,

"refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA"

}

An example unsuccessful response:

HTTP/1.1 400 Bad Request

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"error":"invalid\_request",

"error\_description":"Request was missing the client\_id parameter.",

"error\_uri":”https://tools.ietf.org/html/rfc6749”,

}

## Implicit grant flow with opened (deprecated)

The latest security recommendations recommends not using this flow (<https://tools.ietf.org/html/draft-ietf-oauth-security-topics-16#section-2.1.2>) The recommended flow is the [authorization code grant flow with a public client with PKCE](#_PKCE_(Proof_Key) (no security needed while contacting the backend to exchange the authorization code and a code challenge to link the frontend and backend requests together)

### Implicit grant flow request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

***response\_type***

REQUIRED. OAuth 2.0 Response Type value that determines the authorization processing flow to be used, including what parameters are returned from the endpoints used. When using the Implicit Flow, this value is id\_token token or id\_token. The meanings of both of these values are defined in OAuth 2.0 Multiple Response Type Encoding Practices [OAuth.Responses]. No Access Token is returned when the value is id\_token.

***client\_id***

REQUIRED. The client identifier as described in « Definitions ».

***redirect\_uri***

REQUIRED. As described in « Definitions » under « redirection endpoint ».

***scope***

REQUIRED. The scope of the access request as described « Definitions » under « access token scope». The scope must contain at least “openid” when using the response\_type “id\_token”.

***state***

RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter is used for preventing cross-site request forgery attacks.

***nonce***

REQUIRED. String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token. Sufficient entropy MUST be present in the nonce values used to prevent attackers from guessing values.

***prompt=none***

OPTIONAL. The Authorization Server MUST NOT display any authentication or consent user interface pages. An error is returned if an End-User is not already authenticated or the Client does not have pre-configured consent for the requested Claims or does not fulfill other conditions for processing the request. The error code will typically be *login\_required*, *interaction\_required.* This can be used as a method to check for existing authentication and/or consent. The Authorization request with this parameter cannot be used in an ajax call. It must be called from an hidden iframe.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using TLS:

GET /authorize?response\_type=token&client\_id=example:client&state=xyz HTTP/1.1

Host: https://services.socialsecurity.be/REST/oauth/v5

The authorization server validates the request to ensure that all required parameters are present and valid. The authorization server also verifies that the optional redirection URI (if present in request) to which it will redirect the access token matches a redirection URI registered by the client.

If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

### Implicit grant flow response

If the resource owner grants the access request, the authorization server issues an access token and delivers it to the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format:

***access\_token***

The access token issued by the authorization server.

***token\_type***

The case insensitive type of the token issued as described in [RFC6749 section 7.1](https://tools.ietf.org/html/rfc6749#section-7.1).

**id\_token**

ID Token value associated with the authenticated session. Its content is described in section “OpenId Connect”

***expires\_in***

The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.

***scope***

The scope of the access token which is omitted if identical to the scope requested by the client.

***state***

The exact value received from the client.

The authorization server will NOT issue a refresh token.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: http://example.com/cb#access\_token=2YotnFZFEjr1zCsicMWpAA

 &state=xyz&token\_type=example&expires\_in=3600

The client MUST ignore unrecognized response parameters and avoid making assumptions about value sizes. Wali response parameters are passed as query param instead of in the fragment.

### Error response

If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server informs the resource owner of the error and does not automatically redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format:

***error***

A single ASCII error code from the following:

*invalid*\_*request*

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.

*interaction\_required*

The Authorization Server requires End-User interaction of some form to proceed. This error MAY be returned when the prompt parameter value in the Authentication Request is none, but the Authentication Request cannot be completed without displaying a user interface for End-User interaction.

*login\_required*

The Authorization Server requires End-User authentication. This error MAY be returned when the promptparameter value in the Authentication Request is none, but the Authentication Request cannot be completed without displaying a user interface for End-User authentication.

*consent\_required*

The Authorization Server requires End-User consent. This error MAY be returned when the prompt parameter value in the Authentication Request is none, but the Authentication Request cannot be completed without displaying a user interface for End-User consent.

