|  |
| --- |
| **[TOC](" \l "toc)** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Draft | N. Sakimura | |  | NRI | |  | J. Bradley | |  | Ping Identity | |  | M. Jones | |  | Microsoft | |  | B. de Medeiros | |  | Google | |  | E. Jay | |  | Illumila | |  | May 30, 2013 | |

# OpenID Connect Standard 1.0 - draft 20

### Abstract

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

OpenID Connect Standard 1.0 is an HTTP protocol binding for the OpenID Connect Messages 1.0 request and response messages.

### Table of Contents

[**1.**](#Introduction)  Introduction  
    [**1.1.**](#rnc)  Requirements Notation and Conventions  
    [**1.2.**](#terminology)  Terminology  
[**2.**](#AuthorizationEndpoint)  Authorization Endpoint  
    [**2.1.**](#protocol_flows)  Protocol Flows  
        [**2.1.1.**](#retrieving_code_token)  Obtaining the Authorization Code, ID Token, and Access Token  
        [**2.1.2.**](#code_flow)  Authorization Code Flow  
        [**2.1.3.**](#implicit_flow)  Implicit Flow  
    [**2.2.**](#auth_request)  Authorization Request  
        [**2.2.1.**](#AuthorizationRequest)  Client Prepares Authorization Request  
            [**2.2.1.1.**](#RequestParameters)  Request Parameters  
        [**2.2.2.**](#RequestMethods)  Request Methods  
            [**2.2.2.1.**](#SimpleRequestMethod)  Simple Request Method  
                [**2.2.2.1.1.**](#norm_req)  Client Sends Simple Request  
            [**2.2.2.2.**](#req_param_method)  Request Parameter Method  
                [**2.2.2.2.1.**](#request_req)  Client Sends Request using "request" Parameter  
            [**2.2.2.3.**](#Request_File_Method)  Request File Method  
                [**2.2.2.3.1.**](#rurl_create)  Client Generates the URL of the Request File  
                [**2.2.2.3.2.**](#art_req)  Client Sends Request using "request\_uri" Parameter  
                [**2.2.2.3.3.**](#FetchesRequestFile)  Authorization Server Fetches Request File  
        [**2.2.3.**](#ValidatesRequestObject)  Authorization Server Validates Request Object  
        [**2.2.4.**](#Authenticates)  Authorization Server Authenticates End-User  
        [**2.2.5.**](#Consent)  Authorization Server Obtains End-User Consent/Authorization  
        [**2.2.6.**](#art_res)  Authorization Server Sends End-User Back to Client  
            [**2.2.6.1.**](#art_res_ok)  End-User Grants Authorization  
            [**2.2.6.2.**](#authz_error)  End-User Denies Authorization or Invalid Request  
[**3.**](#token_ep)  Token Endpoint  
    [**3.1.**](#RequestingAccessToken)  Requesting Access Token  
        [**3.1.1.**](#AccessTokenRequest)  Access Token Request  
        [**3.1.2.**](#AccessTokenResponse)  Access Token Response  
        [**3.1.3.**](#AccessTokenErrorResponse)  Access Token Error Response  
    [**3.2.**](#RefreshingAccessToken)  Refreshing Access Token  
        [**3.2.1.**](#RefreshTokenResponse)  Refresh Token Response  
        [**3.2.2.**](#RefreshTokenErrorResponse)  Refresh Token Error Response  
[**4.**](#userinfo)  UserInfo Endpoint  
    [**4.1.**](#UserInfoRequest)  UserInfo Request  
    [**4.2.**](#UserInfoResponse)  UserInfo Response  
    [**4.3.**](#UserInfoErrorResponse)  UserInfo Error Response  
[**5.**](#self_issued)  Self-Issued OpenID Provider  
[**6.**](#client_Initiate_login)  Initiating Login at Client from Third Party  
[**7.**](#Serializations)  Serializations  
    [**7.1.**](#qss)  Query String Serialization  
    [**7.2.**](#form_serialization)  Form Serialization  
    [**7.3.**](#js)  JSON Serialization  
[**8.**](#ImplementationConsiderations)  Implementation Considerations  
    [**8.1.**](#disco_reg)  Discovery and Registration  
[**9.**](#security_considerations)  Security Considerations  
    [**9.1.**](#ImplicitGrantFlowThreats)  Implicit Grant Flow Threats  
    [**9.2.**](#TLS_requirements)  TLS Requirements  
[**10.**](#privacy_considerations)  Privacy Considerations  
[**11.**](#IANA)  IANA Considerations  
[**12.**](#rfc.references1)  References  
    [**12.1.**](#rfc.references1)  Normative References  
    [**12.2.**](#rfc.references2)  Informative References  
[**Appendix A.**](#Acknowledgements)  Acknowledgements  
[**Appendix B.**](#Notices)  Notices  
[**Appendix C.**](#History)  Document History  
[**§**](#rfc.authors)  Authors' Addresses

|  |
| --- |
| [**TOC**](#toc) |

### 1.  Introduction

The [OAuth 2.0 Authorization Framework (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749] and [OAuth 2.0 Bearer Token Usage (Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.)](#RFC6750) [RFC6750] specifications provide a general framework for third-party applications to obtain and use limited access to HTTP resources. They define mechanisms to obtain and use Access Tokens to access resources but do not define standard methods to provide identity information. Notably, without profiling OAuth 2.0, it is incapable of providing information about the authentication of an End-User.

This specification is a binding of the messages defined in the [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] specification to RFC6749 and RFC6750, including messages that provide identity and authentication information, allowing services to securely exchange identity information. This binding builds an identity layer on top of OAuth 2.0. Using this specification, deployments are able to share authentication and attributes on OAuth 2.0 based systems.

|  |
| --- |
| [**TOC**](#toc) |

### 1.1.  Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.)](#RFC2119) .

Throughout this document, values are quoted to indicate that they are to be taken literally. When using these values in protocol messages, the quotes MUST NOT be used as part of the value.

All uses of [JSON Web Signature (JWS) (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)](#JWS) [JWS] and [JSON Web Encryption (JWE) (Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.)](#JWE) [JWE] data structures in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used.

|  |
| --- |
| [**TOC**](#toc) |

### 1.2.  Terminology

This specification uses the terms "Access Token", "Refresh Token", "Authorization Code", "Authorization Grant", "Authorization Server", "Authorization Endpoint", "Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", and "Token Endpoint" defined by [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749], and the terms defined by [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

This specification also defines the following terms:

Request File

Document whose content is a Request Object representing a set of Authorization Request parameters.

Request File URI

URL that references a Request File. The Request File contents MUST be retrievable by the Authorization Server.

|  |
| --- |
| [**TOC**](#toc) |

### 2.  Authorization Endpoint

The Authorization Endpoint performs authentication for the End-User and requests authorization from the End-User to release information to OpenID Connect Relying Party (Client). When an End-User accesses a Relying Party that requires the End-User's identifier and other information, it sends the End-User to the Authorization Server's Authorization Endpoint for authentication and authorization. The Authorization Server then issues an ID Token that asserts the End-User's identity and an Access Token that allows the Client to access the End-User's information at Protected Resource endpoints. Protected Resource endpoints MAY perform different actions or return different information based on the scopes associated with the presented Access Token. Clients MUST specify how the Access Token and ID Token are to be returned by using the response\_type parameter in the Authorization Request.

|  |
| --- |
| [**TOC**](#toc) |

### 2.1.  Protocol Flows

Authorization Requests follow two main paths to obtain Access Tokens and ID Tokens, the Implicit Flow and the Authorization Code Flow. The flows determine how the Access Token and ID Token are returned to the Client. Access Tokens are credentials used to access Protected Resources, as defined in Section 1.4 of [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749]. Access Tokens represent a Resource Owner's authorization and MUST NOT be exposed to unauthorized parties.

The Implicit Flow is mainly used by Clients implemented in a browser using a scripting language. The Access Token and ID Token are returned directly to the Client, which MAY expose them to the Resource Owner and other applications that have access to the Resource Owner's User-Agent. The Authorization Server does not perform Client Authentication before issuing the Access Token.

The Authorization Code Flow returns an Authorization Code to the Client, which can then exchange it for an Access Token directly. This provides the added benefit of not exposing the Access Token to the Resource Owner and possibly other malicious applications with access to the Resource Owner's User-Agent. The Authorization Server can also authenticate the Client before exchanging the Authorization Code for an Access Token. The Authorization Code flow is suitable for Clients that can securely maintain a Client Secret between themselves and the Authorization Server whereas the Implicit flow is suitable for Clients that cannot.

|  |
| --- |
| [**TOC**](#toc) |

### 2.1.1.  Obtaining the Authorization Code, ID Token, and Access Token

In this specification, the Client sends the Authorization Request to the Authorization Endpoint through the User-Agent to obtain the ID Token and Access Token. It can obtain them from the Token Endpoint utilizing the Authorization Code that it obtained from the Authorization Endpoint using the Authorization Code Flow or from the Authorization Endpoint using the Implicit Flow.

|  |
| --- |
| [**TOC**](#toc) |

### 2.1.2.  Authorization Code Flow

The Authorization Code Flow goes through the following steps.

1. Client prepares an Authorization Request containing the desired request parameters.
2. Client sends a request to the Authorization Server.
3. Authorization Server Authenticates the End-User.
4. Authorization Server Obtains the End-User Consent/Authorization.
5. Authorization Server Sends the End-User back to the Client with an Authorization Code.
6. Client requests a response using the Authorization Code at the Token Endpoint, per [Section 3 (Token Endpoint)](#token_ep).
7. Client receives a response that contains an Access Token and ID Token in the response body.
8. Client validates the ID Token and retrieves the End-User's subject identifier.
9. (OPTIONAL) Client accesses the UserInfo Endpoint with the Access Token, per [Section 4 (UserInfo Endpoint)](#userinfo).
10. (OPTIONAL) Client receives UserInfo Response.

Note that in each step, the party that receives a message MUST validate it according to the validation rules in [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

|  |
| --- |
| [**TOC**](#toc) |

### 2.1.3.  Implicit Flow

The Implicit Flow follows the following steps:

1. Client prepares an Authorization Request containing the desired request parameters.
2. Client sends a request to the Authorization Server.
3. Authorization Server Authenticates the End-User.
4. Authorization Server Obtains the End-User Consent/Authorization.
5. Authorization Server Sends the End-User back to the Client with an Access Token and an ID Token if requested.
6. Client validates the ID Token and retrieves the End-User's subject identifier.
7. (OPTIONAL) Client accesses the UserInfo Endpoint with the Access Token, per [Section 4 (UserInfo Endpoint)](#userinfo).
8. (OPTIONAL) Client receives UserInfo Response.

Note that in each step, the party that receives a message MUST validate it according to the validation rules in [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.  Authorization Request

When the Client wishes to access a Protected Resource and the End-User Authorization has not yet been obtained, the Client prepares an Authorization Request to the Authorization Endpoint.

Communication with the Authorization Endpoint MUST utilize TLS. See [Section 9.2 (TLS Requirements)](#TLS_requirements) for more information on using TLS.

