



**International
Standard**

ISO/IEC 26131

**Information technology — OpenID
connect — OpenID connect core 1.0
incorporating errata set 2**

**First edition
2024-10**

Reference number
ISO/IEC 26131:2024(en)

Full document details: [ISO IEC 26131-2024](#)

 **Documents**
© ISO/IEC 2024 Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2024

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and <https://patents.iec.ch>. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by the OpenID Foundation (OIDF) (as OpenID Connect Core 1.0 incorporating errata set 2) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Abstract

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

This specification defines the core OpenID Connect functionality: authentication built on top of OAuth 2.0 and the use of Claims to communicate information about the End-User. It also describes the security and privacy considerations for using OpenID Connect.

Table of Contents

<u>1.</u>	Introduction
<u>1.1.</u>	Requirements Notation and Conventions
<u>1.2.</u>	Terminology
<u>1.3.</u>	Overview
<u>2.</u>	ID Token
<u>3.</u>	Authentication
<u>3.1.</u>	Authentication using the Authorization Code Flow
<u>3.1.1.</u>	Authorization Code Flow Steps
<u>3.1.2.</u>	Authorization Endpoint
<u>3.1.2.1.</u>	Authentication Request
<u>3.1.2.2.</u>	Authentication Request Validation
<u>3.1.2.3.</u>	Authorization Server Authenticates End-User
<u>3.1.2.4.</u>	Authorization Server Obtains End-User Consent/Authorization
<u>3.1.2.5.</u>	Successful Authentication Response
<u>3.1.2.6.</u>	Authentication Error Response
<u>3.1.2.7.</u>	Authentication Response Validation
<u>3.1.3.</u>	Token Endpoint
<u>3.1.3.1.</u>	Token Request
<u>3.1.3.2.</u>	Token Request Validation
<u>3.1.3.3.</u>	Successful Token Response
<u>3.1.3.4.</u>	Token Error Response
<u>3.1.3.5.</u>	Token Response Validation
<u>3.1.3.6.</u>	ID Token
<u>3.1.3.7.</u>	ID Token Validation
<u>3.1.3.8.</u>	Access Token Validation
<u>3.2.</u>	Authentication using the Implicit Flow
<u>3.2.1.</u>	Implicit Flow Steps
<u>3.2.2.</u>	Authorization Endpoint
<u>3.2.2.1.</u>	Authentication Request
<u>3.2.2.2.</u>	Authentication Request Validation
<u>3.2.2.3.</u>	Authorization Server Authenticates End-User
<u>3.2.2.4.</u>	Authorization Server Obtains End-User Consent/Authorization
<u>3.2.2.5.</u>	Successful Authentication Response
<u>3.2.2.6.</u>	Authentication Error Response
<u>3.2.2.7.</u>	Redirect URI Fragment Handling
<u>3.2.2.8.</u>	Authentication Response Validation
<u>3.2.2.9.</u>	Access Token Validation
<u>3.2.2.10.</u>	ID Token
<u>3.2.2.11.</u>	ID Token Validation
<u>3.3.</u>	Authentication using the Hybrid Flow
<u>3.3.1.</u>	Hybrid Flow Steps
<u>3.3.2.</u>	Authorization Endpoint

- [3.3.2.1.](#) **Authentication Request**
 - [3.3.2.2.](#) **Authentication Request Validation**
 - [3.3.2.3.](#) **Authorization Server Authenticates End-User**
 - [3.3.2.4.](#) **Authorization Server Obtains End-User Consent/Authorization**
 - [3.3.2.5.](#) **Successful Authentication Response**
 - [3.3.2.6.](#) **Authentication Error Response**
 - [3.3.2.7.](#) **Redirect URI Fragment Handling**
 - [3.3.2.8.](#) **Authentication Response Validation**
 - [3.3.2.9.](#) **Access Token Validation**
 - [3.3.2.10.](#) **Authorization Code Validation**
 - [3.3.2.11.](#) **ID Token**
 - [3.3.2.12.](#) **ID Token Validation**
- [3.3.3.](#) **Token Endpoint**
 - [3.3.3.1.](#) **Token Request**
 - [3.3.3.2.](#) **Token Request Validation**
 - [3.3.3.3.](#) **Successful Token Response**
 - [3.3.3.4.](#) **Token Error Response**
 - [3.3.3.5.](#) **Token Response Validation**
 - [3.3.3.6.](#) **ID Token**
 - [3.3.3.7.](#) **ID Token Validation**
 - [3.3.3.8.](#) **Access Token**
 - [3.3.3.9.](#) **Access Token Validation**
- [4.](#) **Initiating Login from a Third Party**
- [5.](#) **Claims**
 - [5.1.](#) **Standard Claims**
 - [5.1.1.](#) **Address Claim**
 - [5.1.2.](#) **Additional Claims**
 - [5.2.](#) **Claims Languages and Scripts**
 - [5.3.](#) **UserInfo Endpoint**
 - [5.3.1.](#) **UserInfo Request**
 - [5.3.2.](#) **Successful UserInfo Response**
 - [5.3.3.](#) **UserInfo Error Response**
 - [5.3.4.](#) **UserInfo Response Validation**
 - [5.4.](#) **Requesting Claims using Scope Values**
 - [5.5.](#) **Requesting Claims using the "claims" Request Parameter**
 - [5.5.1.](#) **Individual Claims Requests**
 - [5.5.1.1.](#) **Requesting the "acr" Claim**
 - [5.5.2.](#) **Languages and Scripts for Individual Claims**
 - [5.6.](#) **Claim Types**
 - [5.6.1.](#) **Normal Claims**
 - [5.6.2.](#) **Aggregated and Distributed Claims**
 - [5.6.2.1.](#) **Example of Aggregated Claims**
 - [5.6.2.2.](#) **Example of Distributed Claims**
 - [5.7.](#) **Claim Stability and Uniqueness**
- [6.](#) **Passing Request Parameters as JWTs**
 - [6.1.](#) **Passing a Request Object by Value**

- [6.1.1.](#) Request using the "request" Request Parameter
- [6.2.](#) Passing a Request Object by Reference
 - [6.2.1.](#) URI Referencing the Request Object
 - [6.2.2.](#) Request using the "request_uri" Request Parameter
 - [6.2.3.](#) Authorization Server Fetches Request Object
 - [6.2.4.](#) "request_uri" Rationale
- [6.3.](#) Validating JWT-Based Requests
 - [6.3.1.](#) Encrypted Request Object
 - [6.3.2.](#) Signed Request Object
 - [6.3.3.](#) Request Parameter Assembly and Validation
- [7.](#) Self-Issued OpenID Provider
 - [7.1.](#) Self-Issued OpenID Provider Discovery
 - [7.2.](#) Self-Issued OpenID Provider Registration
 - [7.2.1.](#) Providing Information with the "registration" Request Parameter
 - [7.3.](#) Self-Issued OpenID Provider Request
 - [7.4.](#) Self-Issued OpenID Provider Response
 - [7.5.](#) Self-Issued ID Token Validation
- [8.](#) Subject Identifier Types
 - [8.1.](#) Pairwise Identifier Algorithm
- [9.](#) Client Authentication
- [10.](#) Signatures and Encryption
 - [10.1.](#) Signing
 - [10.1.1.](#) Rotation of Asymmetric Signing Keys
 - [10.2.](#) Encryption
 - [10.2.1.](#) Rotation of Asymmetric Encryption Keys
- [11.](#) Offline Access
- [12.](#) Using Refresh Tokens
 - [12.1.](#) Refresh Request
 - [12.2.](#) Successful Refresh Response
 - [12.3.](#) Refresh Error Response
- [13.](#) Serializations
 - [13.1.](#) Query String Serialization
 - [13.2.](#) Form Serialization
 - [13.3.](#) JSON Serialization
- [14.](#) String Operations
- [15.](#) Implementation Considerations
 - [15.1.](#) Mandatory to Implement Features for All OpenID Providers
 - [15.2.](#) Mandatory to Implement Features for Dynamic OpenID Providers
 - [15.3.](#) Discovery and Registration
 - [15.4.](#) Mandatory to Implement Features for Relying Parties
 - [15.5.](#) Implementation Notes
 - [15.5.1.](#) Authorization Code Implementation Notes