*unauthorized\_client*

The client is not authorized to request an authorization code using this method.

*access\_denied*

The resource owner or authorization server denied the request.

*unsupported\_response\_type*

The authorization server does not support obtaining an authorization code using this method.

*invalid\_scope*

The requested scope is invalid, unknown, or malformed.

*server\_error*

The authorization server encountered an unexpected condition that prevented it from fulfilling the request (needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect).

*temporarily\_unavailable*

The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server (needed because a 503 Service Unavailable HTTP status code cannot be returned the client via an HTTP redirect).

Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

***error\_description***

Human-readable ASCII text providing additional information, used to assist the client developer in understanding the error that occurred.

***error\_uri***

A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.

***state***

The exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

HTTP/1.1 302 Found

Location: https://client.example.com/cb#error=access\_denied&state=xyz

## Accessing userinfo endpoint

The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User. To obtain the requested Claims about the End-User, the Client makes a request to the UserInfo Endpoint using an Access Token obtained through OpenID Connect Authentication. These Claims are normally represented by a JSON object that contains a collection of name and value pairs for the Claims.

### Userinfo request

The Client sends the UserInfo Request using either HTTP GET or HTTP POST. The Access Token obtained from an OpenID Connect Authentication Request MUST be sent as a Bearer Token, per Section 2 of [OAuth 2.0 Bearer Token Usage](https://tools.ietf.org/html/rfc6750) [RFC6750].

The request use the HTTP GET method and the Access Token be sent using the Authorization header field.

The following is a non-normative example of a UserInfo Request:

GET /userinfo HTTP/1.1

Host: <https://services.socialsecurity.be/REST/openIdConnect/v1>

Authorization: Bearer SlAV32hkKG

### Userinfo response

The UserInfo Claims are returned as the members of a JSON object. OpenID Connect Clients use scope values to specify what access privileges are being requested for Access Tokens. The scopes associated with Access Tokens determine what resources will be available when they are used to access OAuth 2.0 protected endpoints.

For OpenID Connect, scopes can be used to request that specific sets of information be made available as Claim Values.

OpenID Connect defines the following scope values that are used to request Claims:

**profile**

This scope value requests access to the End-User's default profile Claims, which are: name, family\_name, given\_name, middle\_name, nickname, preferred\_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated\_at.

***email***

This scope value requests access to the email and email\_verified Claims.

***address***

This scope value requests access to the address Claim.

***phone***

This scope value requests access to the phone\_number and phone\_number\_verified Claims.

The detail of these claims is described in section 5.1 of the [OpenID specification](http://openid.net/specs/openid-connect-core-1_0.html#StandardClaims).

Multiple scope values MAY be used by creating a space delimited, case sensitive list of ASCII scope values.

### Error response

When an error condition occurs, the UserInfo Endpoint returns an Error Response as defined in Section 3 of [OAuth 2.0 Bearer Token Usage](http://openid.net/specs/openid-connect-core-1_0.html#RFC6750) [RFC6750]. (HTTP errors unrelated to RFC 6750 are returned to the User Agent using the appropriate HTTP status code.)

The following is a non-normative example of a UserInfo Error Response:

 HTTP/1.1 401 Unauthorized

 WWW-Authenticate: error="invalid\_token", error\_description="The Access Token expired"

# OAuth for desktop and mobile applications

This flow is based on the [RFC8252](https://tools.ietf.org/html/rfc8252) which defines the flow to be used for Native applications.

For authorizing users in native apps, the best current practice is to perform the OAuth authorization request in an external user-agent (typically the browser) rather than an embedded user-agent (such as one implemented with web-views).

Native app authorization requests that use the browser are more secure and can take advantage of the user's authentication state. Being able to use the existing authentication session in the browser enables single sign-on, as users don't need to authenticate to the authorization server each time they use a new app (unless required by the authorization server policy).

This flow is based on the Authorization code public client grant flow with PKCE and is composed of the following steps:

1. Client app opens a browser tab with the authorization request.
2. Authorization endpoint receives the authorization request, authenticates the user, and obtains authorization. Authenticating the user may involve chaining to other authentication systems.
3. Authorization server issues an authorization code to the redirect URI.
4. Client receives the authorization code from the redirect URI.
5. Client app presents the authorization code at the token endpoint.
6. Token endpoint validates the authorization code and issues the tokens requested.