Authorization Servers MUST support the use of the HTTP GET and POST methods defined in [RFC 2616 (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)](#RFC2616) [RFC2616] at the Authorization Endpoint.

Clients MAY use the HTTP GET or POST methods to send the Authorization Request to the Authorization Server. If using the HTTP GET method, the request parameters are serialized using URI Query String Serialization, per [Section 7.1 (Query String Serialization)](#qss). If using the HTTP POST method, the request parameters are serialized using Form Serialization, per [Section 7.2 (Form Serialization)](#form_serialization).

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.1.  Client Prepares Authorization Request

The Client prepares an Authorization Request to the Authorization Endpoint with the request parameters using the HTTP GET or POST method. The scheme used in the Authorization URL MUST be https.

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.1.1.  Request Parameters

OpenID Connect uses the following OAuth 2.0 request parameters:

response\_type

REQUIRED. OAuth 2.0 registered response type value that determines how the Authorization Response is returned to the Client. As described in [OAuth 2.0 Multiple Response Type Encoding Practices (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)](#OAuth.Responses) [OAuth.Responses], the following registered values are supported by OpenID Connect:

* code
* code id\_token
* id\_token
* token id\_token
* code token
* code token id\_token

client\_id

REQUIRED. OAuth 2.0 Client Identifier.

scope

REQUIRED. Space delimited, case sensitive list of ASCII OAuth 2.0 scope values. OpenID Connect requests MUST contain the openid scope value. OPTIONAL scope values of profile, email, address, phone, and offline\_access are also defined. Section 2.4 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] defines the OpenID Connect scope values.

redirect\_uri

REQUIRED. Redirection URI to which the response will be sent. The Scheme, Host, Path, and Query Parameter segments of this URI MUST match one of the redirect\_uris registered for the client\_id in the [OpenID Connect Dynamic Client Registration 1.0 (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)](#OpenID.Registration) [OpenID.Registration] specification.

state

RECOMMENDED. Opaque value used to maintain state between the request and the callback. Typically, Cross-Site Request Forgery mitigation is done through cryptographically binding the value of this parameter with the browser cookiee.

This specification also uses the following request parameters. Refer to [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] for more information about these parameters.

nonce

REQUIRED or OPTIONAL. 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 Authorization Request to the ID Token. Use of the nonce is REQUIRED when using the implicit flow and OPTIONAL when using the code flow.

display

OPTIONAL. ASCII string value that specifies how the Authorization Server displays the authentication and consent user interface pages to the End-User. The defined values are: page, popup, touch, and wap.

prompt

OPTIONAL. Space delimited, case sensitive list of ASCII string values that specifies whether the Authorization Server prompts the End-User for reauthentication and consent. The defined values are: none, login, consent, and select\_account.

max\_age

OPTIONAL. Maximum Authentication Age. Specifies that the End-User MUST be actively authenticated if the End-User was authenticated longer ago than the specified number of seconds.

ui\_locales

OPTIONAL. End-User's preferred languages and scripts for the user interface, represented as a space-separated list of [BCP47 (Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.)](#RFC5646) [RFC5646] language tag values, ordered by preference.

claims\_locales

OPTIONAL. End-User's preferred languages and scripts for Claims being returned, represented as a space-separated list of [BCP47 (Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.)](#RFC5646) [RFC5646] language tag values, ordered by preference.

id\_token\_hint

OPTIONAL. Previously issued ID Token passed to the Authorization Server as a hint about the End-User this request is about.

login\_hint

OPTIONAL. Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary).

acr\_values

OPTIONAL. Requested Authentication Context Class Reference values. Space-separated string that specifies the acr values that the Authorization Server MUST use for processing requests from this Client.

claims

OPTIONAL. This parameter is used to request that specific Claims be returned. The value is a JSON object listing the requested Claims.

registration

OPTIONAL. This parameter is used by the Client to provide information about itself to a Self-Issued OP that would normally be provided to an OP during Dynamic Client Registration.

request

OPTIONAL. Request Object value.

request\_uri

OPTIONAL. URL that references a resource containing a Request Object value.

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.  Request Methods

There are three methods to construct and send the request to the Authorization Endpoint:

1. Simple Request Method
2. Request Parameter Method
3. Request File Method

The Simple Request Method can be used in cases where signed or encrypted requests are not needed and where the size of the request does not exceed limits imposed by User-Agents.

The Request Parameter Method is used when the Client wants or needs to send an OpenID Connect request as a single, self-contained Request Object value. This method enables requests to be signed and optionally encrypted. Like the Simple Request Method, some requests using this method can exceed limits imposed by User-Agents.

The Request File Method works similarly to the Request Parameter Method but differs in that it sends a URL as a reference to the Request Object. It enables large requests to be sent securely and compactly even on User-Agents with limited capabilities. Clients MAY use the Request File Method to minimize the request size.

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.1.  Simple Request Method

The Client prepares an Authorization Request to the Authorization Endpoint using the appropriate parameters. If using the HTTP GET method, the request parameters are serialized using URI Query String Serialization, per [Section 7.1 (Query String Serialization)](#qss). If using the HTTP POST method, the request parameters are serialized using Form Serialization, per [Section 7.2 (Form Serialization)](#form_serialization).

The following is a non-normative example of an Authorization Request URL (with line wraps within the values for display purposes only):

https://server.example.com/op/authorize?

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.1.1.  Client Sends Simple Request

Having constructed the Authorization Request, the Client sends it to the Authorization Endpoint using HTTPS.

Following is a non-normative example using HTTP redirect (with line wraps within the values for display purposes only):

HTTP/1.1 302 Found

Location: https://server.example.com/authorize?

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.2.  Request Parameter Method

The Client prepares an Authorization Request to the Authorization Endpoint using the appropriate HTTP parameter serialization. The Client SHOULD construct the request using the HTTP POST method, but MAY use the HTTP GET method.

The Authorization Request MUST include the request parameter defined in [Section 2.2.1.1 (Request Parameters)](#RequestParameters). The Authorization Request MUST NOT include the request\_uri parameter.

The request parameter is a Request Object represented as a [JWT (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)](#JWT) [JWT] containing a set of OpenID Connect request parameters. The Request Object MAY be a Plaintext JWT, signed, or signed and encrypted using [JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)](#JWS) [JWS] and/or [JWE (Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.)](#JWE) [JWE], thereby enabling authentication, integrity, non-repudiation, and/or confidentiality to be achieved.

The following is a non-normative example of the Claims in a Request Object before base64url encoding and signing:

{

"response\_type": "code id\_token",

"client\_id": "s6BhdRkqt3",

"redirect\_uri": "https://client.example.org/cb",

"scope": "openid",

"state": "af0ifjsldkj",

"nonce": "n-0S6\_WzA2Mj",

"max\_age": 86400,

"claims":

{

"userinfo":

{

"given\_name": {"essential": true},

"nickname": null,

"email": {"essential": true},

"email\_verified": {"essential": true},

"picture": null

},

"id\_token":

{

"gender": null,

"birthdate": {"essential": true},

"acr": {"values": ["urn:mace:incommon:iap:silver"]}

}

}

}

Signing it with the RS256 algorithm results in this Request Object value (with line breaks for display purposes only):

eyJhbGciOiJSUzI1NiJ9.ew0KICJyZXNwb25zZV90eXBlIjogImNvZGUgaWRfdG9rZW

4iLA0KICJjbGllbnRfaWQiOiAiczZCaGRSa3F0MyIsDQogInJlZGlyZWN0X3VyaSI6I

CJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsDQogInNjb3BlIjogIm9wZW5p

ZCIsDQogInN0YXRlIjogImFmMGlmanNsZGtqIiwNCiAibm9uY2UiOiAibi0wUzZfV3p

BMk1qIiwNCiAibWF4X2FnZSI6IDg2NDAwLA0KICJjbGFpbXMiOiANCiAgew0KICAgIn

VzZXJpbmZvIjogDQogICAgew0KICAgICAiZ2l2ZW5fbmFtZSI6IHsiZXNzZW50aWFsI

jogdHJ1ZX0sDQogICAgICJuaWNrbmFtZSI6IG51bGwsDQogICAgICJlbWFpbCI6IHsi

ZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJlbWFpbF92ZXJpZmllZCI6IHsiZXNzZW5

0aWFsIjogdHJ1ZX0sDQogICAgICJwaWN0dXJlIjogbnVsbA0KICAgIH0sDQogICAiaW

RfdG9rZW4iOiANCiAgICB7DQogICAgICJnZW5kZXIiOiBudWxsLA0KICAgICAiYmlyd

GhkYXRlIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgImFjciI6IHsidmFsdWVz

IjogWyIyIl19DQogICAgfQ0KICB9DQp9.bOD4rUiQfzh4QPIs\_f\_R2GVBhNHcc1p2cQ

TgixB1tsYRs52xW4TO74USgb-nii3RPsLdfoPlsEbJLmtbxG8-TQBHqGAyZxMDPWy3p

hjeRt9ApDRnLQrjYuvsCj6byu9TVaKX9r1KDFGT-HLqUNlUTpYtCyM2B2rLkWM08ufB

q9JBCEzzaLRzjevYEPMaoLAOjb8LPuYOYTBqshRMUxy4Z380-FJ2Lc7VSfSu6HcB2nL

SjiKrrfI35xkRJsaSSmjasMYeDZarYCl7r4o17rFclk5KacYMYgAs-JYFkwab6Dd56Z

rAzakHt9cExMpg04lQIux56C-Qk6dAsB6W6W91AQ

The following is the RSA public key in JWK format that can be used to validate the Request Object signature in the above example (with line breaks for display purposes only):

{

"kty":"RSA",

"n":"y9Lqv4fCp6Ei-u2-ZCKq83YvbFEk6JMs\_pSj76eMkddWRuWX2aBKG

HAtKlE5P7\_vn\_\_PCKZWePt3vGkB6ePgzAFu08NmKemwE5bQI0e6kIChtt\_6KzT5OaaXDF

I6qCLJmk51Cc4VYFaxgqevMncYrzaW\_50mZ1yGSFIQzLYP8bijAHGVjdEFgZaZEN9lsn\_

GdWLaJpHrB3ROlS50E45wxrlg9xMncVb8qDPuXZarvghLL0HzOuYRadBJVoWZowDNTpKp

k2RklZ7QaBO7XDv3uR7s\_sf2g-bAjSYxYUGsqkNA9b3xVW53am\_UZZ3tZbFTIh557JICW

KHlWj5uzeJXaw",

"e":"AQAB"

}

The following is a non-normative example of an Authorization Request using the request parameter (with line wraps within the values for display purposes only):

https://server.example.com/authorize?

response\_type=code%02id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&state=af0ifjsldkj