- [15.5.2.](#) **Nonce Implementation Notes**
 - [15.5.3.](#) **Redirect URI Fragment Handling**
- Implementation Notes**
- [15.6.](#) **Compatibility Notes**
 - [15.7.](#) **Related Specifications and Implementer's Guides**
- [16.](#) **Security Considerations**
 - [16.1.](#) **Request Disclosure**
 - [16.2.](#) **Server Masquerading**
 - [16.3.](#) **Token Manufacture/Modification**
 - [16.4.](#) **Access Token Disclosure**
 - [16.5.](#) **Server Response Disclosure**
 - [16.6.](#) **Server Response Repudiation**
 - [16.7.](#) **Request Repudiation**
 - [16.8.](#) **Access Token Redirect**
 - [16.9.](#) **Token Reuse**
 - [16.10.](#) **Eavesdropping or Leaking Authorization Codes (Secondary Authenticator Capture)**
 - [16.11.](#) **Token Substitution**
 - [16.12.](#) **Timing Attack**
 - [16.13.](#) **Other Crypto Related Attacks**
 - [16.14.](#) **Signing and Encryption Order**
 - [16.15.](#) **Issuer Identifier**
 - [16.16.](#) **Implicit Flow Threats**
 - [16.17.](#) **TLS Requirements**
 - [16.18.](#) **Lifetimes of Access Tokens and Refresh**
- Tokens**
 - [16.19.](#) **Symmetric Key Entropy**
 - [16.20.](#) **Need for Signed Requests**
 - [16.21.](#) **Need for Encrypted Requests**
 - [16.22.](#) **HTTP 307 Redirects**
 - [16.23.](#) **Custom URI Schemes on iOS**
- [17.](#) **Privacy Considerations**
 - [17.1.](#) **Personally Identifiable Information**
 - [17.2.](#) **Data Access Monitoring**
 - [17.3.](#) **Correlation**
 - [17.4.](#) **Offline Access**
- [18.](#) **IANA Considerations**
 - [18.1.](#) **JSON Web Token Claims Registration**
 - [18.1.1.](#) **Registry Contents**
 - [18.2.](#) **OAuth Parameters Registration**
 - [18.2.1.](#) **Registry Contents**
 - [18.3.](#) **OAuth Extensions Error Registration**
 - [18.3.1.](#) **Registry Contents**
 - [18.4.](#) **URI Scheme Registration**
 - [18.4.1.](#) **Registry Contents**
- [19.](#) **References**
 - [19.1.](#) **Normative References**
 - [19.2.](#) **Informative References**

Appendix A. Authorization Examples

- [A.1.](#) Example using response_type=code**
- [A.2.](#) Example using response_type=id_token**
- [A.3.](#) Example using response_type=id_token token**
- [A.4.](#) Example using response_type=code id_token**
- [A.5.](#) Example using response_type=code token**
- [A.6.](#) Example using
response_type=code id_token token**
- [A.7.](#) RSA Key Used in Examples**

Information technology — OpenID Connect — OpenID Connect Core 1.0 incorporating errata set 2

1. Introduction

[TOC](#)

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 [\[RFC6749\]](#) protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

The OpenID Connect Core 1.0 specification defines the core OpenID Connect functionality: authentication built on top of OAuth 2.0 and the use of Claims to communicate information about the End-User. It also describes the security and privacy considerations for using OpenID Connect.

As background, the [OAuth 2.0 Authorization Framework](#) [RFC6749] and [OAuth 2.0 Bearer Token Usage](#) [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. Readers are expected to be familiar with these specifications.

OpenID Connect implements authentication as an extension to the OAuth 2.0 authorization process. Use of this extension is requested by Clients by including the `openid` scope value in the Authorization Request. Information about the authentication performed is returned in a [JSON Web Token \(JWT\)](#) [JWT] called an ID Token (see [Section 2](#)). OAuth 2.0 Authentication Servers implementing OpenID Connect are also referred to as OpenID Providers (OPs). OAuth 2.0 Clients using OpenID Connect are also referred to as Relying Parties (RPs).

This specification assumes that the Relying Party has already obtained configuration information about the OpenID Provider, including its Authorization Endpoint and Token Endpoint locations. This information is normally obtained via Discovery, as described in [OpenID Connect Discovery 1.0](#) [OpenID.Discovery], or may be obtained via other mechanisms.

Likewise, this specification assumes that the Relying Party has already obtained sufficient credentials and provided information needed to use the OpenID Provider. This is normally done via Dynamic Registration, as described in [OpenID Connect Dynamic Client Registration 1.0](#) [OpenID.Registration], or may be obtained via other mechanisms.

The previous versions of this specification are:

- [OpenID Connect Core 1.0 incorporating errata set 1](#) [OpenID.Core.Errata1]
- [OpenID Connect Core 1.0 \(final\)](#) [OpenID.Core.Final]

1.1. Requirements Notation and Conventions

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [RFC2119].

In the .txt version of this specification, 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. In the HTML version of this specification, values to be taken literally are indicated by the use of `this fixed-width font`.

All uses of [JSON Web Signature \(JWS\)](#) [JWS] and [JSON Web Encryption \(JWE\)](#) [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.

1.2. Terminology

TOC

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Authorization Server", "Client", "Client Authentication", "Client Identifier", "Client Secret", "Grant Type", "Protected Resource", "Redirection URI", "Refresh Token", "Resource Server", "Response Type", and "Token Endpoint" defined by [OAuth 2.0](#) [RFC6749], the terms "Claim Name", "Claim Value", "JSON Web Token (JWT)", "JWT Claims Set", and "Nested JWT" defined by [JSON Web Token \(JWT\)](#) [JWT], the terms "Base64url Encoding",

"Header Parameter", and "JOSE Header" defined by [JSON Web Signature \(JWS\)](#) [JWS], the term "User Agent" defined by [RFC 7230](#) [RFC7230], and the term "Response Mode" defined by [OAuth 2.0 Multiple Response Type Encoding Practices](#) [OAuth.Responses].

This specification also defines the following terms:

Authentication

Process used to achieve sufficient confidence in the binding between the Entity and the presented Identity.

Authentication Request

OAuth 2.0 Authorization Request using extension parameters and scopes defined by OpenID Connect to request that the End-User be authenticated by the Authorization Server, which is an OpenID Connect Provider, to the Client, which is an OpenID Connect Relying Party.

Authentication Context

Information that the Relying Party can require before it makes an entitlement decision with respect to an authentication response. Such context can include, but is not limited to, the actual authentication method used or level of assurance such as [ISO/IEC 29115](#) [ISO29115] entity authentication assurance level.

Authentication Context Class

Set of authentication methods or procedures that are considered to be equivalent to each other in a particular context.