## Initiating the Authorization Request from a Native App

Native apps needing user authorization create an authorization request URI with the authorization code grant type with a public client using a redirect URI capable of being received by the native app.

Public native app clients MUST implement the Proof Key for Code Exchange (PKCE [RFC7636]) extension to OAuth.

## Receiving the Authorization Response in a Native App

There are several redirect URI options available to native apps for receiving the authorization response from the browser, the availability and user experience of which varies by platform.

Native apps MAY use whichever redirect option suits their needs best, taking into account platform-specific implementation details.

### Private-Use URI Scheme Redirection (For mobile app)

Many mobile and desktop computing platforms support inter-app communication via URIs by allowing apps to register private-use URI schemes (sometimes colloquially referred to as "custom URL schemes") like "com.example.app". When the browser or another app attempts to load a URI with a private-use URI scheme, the app that registered it is launched to handle the request.

To perform an OAuth 2.0 authorization request with a private-use URI scheme redirect, the native app launches the browser with a standard authorization request, but one where the redirection URI utilizes a private-use URI scheme it registered with the operating system.

When choosing a URI scheme to associate with the app, apps MUST use a URI scheme based on a domain name under their control, expressed in reverse order, as recommended by Section 3.8 of [RFC7595] for private-use URI schemes.

For example, an app that controls the domain name "app.example.com" can use "com.example.app" as their scheme. Some authorization servers assign client identifiers based on domain names, for example, "client1234.usercontent.example.net", which can also be used as the domain name for the scheme when reversed in the same manner. A scheme such as "myapp", however, would not meet this requirement, as it is not based on a domain name.

When there are multiple apps by the same publisher, care must be taken so that each scheme is unique within that group. On platforms that use app identifiers based on reverse-order domain names, those identifiers can be reused as the private-use URI scheme for the OAuth redirect to help avoid this problem.

Following the requirements of Section 3.2 of [RFC3986], as there is no naming authority for private-use URI scheme redirects, only a single slash ("/") appears after the scheme component. A complete example of a redirect URI utilizing a private-use URI scheme is:

com.example.app:/oauth2redirect/example-provider

When the authorization server completes the request, it redirects to the client's redirection URI as it would normally. As the redirection URI uses a private-use URI scheme, it results in the operating system launching the native app, passing in the URI as a launch parameter. Then, the native app uses normal processing for the authorization response.

### Claimed "https" Scheme URI Redirection (For mobile app)

Some operating systems allow apps to claim "https" scheme [RFC7230] URIs in the domains they control. When the browser encounters a claimed URI, instead of the page being loaded in the browser, the native app is launched with the URI supplied as a launch parameter.

Such URIs can be used as redirect URIs by native apps. They are indistinguishable to the authorization server from a regular web- based client redirect URI. An example is:

https://app.example.com/oauth2redirect/example-provider

As the redirect URI alone is not enough to distinguish public native app clients from confidential web clients, it is REQUIRED in Section 8.4 that the client type be recorded during client registration to enable the server to determine the client type and act accordingly.

App-claimed "https" scheme redirect URIs have some advantages compared to other native app redirect options in that the identity of the destination app is guaranteed to the authorization server by the operating system. For this reason, native apps SHOULD use them over the other options where possible.

### Loopback Interface Redirection (For Desktop application)

Native apps that are able to open a port on the loopback network interface without needing special permissions (typically, those on desktop operating systems) can use the loopback interface to receive the OAuth redirect.

Loopback redirect URIs use the "http" scheme and are constructed with the loopback IP literal and whatever port the client is listening on.

That is, "http://127.0.0.1:{port}/{path}" for IPv4, and "http://[::1]:{port}/{path}" for IPv6. An example redirect using the IPv4 loopback interface with a randomly assigned port:

http://127.0.0.1:51004/oauth2redirect/example-provider

An example redirect using the IPv6 loopback interface with a randomly assigned port:

http://[::1]:61023/oauth2redirect/example-provider

The authorization server allows any port to be specified at the time of the request for loopback IP redirect URIs, to accommodate clients that obtain an available ephemeral port from the operating system at the time of the request.