&nonce=n-0S6\_WzA2Mj

&request=eyJhbGciOiJSUzI1NiJ9.ew0KICJyZXNwb25zZV90eXBlIjogImNvZG

UgaWRfdG9rZW4iLA0KICJjbGllbnRfaWQiOiAiczZCaGRSa3F0MyIsDQogInJlZG

lyZWN0X3VyaSI6ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsDQogIn

Njb3BlIjogIm9wZW5pZCIsDQogInN0YXRlIjogImFmMGlmanNsZGtqIiwNCiAibm

9uY2UiOiAibi0wUzZfV3pBMk1qIiwNCiAibWF4X2FnZSI6IDg2NDAwLA0KICJjbG

FpbXMiOiANCiAgew0KICAgInVzZXJpbmZvIjogDQogICAgew0KICAgICAiZ2l2ZW

5fbmFtZSI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJuaWNrbmFtZSI6IG

51bGwsDQogICAgICJlbWFpbCI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgIC

JlbWFpbF92ZXJpZmllZCI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJwaW

N0dXJlIjogbnVsbA0KICAgIH0sDQogICAiaWRfdG9rZW4iOiANCiAgICB7DQogIC

AgICJnZW5kZXIiOiBudWxsLA0KICAgICAiYmlydGhkYXRlIjogeyJlc3NlbnRpYW

wiOiB0cnVlfSwNCiAgICAgImFjciI6IHsidmFsdWVzIjogWyIyIl19DQogICAgfQ

0KICB9DQp9.bOD4rUiQfzh4QPIs\_f\_R2GVBhNHcc1p2cQTgixB1tsYRs52xW4TO7

4USgb-nii3RPsLdfoPlsEbJLmtbxG8-TQBHqGAyZxMDPWy3phjeRt9ApDRnLQrjY

uvsCj6byu9TVaKX9r1KDFGT-HLqUNlUTpYtCyM2B2rLkWM08ufBq9JBCEzzaLRzj

evYEPMaoLAOjb8LPuYOYTBqshRMUxy4Z380-FJ2Lc7VSfSu6HcB2nLSjiKrrfI35

xkRJsaSSmjasMYeDZarYCl7r4o17rFclk5KacYMYgAs-JYFkwab6Dd56ZrAzakHt

9cExMpg04lQIux56C-Qk6dAsB6W6W91AQ

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.2.1.  Client Sends Request using "request" Parameter

Having constructed the Authorization Request, the Client sends it to the Authorization Endpoint using HTTPS.

Following is a non-normative example using HTTP redirect (with line wraps within the values for display purposes only):

HTTP/1.1 302 Found

Location: https://server.example.com/authorize?

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&state=af0ifjsldkj

&nonce=n-0S6\_WzA2Mj

&request=eyJhbGciOiJSUzI1NiJ9.ew0KICJyZXNwb25zZV90eXBlIjogImNvZG

UgaWRfdG9rZW4iLA0KICJjbGllbnRfaWQiOiAiczZCaGRSa3F0MyIsDQogInJlZG

lyZWN0X3VyaSI6ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsDQogIn

Njb3BlIjogIm9wZW5pZCIsDQogInN0YXRlIjogImFmMGlmanNsZGtqIiwNCiAibm

9uY2UiOiAibi0wUzZfV3pBMk1qIiwNCiAibWF4X2FnZSI6IDg2NDAwLA0KICJjbG

FpbXMiOiANCiAgew0KICAgInVzZXJpbmZvIjogDQogICAgew0KICAgICAiZ2l2ZW

5fbmFtZSI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJuaWNrbmFtZSI6IG

51bGwsDQogICAgICJlbWFpbCI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgIC

JlbWFpbF92ZXJpZmllZCI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJwaW

N0dXJlIjogbnVsbA0KICAgIH0sDQogICAiaWRfdG9rZW4iOiANCiAgICB7DQogIC

AgICJnZW5kZXIiOiBudWxsLA0KICAgICAiYmlydGhkYXRlIjogeyJlc3NlbnRpYW

wiOiB0cnVlfSwNCiAgICAgImFjciI6IHsidmFsdWVzIjogWyIyIl19DQogICAgfQ

0KICB9DQp9.bOD4rUiQfzh4QPIs\_f\_R2GVBhNHcc1p2cQTgixB1tsYRs52xW4TO7

4USgb-nii3RPsLdfoPlsEbJLmtbxG8-TQBHqGAyZxMDPWy3phjeRt9ApDRnLQrjY

uvsCj6byu9TVaKX9r1KDFGT-HLqUNlUTpYtCyM2B2rLkWM08ufBq9JBCEzzaLRzj

evYEPMaoLAOjb8LPuYOYTBqshRMUxy4Z380-FJ2Lc7VSfSu6HcB2nLSjiKrrfI35

xkRJsaSSmjasMYeDZarYCl7r4o17rFclk5KacYMYgAs-JYFkwab6Dd56ZrAzakHt

9cExMpg04lQIux56C-Qk6dAsB6W6W91AQ

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.3.  Request File Method

The Request File Method differs from the other methods in that it uses a Request File that contains a Request Object. It then sends the Request File URL as part of the Authorization Request.

The Client prepares an Authorization Request using the desired HTTP GET or POST method. The Client SHOULD use the HTTP GET method, but MAY use the HTTP POST method. The scheme used in the Authorization URL MUST be https.

The Authorization Request MUST NOT include the request parameter. The Authorization Request MUST include the request\_uri parameter. The contents of the resource referenced by the URL MUST be a Request Object. The scheme used in the request\_uri value MUST be https, unless the target Request Object is signed in a way that is verifiable by the Authorization Server. The request\_uri value MUST be reachable by the Authorization Server, and SHOULD be reachable by the Client.

Following is a non-normative example of the contents of a Request File (with line wraps within the values for display purposes only):

eyJhbGciOiJSUzI1NiJ9.ew0KICJyZXNwb25zZV90eXBlIjogImNvZGUgaWRfdG9rZ

W4iLA0KICJjbGllbnRfaWQiOiAiczZCaGRSa3F0MyIsDQogInJlZGlyZWN0X3VyaSI

6ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsDQogInNjb3BlIjogIm9wZ

W5pZCIsDQogInN0YXRlIjogImFmMGlmanNsZGtqIiwNCiAibm9uY2UiOiAibi0wUzZ

fV3pBMk1qIiwNCiAibWF4X2FnZSI6IDg2NDAwLA0KICJjbGFpbXMiOiANCiAgew0KI

CAgInVzZXJpbmZvIjogDQogICAgew0KICAgICAiZ2l2ZW5fbmFtZSI6IHsiZXNzZW5

0aWFsIjogdHJ1ZX0sDQogICAgICJuaWNrbmFtZSI6IG51bGwsDQogICAgICJlbWFpb

CI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJlbWFpbF92ZXJpZmllZCI6IHs

iZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJwaWN0dXJlIjogbnVsbA0KICAgIH0sD

QogICAiaWRfdG9rZW4iOiANCiAgICB7DQogICAgICJnZW5kZXIiOiBudWxsLA0KICA

gICAiYmlydGhkYXRlIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgImFjciI6I

HsidmFsdWVzIjogWyIyIl19DQogICAgfQ0KICB9DQp9.bOD4rUiQfzh4QPIs\_f\_R2G

VBhNHcc1p2cQTgixB1tsYRs52xW4TO74USgb-nii3RPsLdfoPlsEbJLmtbxG8-TQBH

qGAyZxMDPWy3phjeRt9ApDRnLQrjYuvsCj6byu9TVaKX9r1KDFGT-HLqUNlUTpYtCy

M2B2rLkWM08ufBq9JBCEzzaLRzjevYEPMaoLAOjb8LPuYOYTBqshRMUxy4Z380-FJ2

Lc7VSfSu6HcB2nLSjiKrrfI35xkRJsaSSmjasMYeDZarYCl7r4o17rFclk5KacYMYg

As-JYFkwab6Dd56ZrAzakHt9cExMpg04lQIux56C-Qk6dAsB6W6W91AQ

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.3.1.  Client Generates the URL of the Request File

The Client stores the Request File either locally or remotely with a URI that the server can reach. This is the Request URI, request\_uri. Servers MAY cache the contents of the files referenced by request URIs. If the contents of the Request File could ever change, the URI SHOULD include the base64url encoded SHA-256 hash of the referenced file contents as the fragment component of the URI. If the fragment value used for a URI changes, that signals the server that any cached value for that URI with the old fragment value is no longer valid.

Note that Clients MAY pre-register request\_uri values using the request\_uris parameter defined in Section 2 of the [OpenID Connect Dynamic Client Registration 1.0 (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)](#OpenID.Registration) [OpenID.Registration] specification. OPs can require that request\_uri values used be pre-registered with the require\_request\_uri\_registration discovery parameter.

If the Request File includes attribute values, it MUST NOT be revealed to anybody but the Authorization Server. As such, the request\_uri MUST have appropriate entropy for its lifetime, and MUST be removed after successful authentication or a reasonable timeout unless there is an access control measures.

The Client then records the Request File either locally or remotely and obtains the Request File URI, request\_uri.

Following is a non-normative example (with line wraps for display purposes only):

https://client.example.org/rf.txt

#GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQOV7PZHAdM

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.3.2.  Client Sends Request using "request\_uri" Parameter

The Client sends the Authorization Request to the Authorization Endpoint.

The entire request URL MUST NOT exceed 512 ASCII characters.

Following is a non-normative example (with line wraps within the values for display purposes only):

HTTP/1.1 302 Found

Location: https://server.example.com/authorize

?response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&request\_uri=https%3A%2F%2Fclient.example.org%2Frf.txt

%23GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO\_V7PZHAdM

&state=af0ifjsldkj&nonce=n-0S6\_WzA2Mj&scope=openid

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.2.3.3.  Authorization Server Fetches Request File

Upon receipt of the Request, the Authorization Server MUST send a GET request to the request\_uri to retrieve the content unless it is already cached and parse it to recreate the Authorization Request parameters.

Note that the RP SHOULD use a unique URI for each request utilizing distinct parameters, or otherwise prevent the Authorization Server from caching the request\_uri.

Following is a non-normative example of this fetch process:

GET /rf.txt HTTP/1.1

Host: client.example.org

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.3.  Authorization Server Validates Request Object

The Authorization Server validates the request according to Section 5.1 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.4.  Authorization Server Authenticates End-User

The Authorization Server validates the request to ensure all REQUIRED parameters are present and all parameters are valid. If the request is valid, the Authorization Server attempts to log in the End-User or determines whether he is logged in, depending upon the request parameter values used. The methods used by the Authorization Server to log in the End-User (e.g. username and password, session cookies, etc.) are beyond the scope of this specification. An authentication user interface MAY be displayed by the Authorization Server, depending upon the request parameter values used and the authentication methods used.

The Authorization Server MUST attempt to log in the End-User in the following cases:

* The End-User is not already logged in.
* The Authorization Request contains the prompt parameter with the value login. In this case, the Authorization Server MUST reauthenticate the End-User even if the End-User is already authenticated.

The Authorization Server MUST NOT interact with the End-User in the following case:

* The Authorization Request contains the prompt parameter with the value none. In this case, the Authorization Server MUST return an error if the End-User is not already logged in or could not be silently logged in.