Authentication Context Class Reference

Identifier for an Authentication Context Class.

Authorization Code Flow

OAuth 2.0 flow in which an Authorization Code is returned from the Authorization Endpoint and all tokens are returned from the Token Endpoint.

Authorization Request

OAuth 2.0 Authorization Request as defined by [RFC6749](#).

Claim

Piece of information asserted about an Entity.

Claim Type

Syntax used for representing a Claim Value. This specification defines Normal, Aggregated, and Distributed Claim Types.

Claims Provider

Server that can return Claims about an Entity.

Credential

Data presented as evidence of the right to use an identity or other resources.

End-User

Human participant.

Entity

Something that has a separate and distinct existence and that can be identified in a context. An End-User is one example of an Entity.

Essential Claim

Claim specified by the Client as being necessary to ensure a smooth authorization experience for the specific task requested by the End-User.

Hybrid Flow

OAuth 2.0 flow in which an Authorization Code is returned from the Authorization Endpoint, some tokens are returned from the Authorization Endpoint, and others are returned from the Token Endpoint.

ID Token

[JSON Web Token \(JWT\)](#) [JWT] that contains Claims about the Authentication event. It MAY contain other Claims.

Identifier

Value that uniquely characterizes an Entity in a specific context.

Identity

Set of attributes related to an Entity.

Implicit Flow

OAuth 2.0 flow in which all tokens are returned from the Authorization Endpoint and neither the Token Endpoint nor an Authorization Code are used.

Issuer

Entity that issues a set of Claims.

Issuer Identifier

Verifiable Identifier for an Issuer. An Issuer Identifier is a case-sensitive URL using the [https](#) scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.

Message

Request or a response between an OpenID Relying Party and an OpenID Provider.

OpenID Provider (OP)

OAuth 2.0 Authorization Server that is capable of Authenticating the End-User and providing Claims to a Relying Party about the Authentication event and the End-User.

Request Object

JWT that contains a set of request parameters as its Claims.

Request URI

URL that references a resource containing a Request Object. The Request URI contents **MUST** be retrievable by the Authorization Server.

Pairwise Pseudonymous Identifier (PPID)

Identifier that identifies the Entity to a Relying Party that cannot be correlated with the Entity's PPID at another Relying Party.

Personally Identifiable Information (PII)

Information that (a) can be used to identify the natural person to whom such information relates, or (b) is or might be directly or indirectly linked to a natural person to whom such information relates.

Relying Party (RP)

OAuth 2.0 Client application requiring End-User Authentication and Claims from an OpenID Provider.

Sector Identifier

Host component of a URL used by the Relying Party's organization that is an input to the computation of pairwise Subject Identifiers for that Relying Party.

Self-Issued OpenID Provider

Personal, self-hosted OpenID Provider that issues self-signed ID Tokens.

Subject Identifier

Locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client.

UserInfo Endpoint

Protected Resource that, when presented with an Access Token by the Client, returns authorized information about the End-User represented by the corresponding Authorization Grant. The UserInfo Endpoint URL MUST use the [https](#) scheme and MAY contain port, path, and query parameter components.

Validation

Process intended to establish the soundness or correctness of a construct.

Verification

Process intended to test or prove the truth or accuracy of a fact or value.

Voluntary Claim

Claim specified by the Client as being useful but not Essential for the specific task requested by the End-User.

IMPORTANT NOTE TO READERS: The terminology definitions in this section are a normative portion of this specification, imposing requirements upon implementations. All the capitalized words in the text of this specification, such as "Issuer Identifier", reference these defined terms. Whenever the reader encounters them, their definitions found in this section must be followed.

For more background on some of the terminology used, see [Internet Security Glossary, Version 2](#) [RFC4949], [ISO/IEC 29115 Entity Authentication Assurance](#) [ISO29115], and [ITU-T X.1252](#) [X.1252].

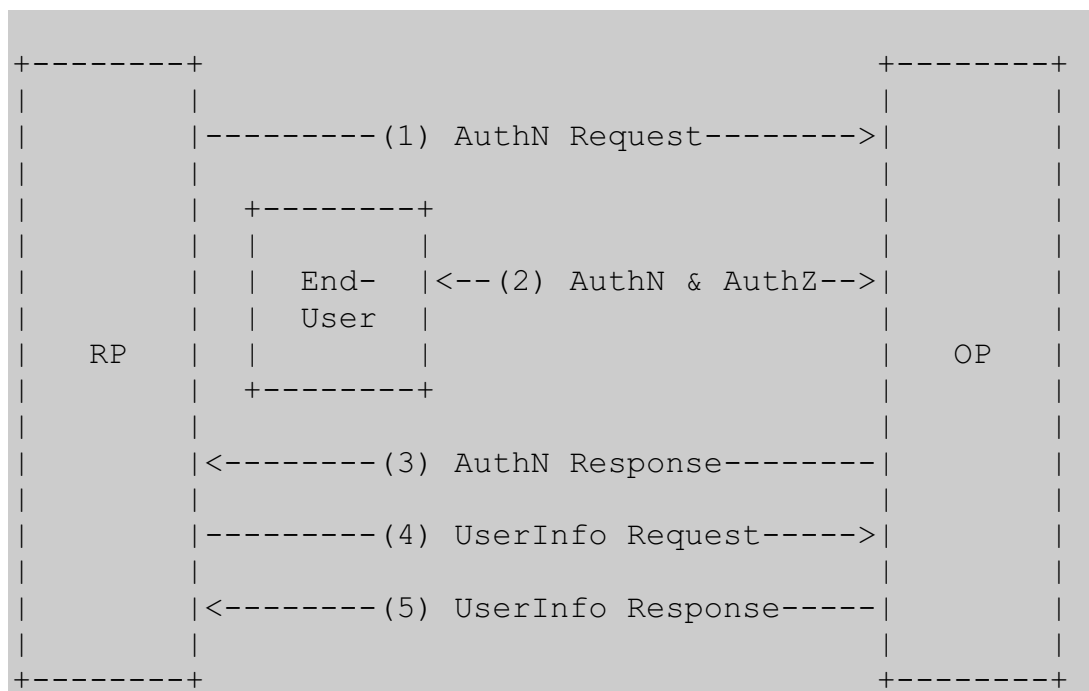
1.3. Overview

TOC

The OpenID Connect protocol, in abstract, follows the following steps.

1. The RP (Client) sends a request to the OpenID Provider (OP).
2. The OP authenticates the End-User and obtains authorization.
3. The OP responds with an ID Token and usually an Access Token.
4. The RP can send a request with the Access Token to the UserInfo Endpoint.
5. The UserInfo Endpoint returns Claims about the End-User.

These steps are illustrated in the following diagram:



2. ID Token

TOC

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

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

`iss`

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

`sub`

REQUIRED. Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client, e.g., `24400320` or `AItoawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4`. It MUST NOT exceed 255 ASCII [\[RFC20\]](#) characters in length. The `sub` value is a case-sensitive string.