Clients SHOULD NOT assume that the device supports a particular version of the Internet Protocol.

## Fat Client userguide (For Desktop java application)

The fatClient library is used to access the OAuth authentication server from a desktop java application to get:

* An access token
* A refresh token
* The list of scopes accepted by the user
* The SSIN of the user
* The name of the user
* The CBE of the organization

To get this info, the fatClient library uses the OAuth authentication code grant flow with PKCE.

For this, the fatClient starts a local browser and calls the OAuth authentication server in this browser. This way, the resource owner can authenticate himself and give the client application his consent to the requested scopes. A local server, started by the fatClient, will wait for the authorization code send by the authentication server. With this code the fatClient will call the token and/or userinfo service of the OAuth server to collect the access/refresh token and optionally the user info.

**SSIN , CBE and the name of the user are only accessible with scope openid and profile.**

##### Maven Dependency

To use the OAuth fatClient, add this maven dependency to your application:

 <dependency>

 <groupId>be.smals.oauth2</groupId>

 <artifactId>oauth2-fatclient</artifactId>

 <version>3.0.0</version>

 </dependency>

This lib is compiled in java 8

Once the maven dependency is added to the project, the OAuth fatClient can be used. To use the fatClient in your application, it has to be configured first. This is an example:

List<String> scopes = new ArrayList<String>();

scopes.add("openid");

scopes.add("profile");

FatClientConfiguration clientConfiguration = new FatClientConfiguration()

 .setAuthorizationServerUrl("http://oauth-v3-mock.int.ext.socialsecurity.be/oauth2/authorize")

 .setClient\_id("oauth2:test:feature:fat:client")

 .setScopes(scopes);

Here are the parameters of this library:

***authorizationServerUrl***

REQUIREDThe url of the oauth authorization server that will supply the authorization code.

* INT (impersonate) : <http://oauth-v5-mock.int.socialsecurity.be/oauth2/authorize>
* INT: <https://oauth-v5.int.socialsecurity.be/oauth2/authorize>
* ACC: <https://oauth-v5.acc.socialsecurity.be/oauth2/authorize>
* PRD: <https://oauth-v5.socialsecurity.be/oauth2/authorize>

***client\_id***

REQUIREDThe client ID of the client that wants to access the protected resource.

***scopes***

REQUIRED The requested scope(s). If you need the inss, the firstName, the lastName or the CBE of the user, the list of scope MUST contain scopes openid and profil.

***browser***

OPTIONAL The browser used to authorize of the resource owner to grant access the protected resource. The default value is DEFAULT. Possible values are:

* IEXPLORE
* FIREFOX
* CHROME
* DEFAULT (the default browser configured by the system)

***successResponseUrl***

OPTIONAL At the end when the fatClient has done its job (after the resource owner has granted his consent), when there are no problems detected, the browser will be redirected to this success page. This value is optional. When not specified, the browser will be redirected to a default success response portal page. It is possible to define your own success response page to inform the user of the result of the authentication process. Some params will be added to this redirect URL by the fatClient:

* language = nl / fr / de

***errorResponseUrl***

OPTIONAL At the end when the fatClient has done its job when there are problems detected, the browser will be redirected to this errorpage.

This value is optional. When not specified, the browser will be redirected to a default error response portal page. It is possible to define your own error response page to inform the user of the problem of the authentication process. Some params will be added to this redirect URL by the fatClient:

* language = nl / fr / de
* error\_code = <The error code>
* error\_message = <Error explanation>
* error\_uri = <Error URI>

***privateMode*** (true/false)

 OPTIONAL When true, the browser will be opened in private (incognito for Chrome) mode. This value is optional. The default is false.

***relativeCallbackUrl*** ("/codeCallback")

 OPTIONAL After successful authentication and after the resource owner has given his consent, the oauth server will call the local fatClient server with this relative callback url. This value is optional. The default value is "/codeCallback”.