The Authorization Server MUST employ appropriate measures against Cross-Site Request Forgery and Clickjacking as, described in Sections 10.12 and 10.13 of [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749].

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.5.  Authorization Server Obtains End-User Consent/Authorization

Once the End-User is authenticated, the Authorization Server MUST obtain an authorization decision. This MAY be done by presenting the End-User with a dialogue that allows the End-User to recognize what he is consenting to and obtain his consent or by establishing consent via conditions for processing or other means (for example, via previous administrative consent).

The Authorization Server MUST attempt to request authorization from the End-User in the following cases:

* The End-User has not pre-authorized the Client for the Authorization Request.
* The Authorization Request contains the prompt parameter with the value consent. The Authorization Server SHOULD request End-User authorization even if the End-User has previously authorized the Client.

The Authorization Server MUST NOT request End-User authorization in the following cases:

* The Authorization Request contains the prompt parameter with the value none. The Authorization Server MUST return an error if the End-User has not pre-authorized the Client.

As in the previous section, the Authorization Server MUST employ countermeasures against Cross-Site Request Forgery and Clickjacking when interacting with the End-User.

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.6.  Authorization Server Sends End-User Back to Client

Once the authorization is determined, the Authorization Server returns a successful or error response.

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.6.1.  End-User Grants Authorization

If the Resource Owner grants the access request, the Authorization Server issues an Authorization Response as described in Section 2.1.2 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] to the Client by adding the response parameters to redirect\_uri specified in the Authorization Request using the "application/x-www-form-urlencoded" format.

Note that if the response\_type parameter in the Authorization Request includes the string value token or id\_token, all response parameters SHOULD be added to the fragment component of the redirection URI. Otherwise, the response parameters are added to the query component of the redirection URI.

The Client MUST validate the response as follows:

Case 1: response\_type=code

1. Validate the response according to RFC 6749, especially Sections 4.1.2 and 10.12.

Case 2: response\_type=token id\_token

1. Verify that the response conforms to Section 5 of [[OAuth.Responses] (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)](" \l "OAuth.Responses).
2. Follow the validation rules in RFC 6749, especially those in Sections 4.2.2 and 10.12.
3. Follow the validation rules in Sections 4.2 and 4.4 of [[OpenID.Messages] (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](" \l "OpenID.Messages).

Case 3: response\_type=code id\_token

* 1. Verify that the response conforms to Section 5 of [[OAuth.Responses] (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)](" \l "OAuth.Responses).

1. Follow the validation rules in RFC 6749, especially those in Sections 4.2.2 and 10.12.
2. Follow the validation rules in Sections 4.2 and 4.5 of [[OpenID.Messages] (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](" \l "OpenID.Messages).

Case 4: response\_type=token code

1. Verify that the response conforms to Section 5 of [[OAuth.Responses] (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)](" \l "OAuth.Responses).
2. Follow the validation rules in RFC 6749, especially those in Sections 4.2.2 and 10.12.

Case 5: response\_type=token code id\_token

1. Verify that the response conforms to Section 5 of [[OAuth.Responses] (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)](" \l "OAuth.Responses).
2. Follow the validation rules in RFC 6749, especially those in Sections 4.2.2 and 10.12.
3. Follow the validation rules in Sections 4.2, 4.4, and 4.5 of [[OpenID.Messages] (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](" \l "OpenID.Messages).

Case 6: response\_type=code id\_token

1. Verify that the response conforms to Section 5 of [[OAuth.Responses] (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)](" \l "OAuth.Responses).
2. Follow the validation rules in RFC 6749, especially those in Sections 4.2.2 and 10.12.
3. Follow the validation rules in Sections 4.2 and 4.5 of [[OpenID.Messages] (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](" \l "OpenID.Messages).

The following are non-normative examples of requests with differing response\_type values and their responses (with line wraps within the values for display purposes only):

Case 1: response\_type=code

https://server.example.com/op/authorize?

response\_type=code

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

HTTP/1.1 302 Found

Location: https://client.example.org/cb?

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf11jnpEX3Tgfvk

&state=af0ifjsldkj

Case 2: response\_type=token id\_token

https://server.example.com/op/authorize?

response\_type=token%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

HTTP/1.1 302 Found

Location: https://client.example.org/cb#

access\_token=jHkWEdUXMU1BwAsC4vtUsZwnNvTIxEl0z9K3vx5KF0Y

&token\_type=Bearer

&id\_token=eyJhbGciOiJSUzI1NiJ9.ew0KICJpc3MiOiAiaHR0cDovL3NlcnZlc

i5leGFtcGxlLmNvbSIsDQogInN1YiI6ICIyNDgyODk3NjEwMDEiLA0KICJhdWQiO

iAiczZCaGRSa3F0MyIsDQogIm5vbmNlIjogIm4tMFM2X1d6QTJNaiIsDQogImV4c

CI6IDEzMTEyODE5NzAsDQogImlhdCI6IDEzMTEyODA5NzAsDQogImF0X2hhc2giO

iAiNzdRbVVQdGpQZnpXdEYyQW5wSzlSUSINCn0.g7UR4IDBNIjoPFV8exQCosUNV

eh8bNUTeL4wdQp-2WXIWnly04ZK0sh4A4uddfenzo4Cjh4wuPPrSw6lMeujYbGy

zKspJrRYL3iiYWc2VQcl8RKdHPzG-7yf5enut1YE8v7PhKucPJCRRoobMjqD73f

1nJNwQ9KBrfh21Ggbx1p8hNqQeeLLXb9b63JD84hVOXwyHmmcVgvZskge-wExwnh

Ivv\_cxTzxIXsSxcYlh3d9hnu0wdxPZOGjT0nNZJxvdIwDD4cAT\_LE5Ae447qB90

ZF89Nmb0Oj2b1GdGVQEIr8-FXrHlyD827f0NhLYPdZ73YK6p10qY9oRtMimg

&state=af0ifjsldkj

Verifying and decoding the ID Token will yield the following Claims:

{

"iss": "http://server.example.com",

"sub": "248289761001",

"aud": "s6BhdRkqt3",

"nonce": "n-0S6\_WzA2Mj",

"exp": 1311281970,

"iat": 1311280970,

"at\_hash": "77QmUPtjPfzWtF2AnpK9RQ"

}

Case 3: response\_type=code id\_token

https://server.example.com/op/authorize?

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

HTTP/1.1 302 Found

Location: https://client.example.org/cb#

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf11jnpEX3Tgfvk

&id\_token=eyJhbGciOiJSUzI1NiJ9.ew0KICJpc3MiOiAiaHR0cDovL3NlcnZlc

i5leGFtcGxlLmNvbSIsDQogInN1YiI6ICIyNDgyODk3NjEwMDEiLA0KICJhdWQiO

iAiczZCaGRSa3F0MyIsDQogIm5vbmNlIjogIm4tMFM2X1d6QTJNaiIsDQogImV4c

CI6IDEzMTEyODE5NzAsDQogImlhdCI6IDEzMTEyODA5NzAsDQogImNfaGFzaCI6I

CJMRGt0S2RvUWFrM1BrMGNuWHhDbHRBIg0KfQ.dAVXerlNOJtqMUysD\_k1Q\_bRX

RJbLkTOsCPVxpKUis5V6xMRvtjfRg8gUfPuAMYrKQMEqZZmL87Hxkv6cFKavb4ft

BUrY2qUnrvqe\_bNjVEz89QSdxGmdFwSTgFVGWkDf5dV5eIiRxXfIkmlgCltPNocR

AyvdNrsWC661rHz5F9MzBho2vgi5epUa\_KAl6tK4ksgl68pjZqlBqsWfTbGEsWQX

Efu664dJkdXMLEnsPUeQQLjMhLH7qpZk2ry0nRx0sS1mRwOMQ0Xmps0vOkNn284

pMUpmWEAjqklWITgtVYXOzF4ilbmZK6ONpFyKCpnSkAYtTEuqz-m7MoLCDA

&state=af0ifjsldkj

Verifying and decoding the ID Token will yield the following Claims:

{

"iss": "http://server.example.com",

"sub": "248289761001",

"aud": "s6BhdRkqt3",

"nonce": "n-0S6\_WzA2Mj",

"exp": 1311281970,

"iat": 1311280970,

"c\_hash": "LDktKdoQak3Pk0cnXxCltA"

}

Case 4: response\_type=token code

https://server.example.com/op/authorize?

response\_type=token%20code

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

HTTP/1.1 302 Found

Location: https://client.example.org/cb#

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf\_11jnpEX3Tgfvk

&access\_token=jHkWEdUXMU1BwAsC4vtUsZwnNvTIxEl0z9K3vx5KF0Y

&token\_type=Bearer

&state=af0ifjsldkj

Case 5: response\_type=token code id\_token

https://server.example.com/op/authorize?

response\_type=token%20code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

HTTP/1.1 302 Found

Location: https://client.example.org/cb#

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf\_11jnpEX3Tgfvk

&access\_token=jHkWEdUXMU1BwAsC4vtUsZwnNvTIxEl0z9K3vx5KF0Y

&token\_type=Bearer

&id\_token=eyJhbGciOiJSUzI1NiJ9.ew0KICJpc3MiOiAiaHR0cDovL3NlcnZlc

i5leGFtcGxlLmNvbSIsDQogInN1YiI6ICIyNDgyODk3NjEwMDEiLA0KICJhdWQiO

iAiczZCaGRSa3F0MyIsDQogIm5vbmNlIjogIm4tMFM2X1d6QTJNaiIsDQogImV4c

CI6IDEzMTEyODE5NzAsDQogImlhdCI6IDEzMTEyODA5NzAsDQogImF0X2hhc2giO

iAiNzdRbVVQdGpQZnpXdEYyQW5wSzlSUSIsDQogImNfaGFzaCI6ICJMRGt0S2RvU

WFrM1BrMGNuWHhDbHRBIg0KfQ.JQthrBsOirujair9aD5gj1Yd5qEv0j4fhLgl8h

3RaH3soYhwPOiN2Iyyb7wMCO6I3bPoGJc3zCkpjgUtdB4O2eEhFqXHdwnE4c0oV

TaTHJi\_PdV2ox9g-1ikDB0ckWk0f0SzBd7yM2RoYYxJCiGBQlsSSRQz6ehykonI3

hLAhXFdpfbK-3a3HBNKOv9MrJJrz2pqSygk5IBNvwzf1ouVeM91KKvr7EdriK

N8ysk68fctbFAga1p8rE3cfBOX7Acn4p9QSNpUx0ix4WHktyKDvHhLdUw91Fql

UOgMP\_9h8TYdkAjcq8n1tFzaO7kVaazlZ5SM32J7OSDgNSA

&state=af0ifjsldkj

Verifying and decoding the ID Token will yield the following Claims:

{

"iss": "http://server.example.com",

"sub": "248289761001",

"aud": "s6BhdRkqt3",

"nonce": "n-0S6\_WzA2Mj",

"exp": 1311281970,

"iat": 1311280970,

"at\_hash": "77QmUPtjPfzWtF2AnpK9RQ",

"c\_hash": "LDktKdoQak3Pk0cnXxCltA"

}

This following example makes a request using a request parameter value requesting that specific Claims be returned in the ID Token. The sample Request Object used is described in [Section 2.2.2.2 (Request Parameter Method)](#req_param_method).