`aud`

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

`exp`

REQUIRED. Expiration time on or after which the ID Token MUST NOT be accepted by the RP when performing authentication with the OP. The processing of this parameter requires that the current date/time MUST be before the expiration date/time listed in the value. Implementers MAY provide for some small leeway, usually no more than a few

minutes, to account for clock skew. Its value is a JSON [\[RFC8259\]](#) number representing the number of seconds from 1970-01-01T00:00:00Z as measured in UTC until the date/time. See [RFC 3339](#) [RFC3339] for details regarding date/times in general and UTC in particular. NOTE: The ID Token expiration time is unrelated the lifetime of the authenticated session between the RP and the OP.

iat

REQUIRED. Time at which the JWT was issued. Its value is a JSON number representing the number of seconds from 1970-01-01T00:00:00Z as measured in UTC until the date/time.

auth_time

Time when the End-User authentication occurred. Its value is a JSON number representing the number of seconds from 1970-01-01T00:00:00Z as measured in UTC until the date/time. When a [max_age](#) request is made or when [auth_time](#) is requested as an Essential Claim, then this Claim is REQUIRED; otherwise, its inclusion is OPTIONAL. (The [auth_time](#) Claim semantically corresponds to the OpenID 2.0 [PAPE](#) [OpenID.PAPE] [auth_time](#) response parameter.)

nonce

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

acr

OPTIONAL. Authentication Context Class Reference. String specifying an Authentication Context Class Reference value that identifies the Authentication Context Class that the authentication performed satisfied. The value "0" indicates the End-User authentication did not meet the requirements of [ISO/IEC 29115](#) [ISO29115] level 1. For historic reasons, the value "0" is used to indicate that there is no confidence that the same person is actually there. Authentications with

level 0 SHOULD NOT be used to authorize access to any resource of any monetary value. (This corresponds to the OpenID 2.0 [PAPE](#) [OpenID.PAPE] `nist_auth_level 0`.) An absolute URI or an [RFC 6711](#) [RFC6711] registered name SHOULD be used as the `acr` value; registered names MUST NOT be used with a different meaning than that which is registered. Parties using this claim will need to agree upon the meanings of the values used, which may be context specific. The `acr` value is a case-sensitive string.

amr

OPTIONAL. Authentication Methods References. JSON array of strings that are identifiers for authentication methods used in the authentication. For instance, values might indicate that both password and OTP authentication methods were used. The `amr` value is an array of case-sensitive strings. Values used in the `amr` Claim SHOULD be from those registered in the IANA Authentication Method Reference Values registry [\[IANA.AMR\]](#) established by [\[RFC8176\]](#); parties using this claim will need to agree upon the meanings of any unregistered values used, which may be context specific.

azp

OPTIONAL. Authorized party - the party to which the ID Token was issued. If present, it MUST contain the OAuth 2.0 Client ID of this party. The `azp` value is a case-sensitive string containing a StringOrURI value. Note that in practice, the `azp` Claim only occurs when extensions beyond the scope of this specification are used; therefore, implementations not using such extensions are encouraged to not use `azp` and to ignore it when it does occur.

ID Tokens MAY contain other Claims. Any Claims used that are not understood MUST be ignored. See Sections [3.1.3.6](#), [3.3.2.11](#), [5.1](#), and [7.4](#) for additional Claims defined by this specification.

ID Tokens MUST be signed using [JWS](#) [JWS] and optionally both signed and then encrypted using [JWS](#) [JWS] and [JWE](#) [JWE] respectively, thereby providing authentication, integrity, non-repudiation, and optionally, confidentiality, per [Section 16.14](#). If the ID Token is encrypted, it MUST be signed then encrypted, with the result being a Nested JWT, as defined in [\[JWT\]](#). ID Tokens MUST NOT use `none` as the `alg` value unless the Response Type used returns no ID Token from the Authorization Endpoint (such as when using the Authorization Code Flow) and the Client explicitly requested the use of `none` at Registration time.

ID Tokens SHOULD NOT use the JWS or JWE `x5u`, `x5c`, `jku`, or `jwk` Header Parameter fields. Instead, references to keys used are communicated in advance using Discovery and Registration parameters, per [Section 10](#).

The following is a non-normative example of the set of Claims (the JWT Claims Set) in an ID Token:

```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "auth_time": 1311280969,
  "acr": "urn:mace:incommon:iap:silver"
}
```

3. Authentication

TOC

OpenID Connect performs authentication to log in the End-User or to determine that the End-User is already logged in. OpenID Connect returns the result of the Authentication performed by the Server to the Client in a secure manner so that the Client can rely on it. For this reason, the Client is called Relying Party (RP) in this case.

The Authentication result is returned in an ID Token, as defined in [Section 2](#). It has Claims expressing such information as the Issuer, the Subject Identifier, when the authentication was performed, etc.

Authentication can follow one of three paths: the Authorization Code Flow (`response_type=code`), the Implicit Flow (`response_type=id_token` or `response_type=id_token`), or the Hybrid Flow (using other Response Type values defined in [OAuth 2.0 Multiple Response Type Encoding Practices](#) [OAuth.Responses]). The flows determine how the ID Token and Access Token are returned to the Client.

The characteristics of the three flows are summarized in the following non-normative table. The table is intended to provide some guidance on which flow to choose in particular contexts.

Property	Authorization Code Flow	Implicit Flow	Hybrid Flow
All tokens returned from Authorization Endpoint	no	yes	no
All tokens returned from Token Endpoint	yes	no	no
Tokens not revealed to User Agent	yes	no	no
Client can be authenticated	yes	no	yes
Refresh Token possible	yes	no	yes
Communication in one round trip	no	yes	no
Most communication server-to-server	yes	no	varies

OpenID Connect Authentication Flows

The flow used is determined by the `response_type` value contained in the Authorization Request. These `response_type` values select these flows:

"response_type" value	Flow
<code>code</code>	Authorization Code Flow
<code>id_token</code>	Implicit Flow
<code>id_token token</code>	Implicit Flow
<code>code id_token</code>	Hybrid Flow
<code>code token</code>	Hybrid Flow
<code>code id_token token</code>	Hybrid Flow

All but the `code` Response Type value, which is defined by [OAuth 2.0 \[RFC6749\]](#), are defined in the [OAuth 2.0 Multiple Response Type Encoding Practices](#) [OAuth.Responses] specification. NOTE: While OAuth 2.0 also defines the `token` Response Type value for the Implicit Flow, OpenID Connect does not use this Response Type, since no ID Token would be returned.

3.1. Authentication using the Authorization Code Flow

TOC

This section describes how to perform authentication using the Authorization Code Flow. When using the Authorization Code Flow, all tokens are returned from the Token Endpoint.

The Authorization Code Flow returns an Authorization Code to the Client, which can then exchange it for an ID Token and an Access Token directly. This provides the benefit of not exposing any tokens to the User Agent and possibly other malicious applications with access to the 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.

3.1.1. Authorization Code Flow Steps

TOC

The Authorization Code Flow goes through the following steps.

1. Client prepares an Authentication Request containing the desired request parameters.
2. Client sends the request to the Authorization Server.
3. Authorization Server Authenticates the End-User.

4. Authorization Server obtains 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.
 7. Client receives a response that contains an ID Token and Access Token in the response body.
 8. Client validates the ID token and retrieves the End-User's Subject Identifier.
-

3.1.2. Authorization Endpoint

TOC

The Authorization Endpoint performs Authentication of the End-User. This is done by sending the User Agent to the Authorization Server's Authorization Endpoint for Authentication and Authorization, using request parameters defined by OAuth 2.0 and additional parameters and parameter values defined by OpenID Connect.

Communication with the Authorization Endpoint MUST utilize TLS. See [Section 16.17](#) for more information on using TLS.

3.1.2.1. Authentication Request

TOC

An Authentication Request is an OAuth 2.0 Authorization Request that requests that the End-User be authenticated by the Authorization Server.