***tokenEndpointUrl***

OPTIONAL The endpoint of the token service that will supply the access/refresh tokens. This value is optional. When not set, de default URL endpoint location that corresponds to the authorization server set by setAuthorizationServerUrl will be used.

***userInfoEndpointUrl***

OPTIONAL The endpoint of the userinfo service that will supply the user info when opened and profile scopes where requested. This value is optional. When not set, de default URL endpoint location that corresponds to the authorization server set by setAuthorizationServerUrl will be used.

**serverPortRange**(5000, 6000)

 OPTIONAL The fatClient will launch a local server that will wait for the authorization code from the authorization server. The port this server will listen to will be chosen in this range (the first available). This setter is optional. The default values are between 5000 and 6000.

**localhost**

 OPTIONAL Localhost name to be used for redirection to local server to collect the authorization code. When not set (=default), “127.0.0.1” is used for all browsers except for default browser. For default system browser, “localhost” is used because the Edge browser (possibly the default browser) does not accept 127.0.0.1. Possible settings are:

* LOCALHOST (=”localhost” )
* LOCALHOST\_IPV4 (=”127.0.0.1 )
* LOCALHOST\_IPV6 (=”[::1]” )
* DEFAULT

## Using the fatClient

To use the fatClient, create the AuthenticationService (=fatClient) with the AuthenticationServiceFactory class using the config object from the previous chapter as getInstance parameter and call the authenticate method. This results in an AuthenticationResponse:

AuthenticationService fatClient = AuthenticationServiceFactory.getInstance(clientConfiguration);

AuthenticationResponse response = fatClient.authenticate();

System.out.println("Access\_token: " + response.getAccess\_token());

System.out.println("Client\_id: " + response.getClient\_id());

System.out.println("Expires\_at: " + response.getExpires\_at());

System.out.println("Token\_type: " + response.getToken\_type());

System.out.println("Expires\_in: " + response.getExpires\_in());

System.out.println("Scopes: " + response.getScopes());

**DO NOT save the access\_token String !!**

When you need the access\_token you **MUST use AuthenticationResponse#getAccess\_token().** This méthode refresh the token when it expire.

### Properties AuthenticationResponse

***Access****\_****token***

The OAuth access token.

***RefreshToken***

The OAuth refresh token that can be used to get a new access token when the access token is expired.

***Token\_type***

The token type (e.g. Bearer)

***Expires\_at***

Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire.

***Expires\_in***

The number of seconds before expiration of this token.

***Client\_id***

The ID of the client that wants to access the protected resource.

***Scopes***

The accepted scopes.

***OauthIdentity***

The identity of the resource owner. This info is only returned when the requested scopes contain "profile" and "openid". Available identity attributes are:

* firstName
* lastName
* niss
* cbe
* userType
* CompanyName
* ErrorCode: Only when problem with authentication (unauthorized client, technical problem,…)
* ErrorMessage: Error message explaining the problem.

***ErrorUri***

Error URI.

When the client application tries to get the access token (from the AuthenticationResponse)  and the access token has expired, the fatClient will automatically get a new one from the OAuth authentication server using the refresh token.

### Logging

The OAuth fatClient uses slf4j for logging. Most logging done by the is done at DEBUG level.

To activate the logging in your application, you have to add an slf4j implementation to  your applications, like:

 <dependency>

 <groupId>org.slf4j</groupId>

 <artifactId>slf4j-simple</artifactId>

 <version>1.7.26</version>

 </dependency>

To view the debug logs, change de logging level to DEBUG. In case of slf4j-simple, this can be done by adding this system property:

    -Dorg.slf4j.simpleLogger.defaultLogLevel=debug

 or by adding a logging config file (with name simplelogger.properties) to the classpath of your application. This file should contain at least:

 # Default logging detail level for all instances of SimpleLogger.

 # Must be one of ("trace", "debug", "info", "warn", or "error").

 # If not specified, defaults to "info".

 org.slf4j.simpleLogger.defaultLogLevel=debug

Other slf4j implementations (like logback or jdk) are of course possible: more information can be found here:

<https://dzone.com/articles/how-configure-slf4j-different>

# Token introspection

The introspection endpoint is an OAuth 2.0 endpoint that takes a parameter representing an OAuth 2.0 token and returns a JSON [RFC7159] document representing the meta information surrounding the token, including whether this token is currently active.