Case 6: response\_type=code id\_token

https://server.example.com/op/authorize?

response\_type=code%20id\_token

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

&scope=openid

&request=eyJhbGciOiJSUzI1NiJ9.ew0KICJyZXNwb25zZV90eXBlIjogImNvZG

UgaWRfdG9rZW4iLA0KICJjbGllbnRfaWQiOiAiczZCaGRSa3F0MyIsDQogInJlZG

lyZWN0X3VyaSI6ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsDQogIn

Njb3BlIjogIm9wZW5pZCIsDQogInN0YXRlIjogImFmMGlmanNsZGtqIiwNCiAibm

9uY2UiOiAibi0wUzZfV3pBMk1qIiwNCiAibWF4X2FnZSI6IDg2NDAwLA0KICJjbG

FpbXMiOiANCiAgew0KICAgInVzZXJpbmZvIjogDQogICAgew0KICAgICAiZ2l2ZW

5fbmFtZSI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJuaWNrbmFtZSI6IG

51bGwsDQogICAgICJlbWFpbCI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgIC

JlbWFpbF92ZXJpZmllZCI6IHsiZXNzZW50aWFsIjogdHJ1ZX0sDQogICAgICJwaW

N0dXJlIjogbnVsbA0KICAgIH0sDQogICAiaWRfdG9rZW4iOiANCiAgICB7DQogIC

AgICJnZW5kZXIiOiBudWxsLA0KICAgICAiYmlydGhkYXRlIjogeyJlc3NlbnRpYW

wiOiB0cnVlfSwNCiAgICAgImFjciI6IHsidmFsdWVzIjogWyIyIl19DQogICAgfQ

0KICB9DQp9.bOD4rUiQfzh4QPIsfR2GVBhNHcc1p2cQTgixB1tsYRs52xW4TO7

4USgb-nii3RPsLdfoPlsEbJLmtbxG8-TQBHqGAyZxMDPWy3phjeRt9ApDRnLQrjY

uvsCj6byu9TVaKX9r1KDFGT-HLqUNlUTpYtCyM2B2rLkWM08ufBq9JBCEzzaLRzj

evYEPMaoLAOjb8LPuYOYTBqshRMUxy4Z380-FJ2Lc7VSfSu6HcB2nLSjiKrrfI35

xkRJsaSSmjasMYeDZarYCl7r4o17rFclk5KacYMYgAs-JYFkwab6Dd56ZrAzakHt

9cExMpg04lQIux56C-Qk6dAsB6W6W91AQ

&nonce=n-0S6\_WzA2Mj

&state=af0ifjsldkj

HTTP/1.1 302 Found

Location: https://client.example.org/cb#

code=Qcb0Orv1zh30vL1MPRsbm-diHiMwcLyZvn1arpZv-Jxf\_11jnpEX3Tgfvk

&token\_type=Bearer

&id\_token=eyJhbGciOiJSUzI1NiJ9.ew0KICJpc3MiOiAiaHR0cDovL3NlcnZlc

i5leGFtcGxlLmNvbSIsDQogInN1YiI6ICIyNDgyODk3NjEwMDEiLA0KICJhdWQiO

iAiczZCaGRSa3F0MyIsDQogIm5vbmNlIjogIm4tMFM2X1d6QTJNaiIsDQogImV4c

CI6IDEzMTEyODE5NzAsDQogImlhdCI6IDEzMTEyODA5NzAsDQogImdlbmRlciI6I

CJmZW1hbGUiLA0KICJiaXJ0aGRhdGUiOiAiMDAwMC0xMC0zMSIsDQogImFjciI6I

CIyIiwNCiAiY19oYXNoIjogIkxEa3RLZG9RYWszUGswY25YeENsdEEiDQp9.d8vv

NPuPR01zA\_6RTArFMCtct1Oy6Q0gTCTXa9oDEuk7Sd3Hv0i34M6xFccrrbYtCzVn

AjnJnJCgixJSlMsYJBTx168HgqdBcBwFLmd28i1Q\_7cyfoglw4GkblXn7UxUgdAZ

1ILyDAOV3dMS2ZxaCPK\_HE2-\_kzlIXtOamdWruHickUBA3\_xlzMorMvbmHnU47qA

u3AgCJF9RmKhgVY1cjUiHmEF8GBugxMn6eE7-ji6houeM-WL0nudHJEBikVmpnhQ

ezFyj6tZhfeVcWoEsCwzbDts4pPQVcDHFq2dSnq1ZuhX1svDYnV-TRUW9ri7V9ik

K7TBhtqtk7e5cIWSPw

&state=af0ifjsldkj

Verifying and decoding the ID Token will yield the following Claims:

{

"iss": "http://server.example.com",

"sub": "248289761001",

"aud": "s6BhdRkqt3",

"nonce": "n-0S6\_WzA2Mj",

"exp": 1311281970,

"iat": 1311280970,

"gender": "female",

"birthdate": "0000-10-31",

"acr": "urn:mace:incommon:iap:silver",

"c\_hash": "LDktKdoQak3Pk0cnXxCltA"

}

The following is the RSA public key in JWK format that can be used to validate the Request Object signature in the above examples (with line breaks for display purposes only):

{

"kty":"RSA",

"n":"zhEWTBJVTfcUeqnMzOQFMCEVQWOyOUZwP8LrBWh88tKrZyPGCvBkT

Dp-E2BzyHMQV4pK51Uys2YOwzL9se5THDWMda9rtsCJVcj1V7WaE7wPgl-kIIdWWf4o2g

6ZszOy\_Fp4q0nG3OTtDRCkBu2iEP21j82pRSRrkCBxnzaChflA7KZbI1n\_yhKtxyA7FdA

480LaSVZyKApvrKiYhocACSwf0y6CQ-wkEi6mVXRJt1aBSywlLYA08ojp5hkZQ39eCM2k

1EdXdhbar998Q9PZTwXA1cfvuGTZbDWxEKLjMKVuKrT1Yvs-2NTXhZAW1KjFS\_3UwLkDk

-w4dVN-x5tDnw",

"e":"AQAB"

}

|  |
| --- |
| [**TOC**](#toc) |

### 2.2.6.2.  End-User Denies Authorization or Invalid Request

If the End-User denies the authorization or the user authentication fails, the Authorization Server MUST return the Authorization Error Response as defined in Section 2.1.3 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages]. The Authorization Server returns the Client to the redirection URI specified in the Authorization Request with the appropriate error parameters. No other parameters SHOULD be returned.

The error response parameters are the following:

error

REQUIRED. Error code.

error\_description

OPTIONAL. Human-readable ASCII encoded text description of the error.

error\_uri

OPTIONAL. URI of a web page that includes additional information about the error.

state

OAuth 2.0 state value. REQUIRED if the Authorization Request included the state parameter. Set to the exact value received from the Client.

If the response\_type parameter in the Authorization Request includes the string value token or id\_token, all error response parameters SHOULD be added to the fragment component of the redirection URI. Otherwise, the response parameters are added to the query component of the redirection URI.

The following is a non-normative example (with line wraps within the values for the display purposes only):

HTTP/1.1 302 Found

Location: https://client.example.org/cb?

error=invalid\_request

&error\_description=

the%20request%20is%20not%20valid%20or%20malformed

&state=af0ifjsldkj

|  |
| --- |
| [**TOC**](#toc) |

### 3.  Token Endpoint

The Token Endpoint handles requests for retrieving and refreshing Access Tokens as well as ID Token and other variables.

Clients MUST use the HTTP POST method to make requests to the Token Endpoint. Request parameters are added using Form Serialization, per [Section 7.2 (Form Serialization)](#form_serialization).

Clients MAY provide authentication parameters in the request to the Token Endpoint as described in Section 2.2.1 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

The Token Endpoint MUST support the use of the HTTP POST method defined in [RFC 2616 (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)](#RFC2616) [RFC2616] at the Token Endpoint.

Communication with the Token Endpoint MUST utilize TLS. See [Section 9.2 (TLS Requirements)](#TLS_requirements) for more information on using TLS.

All Token Endpoint responses that contain tokens, secrets, or other sensitive information MUST include the following HTTP response header fields and values:

|  |  |
| --- | --- |
| **Header Name** | **Header Value** |
| Cache-Control | no-store |
| Pragma | no-cache |

|  |
| --- |
| **HTTP Response Headers and Values** |

|  |
| --- |
| [**TOC**](#toc) |

### 3.1.  Requesting Access Token

To retrieve an Access Token when using the Authorization Code Flow, a Client MUST have an Authorization Code obtained as described in [Section 2.1.2 (Authorization Code Flow)](#code_flow).

|  |
| --- |
| [**TOC**](#toc) |

### 3.1.1.  Access Token Request

To obtain an Access Token, Refresh Token or ID Token, the Client MUST authenticate to the Token Endpoint using the authentication method registered for its client\_id, as documented in Section 2.2.1 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages]. The Client sends the parameters via HTTPS POST to the Token Endpoint using Form Serialization, per [Section 7.2 (Form Serialization)](#form_serialization), as described in Section 4.1.3 of [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749].

The following is a non-normative example of an Access Token Request:

POST /token HTTP/1.1

Host: server.example.com

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

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant\_type=authorization\_code&code=SplxlOBeZQQYbYS6WxSbIA

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

The Authorization Server MUST:

* Authenticate any Clients that were issued Client Credentials (or for which other Client Authentication methods can be used),
* Ensure the Authorization Code was issued to the authenticated Client,
* Verify that the Authorization Code is valid, and
* Ensure that the Scheme, Host, Path, and Query Parameter segments in the redirect\_uri parameter are identical to the redirect\_uri parameter value that was included in the initial Authorization Request. If the redirect\_uri parameter value not be present when there is only one registered redirect\_uri value, the Authorization Server MAY return an error (since the Client should have included the parameter) or MAY proceed without an error (since OAuth 2.0 permits the parameter to be omitted in this case).

|  |
| --- |
| [**TOC**](#toc) |

### 3.1.2.  Access Token Response

Upon receipt of the Token Request, the Authorization Server MUST return either a successful response or an error response that corresponds to the received Authorization Code.