Authorization Servers MUST support the use of the HTTP `GET` and `POST` methods defined in [RFC 7231](#) [RFC7231] 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 13.1](#). If using the HTTP `POST` method, the request parameters are serialized using Form Serialization, per [Section 13.2](#).

OpenID Connect uses the following OAuth 2.0 request parameters with the Authorization Code Flow:

scope

REQUIRED. OpenID Connect requests MUST contain the `openid` scope value. If the `openid` scope value is not present, the behavior is entirely unspecified. Other scope values MAY be present. Scope values used that are not understood by an implementation SHOULD be ignored. See Sections 5.4 and 11 for additional scope values defined by this specification.

response_type

REQUIRED. OAuth 2.0 Response Type value that determines the authorization processing flow to be used, including what parameters are returned from the endpoints used. When using the Authorization Code Flow, this value is `code`.

client_id

REQUIRED. OAuth 2.0 Client Identifier valid at the Authorization Server.

redirect_uri

REQUIRED. Redirection URI to which the response will be sent. This URI MUST exactly match one of the Redirection URI values for the Client pre-registered at the OpenID Provider, with the matching performed as described in Section 6.2.1 of [RFC3986] (Simple String Comparison). When using this flow, the Redirection URI SHOULD use the `https` scheme; however, it MAY use the `http` scheme, provided that the Client Type is `confidential`, as defined in Section 2.1 of OAuth 2.0, and provided the OP allows the use of `http` Redirection URIs in this case. Also, if the Client is a native application, it MAY use the `http` scheme with `localhost` or the IP loopback literals `127.0.0.1` or `:::1` as the hostname. The Redirection URI MAY use an alternate scheme, such as one that is intended to identify a callback into a native application.

state

RECOMMENDED. Opaque value used to maintain state between the request and the callback. Typically, Cross-Site Request Forgery (CSRF, XSRF) mitigation is done by cryptographically binding the value of this parameter with a browser cookie.

OpenID Connect also uses the following OAuth 2.0 request parameter, which is defined in [OAuth 2.0 Multiple Response Type Encoding Practices](#) [OAuth.Responses]:

response_mode

OPTIONAL. Informs the Authorization Server of the mechanism to be used for returning parameters from the Authorization Endpoint. This use of this parameter is NOT RECOMMENDED when the Response Mode that would be requested is the default mode specified for the Response Type.

This specification also defines the following request parameters:

nonce

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 Authentication Request to the ID Token. Sufficient entropy MUST be present in the [nonce](#) values used to prevent attackers from guessing values. For implementation notes, see [Section 15.5.2](#).

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

The Authorization Server SHOULD display the authentication and consent UI consistent with a full User Agent page view. If the display parameter is not specified, this is the default display mode.

popup

The Authorization Server SHOULD display the authentication and consent UI consistent with a popup User Agent window. The popup User Agent window should be of an appropriate size for a login-focused dialog and should not obscure the entire window that it is popping up over.

touch

The Authorization Server SHOULD display the authentication and consent UI consistent with a device that leverages a touch interface.

wap

The Authorization Server SHOULD display the authentication and consent UI consistent with a "feature phone" type display.

The Authorization Server MAY also attempt to detect the capabilities of the User Agent and present an appropriate display.

If an OP receives a `display` value outside the set defined above that it does not understand, it MAY return an error or it MAY ignore it; in practice, not returning errors for not-understood values will help facilitate phasing in extensions using new `display` values.

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

The Authorization Server MUST NOT display any authentication or consent user interface pages. An error is returned if an End-User is not already authenticated or the Client does not have pre-configured consent for the requested Claims or does not fulfill other conditions for processing the request. The error code will typically be `login_required`, `interaction_required`, or another code defined in [Section 3.1.2.6](#). This can be used as a method to check for existing authentication and/or consent.

login

The Authorization Server SHOULD prompt the End-User for reauthentication. If it cannot reauthenticate the End-User, it MUST return an error, typically `login_required`.

consent

The Authorization Server SHOULD prompt the End-User for consent before returning information to the Client. If it cannot obtain consent, it MUST return an error, typically `consent_required`.

select_account

The Authorization Server SHOULD prompt the End-User to select a user account. This enables an End-User who has multiple accounts at the Authorization Server to select amongst the multiple accounts that they might have current sessions for. If it cannot obtain an account selection choice made by the End-User, it MUST return an error, typically `account_selection_required`.

The `prompt` parameter can be used by the Client to make sure that the End-User is still present for the current session or to bring attention to the request. If this parameter contains `none` with any other value, an error is returned.

If an OP receives a `prompt` value outside the set defined above that it does not understand, it MAY return an error or it MAY ignore it; in practice, not returning errors for not-understood values will help facilitate phasing in extensions using new `prompt` values.

max_age

OPTIONAL. Maximum Authentication Age. Specifies the allowable elapsed time in seconds since the last time the End-User was actively authenticated by the OP. If the elapsed time is greater than this value, the OP MUST attempt to actively re-authenticate the End-User. (The `max_age` request parameter corresponds to the OpenID 2.0 [PAPE](#) [OpenID.PAPE] `max_auth_age` request parameter.) When `max_age` is used, the ID Token returned MUST include an `auth_time` Claim Value. Note that `max_age=0` is equivalent to `prompt=login`.

ui_locales

OPTIONAL. End-User's preferred languages and scripts for the user interface, represented as a space-separated list of [BCP47](#) [RFC5646] language tag values, ordered by preference. For instance, the value "fr-CA fr en" represents a preference for French as spoken in Canada, then French (without a region designation), followed by English (without

a region designation). An error SHOULD NOT result if some or all of the requested locales are not supported by the OpenID Provider.

id_token_hint

OPTIONAL. ID Token previously issued by the Authorization Server being passed as a hint about the End-User's current or past authenticated session with the Client. If the End-User identified by the ID Token is already logged in or is logged in as a result of the request (with the OP possibly evaluating other information beyond the ID Token in this decision), then the Authorization Server returns a positive response; otherwise, it MUST return an error, such as `login_required`. When possible, an `id_token_hint` SHOULD be present when `prompt=none` is used and an `invalid_request` error MAY be returned if it is not; however, the server SHOULD respond successfully when possible, even if it is not present. The Authorization Server need not be listed as an audience of the ID Token when it is used as an `id_token_hint` value.

If the ID Token received by the RP from the OP is encrypted, to use it as an `id_token_hint`, the Client MUST decrypt the signed ID Token contained within the encrypted ID Token. The Client MAY re-encrypt the signed ID token to the Authentication Server using a key that enables the server to decrypt the ID Token and use the re-encrypted ID token as the `id_token_hint` value.

login_hint

OPTIONAL. Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary). This hint can be used by an RP if it first asks the End-User for their e-mail address (or other identifier) and then wants to pass that value as a hint to the discovered authorization service. It is RECOMMENDED that the hint value match the value used for discovery. This value MAY also be a phone number in the format specified for the `phone_number` Claim. The use of this parameter is left to the OP's discretion.

acr_values

OPTIONAL. Requested Authentication Context Class Reference values. Space-separated string that specifies the `acr` values that the Authorization Server is being requested to use for processing this Authentication Request, with the values appearing in order of preference. The Authentication Context Class satisfied by the authentication performed is returned as the `acr` Claim Value, as specified in [Section 2](#).

The `acr` Claim is requested as a Voluntary Claim by this parameter.

Other parameters MAY be sent. See Sections [3.2.2](#), [3.3.2](#), [5.2](#), [5.5](#), [6](#), and [7.2.1](#) for additional Authorization Request parameters and parameter values defined by this specification.