The definition of an active token is dependent upon the authorization server, but this is commonly a token that has been issued by this authorization server, is not expired, has not been revoked, and is valid for use at the protected resource making the introspection call.

## Introspection request

The protected resource calls the introspection endpoint using an HTTP POST [RFC7231] request with parameters sent as "application/x-www-form-urlencoded" data as defined in [W3C.REC-html5-20141028]. The protected resource sends a parameter representing the token along with optional parameters representing additional context that is known by the protected resource to aid the authorization server in its response.

***token***

REQUIRED. The string value of the token. For access tokens, this is the "access\_token" value returned from the token endpoint defined in OAuth 2.0 [RFC6749], Section 5.1. For refresh tokens, this is the "refresh\_token" value returned from the token endpoint as defined in OAuth 2.0 [RFC6749],

***token\_type\_hint***

OPTIONAL. A hint about the type of the token submitted for introspection. The protected resource MAY pass this parameter to help the authorization server optimize the token lookup. If the server is unable to locate the token using the given hint, it MUST extend its search across all of its supported token types. Values for this field are either “access\_token” or “refresh\_token”.

To prevent token scanning attacks, the endpoint MUST also require some form of authorization to access this endpoint using a separate OAuth 2.0 access token such as the bearer token described in OAuth 2.0 Bearer Token Usage [RFC6750].

The following is a non-normative example request:

 POST /introspect HTTP/1.1

 Host: https://services.socialsecurity.be/REST/oauth/v5/

 Accept: application/json

 Content-Type: application/x-www-form-urlencoded

 Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

 token=kbfqzpuibhvoqizfqvmn&token\_type\_hint=access\_token

## Introspection response

The server responds with a JSON object [RFC7159] in "application/

json" format with the following top-level members.

***active***

REQUIRED. Boolean indicator of whether or not the presented token is currently active. The specifics of a token's "active" state will vary depending on the implementation of the authorization server and the information it keeps about its tokens, but a "true" value return for the "active" property will generally indicate that a given token has been issued by this authorization server, has not been revoked by the resource owner, and is within its given time window of validity (e.g., after its issuance time and before its expiration time).

***scope***

OPTIONAL. A JSON string containing a space-separated list of scopes associated with this token, in the format described in Section 3.3 of OAuth 2.0 [RFC6749].

***client\_id***

OPTIONAL. Client identifier for the OAuth 2.0 client that requested this token.

***username***

OPTIONAL. Human-readable identifier for the resource owner who authorized this token.

***token\_type***

OPTIONAL. Type of the token as defined in Section 5.1 of OAuth 2.0 [RFC6749].

***exp***

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire, as defined in JWT [RFC7519].

***iat***

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token was originally issued, as defined in JWT [RFC7519].

***nbf***

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token is not to be used before, as defined in JWT [RFC7519].

***sub***

OPTIONAL. Subject of the token, as defined in JWT [RFC7519]. Usually a machine-readable identifier of the resource owner who authorized this token.

***aud***

OPTIONAL. Service-specific string identifier or list of string identifiers representing the intended audience for this token, as defined in JWT [RFC7519].

***iss***

OPTIONAL. String representing the issuer of this token, as defined in JWT [RFC7519].

***jti***

OPTIONAL. String identifier for the token, as defined in JWT [RFC7519].

This structure is extended with the principals attributes of the resource owner.

The authorization server MAY respond differently to different protected resources making the same request. For instance, an authorization server MAY limit which scopes from a given token are returned for each protected resource to prevent a protected resource from learning more about the larger network than is necessary for its operation.

The response MAY be cached by the protected resource to improve performance and reduce load on the introspection endpoint, but at the cost of liveness of the information used by the protected resource to make authorization decisions.

# JWKS Uri

The public keys of the Authorization Sever are exposed as a JSON Web Key Set (JWKS). This JWKS is published on an URL (the “JWKS URI”). The public keys included in the JWKS should be used to verify any JWT issued by the authorization server.