A successful response returns the application/json media type and the response body is the Access Token Response documented in Section 2.2.3 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

Following is a non-normative example of a successful response:

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"access\_token": "SlAV32hkKG",

"token\_type": "Bearer",

"refresh\_token": "8xLOxBtZp8",

"expires\_in": 3600,

"id\_token": "eyJhbGciOiJSUzI1NiJ9.ew0KICAgICJpc3MiOiAiaHR0cDovL

3NlcnZlci5leGFtcGxlLmNvbSIsDQogICAgInVzZXJfaWQiOiAiMjQ4Mjg5NzYxM

DAxIiwNCiAgICAiYXVkIjogInM2QmhkUmtxdDMiLA0KICAgICJub25jZSI6ICJuL

TBTNl9XekEyTWoiLA0KICAgICJleHAiOiAxMzExMjgxOTcwLA0KICAgICJpYXQiO

iAxMzExMjgwOTcwDQp9.lsQI\_KNHpl58YY24G9tUHXr3Yp7OKYnEaVpRL0KI4szT

D6GXpZcgxIpkOCcajyDiIv62R9rBWASV191Akk1BM36gUMm8H5s8xyxNdRfBViCa

xTqHA7X\_vV3U-tSWl6McR5qaSJaNQBpg1oGPjZdPG7zWCG-yEJC4-Fbx2FPOS7-h

5V0k33O5Okd-OoDUKoFPMd6ur5cIwsNyBazcsHdFHqWlCby5nl\_HZdW-PHq0gjzy

JydB5eYIvOfOHYBRVML9fKwdOLM2xVxJsPwvy3BqlVKc593p2WwItIg52ILWrc6A

tqkqHxKsAXLVyAoVInYkl\_NDBkCqYe2KgNJFzfEC8g"

}

|  |
| --- |
| [**TOC**](#toc) |

### 3.1.3.  Access Token Error Response

If the Token Request is invalid or unauthorized, the Authorization Server constructs the response by returning the Token Error Response defined in [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] in the entity body of the HTTP response using the application/json media type with HTTP response code 400.

Following is a non-normative example:

HTTP/1.1 400 Bad Request

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"error": "invalid\_request"

}

|  |
| --- |
| [**TOC**](#toc) |

### 3.2.  Refreshing Access Token

To refresh an Access Token, the Client MUST authenticate to the Token Endpoint using the authentication method registered for its client\_id, as documented in Section 2.2.1 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages]. The Client sends the parameters via HTTPS POST to the Token Endpoint using Form Serialization, per [Section 7.2 (Form Serialization)](#form_serialization), as described in Section 6 of [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749]:

The Authorization Server MUST validate the Refresh Token.

|  |
| --- |
| [**TOC**](#toc) |

### 3.2.1.  Refresh Token Response

Upon receipt of the Refresh Token Request, the Authorization Server MUST return either a successful response or an error response that corresponds to the received Refresh Token.

Upon successful validation of the Refresh Token, a successful response returns the application/json media type and the response body is the Access Token Response of Section 2.2.3 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] except that it MUST NOT return an id\_token.

Following is a non-normative example of a Refresh Token Request and response:

POST /token HTTP/1.1

Host: server.example.com

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

client\_id=s6BhdRkqt3

&client\_secret=some\_secret12345

&grant\_type=refresh\_token

&refresh\_token=8xLOxBtZp8

&scope=openid%20profile

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"access\_token": "TlBN45jURg",

"token\_type": "Bearer",

"refresh\_token": "9yNOxJtZa5",

"expires\_in": 3600

}

|  |
| --- |
| [**TOC**](#toc) |

### 3.2.2.  Refresh Token Error Response

If the Refresh Token Request is invalid or unauthorized, the Authorization Server returns the Token Error Response as defined in Section 5.2 of [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749].

|  |
| --- |
| [**TOC**](#toc) |

### 4.  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 GET or POST request to the UserInfo Endpoint as in Section 2.3.1 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

Communication with the UserInfo Endpoint MUST utilize TLS. See [Section 9.2 (TLS Requirements)](#TLS_requirements) for more information on using TLS.

The UserInfo Endpoint MUST support the use of the HTTP GET and HTTP POST methods defined in [RFC 2616 (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)](#RFC2616) [RFC2616] at the UserInfo Endpoint.

The UserInfo Endpoint MUST accept Access Tokens as [OAuth 2.0 Bearer Token Usage (Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.)](#RFC6750) [RFC6750].

The UserInfo Endpoint SHOULD support the use of [Cross Origin Resource Sharing (CORS) (Opera Software ASA, “Cross-Origin Resource Sharing,” July 2010.)](#CORS) [CORS] and or other methods as appropriate to enable Java Script Clients to access the endpoint.

|  |
| --- |
| [**TOC**](#toc) |

### 4.1.  UserInfo Request

Client SHOULD send the UserInfo Request defined in Section 2.3.1 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] either in an HTTPS GET or HTTPS POST request.

The Access Token obtained from an OpenID Connect Authorization Request MUST be sent as a Bearer Token. Section 2 of the [OAuth 2.0 Bearer Token Usage (Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.)](#RFC6750) [RFC6750] specification documents the permissible methods of sending the Access Token.

It is RECOMMENDED that the Client use the Authorization header field method for all requests and that they use the GET method.

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

GET /userinfo HTTP/1.1

Host: server.example.com

Authorization: Bearer SlAV32hkKG

|  |
| --- |
| [**TOC**](#toc) |

### 4.2.  UserInfo Response

The sub (subject) Claim in the UserInfo Endpoint response MUST exactly match the sub Claim in the ID Token, before using additional UserInfo Endpoint Claims.

Upon receipt of the UserInfo request, the UserInfo Endpoint MUST return the JSON Serialization of the UserInfo response as in Section 2.3.2 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] in the HTTP response body unless a different format was specified during Registration [[OpenID.Registration] (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)](#OpenID.Registration). The content-type of the HTTP response MUST be set to application/json if the response body is a text JSON structure; the response body SHOULD be encoded using UTF-8. If the JSON response is signed or encrypted, then the content-type MUST be set to application/jwt.

Upon receipt of the UserInfo Response, the Client MUST verify the response in accordance with Section 5.3 (UserInfo Response Validation) of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

Following is a non-normative example of such response:

HTTP/1.1 200 OK

Content-Type: application/json

{

"sub": "248289761001",

"name": "Jane Doe"

"given\_name": "Jane",

"family\_name": "Doe",

"email": "janedoe@example.com",

"picture": "http://example.com/janedoe/me.jpg"

}

|  |
| --- |
| [**TOC**](#toc) |

### 4.3.  UserInfo Error Response

When an error condition occurs, the UserInfo Endpoint returns an Error Response as defined in Section 3 of the [OAuth 2.0 Bearer Token Usage (Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.)](#RFC6750) [RFC6750] specification utilizing an error code as specified in Section 2.3.3 of [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages].

Following is a non-normative example of an error response:

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Bearer realm="example.com",

error="invalid\_token",

error\_description="The Access Token expired"

|  |
| --- |
| [**TOC**](#toc) |

### 5.  Self-Issued OpenID Provider

OpenID Connect supports Self-Issued OpenID Providers - personal OPs that issue self-signed ID Tokens. Self-Issued OPs use the special Issuer Identifier https://self-issued.me.

The messages used to communicate with Self-Issued OPs are mostly the same as those used to communicate with other OPs. Specifications for the few additional parameters used and for the values of some parameters in the Self-Issued case are described in Section 7 of the [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] specification.

|  |
| --- |
| [**TOC**](#toc) |

### 6.  Initiating Login at Client from Third Party

In some cases, the login flow can start at the Authorization Server or another party by contacting the Client via a stored link. The target resource at the Client can be a deep link, rather than a default landing page.

The Client MAY optionally [register (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)](#OpenID.Registration) [OpenID.Registration] an initiate\_login\_uri that can be used by the Authorization Server or another party to initiate a login for an End-User at the Client.

The Authorization Server or a third party sends a Login Initiation Request to the Client Initiation URI with the following parameters:

login\_hint

REQUIRED. Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary).

iss

REQUIRED. Issuer Identifier for the Issuer that the Client is to send the authentication request to. Its value MUST be a URL using the https scheme.

target\_link\_uri

OPTIONAL. URI that the Client is requested to redirect to after authentication. Clients MUST verify the value of the target\_link\_uri to prevent being used as an open redirector to external sites.

Other parameters MAY be sent, if defined by extensions. Any parameters used that are not understood MUST be ignored by the Client.

Clients SHOULD employ frame busting and other techniques to prevent End-Users from being logged in by third party sites without their knowledge.

|  |
| --- |
| [**TOC**](#toc) |

### 7.  Serializations

Messages are serialized using one of the following methods:

1. Query String Serialization
2. Form Serialization
3. JSON Serialization

Not all methods can be used for all messages.

|  |
| --- |
| [**TOC**](#toc) |

### 7.1.  Query String Serialization

In order to serialize the parameters using the Query String Serialization, the Client constructs the string by adding the parameters and values to the query component of a URL using the application/x-www-form-urlencoded format as defined by [[W3C.REC‑html401‑19991224] (Hors, A., Raggett, D., and I. Jacobs, “HTML 4.01 Specification,” December 1999.)](#W3C.REC-html401-19991224). Query String Serialization is typically used in HTTP GET requests.

Following is a non-normative example of such serialization:

GET /authorize?scope=openid&response\_type=code

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1

Host: server.example.com

|  |
| --- |
| [**TOC**](#toc) |

### 7.2.  Form Serialization

Parameters and their values are Form Serialized by adding the parameter names and values to the entity body of the HTTP request using the application/x-www-form-urlencoded format as defined by [[W3C.REC‑html401‑19991224] (Hors, A., Raggett, D., and I. Jacobs, “HTML 4.01 Specification,” December 1999.)](#W3C.REC-html401-19991224). Form Serialization is typically used in HTTP POST requests.

Following is a non-normative example of such serialization:

POST /authorize HTTP/1.1

Host: server.example.com

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

scope=openid&response\_type=code

&client\_id=s6BhdRkqt3

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

|  |
| --- |
| [**TOC**](#toc) |

### 7.3.  JSON Serialization

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. Each parameter MAY have a JSON structure as its value.

Following is a non-normative example of such serialization:

{

"access\_token":"SlAV32hkKG",

"expires\_in":3600,

"refresh\_token":"8xLOxBtZp8"

}

|  |
| --- |
| [**TOC**](#toc) |

### 8.  Implementation Considerations

This specification defines features used by both Relying Parties and OpenID Providers. Features that are mandatory to implement for Relying Parties are already described in the [OpenID Connect Basic Client Profile 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “OpenID Connect Basic Client Profile 1.0,” May 2013.)](#OpenID.Basic) [OpenID.Basic] and [OpenID Connect Implicit Client Profile 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Implicit Client Profile 1.0,” May 2013.)](#OpenID.Implicit) [OpenID.Implicit] specifications, and so are not discussed again here.

All OpenID Providers MUST implement the features that are already listed elsewhere in this specification as being "REQUIRED" or are described with a "MUST". Since this specification is a protocol binding for the [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages] specification, all the implementation considerations described in that specification also apply here.

|  |
| --- |
| [**TOC**](#toc) |

### 8.1.  Discovery and Registration

Some OpenID Connect installations can use a pre-configured set of OpenID Providers and/or Relying Parties. In those cases, it might not be necessary to support dynamic discovery of information about identities or services or dynamic registration of Clients.