The following is a non-normative example HTTP 302 redirect response by the Client, which triggers the User Agent to make an Authentication Request to the Authorization Endpoint (with line wraps within values for display purposes only):

```
HTTP/1.1 302 Found
Location: https://server.example.com/authorize?
  response_type=code
  &scope=openid%20profile%20email
  &client_id=s6BhdRkqt3
  &state=af0ifjsldkj
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

The following is the non-normative example request that would be sent by the User Agent to the Authorization Server in response to the HTTP 302 redirect response by the Client above (with line wraps within values for display purposes only):

```
GET /authorize?
  response_type=code
  &scope=openid%20profile%20email
  &client_id=s6BhdRkqt3
  &state=af0ifjsldkj
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
HTTP/1.1
Host: server.example.com
```

3.1.2.2. Authentication Request Validation

TOC

The Authorization Server MUST validate the request received as follows:

1. The Authorization Server MUST validate all the OAuth 2.0 parameters according to the OAuth 2.0 specification.
2. Verify that a `scope` parameter is present and contains the `openid` scope value. (If no `openid` scope value is present, the request may still be a valid OAuth 2.0 request but is not an OpenID Connect request.)

3. The Authorization Server MUST verify that all the REQUIRED parameters are present and their usage conforms to this specification.
4. If the `sub` (subject) Claim is requested with a specific value for the ID Token, the Authorization Server MUST only send a positive response if the End-User identified by that `sub` value has an active session with the Authorization Server or has been Authenticated as a result of the request. The Authorization Server MUST NOT reply with an ID Token or Access Token for a different user, even if they have an active session with the Authorization Server. Such a request can be made either using an `id_token_hint` parameter or by requesting a specific Claim Value as described in [Section 5.5.1](#), if the `claims` parameter is supported by the implementation.
5. When an `id_token_hint` is present, the OP MUST validate that it was the issuer of the ID Token. The OP SHOULD accept ID Tokens when the RP identified by the ID Token has a current session or had a recent session at the OP, even when the `exp` time has passed.

As specified in [OAuth 2.0](#) [RFC6749], Authorization Servers SHOULD ignore unrecognized request parameters.

If the Authorization Server encounters any error, it MUST return an error response, per [Section 3.1.2.6](#).

3.1.2.3. Authorization Server Authenticates End-User

TOC

If the request is valid, the Authorization Server attempts to Authenticate the End-User or determines whether the End-User is Authenticated, depending upon the request parameter values used. The methods used by the Authorization Server to Authenticate 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 Authenticate the End-User in the following cases:

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

When interacting with the End-User, 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](#) [RFC6749].

3.1.2.4. Authorization Server Obtains End-User Consent/Authorization

[TOC](#)

Once the End-User is authenticated, the Authorization Server MUST obtain an authorization decision before releasing information to the Relying Party. When permitted by the request parameters used, this MAY be done through an interactive dialogue with the End-User that makes it clear what is being consented to or by establishing consent via conditions for processing the request or other means (for example, via previous administrative consent). Sections [2](#) and [5.3](#) describe information release mechanisms.

3.1.2.5. Successful Authentication Response

[TOC](#)

An Authentication Response is an OAuth 2.0 Authorization Response message returned from the OP's Authorization Endpoint in response to the Authorization Request message sent by the RP.

When using the Authorization Code Flow, the Authorization Response MUST return the parameters defined in Section 4.1.2 of [OAuth 2.0](#) [RFC6749] by adding them as query parameters to the `redirect_uri` specified in the Authorization Request using the `application/x-www-form-urlencoded` format, unless a different Response Mode was specified.

The following is a non-normative example successful response using this flow (with line wraps within values for display purposes only):

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
  code=Sp1xl0BeZQQYbYS6WxSbIA
  &state=af0ifjsldkj
```

For implementation notes on the contents of the Authorization Code, see [Section 15.5.1](#).

3.1.2.6. Authentication Error Response

TOC

An Authentication Error Response is an OAuth 2.0 Authorization Error Response message returned from the OP's Authorization Endpoint in response to the Authorization Request message sent by the RP.

If the End-User denies the request or the End-User authentication fails, the OP (Authorization Server) informs the RP (Client) by using the Error Response parameters defined in Section 4.1.2.1 of [OAuth 2.0](#) [RFC6749]. (HTTP errors unrelated to RFC 6749 are returned to the User Agent using the appropriate HTTP status code.)

Unless the Redirection URI is invalid, the Authorization Server returns the Client to the Redirection URI specified in the Authorization Request with the appropriate error and state parameters. Other parameters SHOULD NOT be returned. If the Redirection URI is invalid, the Authorization Server MUST NOT redirect the User Agent to the invalid Redirection URI.

If the Response Mode value is not supported, the Authorization Server returns an HTTP response code of 400 (Bad Request) without Error Response parameters, since understanding the Response Mode is necessary to know how to return those parameters.

In addition to the error codes defined in Section 4.1.2.1 of OAuth 2.0, this specification also defines the following error codes:

`interaction_required`

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

`login_required`

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

`account_selection_required`

The End-User is REQUIRED to select a session at the Authorization Server. The End-User MAY be authenticated at the Authorization Server with different associated accounts, but the End-User did not select a session. This error MAY be returned when the `prompt` parameter value in the Authentication Request is `none`, but the Authentication Request cannot be completed without displaying a user interface to prompt for a session to use.

`consent_required`

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

`invalid_request_uri`

The `request_uri` in the Authorization Request returns an error or contains invalid data.

`invalid_request_object`

The `request` parameter contains an invalid Request Object.

`request_not_supported`

The OP does not support use of the `request` parameter defined in [Section 6](#).

`request_uri_not_supported`

The OP does not support use of the `request_uri` parameter defined in [Section 6](#).

`registration_not_supported`

The OP does not support use of the `registration` parameter defined in [Section 7.2.1](#).

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 value received from the Client.

When using the Authorization Code Flow, the error response parameters are added to the query component of the Redirection URI, unless a different Response Mode was specified.

The following is a non-normative example error response using this flow (with line wraps within values for display purposes only):