***"kty" (Key Type) Parameter*** The "kty" (key type) parameter identifies the cryptographic algorithm family used with the key, such as "RSA" or "EC”.

***"use" (Public Key Use) Parameter***

 The "use" (public key use) parameter identifies the intended use of the public key. The "use" parameter is employed to indicate whether a public key is used for encrypting data or verifying the signature on data.

 Values defined by this specification are:

* "sig" (signature)
* "enc" (encryption)

 Other values MAY be used. The "use" value is a case-sensitive string.

***"key\_ops" (Key Operations) Parameter***

 The "key\_ops" (key operations) parameter identifies the operation(s)

 for which the key is intended to be used.

### *"kid" (Key ID) Parameter*

 The "kid" (key ID) parameter is used to match a specific key. This is used, for instance, to choose among a set of keys within a JWK Set during key rollover.

### *"n" Parameter*

 The "n" parameter is the modulus of the RSA public and private keys. It is represented as a base64url-encoded value.

### *"e" Parameter*

 The "e" parameter is the exponent of the RSA public key. It is represented as a base64url-encoded value.

All details are in the [rfc 7517](https://tools.ietf.org/html/rfc7517).

# PKCE (Proof Key for Code Exchange by OAuth Public Clients)

OAuth 2.0 [[RFC6749](https://tools.ietf.org/html/rfc6749)] public clients are susceptible to the authorization code interception attack.

In this attack, the attacker intercepts the authorization code returned from the authorization endpoint within a communication path not protected by Transport Layer Security (TLS), such as inter-application communication within the client's operating system.

Once the attacker has gained access to the authorization code, it can use it to obtain the access token.

To mitigate this attack, this extension utilizes a dynamically created cryptographically random key called "code verifier". A unique code verifier is created for every authorization request, and its transformed value, called "code challenge", is sent to the authorization server to obtain the authorization code. The authorization code obtained is then sent to the token endpoint with the "code verifier", and the server compares it with the previously received request code so that it can perform the proof of possession of the "code verifier" by the client. This works as the mitigation since the attacker would not know this one-time key, since it is sent over TLS and cannot be intercepted.

## Protocol

### Client creates a Code Verifier

The client first creates a code verifier, "code\_verifier", for each OAuth 2.0 [[RFC6749](https://tools.ietf.org/html/rfc6749)] Authorization Request, in the following manner:

code\_verifier = high-entropy cryptographic random STRING using the unreserved characters [A-Z] / [a-z] / [0-9] / "-" / "." / "\_" / "~" from [Section 2.3 of [RFC3986]](https://tools.ietf.org/html/rfc3986#section-2.3), with a minimum length of 43 characters and a maximum length of 128 characters.

### Client Creates the Code Challenge

The client then creates a code challenge derived from the code verifier by using one of the following transformations on the code verifier:

**plain**

 code\_challenge = code\_verifier

**S256**

 code\_challenge = BASE64URL-ENCODE(SHA256(ASCII(code\_verifier)))

If the client is capable of using "S256", it MUST use "S256", as "S256" is Mandatory To Implement (MTI) on the server. Clients are permitted to use "plain" only if they cannot support "S256" for some technical reason and know via out-of-band configuration that the server supports "plain".

###  Client Sends the Code Challenge with the Authorization Request

The client sends the code challenge as part of the OAuth 2.0 Authorization Request ([Section 4.1.1 of [RFC6749]](https://tools.ietf.org/html/rfc6749#section-4.1.1)) using the following additional parameters:

 **code\_challenge**

 REQUIRED. Code challenge.

 **code\_challenge\_method**

 OPTIONAL. Defaults to "plain" if not present in the request. Code verifier transformation method is "S256" or "plain".

### Server Returns the Code

When the server issues the authorization code in the authorization response, it MUST associate the "code\_challenge" and "code\_challenge\_method" values with the authorization code so it can be verified later.