However, if installations choose to support unanticipated interactions between Relying Parties and OpenID Providers that do not have pre-configured relationships, they SHOULD accomplish this by implementing the facilities defined in the [OpenID Connect Discovery 1.0 (Sakimura, N., Bradley, J., Jones, M., and E. Jay, “OpenID Connect Discovery 1.0,” May 2013.)](#OpenID.Discovery) [OpenID.Discovery] and [OpenID Connect Dynamic Client Registration 1.0 (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)](#OpenID.Registration) [OpenID.Registration] specifications.

|  |
| --- |
| [**TOC**](#toc) |

### 9.  Security Considerations

This specification references the security considerations defined in [OpenID Connect Messages 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Messages 1.0,” May 2013.)](#OpenID.Messages) [OpenID.Messages], Section 10 of [OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)](#RFC6749) [RFC6749], and Section 5 of [OAuth 2.0 Bearer Token Usage (Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.)](#RFC6750) [RFC6750]. Furthermore, the [OAuth 2.0 Threat Model and Security Considerations (Lodderstedt, T., McGloin, M., and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations,” January 2013.)](#RFC6819) [RFC6819] specification provides an extensive list of threats and controls that apply to this specification as well, given that it is an OAuth 2.0 binding of OpenID Connect Messages 1.0. Implementers are highly advised to read these references in detail and apply the countermeasures described therein.

In addition, the following list of attack vectors and remedies are also considered.

|  |
| --- |
| [**TOC**](#toc) |

### 9.1.  Implicit Grant Flow Threats

In the implicit grant flow, the Access Token is returned in the fragment part of the Client's redirect\_uri through HTTPS, thus it is protected between the OP and the User-Agent, and User-Agent and the RP. The only the place it can be captured is the User-Agent where the TLS session is terminated, and is possible if the User-Agent is infested by malware.

|  |
| --- |
| [**TOC**](#toc) |

### 9.2.  TLS Requirements

Implementations MUST support TLS. Which version(s) ought to be implemented will vary over time, and depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [[RFC5246] (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.)](#RFC5246) is the most recent version, but has very limited actual deployment, and might not be readily available in implementation toolkits. TLS version 1.0 [[RFC2246] (Dierks, T. and C. Allen, “The TLS Protocol Version 1.0,” January 1999.)](#RFC2246) is the most widely deployed version, and will give the broadest interoperability.

To protect against information disclosure and tampering, confidentiality protection MUST be applied using TLS with a ciphersuite that provides confidentiality and integrity protection.

Whenever TLS is used, a TLS server certificate check MUST be performed, per [RFC 6125 (Saint-Andre, P. and J. Hodges, “Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS),” March 2011.)](#RFC6125) [RFC6125].

|  |
| --- |
| [**TOC**](#toc) |

### 10.  Privacy Considerations

The UserInfo response typically contains Personally Identifiable Information. As such, End-User consent for the release of the information for the specified purpose SHOULD be obtained at or prior to the authorization time in accordance with relevant regulations. The purpose of use is typically registered in association with the redirect\_uri.

Only necessary UserInfo data should be stored at the Client and the Client SHOULD associate the received data with the purpose of use statement.

The Resource Server SHOULD make the UserInfo access log available to the End-User so that the End-User can monitor who accessed his data.

To protect the End-User from a possible correlation among Clients, the use of a Pairwise Pseudonymous Identifier (PPID) as the sub (subject) SHOULD be considered.

|  |
| --- |
| [**TOC**](#toc) |

### 11.  IANA Considerations

This document makes no requests of IANA.

|  |
| --- |
| [**TOC**](#toc) |

### 12.  References

|  |
| --- |
| [**TOC**](#toc) |

### 12.1. Normative References

|  |  |
| --- | --- |
| **[JWE]** | Jones, M., Rescorla, E., and J. Hildebrand, “[JSON Web Encryption (JWE)](http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption),” draft-ietf-jose-json-web-encryption (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-11)). |
| **[JWS]** | Jones, M., Bradley, J., and N. Sakimura, “[JSON Web Signature (JWS)](http://tools.ietf.org/html/draft-ietf-jose-json-web-signature),” draft-ietf-jose-json-web-signature (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-11)). |
| **[JWT]** | Jones, M., Bradley, J., and N. Sakimura, “[JSON Web Token (JWT)](http://tools.ietf.org/html/draft-ietf-oauth-json-web-token),” draft-ietf-oauth-json-web-token (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-08)). |
| **[OAuth.Responses]** | de Medeiros, B., Scurtescu, M., and P. Tarjan, “[OAuth 2.0 Multiple Response Type Encoding Practices](http://openid.net/specs/oauth-v2-multiple-response-types-1_0-07.html),” June 2013. |
| **[OpenID.Discovery]** | Sakimura, N., Bradley, J., Jones, M., and E. Jay, “[OpenID Connect Discovery 1.0](http://openid.net/specs/openid-connect-discovery-1_0-16.html),” May 2013. |
| **[OpenID.Messages]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “[OpenID Connect Messages 1.0](http://openid.net/specs/openid-connect-messages-1_0-19.html),” May 2013. |
| **[OpenID.Registration]** | Sakimura, N., Bradley, J., and M. Jones, “[OpenID Connect Dynamic Client Registration 1.0](http://openid.net/specs/openid-connect-registration-1_0-18.html),” May 2013. |
| **[RFC2119]** | [Bradner, S.](mailto:sob@harvard.edu), “[Key words for use in RFCs to Indicate Requirement Levels](http://tools.ietf.org/html/rfc2119),” BCP 14, RFC 2119, March 1997 ([TXT](http://www.rfc-editor.org/rfc/rfc2119.txt), [HTML](http://xml.resource.org/public/rfc/html/rfc2119.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2119.xml)). |
| **[RFC2246]** | [Dierks, T.](mailto:tdierks@certicom.com) and [C. Allen](mailto:callen@certicom.com), “[The TLS Protocol Version 1.0](http://tools.ietf.org/html/rfc2246),” RFC 2246, January 1999 ([TXT](http://www.rfc-editor.org/rfc/rfc2246.txt)). |
| **[RFC2616]** | [Fielding, R.](mailto:fielding@ics.uci.edu), [Gettys, J.](mailto:jg@w3.org), [Mogul, J.](mailto:mogul@wrl.dec.com), [Frystyk, H.](mailto:frystyk@w3.org), [Masinter, L.](mailto:masinter@parc.xerox.com), [Leach, P.](mailto:paulle@microsoft.com), and [T. Berners-Lee](mailto:timbl@w3.org), “[Hypertext Transfer Protocol -- HTTP/1.1](http://tools.ietf.org/html/rfc2616),” RFC 2616, June 1999 ([TXT](http://www.rfc-editor.org/rfc/rfc2616.txt), [PS](http://www.rfc-editor.org/rfc/rfc2616.ps), [PDF](http://www.rfc-editor.org/rfc/rfc2616.pdf), [HTML](http://xml.resource.org/public/rfc/html/rfc2616.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2616.xml)). |
| **[RFC5246]** | Dierks, T. and E. Rescorla, “[The Transport Layer Security (TLS) Protocol Version 1.2](http://tools.ietf.org/html/rfc5246),” RFC 5246, August 2008 ([TXT](http://www.rfc-editor.org/rfc/rfc5246.txt)). |
| **[RFC5646]** | Phillips, A. and M. Davis, “[Tags for Identifying Languages](http://tools.ietf.org/html/rfc5646),” BCP 47, RFC 5646, September 2009 ([TXT](http://www.rfc-editor.org/rfc/rfc5646.txt)). |
| **[RFC6125]** | Saint-Andre, P. and J. Hodges, “[Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)](http://tools.ietf.org/html/rfc6125),” RFC 6125, March 2011 ([TXT](http://www.rfc-editor.org/rfc/rfc6125.txt)). |
| **[RFC6749]** | Hardt, D., “[The OAuth 2.0 Authorization Framework](http://tools.ietf.org/html/rfc6749),” RFC 6749, October 2012 ([TXT](http://www.rfc-editor.org/rfc/rfc6749.txt)). |
| **[RFC6750]** | Jones, M. and D. Hardt, “[The OAuth 2.0 Authorization Framework: Bearer Token Usage](http://tools.ietf.org/html/rfc6750),” RFC 6750, October 2012 ([TXT](http://www.rfc-editor.org/rfc/rfc6750.txt)). |
| **[RFC6819]** | Lodderstedt, T., McGloin, M., and P. Hunt, “[OAuth 2.0 Threat Model and Security Considerations](http://tools.ietf.org/html/rfc6819),” RFC 6819, January 2013 ([TXT](http://www.rfc-editor.org/rfc/rfc6819.txt)). |
| **[W3C.REC-html401-19991224]** | Hors, A., Raggett, D., and I. Jacobs, “[HTML 4.01 Specification](http://www.w3.org/TR/1999/REC-html401-19991224),” World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 ([HTML](http://www.w3.org/TR/1999/REC-html401-19991224)). |

|  |
| --- |
| [**TOC**](#toc) |

### 12.2. Informative References

|  |  |
| --- | --- |
| **[CORS]** | Opera Software ASA, “[Cross-Origin Resource Sharing](http://www.w3.org/TR/access-control/),” July 2010. |
| **[OpenID.Basic]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “[OpenID Connect Basic Client Profile 1.0](http://openid.net/specs/openid-connect-basic-1_0-27.html),” May 2013. |
| **[OpenID.Implicit]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “[OpenID Connect Implicit Client Profile 1.0](http://openid.net/specs/openid-connect-implicit-1_0-10.html),” May 2013. |

|  |
| --- |
| [**TOC**](#toc) |

### Appendix A.  Acknowledgements

The OpenID Community would like to thank the following people for the work they've done in the drafting and editing of this specification.

Naveen Agarwal (naa@google.com), Google

Amanda Anganes ([aanganes@mitre.org](mailto:aanganes@mitre.org)), Mitre Corporation

John Bradley (ve7jtb@ve7jtb.com), Ping Identity

Brian Campbell ([bcampbell@pingidentity.com](mailto:bcampbell@pingidentity.com)), Ping Identity

Blain Cook ([romeda@gmail.com](mailto:romeda@gmail.com)), Independent

Tim Bray (tbray@textuality.com), Google

Vladimir Dzhuvinov (vladimir@nimbusds.com), Nimbus Directory Service

Torsten Lodderstedt ([t.lodderstedt@telecom.de](mailto:t.lodderstedt@telecom.de)), Deutche Telekom

Breno de Medeiros (breno@gmail.com), Google

George Fletcher (gffletch@aol.com), AOL

Roland Hedberg ([roland.hedberg@adm.umu.se](mailto:roland.hedberg@adm.umu.se)), University of Umea

Ryo Ito (ryo.ito@mixi.co.jp), mixi, Inc.

Michael B. Jones (mbj@microsoft.com), Microsoft

Hideki Nara (hdknr@ic-tact.co.jp), Tact Communications

Axel Nennker (axel.nennker@telekom.de), Deutsche Telekom

Justin Richer ([jricher@mitre.org](mailto:jricher@mitre.org)), Mitre Corporation

Nat Sakimura (n-sakimura@nri.co.jp), Nomura Research Institute, Ltd.

|  |
| --- |
| [**TOC**](#toc) |

### Appendix B.  Notices

Copyright (c) 2013 The OpenID Foundation.