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
  error=invalid_request
  &error_description=
    Unsupported%20response_type%20value
  &state=af0ifjsldkj
```

3.1.2.7. Authentication Response Validation

TOC

When using the Authorization Code Flow, the Client MUST validate the response according to RFC 6749, especially Sections 4.1.2 and 10.12.

3.1.3. Token Endpoint

TOC

To obtain an Access Token, an ID Token, and optionally a Refresh Token, the RP (Client) sends a Token Request to the Token Endpoint to obtain a Token Response, as described in Section 3.2 of [OAuth 2.0](#) [RFC6749], when using the Authorization Code Flow.

Communication with the Token Endpoint MUST utilize TLS. See [Section 16.17](#) for more information on using TLS.

3.1.3.1. Token Request

TOC

A Client makes a Token Request by presenting its Authorization Grant (in the form of an Authorization Code) to the Token Endpoint using the `grant_type` value `authorization_code`, as described in Section 4.1.3 of [OAuth 2.0](#) [RFC6749]. If the Client is a Confidential Client, then it MUST authenticate to the Token Endpoint using the authentication method registered for its `client_id`, as described in [Section 9](#).

The Client sends the parameters to the Token Endpoint using the HTTP `POST` method and the Form Serialization, per [Section 13.2](#), as described in Section 4.1.3 of [OAuth 2.0](#) [RFC6749].

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

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
```

```
grant_type=authorization_code&code=Splxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

3.1.3.2. Token Request Validation

TOC

The Authorization Server MUST validate the Token Request as follows:

- Authenticate the Client if it was issued Client Credentials or if it uses another Client Authentication method, per [Section 9](#).
- Ensure the Authorization Code was issued to the authenticated Client.
- Verify that the Authorization Code is valid.
- If possible, verify that the Authorization Code has not been previously used.
- Ensure that the `redirect_uri` parameter value is identical to the `redirect_uri` parameter value that was included in the initial Authorization Request. If the `redirect_uri` parameter value is not 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).
- Verify that the Authorization Code used was issued in response to an OpenID Connect Authentication Request (so that an ID Token will be returned from the Token Endpoint).

3.1.3.3. Successful Token Response

TOC

After receiving and validating a valid and authorized Token Request from the Client, the Authorization Server returns a successful response that includes an ID Token and an Access Token. The parameters in the successful response are defined in Section 4.1.4 of [OAuth 2.0 \[RFC6749\]](#). The response uses the `application/json` media type.

The OAuth 2.0 `token_type` response parameter value MUST be `Bearer`, as specified in [OAuth 2.0 Bearer Token Usage](#) [RFC6750], unless another Token Type has been negotiated with the Client. Servers SHOULD support the `Bearer` Token Type; use of other Token Types is outside the scope of this specification. Note that the `token_type` value is case insensitive.

In addition to the response parameters specified by OAuth 2.0, the following parameters MUST be included in the response:

`id_token`

ID Token value associated with the authenticated session.

All Token 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

HTTP Response Headers and Values

The following is a non-normative example of a successful Token Response. The ID Token signature in the example can be verified with the key at [Appendix A.7](#).

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

{
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "refresh_token": "8xLOxBtZp8",
  "expires_in": 3600,
  "id_token":
  "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
```

```

yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiM
jQ4Mjg5

NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAi
bi0wUzZ

fV3pBMk1qIiwKICJleHAIiOiAxMzExMjg5OTcwLAogImIhdCI6IDEzMTE
yODA5Nz

AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU9
6Sr_P6q
    Jp6IcmD3HP99Obi1PRs-cwh3LO-
p146waJ8IhehcwL7F09JdiJmBqkvPeB2T9CJ
    NqeGpe-
gccMg4vfKjkm8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd

QyHE5lcMiKPXfEIQLVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJ
bOEoRoS

K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEE
BLuVVk4
    XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
}

```

As specified in [OAuth 2.0](#) [RFC6749], Clients SHOULD ignore unrecognized response parameters.

3.1.3.4. Token Error Response

TOC

If the Token Request is invalid or unauthorized, the Authorization Server constructs the error response. The parameters of the Token Error Response are defined as in Section 5.2 of [OAuth 2.0](#) [RFC6749]. The HTTP response body uses the `application/json` media type with HTTP response code of 400.

The following is a non-normative example Token Error Response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "invalid_request"
}

```

3.1.3.5. Token Response Validation

TOC

The Client MUST validate the Token Response as follows:

1. Follow the validation rules in RFC 6749, especially those in Sections 5.1 and 10.12.
 2. Follow the ID Token validation rules in [Section 3.1.3.7](#).
 3. Follow the Access Token validation rules in [Section 3.1.3.8](#).
-

3.1.3.6. ID Token

TOC

The contents of the ID Token are as described in [Section 2](#). When using the Authorization Code Flow, these additional requirements for the following ID Token Claims apply:

`at_hash`

OPTIONAL. Access Token hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the `access_token` value, where the hash algorithm used is the hash algorithm used in the `alg` Header Parameter of the ID Token's JOSE Header. For instance, if the `alg` is `RS256`, hash the `access_token` value with SHA-256, then take the left-most 128 bits and base64url-encode them. The `at_hash` value is a case-sensitive string.

3.1.3.7. ID Token Validation

TOC

Clients MUST validate the ID Token in the Token Response in the following manner:

1. If the ID Token is encrypted, decrypt it using the keys and algorithms that the Client specified during Registration that the OP was to use to encrypt the ID Token. If encryption was negotiated with the OP at Registration time and the ID Token is not encrypted, the RP SHOULD reject it.

2. The Issuer Identifier for the OpenID Provider (which is typically obtained during Discovery) MUST exactly match the value of the `iss` (issuer) Claim.
3. The Client MUST validate that the `aud` (audience) Claim contains its `client_id` value registered at the Issuer identified by the `iss` (issuer) Claim as an audience. The `aud` (audience) Claim MAY contain an array with more than one element. The ID Token MUST be rejected if the ID Token does not list the Client as a valid audience, or if it contains additional audiences not trusted by the Client.
4. If the implementation is using extensions (which are beyond the scope of this specification) that result in the `azp` (authorized party) Claim being present, it SHOULD validate the `azp` value as specified by those extensions.
5. This validation MAY include that when an `azp` (authorized party) Claim is present, the Client SHOULD verify that its `client_id` is the Claim Value.
6. If the ID Token is received via direct communication between the Client and the Token Endpoint (which it is in this flow), the TLS server validation MAY be used to validate the issuer in place of checking the token signature. The Client MUST validate the signature of all other ID Tokens according to [JWS](#) [JWS] using the algorithm specified in the JWT `alg` Header Parameter. The Client MUST use the keys provided by the Issuer.
7. The `alg` value SHOULD be the default of `RS256` or the algorithm sent by the Client in the `id_token_signed_response_alg` parameter during Registration.
8. If the JWT `alg` Header Parameter uses a MAC based algorithm such as `HS256`, `HS384`, or `HS512`, the octets of the UTF-8 [\[RFC3629\]](#) representation of the `client_secret` corresponding to the `client_id` contained in the `aud` (audience) Claim are used as the key to validate the signature. For MAC based algorithms, the behavior is unspecified if the `aud` is multi-valued.
9. The current time MUST be before the time represented by the `exp` Claim.
10. The `iat` Claim can be used to reject tokens that were issued too far away from the current time, limiting the amount of time that nonces need to be stored to prevent attacks. The acceptable range is Client specific.

- 11.If a nonce value was sent in the Authentication Request, a `nonce` Claim MUST be present and its value checked to verify that it is the same value as the one that was sent in the Authentication Request. The Client SHOULD check the `nonce` value for replay attacks. The precise method for detecting replay attacks is Client specific.
- 12.If the `acr` Claim was requested, the Client SHOULD check that the asserted Claim Value is appropriate. The meaning and processing of `acr` Claim Values is out of scope for this specification.
- 13.If the `auth_time` Claim was requested, either through a specific request for this Claim or by using the `max_age` parameter, the Client SHOULD check the `auth_time` Claim value and request re-authentication if it determines too much time has elapsed since the last End-User authentication.

3.1.3.8. Access Token Validation

TOC

When using the Authorization Code Flow, if the ID Token contains an `at_hash` Claim, the Client MAY use it to validate the Access Token in the same manner as for the Implicit Flow, as defined in [Section 3.2.2.9](#), but using the ID Token and Access Token returned from the Token Endpoint.

3.2. Authentication using the Implicit Flow

TOC

This section describes how to perform authentication using the Implicit Flow. When using the Implicit Flow, all tokens are returned from the Authorization Endpoint; the Token Endpoint is not used.

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 End-User and applications that have access to the End-User's User Agent. The Authorization Server does not perform Client Authentication.

3.2.1. Implicit Flow Steps

TOC

The Implicit Flow follows the following steps:

1. Client prepares an Authentication Request containing the desired request parameters.
2. Client sends the request to the Authorization Server.
3. Authorization Server Authenticates the End-User.
4. Authorization Server obtains End-User Consent/Authorization.
5. Authorization Server sends the End-User back to the Client with an ID Token and, if requested, an Access Token.
6. Client validates the ID token and retrieves the End-User's Subject Identifier.

3.2.2. Authorization Endpoint

TOC

When using the Implicit Flow, the Authorization Endpoint is used in the same manner as for the Authorization Code Flow, as defined in [Section 3.1.2](#), with the exception of the differences specified in this section.

3.2.2.1. Authentication Request

TOC

Authentication Requests are made as defined in [Section 3.1.2.1](#), except that these Authentication Request parameters are used as follows:

`response_type`

REQUIRED. OAuth 2.0 Response Type value that determines the authorization processing flow to be used, including what parameters are returned from the endpoints used. When using the Implicit Flow, this value is `id_token token` or `id_token`. The meanings of both of these values are defined in [OAuth 2.0 Multiple Response Type Encoding Practices](#)

[OAuth.Responses]. No Access Token is returned when the value is `id_token`.

NOTE: While OAuth 2.0 also defines the `token` Response Type value for the Implicit Flow, OpenID Connect does not use this Response Type, since no ID Token would be returned.

`redirect_uri`

REQUIRED. Redirection URI to which the response will be sent. This URI MUST exactly match one of the Redirection URI values for the Client pre-registered at the OpenID Provider, with the matching performed as described in Section 6.2.1 of [\[RFC3986\]](#) (Simple String Comparison). When using this flow, the Redirection URI MUST NOT use the `http` scheme unless the Client is a native application, in which case it MAY use the `http` scheme with `localhost` or the IP loopback literals `127.0.0.1` or `:::1` as the hostname.

`nonce`

REQUIRED. String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token. Sufficient entropy MUST be present in the `nonce` values used to prevent attackers from guessing values. For implementation notes, see [Section 15.5.2](#).

The following is a non-normative example request using the Implicit Flow that would be sent by the User Agent to the Authorization Server in response to a corresponding HTTP 302 redirect response by the Client (with line wraps within values for display purposes only):