###  Client Sends the Authorization Code and the Code Verifier to the Token Endpoint

Upon receipt of the Authorization Code, the client sends the Access Token Request to the token endpoint. In addition to the parameters defined in the OAuth 2.0 Access Token Request ([Section 4.1.3 of [RFC6749]](https://tools.ietf.org/html/rfc6749#section-4.1.3)), it sends the following parameter:

 **code\_verifier**

 REQUIRED. Code verifier

The "code\_challenge\_method" is bound to the Authorization Code when the Authorization Code is issued. That is the method that the token endpoint MUST use to verify the "code\_verifier".

###  Server Verifies code\_verifier before Returning the Tokens

Upon receipt of the request at the token endpoint, the server verifies it by calculating the code challenge from the received "code\_verifier" and comparing it with the previously associated "code\_challenge", after first transforming it according to the "code\_challenge\_method" method specified by the client.

If the "code\_challenge\_method" from [Section 4.3](https://tools.ietf.org/html/rfc7636#section-4.3) was "S256", the received "code\_verifier" is hashed by SHA-256, base64url-encoded, and then compared to the "code\_challenge". If the "code\_challenge\_method" from [Section 4.3](https://tools.ietf.org/html/rfc7636#section-4.3) was "plain", they are compared directly.

If the values are equal, the token endpoint MUST continue processing as normal (as defined by OAuth 2.0 [[RFC6749](https://tools.ietf.org/html/rfc6749)]). If the values are not equal, an error response indicating "invalid\_grant" as described in [Section 5.2 of [RFC6749]](https://tools.ietf.org/html/rfc6749#section-5.2) MUST be returned.

# Environments

## Integration

The different endpoints are:

* Back channel:[**https://services-int.socialsecurity.be/REST/oauth/v5/token**](https://services-int.socialsecurity.be/REST/oauth/v5/token)
* Front channel: **<https://oauth-v5.int.socialsecurity.be/oauth2/authorize>**
* Introspect : **<https://services-int.socialsecurity.be/REST/oauth/v5/introspect>**
* Userinfo : **<https://services-int.socialsecurity.be/REST/openIdConnect/v1/userinfo>**
* JWKS :[**https://services-int.socialsecurity.be/REST/oauth/v5/jwks**](https://services-int.socialsecurity.be/REST/oauth/v5/jwks)

The expected audience in JWT authentication is **<https://oauth-v5.int.socialsecurity.be>**The expected opaque JWT issuer is **<https://oauth-v5.int.pub.socialsecurity.be>**

## Acceptation

The different endpoints are:

* Back channel: **<https://services-acpt.socialsecurity.be/REST/oauth/v5/token>**
* Front channel: **<https://oauth-v5.acc.socialsecurity.be/oauth2/authorize>**
* Introspect: **<https://services-acpt.socialsecurity.be/REST/oauth/v5/introspect>**
* Userinfo: **<https://services-acpt.socialsecurity.be/REST/openIdConnect/v1/userinfo>**
* JWKS:[**https://oauth-v5.acc.socialsecurity.be/oauth2/api/jwks**](https://oauth-v5.acc.socialsecurity.be/oauth2/api/jwks)

The expected audience in JWT authentication is **<https://oauth-v5.acc.socialsecurity.be>**The expected opaque JWT issuer is **<https://oauth-v5.acc.socialsecurity.be>**

## Production

The different endpoints are:

* Back channel: **<https://services.socialsecurity.be/REST/oauth/v5/token>**
* Front channel: **<https://oauth-v5.socialsecurity.be/oauth2/authorize>**
* Introspect: **<https://services.socialsecurity.be/REST/oauth/v5/introspect>**
* Userinfo: **<https://services.socialsecurity.be/REST/openIdConnect/v1/userinfo>**
* JWKS:[**https://oauth-v5.socialsecurity.be/oauth2/api/jwks**](https://oauth-v5.socialsecurity.be/oauth2/api/jwks)

The expected audience in JWT authentication is **<https://oauth-v5.socialsecurity.be>**The expected opaque JWT issuer is **<https://oauth-v5.socialsecurity.be>**

# External document references

* The OAuth 2.0 Authorization Framework : <https://tools.ietf.org/html/rfc6749>
* OpenID Connect specification : <http://openid.net/connect/>