The OpenID Foundation (OIDF) grants to any Contributor, developer, implementer, or other interested party a non-exclusive, royalty free, worldwide copyright license to reproduce, prepare derivative works from, distribute, perform and display, this Implementers Draft or Final Specification solely for the purposes of (i) developing specifications, and (ii) implementing Implementers Drafts and Final Specifications based on such documents, provided that attribution be made to the OIDF as the source of the material, but that such attribution does not indicate an endorsement by the OIDF.

The technology described in this specification was made available from contributions from various sources, including members of the OpenID Foundation and others. Although the OpenID Foundation has taken steps to help ensure that the technology is available for distribution, it takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this specification or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any independent effort to identify any such rights. The OpenID Foundation and the contributors to this specification make no (and hereby expressly disclaim any) warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this specification, and the entire risk as to implementing this specification is assumed by the implementer. The OpenID Intellectual Property Rights policy requires contributors to offer a patent promise not to assert certain patent claims against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

|  |
| --- |
| [**TOC**](#toc) |

### Appendix C.  Document History

[[ To be removed from the final specification ]]

-20

* Changed from using the term "byte" to either "octet" or "character".
* Fixed #836 - Clarified language describing signing and encrypting request parameter values.
* Stated that the JWS Compact Serialization and the JWE Compact Serialization are always used for JWS and JWE data structures.
* Fixed #839 - Described requirement to employ countermeasures against Cross-Site Request Forgery and Clickjacking.
* Fixed #841 - Described verification requirements for Authorization Server responses.
* Fixed #846 - Provided a more complete introduction to Standard.

-19

* Fixed #821 - Moved definition of JSON Serialization to where it's used.

-18

* Fixed #801 - Removed schema and id parameters to UserInfo Endpoint. Also fixed related issue #791 - Removed invalid\_schema error.

-17

* Fixed #715 - Delete "profile" from request object example.
* Fixed #722 - Text on "id\_token\_hint" needs to be clarified.
* Fixed #718 - Text on re-encrypting should be clearer.
* Fixed #714 - Clarified text specifying response\_type behaviors, including prohibiting the use of the "token" response\_type, since it returns no ID Token.
* Added Security Considerations section about TLS version requirements and usage.
* Removed language about supporting other transport-layer mechanisms with equivalent security to TLS.
* State that when any validations fail, any operations requiring the information that failed to correctly validate MUST be aborted and the information that failed to validate MUST NOT be used.
* Fixed #744 - Promoted max\_age to being a top-level parameter.
* Fixed #748 - Promoted claims to being a top-level parameter separate from the OpenID Request Object.
* Fixed #597 - Changed representation of omitted year in birthdate from 9999 to 0000.
* Fixed #773 - Added request\_uris registration parameter to pre-register request\_uri values. Also clarified that Request File contents may be cached.
* Deleted top-level policy\_url parameter from the Self-Issued case, since it is already a registration parameter member.
* Fixed #778 - Added registration parameter to Self-Issued request example.
* Fixed #782 - Changed uses of "\_url" in identifiers to "\_uri".
* Fixed #719 - Moved message definitions for Self-Issued OPs to the Messages spec.
* Fixed #788 - Renamed "OpenID Request Object" to "Request Object".
* Use legal acr values in examples.

-16

* Added Implementation Considerations section.
* Fixed #698 - Inconsistent use of articles.
* Fixed #655 - Specify UTF-8 as encoding scheme whenever necessary.
* To remove ambiguity in the self-issued sub computation, changed the text "the concatenation of the key values" to "the concatenation of the bytes of the UTF-8 representations of the base64url encoded key values".
* Renamed the user\_jwk Claim to sub\_jwk, paralleling the change from user\_id to sub.
* Tracked JWK parameter name changes alg -> kty, mod -> n, exp -> e.

-15

* Fixed #687 - Inconsistency between user\_id and prn claims. The fix changed these names: user\_id -> sub, user\_id\_types\_supported -> subject\_types\_supported, user\_id\_type -> subject\_type, and prn -> sub.
* Fixed #689 - Track JWT change that allows JWTs to have multiple audiences.
* Fixed #539 - Add scope for offline access.
* Re #601 add Initiating login at a client from a third party
* Made the OpenID Foundation Artifact Binding Working Group the change controller for the values registered with IANA.
* Moved OAuth error registrations from Standard to Messages since the errors are defined in Messages and not in Standard.
* Re #657 Changed id\_token to id\_token\_hint in Sec 2.3.1

-14

* Fixed #614 - Discovery - 3.2 Distinguishing between signature and integrity parameters for HMAC algorithms. This fix tracks the parameter changes made to the JWE spec in draft-ietf-jose-json-web-encryption-06. It deletes the parameters {userinfo,id\_token}\_encrypted\_response\_int. It replaces the parameters {userinfo,id\_token,request\_object,token\_endpoint}\_algs\_supported with {userinfo,id\_token,request\_object,token\_endpoint}\_signing\_alg\_values\_supported and {userinfo,id\_token,request\_object,token\_endpoint}\_encryption\_{alg,enc}\_values\_supported.
* Fixed #666 - JWS signature validation vs. verification.
* Fixed #681 - Change remaining uses of "birthday" to "birthdate".
* Fixed #669 - Inconsistent treatment of redirect\_uri parameter.
* Referenced OAuth 2.0 RFCs -- RFC 6749 and RFC 6750.

-13

* Defined means of using a self-issued OP

-12

* Updated matching of redirect URI to include query parameters to match Google's implementation

-11

* Removed claims\_in\_id\_token scope value, per decision on June 15, 2012 special working group call

-10

* Changed verified to email\_verified, per issue #564
* Added scope value claims\_in\_id\_token as a switch to indicate that the UserInfo claims should be returned in the id\_token, per issue #561
* Changed optional claim request parameter to essential, per issue #577
* Removed Check ID Endpoint, per issue #570
* Specified that parameters present in both the OpenID Request Object and the OAuth 2.0 Authorization Request MUST exactly match, per issue #575
* Changed OpenID Request Object from being specified as a JWT to being specified as a JWS signed base64url encoded JSON object, per issue #592
* Made use of the nonce REQUIRED when using the implicit flow and OPTIONAL when using the code flow, per issue #569
* Changed client.example.com to client.example.org, per issue #251
* Removed example text for generating a nonce via a signed session cookie, per issue #562
* Use standards track version of JSON Web Token spec (draft-ietf-oauth-json-web-token)

-09

* Added error interaction\_required and removed user\_mismatched, per issue #523
* Changed invalid\_request\_request\_uri to invalid\_request\_uri and invalid\_request\_redirect\_uri to invalid\_redirect\_uri, per issue #553
* Added optional id\_token to authorization request parameters, per issue #535
* Removed use of non-existent scope parameters registry, per issue #558
* Updated Notices
* Updated References

-08

* Updated version number and date
* Fix #543 Section 2.3.1.3 Request file requiring all request params to be included is false
* Fix Section 5.1 to reference Messages 2.4.1 rather than 3.3
* Added reference to CORS for JS clients to userinfo and check\_id endpoints

-07

* Removed definition and usage for assertion and claim object
* Removed Request File Registration Service
* Consistent use of End-User
* email scope allows access to the 'verified' claim
* Removed language pertaining to custom userinfo schemas
* Updated error section for refreshing access token
* Remove 'audience' parameter from Authorization Request
* Moved display=none to prompt=none
* Moved IANA considerations from Messages
* Check ID Endpoint returns only JSON
* Removed PPID scope value
* Reference Messages for validation of request object encryption and signature
* Redefined 'nonce' in Authorization Request. Changed to REQUIRED parameter.
* Changed usage of "approval" to "consent"
* Use RFC 6125 to verify TLS endpoints
* Added Privacy considerations
* Changed 'request\_uri' to require HTTPS unless the referenced content is signed and only needs to be reachable by AS
* Added hash and entropy considerations to 'request\_uri'
* Added requirement to compare user\_id from userinfo endpoint to id\_token
* Check ID Endpoint SHOULD use POST
* Changed UserInfo Error Response to augment and return OAuth 2.0 Bearer Token Error Response
* Added section about string comparison rules needed
* Added Response Encoding according to Multiple Response Types spec
* Allows only 'id\_token' for 'response\_type' parameter in Authorization Request
* Clarified redirect\_uris matching
* Added explanation of select\_account
* Changed Security Considerations to refer to corresponding section in Messages
* Check ID Endpoint uses ID Token as Access Token according to Bearer Token spec
* Update John Bradley email and affiliation for Implementer's Draft
* Removed invalid\_authorization\_code, invalid\_id\_token error codes

-06

* Reworked return type wording in section 4.4.1 per ticket #174.
* Added reference to registered return types.
* Bumped Version number and date.
* Make clear the server passes the value of nonce through untouched. Ticket #97.
* Prevent caching of request\_uri. Ticket #148.
* Add nonce to request examples. Ticket #147.
* Fixed 4.3.1.3 per ticket #150.
* Fixed 4.3.2 to remove display scopes per ticket #172.
* Make scope optional for refresh in 5.2.
* Reference messages 3.2.2 for field definitions in section 5.2.1 per ticket #159.
* Removed scopes from display value in 4.3.1 per ticket #172.
* Make "code" and "id\_token token" response types mandatory for Authorization Servers to support.

-05

* Changed check\_session to check\_id.
* schema=openid now required when requesting UserInfo.
* Removed display values popup, touch, and mobile, since not well defined.
* Resolve issue #135, clarifying that the access\_token MAY be sent in the message body.

-04

* Changes associated with renaming "Lite" to "Basic Client" and replacing "Core" and "Framework" with "Messages" and "Standard".
* Numerous cleanups, including updating references.

-03

* Added secret\_type to the Token Endpoint.
* Minor edits to the samples.

-02

* Incorporates feedback from Nat Sakimura.

-01

* First Draft that incorporates the merge of the Core and Framework specs.

|  |
| --- |
| [**TOC**](#toc) |

### Authors' Addresses

|  |  |
| --- | --- |
|  | Nat Sakimura |
|  | Nomura Research Institute, Ltd. |
| **Email:** | [n-sakimura@nri.co.jp](mailto:n-sakimura@nri.co.jp) |
|  |  |
|  | John Bradley |
|  | Ping Identity |
| **Email:** | [ve7jtb@ve7jtb.com](mailto:ve7jtb@ve7jtb.com) |
|  |  |
|  | Michael B. Jones |
|  | Microsoft |
| **Email:** | [mbj@microsoft.com](mailto:mbj@microsoft.com) |
|  |  |
|  | Breno de Medeiros |
|  | Google |
| **Email:** | [breno@google.com](mailto:breno@google.com) |
|  |  |
|  | Edmund Jay |
|  | Illumila |
| **Email:** | [ejay@mgi1.com](mailto:ejay@mgi1.com) |