```
GET /authorize?
  response_type=id_token%20token
  &client_id=s6BhdRkqt3
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=openid%20profile
  &state=af0ifjsldkj
  &nonce=n-0S6_WzA2Mj HTTP/1.1
Host: server.example.com
```

3.2.2.2. Authentication Request Validation

TOC

When using the Implicit Flow, the Authentication Request is validated in the same manner as for the Authorization Code Flow, as defined in [Section 3.1.2.2](#).

3.2.2.3. Authorization Server Authenticates End-User

TOC

When using the Implicit Flow, End-User Authentication is performed in the same manner as for the Authorization Code Flow, as defined in [Section 3.1.2.3](#).

3.2.2.4. Authorization Server Obtains End-User Consent/Authorization

TOC

When using the Implicit Flow, End-User Consent is obtained in the same manner as for the Authorization Code Flow, as defined in [Section 3.1.2.4](#).

3.2.2.5. Successful Authentication Response

TOC

When using the Implicit Flow, Authentication Responses are made in the same manner as for the Authorization Code Flow, as defined in [Section 3.1.2.5](#), with the exception of the differences specified in this section.

When using the Implicit Flow, all response parameters are added to the fragment component of the Redirection URI, as specified in [OAuth 2.0 Multiple Response Type Encoding Practices](#) [OAuth.Responses], unless a different Response Mode was specified.

These parameters are returned from the Authorization Endpoint:

`access_token`

OAuth 2.0 Access Token. This is returned unless the `response_type` value used is `id_token`.

`token_type`

OAuth 2.0 Token Type value. The value MUST be `Bearer` or another `token_type` value that the Client has negotiated with the Authorization Server. Clients implementing this profile MUST support the [OAuth 2.0 Bearer Token Usage](#) [RFC6750] specification. This profile only describes the use of bearer tokens. This is returned in the same cases as `access_token` is.

`id_token`

REQUIRED. ID Token.

`state`

OAuth 2.0 state value. REQUIRED if the `state` parameter is present in the Authorization Request. Clients MUST verify that the `state` value is equal to the value of `state` parameter in the Authorization Request.

`expires_in`

OPTIONAL. Expiration time of the Access Token in seconds since the response was generated.

Per Section 4.2.2 of [OAuth 2.0](#) [RFC6749], no `code` result is returned when using the Implicit Flow.

The following is a non-normative example of a successful response using the Implicit Flow (with line wraps for the display purposes only):

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb#
  access_token=SlAV32hkKG
  &token_type=bearer
  &id_token=eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu
... ZXso
  &expires_in=3600
  &state=af0ifjsldkj
```

3.2.2.6. Authentication Error Response

TOC

When using the Implicit Flow, Authorization Error Responses are made in the same manner as for the Authorization Code Flow, as defined in [Section 3.1.2.6](#), with the exception of the differences specified in this section.

Whenever Error Response parameters are returned, such as when End-User denies the authorization or the End-User authentication fails, the Authorization Server MUST return the error Authorization Response in the fragment component of the Redirection URI, as defined in Section 4.2.2.1 of [OAuth 2.0](#) [RFC6749] and [OAuth 2.0 Multiple Response Type Encoding Practices](#) [OAuth.Responses], unless a different Response Mode was specified.

3.2.2.7. Redirect URI Fragment Handling

TOC

Since response parameters are returned in the Redirection URI fragment value, the Client needs to have the User Agent parse the fragment encoded values and pass them on to the Client's processing logic for consumption. See [Section 15.5.3](#) for implementation notes on URI fragment handling.

3.2.2.8. Authentication Response Validation

TOC

When using the Implicit Flow, the Client MUST validate the response as follows:

1. Verify that the response conforms to Section 5 of [\[OAuth.Responses\]](#).
2. Follow the validation rules in RFC 6749, especially those in Sections 4.2.2 and 10.12.
3. Follow the ID Token validation rules in [Section 3.2.2.11](#).
4. Follow the Access Token validation rules in [Section 3.2.2.9](#), unless the `response_type` value used is `id_token`.

3.2.2.9. Access Token Validation

TOC

To validate an Access Token issued from the Authorization Endpoint with an ID Token, the Client SHOULD do the following:

1. Hash the octets of the ASCII representation of the `access_token` with the hash algorithm specified in [JWA](#) [JWA] for the `alg` Header Parameter of the ID Token's JOSE Header. For instance, if the `alg` is `RS256`, the hash algorithm used is SHA-256.
2. Take the left-most half of the hash and base64url-encode it.
3. The value of `at_hash` in the ID Token MUST match the value produced in the previous step.

3.2.2.10. ID Token

TOC

The contents of the ID Token are as described in [Section 2](#). When using the Implicit Flow, these additional requirements for the following ID Token Claims apply:

`nonce`

Use of the `nonce` Claim is REQUIRED for this flow.

`at_hash`

Access Token hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the `access_token` value, where the hash algorithm used is the hash algorithm used in the `alg` Header Parameter of the ID Token's JOSE Header. For instance, if the `alg` is `RS256`, hash the `access_token` value with SHA-256, then take the left-most 128 bits and base64url-encode them. The `at_hash` value is a case-sensitive string.

If the ID Token is issued from the Authorization Endpoint with an `access_token` value, which is the case for the `response_type` value `id_token token`, this is REQUIRED; it MAY NOT be used when no Access Token is issued, which is the case for the `response_type` value `id_token`.